Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval (INEX)

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

Editors:

Norbert Fuhr
University of Duisburg-Essen

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/

Preface

The widespread use of XML in digital libraries, product catalogues, scientific data repositories and across the Web prompted the development of appropriate retrieval (searching and browsing) methods for XML documents. This in turn led to the need to evaluate the developed XML retrieval systems.

As part of a large-scale effort to improve the efficiency of research in information retrieval and digital libraries, the INitiative for the Evaluation of XML retrieval (INEX) has started an international, coordinated effort to promote evaluation procedures for content-based XML retrieval. 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 first year of the evaluation effort, in 2002, 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 INEX Workshop, held at the Schloss Dagstuhl Research Centre, concluded the results of this large-scale effort, summarised and addressed the encountered issues and devised a workplan for the evaluation of XML retrieval systems. The workshop brought together researchers in the field of XML retrieval and, in particular, researchers who participated in INEX 2002. The workshop was organised into presentation and workshop sessions. During the presentation sessions participants had the opportunity to present their approaches to XML indexing and retrieval. The workshop sessions served 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 provided by INEX. The results of these discussions have provided valuable input for the organisation of INEX 2003. Finally, the workshops on evaluation measures aimed to provide a forum to develop guidelines and procedures for the evaluation of XML retrieval systems based on the employed relevance dimensions. As a result, the discussed evaluation metrics have been implemented and applied to the INEX 2002 submissions.

This proceeding contains a collection of papers describing the research of the INEX 2002 participants. The papers have been grouped according to the approach to XML retrieval that they report on. The categories have been defined using the following definitions:

- IR-oriented: Research groups that focus on the extension of a specific type of information retrieval (IR) model, which they have applied to standard IR test collections in the past, to deal with XML documents.
- DB-oriented: Groups that are working on extending database (DB) management systems to deal with semistructured data; most of these groups also incorporate uncertainty weights, thus producing ranked results.
- XML-specific: Groups that, instead of aiming to extend existing approaches towards XML, have developed models and systems specifically for XML. Although these groups have very different backgrounds they usually base their work on XML standards (like XSL or XPath).

In addition to the research papers, the proceeding includes an overview paper providing details of the constructed INEX test collection, its construction process and the applied evaluation metrics. Detailed evaluation results are attached in the Appendix.

We would like to thank the participating organisations and people for their contributions to the INEX test collection. Special thanks go to the DELOS Network of Excellence for Digital Libraries for partially funding INEX 2002, and the IEEE Computer Society for kindly donating their XML document collection, without which INEX would not have happened. Additional acknowledgements go to the Deutscher Akadmischer Austausch Dienst (DAAD) and The British Council, who supported INEX through their Academic Research Collaboration (ARC) Programme. We would also like to thank the staff at the Schloss Dagstuhl Research Centre for all their help and efforts in managing the logistics of this Workshop.

Norbert Fuhr
Norbert Gövert
Gabriella Kazai
Mounia Lalmas
Editors
March 2003

Table of Contents

Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2002

Norbert Gövert
University of Dortmund
Germany
goevert@ls6.cs.uni-dortmund.de

Gabriella Kazai
Queen Mary University of London
United Kingdom
gabs@dcs.qmul.ac.uk

The INitiative for the Evaluation of XML retrieval (INEX) aims at providing an infrastructure for evaluating the effectiveness of content-oriented XML retrieval. In the first round of INEX, in 2002, a test collection of real world XML documents along with standard topics and respective relevance assessments has been created. Research groups from 36 different organisations participated in this collaborative effort. In this article we describe the test collection and how it was constructed. An overview of the metrics used to evaluate the effectiveness of XML retrieval approaches and of the evaluation results of 51 submissions from the INEX 2002 participants is also provided.

1 Introduction

The INitiative for the Evaluation of XML retrieval (INEX) was set up at the beginning of 2002 with the aim to establish an infrastructure and to provide means, in the form of a large XML test collection and appropriate scoring methods, for the evaluation of content-oriented retrieval of XML documents. INEX 2002 was the first in a series of future XML retrieval evaluation efforts. As a result of a collaborative effort, during the course of 2002, INEX created an XML test collection consisting of publications of the IEEE Computer Society between 1995 and 2002, 60 topics, and graded relevance assessments. Using the constructed test collection and the developed set of evaluation metrics and procedures, the retrieval effectiveness of the participating organisations' XML retrieval approaches were evaluated and their results compared.

This paper presents an overview of INEX 2002, the constructed test collection and the developed evaluation metrics, and provides a summary of the research in XML retrieval described in detail in the remainder of the proceedings. Although this overview is intended to provide a complete account of INEX 2002, it does not aim to explain or review the underlying research concepts for the evaluation of XML retrieval. On the other hand, for completeness, we cover in this paper some material already published at the SIGIR XML Workshop in 2002 while the initiative was still in progress and which provided an introduction into INEX [2].

The paper is structured as follows. In Section 2 we provide a brief summary of the INEX participants and their systems. Section 3 outlines the evaluation task set by INEX. Section 4 provides an overview of the INEX test collection along with a description of how the collection was constructed. In Section 5 a specification of the evaluation metrics applied for INEX 2002 is given, and Section 6 summarises the evaluation results. We end with conclusions and an outlook on INEX 2003 in Section 7.

2 Participating organisations

In response to the call for participation, issued in March 2002, 49 organisations from 21 countries on four continents registered within six weeks. However, throughout the year a number of groups dropped out due to resource requirements, while a number of new groups joined the initiative at the relevance assessments stage. The final 36 active INEX 2002 groups are listed in Table 1.

Due to the diversity in the background of the participating groups, a wide range of different approaches to XML retrieval were represented within INEX 2002. Although the approaches are quite diverse, we tried to classify them using the following three categories [2]:

IR model-oriented: Research groups that focus on the extension of a specific type of information retrieval (IR) model (e. g. vector space, rule-based, logistic regression, LSI), which they have applied to standard IR test collections in the past, to deal with XML documents.

DB-oriented: Groups that are working on extending database (DB) management systems to deal with semistructured data; most of these groups also incorporate uncertainty weights, thus producing ranked results.

XML-specific: Groups that, instead of aiming to extend existing approaches towards XML, have developed models and systems specifically for XML. Although these groups have very different backgrounds they usually base their work on XML standards (like XSL, XPath or XQuery).

Table 1 shows the approaches followed by the different groups. As it can be seen, most of the retrieval approaches were pure IR, DB or XML, although a few groups combined elements from two categories.

3 The task

Evaluation initiatives for flat document retrieval in IR, such as TREC[1], include several different tracks focusing on tasks such as ad-hoc retrieval, routing, filtering, and interactive retrieval, etc. Although most of these tasks are applicable to XML document retrieval, this being the first year of the initiative, we decided to run only one track, where the task to be performed was set as the ad-hoc retrieval of XML documents. Just as in TREC, the ad-hoc task was defined with the aim to evaluate the performance of systems that search a static set of documents using a new set of topics. This task has been described as a simulation of how a library might be used, where the collection of documents is known, while the queries to be asked are unknown [13]. Compared with flat document retrieval, however, for the evaluation of the ad-hoc retrieval of XML documents, we needed to consider additional requirements.

Given the different approaches to XML document retrieval (Section 2) and the widespread development and use of XML query languages, users of XML retrieval systems are able to issue (directly or indirectly) more complex queries than those used in flat document retrieval. For example, users are able to exploit the structural nature of the data and restrict their search to specific structural elements within an XML collection. This has to be reflected in the queries used for the evaluation of such systems. Content-oriented XML retrieval systems, however, should also support queries that do not specify structural conditions. The need for this type of queries for the evaluation of XML retrieval is well published (even within this proceedings) and stems from the fact that users often do not know the exact structure of the XML documents. Taking this into account, we identified the following two types of queries to be included in the INEX ad-hoc task:

Content-and-structure (CAS) queries are topic statements that contain explicit references to the XML structure, either by restricting the context of interest or the context of certain search concepts.

Content-only (CO) queries ignore the document structure and are, in a sense, the traditional topics used in IR test collections. Their resemblance to traditional IR queries is, however, only in their appearance. They pose a challenge to XML retrieval in that the retrieval results to such queries can be (possibly overlapping) XML elements of varying granularity that fulfill the query.

The objective of the evaluation in INEX, based on the ad-hoc task, is to assess a system's retrieval effectiveness, where effectiveness is measured as a system's ability to satisfy both content and structural aspects of a user's information need and retrieve the most specific relevant document components, which are exhaustive to the topic of request and match its structural constraints.

[1] `http://trec.nist.org/`

Organisation	Retrieval approach	no of runs submitted	Assessed topics
Carnegie Mellon University	IR		07, 28
Centrum voor Wiskunde en Informatica (CWI)	DB+IR	3	02, 03, 36
CSIRO Mathematical and Information Sciences	IR	3	14, 15, 27
doctronic GmbH	IR+XML	1	43
Electronics and Telecommunications Research Institute (ETRI)	DB+XML	1	26, 58
ETH Zurich	DB+IR	1	16, 47
Florida A&M University			59
IBM Haifa Labs	IR	3	08, 09
Institut de Recherche en Informatique de Toulouse (IRIT)	IR	1	
Nara Institute of Science and Technology	IR	1	37, 38
Queen Mary University of London	IR	3	53
Queensland University of Technology	IR+XML	3	29, 60
Royal School of Library and Information Science	other	3	04, 34
Salzburg Research Forschungsgesellschaft	IR	1	
Sejong Cyber University	XML	1	25
Tarragon Consulting Corporation	IR	2	31, 33
Universität Bayreuth	DB	1	05, 06
Universität Dortmund / Universität Duisburg-Essen	IR	3	30
Université Pierre et Marie Curie	IR+XML	3	10, 45, 50
University of Amsterdam	IR	3	01, 42
University of California, Berkeley	IR	3	17, 18
University of California, Los Angeles		1	48, 49
University of Helsinki	IR		19, 51
University of Melbourne	IR	3	20, 52
University of Michigan	DB+XML	2	12, 13
University of Minnesota Duluth	IR	1	11, 46
University of North Carolina at Chapel Hill	IR	1	
University of Rostock	XML		21, 22
University of Twente	DB	3	23, 24
University of Zurich			41
Organisations joined at the relevance assessments stage:			
Dublin City University			39, 40
Ecole Nationale Supérieure des Mines de Saint-Etienne			50
Justus-Liebig-Universität Gießen			50
University of California, San Diego			32
University of East Anglia			40
University of Granada			44

Table 1: List of INEX 2002 participants

id	Publication title	Year	Size (MB)	no of articles
an	IEEE Annals of the History of Computing	1995-2001	13.2	316
cg	IEEE Computer Graphics and Applications	1995-2001	19.1	680
co	Computer	1995-2001	40.4	1 902
cs	IEEE Computational Science & Engineering	1995-1998	14.6	571
	Computing in Science & Engineering	1999-2001		
dt	IEEE Design & Test of Computers	1995-2001	13.6	539
ex	IEEE Expert	1995-1997	20.3	702
	IEEE Intelligent Systems	1998-2001		
ic	IEEE Internet Computing	1997-2001	12.2	547
it	IT Professional	1999-2001	4.7	249
mi	IEEE Micro	1995-2001	15.8	604
mu	IEEE MultiMedia	1995-2001	11.3	465
pd	IEEE Parallel & Distributed Technology	1995-1996	10.7	363
	IEEE Concurrency	1997-2000		
so	IEEE Software	1995-2001	20.9	936
tc	IEEE Transactions on Computers	1995-2002	66.1	1 042
td	IEEE Transactions on Parallel & Distributed Systems	1995-2002	58.8	765
tg	IEEE Transactions on Visualization & Computer Graphics	1995-2002	15.2	225
tk	IEEE Transactions on Knowledge and Data Engineering	1995-2002	48.1	585
tp	IEEE Transactions on Pattern Analysis & Machine Intelligence	1995-2002	62.9	1 046
ts	IEEE Transactions on Software Engineering	1995-2002	46.1	570
Total			494	12 107

Table 2: The INEX document collection

4 The test collection

Similarly to standard IR test collections, the INEX test collection consists of three parts: a set of documents, topics and relevance assessments.

4.1 Documents

The document collection was donated to INEX by the IEEE Computer Society. It consists of the fulltexts of 12 107 articles, marked up in XML, from 12 magazines and 6 transactions of the IEEE Computer Society's publications, covering the period of 1995–2002, and totalling 494 MB in size. Table 2 lists some statistics for the different publications included in the collection. Although the size of the document collection is relatively small compared with TREC, it has a suitably complex XML structure containing 192 different content models in its DTD. On average, an article contains 1 532 XML nodes, where the average depth of a node is 6.9.

All documents in the collection are tagged using XML conforming to one common schema, i. e. DTD. Figure 1 shows the overall structure of a typical article consisting of a front matter (`<fm>`), a body (`<bdy>`), and a back matter (`<bm>`). The front matter contains the article's metadata, such as title, author, publication information, and abstract. Following it is the article's body, which contains the content. The body is structured into sections (`<sec>`), sub-sections (`<ss1>`), and sub-sub-sections (`<ss2>`). These logical units start with a title, followed by a number of paragraphs. In addition, the content has markup for references (citations, tables, figures), item lists, layout (such as emphasised and bold faced text), etc. The back matter contains a bibliography and information about the authors of the article.

4.2 Topics

The topic format and the topic development procedures were based on TREC guidelines, which were modified to accommodate the two types of topics used: CO and CAS (see Section 3).

```
<article>
  <fm>
    ...
    <ti>IEEE Transactions on ...</ti>
    <atl>Construction of ...</atl>
    <au>
      <fnm>John</fnm>
      <snm>Smith</snm>
      <aff>University of ...</aff>
    </au>
    <au>...</au>
    ...
  </fm>
  <bdy>
    <sec>
      <st>Introduction</st>
      <p>...</p>
      ...
    </sec>
    <sec>
      <st>...</st>
      ...
      <ss1>...</ss1>
      <ss1>...</ss1>
      ...
    </sec>
    ...
  </bdy>
  <bm>
    <bib>
      <bb>
        <au>...</au><ti>...</ti>
        ...
      </bb>
      ...
    </bib>
  </bm>
</article>
```

Figure 1: Sketch of the structure of the typical INEX articles

```
<!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 2: Topic DTD

4.2.1 Topic format

The topic format was modified to allow the definition of containment conditions and the specification of target elements (e. g. elements that should be returned to the user). The DTD of an INEX topic is shown in Figure 2. The four main parts of a topic are the topic title, topic description, narrative and keywords.

As in TREC, the topic title is a short version of the topic description and usually consists of a number of keywords that best describe what the user is looking for. In INEX, however, the topic title serves as a summary of both content and structure related requirements of a user's information need. An INEX topic title, hence, may contain a number of different components: target elements (`<te>`), a set of search concepts (`<cw>`), and a set of context elements (`<ce>`). The combination of the latter two corresponds to a containment condition. A search concept may be represented by a set of keywords or phrases. A CO topic title consists only of `<cw>` components as, by definition, it does not specify constraints over the structure of the result elements. For CAS queries, a topic title may specify the target elements of the search and / or the context elements of given search concepts. Both target and context elements may list one or more XML elements (e. g. `<ce>abs, kwd</ce>`), which may be given by their absolute (e. g. `article/fm/au`) or abbreviated path (e. g. `//au`), or by their element type (e. g. `au`). Omitting the target or context element in a topic title indicates that there are no restrictions placed upon the type of element the search should return, or the type of element a given concept should be a subject of.

The topic description is a one- or two-sentence natural language definition of the information need. The narrative is a detailed explanation of the topic statement and a description of what makes a document / component relevant or

```
<INEX-Topic topic-id="09" query-type="CAS" ct-no="048">
  <Title>
    <te>article</te>
    <cw>non-monotonic reasoning</cw> <ce>bdy/sec</ce>
    <cw>1999 2000</cw>               <ce>hdr//yr</ce>
    <cw>-calendar</cw>               <ce>tig/atl</ce>
    <cw>belief revision</cw>
  </Title>
  <Description>
    Retrieve all articles from the years 1999-2000 that deal with works on non-
    monotonic reasoning. Do not retrieve articles that are calendar/call for papers.
  </Description>
  <Narrative>
    Retrieve all articles from the years 1999-2000 that deal with works on non-
    monotonic reasoning. Do not retrieve articles that are calendar/call for papers.
  </Narrative>
  <Keywords>
    non-monotonic reasoning belief revision
  </Keywords>
</INEX-Topic>
```

Figure 3: A CAS topic from the INEX test collection

```
<INEX-Topic topic-id="45" query-type="CO" ct-no="056">
  <Title>
    <cw>augmented reality and medicine</cw>
  </Title>
  <Description>
    How virtual (or augmented) reality can contribute to improve the medical and
    surgical practice.
  </Description>
  <Narrative>
    In order to be considered relevant, a document/component must include
    considerations about applications of computer graphics and especially augmented
    (or virtual) reality to medicine (including surgery).
  </Narrative>
  <Keywords>
    augmented virtual reality medicine surgery improve computer assisted aided image
  </Keywords>
</INEX-Topic>
```

Figure 4: A CO topic from the INEX test collection

not. The keywords component of a topic was added in INEX as a means to keep a record of the list of search terms used for retrieval during the topic development process carried out by the participating groups (see Section 4.2.2).

The three attributes of a topic are: `topic-id` (e. g. 1 to 60), `query-type` (e. g. CAS or CO), and `ct-no`, which refers to the candidate topic number (e. g. 1 to 143). Figures 3 and 4 show examples for both types of topics.

4.2.2 The topic development process

In INEX, the topics were created by the participating groups. We asked each organisation to create a set of candidate topics that were representative of what real users might ask and the type of the service that operational systems may provide. Participants were provided with guidelines to assist them in this task [5]. The guide identified the following stages of the topic creation process: (1) Creation of the initial topic statement, (2) Collection exploration, (3) Topic refinement, and (4) Topic selection. While the first three stages were carried out by the participants, the selection of the final topics was left to us.

During the first stage participants created their initial topic statements. These were treated as a user's description of his / her information need and were formed without regard to system capabilities or collection peculiarities to avoid artificial or collection-biased queries.

	CAS	CO
no of topics	30	30
total no of <cw> components	62	30
avg no of <cw> / topic title	2.06	1.0
avg no of unique words / cw	2.5	4.3
avg no of unique words / topic title	5.1	4.3
total no of <ce> components	49	0
avg no of <ce> / topic title	1.63	–
avg no of XML elements / <ce>	1.53	–
avg no of XML elements / topic title	2.5	–
no of topics with <ce> representing a fact	12	–
no of topics with <ce> representing content	6	–
no of topics with mixed <ce>	12	–
total no of topics with <te> components	25	0
avg no of XML elements / <te>	1.68	–
no of topics with <te> representing a fact	13	–
no of topics with <te> representing content	12	–
no of topics with <te> representing articles	6	–
total no of (<cw>, <ce>) pairs	49	0
avg no of (<cw>, <ce>) pairs / topic title	1.63	–
avg no of words in topic description	18.8	16.1
avg no of words in keywords component	7.06	8.7

Table 3: Statistics on CAS and CO queries in the INEX test collection

During the collection exploration stage, participants estimated the number of relevant documents / components to their candidate topics. Unlike TREC, we did not provide topic authors a retrieval system for this task, but participants used their own retrieval engines. They then judged the top 25 retrieved components and the top 100 results after performing relevance feedback. Keywords used in the retrieval runs were recorded within the topic's keywords component.

In the topic refinement stage the components of a topic were finalised ensuring coherency and that each component could be used in a stand-alone fashion (e. g. retrieval using only the topic title).

After completion of the first three stages, the candidate topics were submitted to INEX. A total of 143 candidate topics were received, of which 60 topics (30 CAS and 30 CO) were selected into the final set of topics. The selection of the final 60 topics was based on the combination of criteria, such as including equal number of CO and CAS topics, having topics that are representative of IR, DB and XML-specific search situations, balancing the load across participants for relevance assessments, and eliminating topics that were considered too ambiguous or too difficult to judge. We also aimed to include topics that were likely to retrieve diverse sets (varying granularity) of relevant components. Furthermore, we based topic selection on the estimated number of relevant components, where we selected topics with at least 2, but no more than 20 relevant items in the top 25 retrieved components. Note that due to the lack of information with respect to the estimated number of relevant components within the top 100 results after relevance feedback, this data was largely ignored during topic selection.

Table 3 shows some statistics on the final set of INEX topics. Note that these figures are different from that in [2] as a result of subsequent changes to the topics. In the statistics we differentiated between context and target elements that represent facts, such as author or title information, or content, such as the text of an article or a part of the article. Looking at the 25 CAS topics that specified target elements, we can see that more than half requested facts to be returned to the user. Furthermore, the majority of the CAS topics contained either only fact (e. g. specifying the publication year and / or the title), or a mixture of fact and content containment conditions (e. g. specifying the author and the subject of a document component).

	CAS topics	CO topics
no of documents submitted	64 024	97 947
no of documents in pools	23 375	30 275
reduction	63 %	69 %
no of components submitted	100 904	139 235
no of components in pools	47 419	60 066
reduction	53 %	57 %

Table 4: Pooling effect for CAS and CO topics

4.3 Submissions

Participating groups evaluated the final set of topics against the document collection and produced, for each topic, a ranked list of XML documents / components (result elements). The top 100 result elements from all sixty sets of ranked lists (one per topic) consisted the results of one retrieval run. Each group was allowed to submit up to three runs. The submission format and procedure is detailed in [7]. Each result element was identified using a combination of file names and XPaths. The file name and file path uniquely identified an article within the document collection, and XPath allowed the location of a given component within the XML tree of the article. The result components varied from author, title and paragraph elements through sub-section and section elements to complete articles and even journals. Associated with a result element were its retrieval rank and / or its relevance status value.

In the first round of INEX, a total of 51 runs were submitted by 25 participating organisations. 42 of the 51 submissions contained results for the CAS topics and 49 contained results for the CO topics.

For each topic, all of the results from the submissions were merged to form the pool for assessment [11]. A median sized assessment pool for CAS topics contained 1 585 document components from 749 different articles. For CO topics the median sized assessment pool contained 1 980 document components from 981 different articles. Table 4 shows the pooling effect for CAS and CO topics.

4.4 Assessments

The assessment pools were then assigned to participants for assessment; either to the original topic authors or when this was not possible, on a voluntary basis, to groups with expertise in the topic's subject area. The topics assessed by the different groups are summarised in Table 1. Note that the list excludes topics 35, 54, 55, 56, and 57 as no groups volunteered to assess them. On the other hand, we obtained multiple assessments for topics 40 and 50, which were assessed by two and three assessors, respectively. We will analyse these sets in the near future to estimate the consistency of the collected assessments.

The assessments were done along the following two dimensions:

Topical relevance, which reflects the extent to which the information contained in a document component satisfies the information need.

Component coverage, which reflects the extent to which a document component is focused on the information need, while being an informative unit.

Both these dimensions were measured using graded scales. For topical relevance we used the following four-point scale [8]:

Irrelevant (0): The document component does not contain any information about the topic of request.

Marginally relevant (1): The document component mentions the topic of request, but only in passing.

Fairly relevant (2): 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.

Highly relevant (3): The document component discusses the topic of request exhaustively. In the case of multi-faceted topics, all or most sub-themes or viewpoints are discussed.

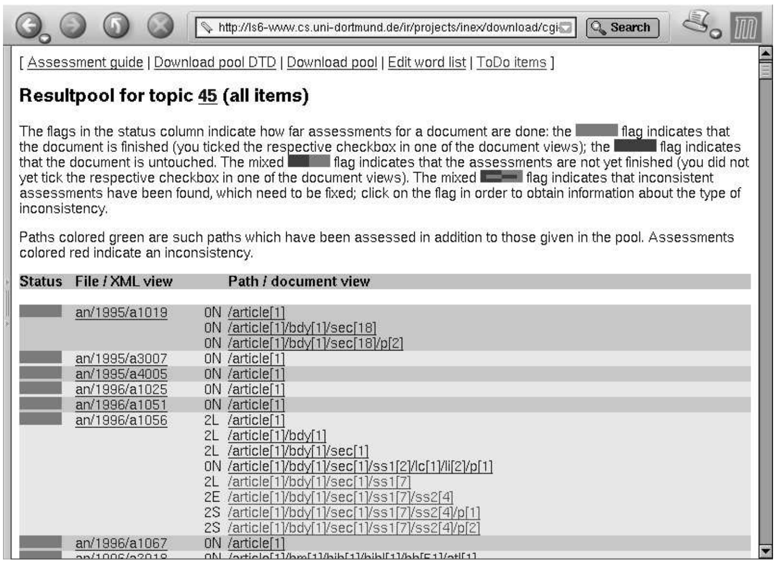


Figure 5: Result pool. Result elements are listed in alphabetical order and grouped within article elements. The relevance and coverage values are shown in front of assessed elements.

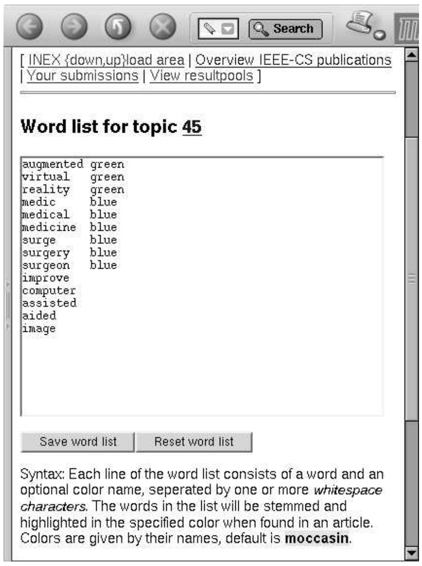


Figure 6: Word list editor. It was used by the assessors to specify a list of cue terms that were then highlighted in the document views.

Component coverage was selected from the following four categories [10]:

No coverage (N): The topic or an aspect of the topic is not a theme of the document component.

Too large (L): The topic or an aspect of the topic is only a minor theme of the document component.

Too small (S): 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.

Exact coverage (E): 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.

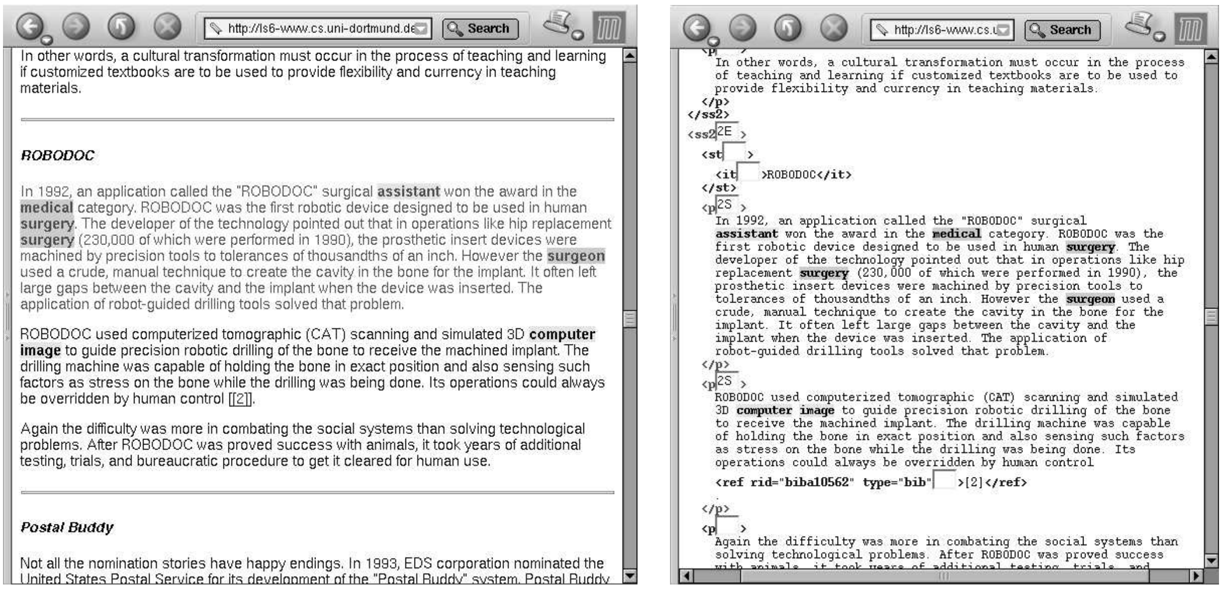
Note that the two assessed dimensions are not perfectly orthogonal to each other. Some combinations of relevance / coverage values do not make sense: A component which has no relevance cannot have any coverage with the topic. Vice versa, if a document component has no coverage with a topic, it cannot be relevant to the topic at the same time. In a similar way, a document component which has a coverage too small, cannot be highly relevant, since this would assume that all or most of the concepts requested by the topic are discussed exhaustively.

Assessors were sent detailed instructions on how to carry out the assessments based on the above two dimensions [6]. Assessments were recorded using an on-line assessment system, which allowed users to view the pooled result set of a given topic, to browse the document collection and view articles and result elements both in XML (i. e. showing the tags) and document view (i. e. formatted for ease of reading). Other features included facilities such as keyword highlighting, and consistency checking of the assessments. Figures 5, 6, and 7 show screenshots of the assessment system.

Table 5 shows a summary of the collected assessments for CAS and CO topics. Here, the relatively large proportion of non-article level elements with exact coverage compared with article elements indicates that for most topics sub-components were considered as the preferred units to be returned to a user; this is emphasised in Figure 8. Figure 9 shows the relative distribution of selected non-article XML elements that were judged relevant.

5 Evaluation metrics

Due to the nature of XML retrieval, metrics from traditional evaluation initiatives like TREC and CLEF could not be applied in INEX without modification. Therefore, it was necessary to develop new evaluation procedures. Here we



a) Document view b) XML view

Figure 7: A section of an article in document and XML view. Result elements are highlighted and cue words are marked as specified in the word list editor. Participants used the XML view to record their assessments, i. e. values of relevance and coverage for a given XML element.

Rel+ Cov	CAS topics article level	CAS topics non-articles	CO topics article level	CO topics non-articles
3E	187	2 304	307	1 087
2E	59	1 128	165	1 107
1E	82	1 770	114	827
3L	173	424	394	1 145
2L	137	507	599	2 295
1L	236	719	854	2 708
2S	21	846	118	3 825
1S	54	1 119	116	3 156
All	949	8 817	2 667	16 150

Table 5: Assessments at article and component levels

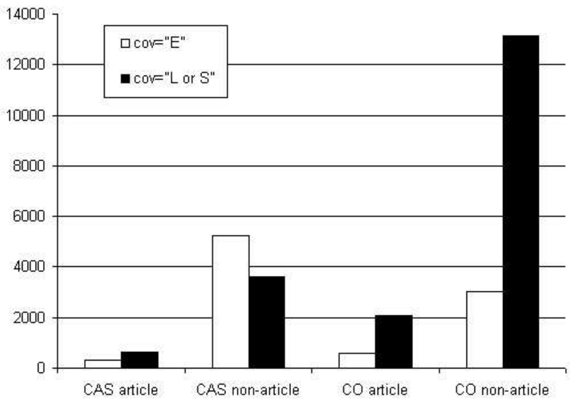


Figure 8: Distribution of relevant article and non-article elements (topical relevance > 0).

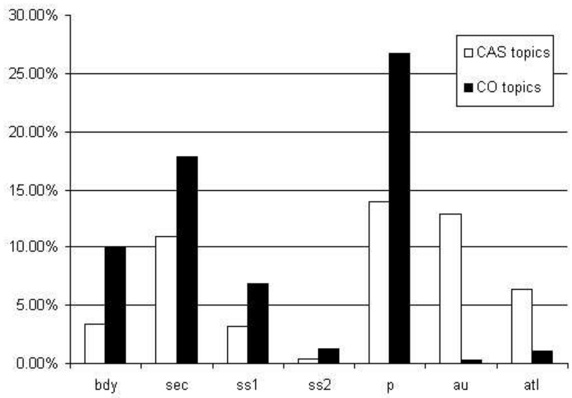


Figure 9: Distribution of relevant non-article elements (topical relevance > 0).

describe the evaluation metrics that were discussed at the INEX Workshop and have been applied to the INEX 2002 submissions. These metrics have been implemented within the `inex_eval` package, which has been distributed to the participants. In addition, a Web-based evaluation interface has also been provided for the participants.

In Section 5.1 we describe how implicit assessments have been derived from the explicit assessments done by the assessors. The evaluation metrics proposed in Section 5.3 are based on established recall / precision metrics. However, in order to apply these in INEX, the two dimensional quality assessments (see Section 4.4) first had to be quantised onto a binary relevance scale. The quantisation functions developed for this purpose are given in Section 5.2.

5.1 Implicit relevance assessments

Due to the nature of the two assessed dimensions (*topical relevance* and *component coverage*) and from the INEX quality assessment guide [6] one can, in certain cases, deduce assessments for nodes which have not been assessed explicitly:

- Due to the definition of the relevance dimension, the relevance level of a parent component of an assessed component is equal to or greater than the relevance of the assessed component.

- For a component which has a coverage assessment of *exact* or *too large* it can be deduced that its parent component has a coverage of *too large*.

These rules have been applied recursively, up to the article level of the documents, in order to add implicit assessments to the explicit assessments done by the assessors. The only exception for applying the rules are CAS topics with *target element* specifications, as it has been agreed to interpret the target element specifications in a strict way in terms of evaluation.

5.2 Quantisation of relevance and coverage

In order to apply traditional recall / precision metrics, values for the two dimensions of relevance and coverage must be quantised by some function $\mathbf{f}_{quant}$ to a single relevance value:

$$\begin{array}{rcccl} \mathbf{f}_{quant} & : & Relevance \times Coverage & \rightarrow & [0,1] \\ & & (rel, cov) & \mapsto & \mathbf{f}_{quant}(rel, cov) \end{array} \tag{1}$$

Here, the set of relevance assessments is $Relevance := \{0, 1, 2, 3\}$, and the set of coverage assessments is $Coverage := \{\mathrm{N}, \mathrm{S}, \mathrm{L}, \mathrm{E}\}$.

Quantisation functions can be selected according to the desired user standpoint. For INEX 2002, two different functions have been selected: $\mathbf{f}_{strict}$ and $\mathbf{f}_{generalised}$.

The quantisation function $\mathbf{f}_{strict}$ is used to evaluate whether a given retrieval method is capable of retrieving highly relevant and highly focused document components:

$$\mathbf{f}_{strict}(rel, cov) \quad := \quad \begin{cases} 1 & \text{if } rel = 3 \text{ and } cov = \mathrm{E}, \\ 0 & \text{else} \end{cases} \tag{2}$$

Other functions can be based on the different possible combinations of relevance degrees and coverage categories, such as $\mathbf{f}_{quant}(rel, cov) = 1$ if $rel > 1$ and $cov = \mathrm{E}$. In order to credit document components according to their *degree of* relevance (generalised recall / precision), the quantisation function $\mathbf{f}_{generalised}$ is used:

$$\mathbf{f}_{generalised}(rel, cov) \quad := \quad \begin{cases} 1.00 & \text{if } (rel, cov) = 3\mathrm{E}, \\ 0.75 & \text{if } (rel, cov) \in \{2\mathrm{E}, 3\mathrm{L}\}, \\ 0.50 & \text{if } (rel, cov) \in \{1\mathrm{E}, 2\mathrm{L}, 2\mathrm{S}\}, \\ 0.25 & \text{if } (rel, cov) \in \{1\mathrm{S}, 1\mathrm{L}\}, \\ 0.00 & \text{if } (rel, cov) = 0\mathrm{N} \end{cases} \tag{3}$$

5.3 Recall / precision metrics

Given the type of quantisation described above, each document component in a result ranking is assigned a single relevance value. In INEX 2002, overlaps of document components in rankings were ignored, thus procedures that calculate recall / precision curves for standard document retrieval could be applied directly to the results of the quantisation functions. The method described by Raghavan et al. in [9] is used for this. Here, precision is interpreted as the probability, $P(rel|retr)$, that a document viewed by a user is relevant. Given that the user stops viewing at the ranking after a given number of relevant document components NR, this probability can be computed as:

$$P(rel|retr)(NR) \quad := \quad \frac{NR}{NR + esl_{NR}} \quad = \quad \frac{NR}{NR + j + s \cdot i/(r+1)}. \tag{4}$$

The expected search length, esl_{NR}, denotes the total number of non-relevant document components that are estimated to be retrieved until the NRth relevant document is retrieved. Let l denote the rank from which the NRth relevant component is drawn. Then j is the number of non-relevant document components within the ranks before rank l, s is the number of relevant components to be taken from rank l, and r and i are the numbers of relevant and non-relevant components in rank l, respectively (details on the derivation are given by Cooper in [1]).

Raghavan et al. also gave theoretical justification, that intermediary real numbers can be used instead of simple recall points only (here, n is the total number of relevant document components with regard to the user request in the collection; $x \in [0, 1]$ denotes an arbitrary recall value):

$$P(rel|retr)(x) \quad := \quad \frac{x \cdot n}{x \cdot n + esl_{x \cdot n}} \quad = \quad \frac{x \cdot n}{x \cdot n + j + s \cdot i/(r+1)} \tag{5}$$

This leads to an intuitive method for employing arbitrary fractional numbers, x, as recall values and thus allows for averaging evaluation results over multiple topic results.

The metric from Raghavan et al. has some theoretical advantages over the metric described in [12]: besides the intuitive method for interpolation it handles weakly ordered ranks correctly. The main advantage, however, is that the variables n, j, i, r, and s in Formula 5 can be interpreted as expectations, thus allowing for a straightforward implementation of the metric for the generalised quantisation function. For example, given a function $\mathrm{assessment}(c)$, which yields the relevance / coverage assessment for a given document component c, the number n of relevant components with respect to a given topic and quantisation function is computed as:

$$n \quad = \sum_{c \in components} \mathbf{f}_{quant}(\mathrm{assessment}(c)). \tag{6}$$

Expectations for the other variables are computed respectively. Table 6 lists the number of relevant document components on a per topic basis, for both quantisation functions $\mathbf{f}_{strict}$ and $\mathbf{f}_{generalised}$.

For computation of the recall / precision curves for a given submission using Raghavan et al.'s method, it is assumed that the submission conceptually ranks all components available through the document collection. In INEX 2002, however, participants were allowed to submit 100 document components per topic only. The evaluation procedure therefore creates a virtual final rank, which enumerates all the components not being part of the set of components explicitly ranked within the submission itself. A theoretical problem which arises in the case of structured document retrieval is the question of the size of this rank (needs to be determined in order to apply Formula 5). Obviously, not every element given by the XML markup of the documents are candidates for retrievable components (most of them would be far too small to serve as a meaningful unit of information). We therefore computed a rough estimation of this figure, based on the assessments available for a given topic. For this, it is assumed that for documents where explicit assessments are available, *all retrievable* components have been assessed (explicitly or implicitly). In addition, it is assumed that retrievable components are distributed equally in all documents, regardless of the fact whether they have been assessed or not. The estimated number of retrievable components for a given topic can then be computed by:

$$|\mathrm{components}| \quad \approx \quad |\mathrm{documents}| \cdot \frac{|\mathrm{components\ assessed}|}{|\mathrm{documents\ assessed}|} \tag{7}$$

The number of components per topic in Table 6 have been computed this way.

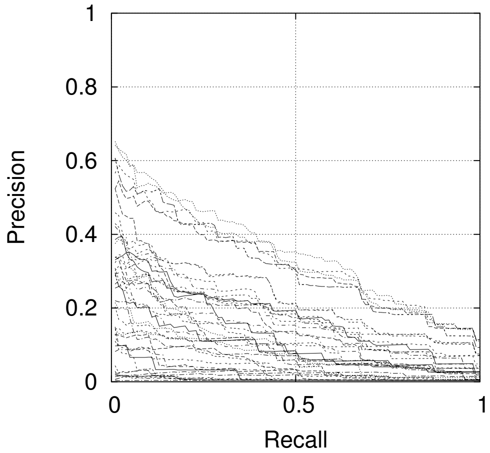
	strict		generalised	
	comp.	rel.	comp.	rel.
01	14 222	44.00	14 222	44.00
02	12 160	567.00	12 160	577.50
03	48 360	125.00	48 360	831.50
04	26 535	41.00	26 535	105.00
05	14 373	79.00	14 373	126.50
06	12 186	17.00	12 186	91.25
07	35 246	55.00	35 246	174.50
08	12 220	8.00	12 220	9.00
09	12 107	10.00	12 107	10.25
10	30 237	57.00	30 237	272.50
11	15 703	73.00	15 703	252.00
12	22 191	30.00	22 191	57.50
13	19 109	1.00	19 109	2.75
14	72 339	30.00	72 339	172.00
15	90 572	39.00	90 572	690.25
16	12 107	91.00	12 107	122.25
17	97 025	21.00	97 025	78.25
18	30 690	7.00	30 690	66.25
19	15 392	71.00	15 392	152.25
20	149 009	33.00	149 009	83.50
21	45 082	9.00	45 082	114.50
22	29 436	73.00	29 436	95.75
23	14 562	29.00	14 562	36.75
24	12 107	6.00	12 107	12.25
25	15 303	8.00	15 303	24.50
26	15 948	174.00	15 948	280.50
27	1 809 996	149.00	1 809 996	149.00
28	12 107	47.00	12 107	47.00
29	33 703	173.00	33 703	618.00
30	47 453	424.00	47 453	758.25

a) CAS topics

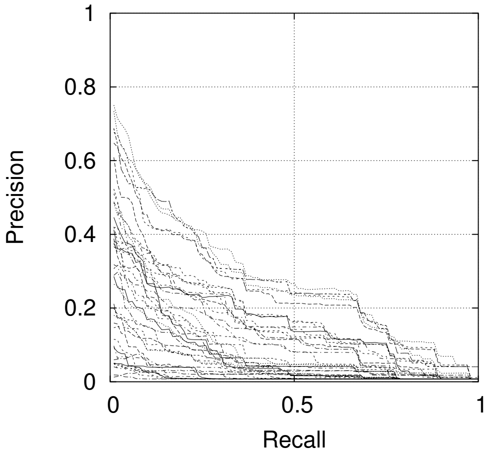
	strict		generalised	
	comp.	rel.	comp.	rel.
31	15 366	4.00	15 366	45.25
32	141 858	35.00	141 858	795.50
33	13 235	2.00	13 235	34.50
34	26 336	66.00	26 336	412.50
35	–	–	–	–
36	17 507	31.00	17 507	138.75
37	42 102	138.00	42 102	860.50
38	48 006	111.00	48 006	1 304.00
39	105 503	48.00	105 503	277.25
40	13 587	124.00	13 587	232.50
41	22 691	57.00	22 691	159.00
42	63 129	91.00	63 129	309.50
43	49 528	15.00	49 528	77.75
44	65 139	36.00	65 139	158.00
45	31 845	57.00	31 845	535.75
46	19 962	26.00	19 962	239.50
47	78 780	22.00	78 780	233.75
48	21 349	65.00	21 349	296.75
49	21 792	9.00	21 792	157.25
50	133 437	0.00	133 437	451.50
51	15 548	26.00	15 548	191.25
52	135 699	15.00	135 699	140.50
53	76 783	34.00	76 783	816.25
54	–	–	–	–
55	–	–	–	–
56	–	–	–	–
57	–	–	–	–
58	28 576	210.00	28 576	722.75
59	–	–	–	–
60	26 318	174.00	26 318	638.50

b) CO topics

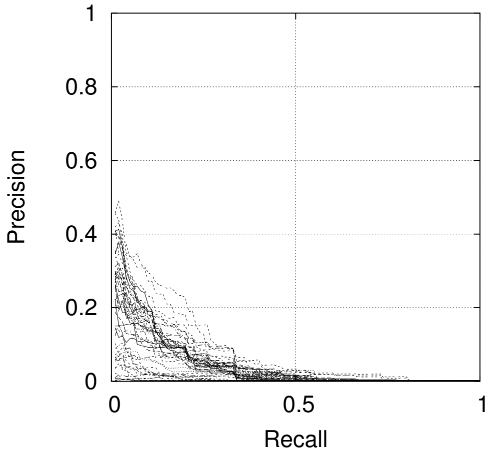
Table 6: Number of components (comp.) and relevant components (rel.) per topic, for both quantisation functions. The number of relevant components has been computed using Equation 6, while the number of components has been estimated using Equation 7.



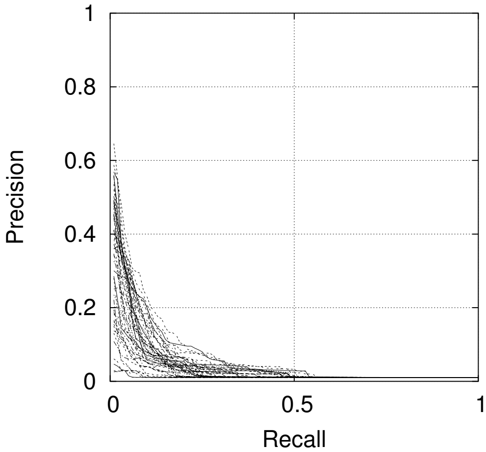
a) CAS topics; strict quantisation



b) CAS topics; generalised quantisation



c) CO topics; strict quantisation



d) CO topics; generalised quantisation

Figure 10: Summary of recall / precision curves for all INEX 2002 submissions

6 Summary of participants' results

For INEX 2002, a total of 51 runs (42 of them contained results for the CAS topics, 49 of them contained results for the CO topics) were submitted by 25 participating organisations. Figure 10 summarises the recall / precision graphs for CAS and CO topics, using the two quantisation functions $\mathbf{f}_{strict}$ and $\mathbf{f}_{generalised}$.[2]

In addition to the recall / precision curves, the `inex_eval` software computes the average precision for 100 recall points. The submissions have been ranked according to the average precision. The top ten submissions for each task and each quantisation function are displayed in Table 7. Detailed evaluation results for the runs submitted for INEX 2002 can be obtained from [4].

When comparing the rankings for the two different quantisation functions it becomes evident that they are quite similar. A regression analysis based on average precision values for the submissions shows a strong linear correlation between results obtained using strict quantisation and results obtained using generalised quantisation. Figure 11 depicts the scatter plots for CAS and CO topics and the respective regression lines. For CAS topics the correlation coefficient is 0.9943, for CO topics 0.8875.

[2] All evaluation results have been compiled using the assessment package version 1.8 and `inex_eval` version 0.007.

rank	avg precision	organisation	run ID
1.	0.3438	CSIRO Mathematical and Information Sciences	manual
2.	0.3411	IBM Haifa Labs	Merge
3.	0.3248	IBM Haifa Labs	ManualNoMerge
4.	0.3093	IBM Haifa Labs	NoMerge
5.	0.3090	University of Michigan	no-duplicate
6.	0.3090	University of Michigan	allow-duplicate
7.	0.2257	University of Amsterdam	UAmsI02NGiSt
8.	0.2233	University of Amsterdam	UAmsI02NGram
9.	0.1865	University of California, Berkeley	Berkeley03
10.	0.1839	University of Amsterdam	UAmsI02Stem

a) CAS topics; strict quantisation

rank	avg precision	organisation	run ID
1.	0.2752	CSIRO Mathematical and Information Sciences	manual
2.	0.2706	IBM Haifa Labs	Merge
3.	0.2634	University of Michigan	allow-duplicate
4.	0.2634	University of Michigan	no-duplicate
5.	0.2535	IBM Haifa Labs	ManualNoMerge
6.	0.2419	IBM Haifa Labs	NoMerge
7.	0.1782	University of Amsterdam	UAmsI02NGiSt
8.	0.1770	University of Amsterdam	UAmsI02NGram
9.	0.1592	University of Amsterdam	UAmsI02Stem
10.	0.1583	Tarragon Consulting Corporation	tgnCAS_base

b) CAS topics; generalised quantisation

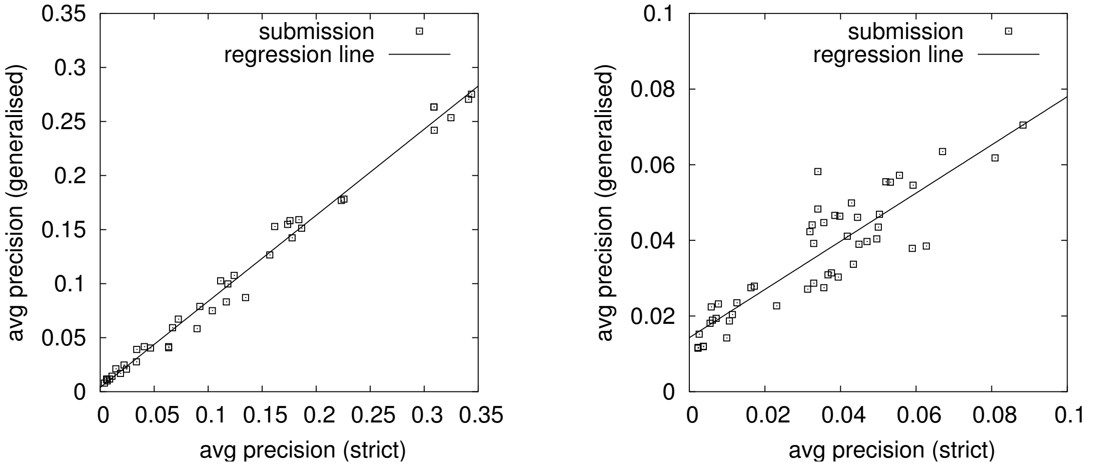
rank	avg precision	organisation	run ID
1.	0.0883	Universität Dortmund / Universität Duisburg-Essen	Epros03
2.	0.0809	Royal School of Library and Information Science	bag-of-words
3.	0.0670	Universität Dortmund / Universität Duisburg-Essen	Epros06
4.	0.0627	Queensland University of Technology	inexresult2.xml
5.	0.0592	University of Amsterdam	UAmsI02NGram
6.	0.0590	Queensland University of Technology	inexresults3.xml
7.	0.0556	Universität Dortmund / Universität Duisburg-Essen	plain hyrex
8.	0.0532	University of Amsterdam	UAmsI02NGiSt
9.	0.0520	Centrum voor Wiskunde en Informatica (CWI)	R_article
10.	0.0503	University of Minnesota Duluth	01

c) CO topics; strict quantisation

rank	avg precision	organisation	run ID
1.	0.0705	Universität Dortmund / Universität Duisburg-Essen	Epros03
2.	0.0635	Universität Dortmund / Universität Duisburg-Essen	Epros06
3.	0.0618	Royal School of Library and Information Science	bag-of-words
4.	0.0582	Sejong Cyber University	TitleKeywordsWLErr
5.	0.0572	Universität Dortmund / Universität Duisburg-Essen	plain hyrex
6.	0.0555	Centrum voor Wiskunde en Informatica (CWI)	R_article
7.	0.0554	University of Amsterdam	UAmsI02NGiSt
8.	0.0546	University of Amsterdam	UAmsI02NGram
9.	0.0499	University of Twente	utwente1pr
10.	0.0483	University of Melbourne	um_mgx2_long

d) CO topics; generalised quantisation

Table 7: Ranking of submissions w. r. t. average precision



a) CAS topics b) CO topics

Figure 11: Scatter plots and regression lines for average precision of submissions, using strict and generalised quantisation.

7 Conclusions and outlook on INEX 2003

Within the first round of INEX in 2002, as a result of a collaborative effort with research groups from 36 different organisations worldwide, an infrastructure has been created for evaluating the effectiveness of content-oriented retrieval of XML documents. A document collection with real world XML documents from the IEEE Computer Society's digital library has been set up; 60 topics were created; the INEX 2002 participants provided assessments for 55 of these topics. Based on the notion of recall and precision, a metric for evaluating the effectiveness of XML retrieval has been developed and applied for evaluating the participants' submissions.

At the time of this writing, the call for participation in the INEX 2003 round has been published already. In 2003 we aim to extend the test collection with additional topics. The retrieval task, ad-hoc retrieval with CAS and CO topics, will remain the same. However, participants now can benefit from the test collection created in 2002 and optimise their retrieval approaches accordingly. We are looking forward to many participating organisations again with a broad range of retrieval approaches, thus promoting research in the field of XML retrieval.

8 Acknowledgements

We would like to thank the DELOS Network of Excellence for Digital Libraries[3] for partially funding the INEX initiative. Special thanks go to the IEEE Computer Society[4]: Without their XML document collection INEX would not have happened. Additional acknowledgements go to *Deutscher Akadmischer Austausch Dienst (DAAD)*[5] and *The British Council*[6] who supported INEX through their Academic Research Collaboration (ARC) Programme. Last but not least, we would like to thank the participating organisations and people for their contributions to the INEX test collection.

References

[1] W. S. Cooper. Expected search length: A single measure of retrieval effectiveness based on weak ordering action of retrieval systems. *Journal of the American Society for Information Science*, 19:30–41, 1968.

[3] `http://delos-noe.org/`
[4] `http://computer.org/`
[5] `http://www.daad.de/`
[6] `http://www.britishcouncil.org/`

[2] Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas. INEX: INitiative for the Evaluation of XML retrieval. In Ricardo Baeza-Yates, Norbert Fuhr, and Yoelle S. Maarek, editors, *Proceedings of the SIGIR 2002 Workshop on XML and Information Retrieval*, 2002.

[3] Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas, editors. *INitiative for the Evaluation of XML Retrieval (INEX). Proceedings of the First INEX Workshop. Dagstuhl, Germany, December 8–11, 2002*, ERCIM Workshop Proceedings, Sophia Antipolis, France, March 2003. ERCIM.

[4] INEX. INEX 2002 evaluation results in detail. In Fuhr et al. [3].

[5] INEX. INEX guidelines for topic development. In Fuhr et al. [3].

[6] INEX. INEX relevance assessment guide. In Fuhr et al. [3].

[7] INEX. INEX retrieval result submission format. In Fuhr et al. [3].

[8] Jaana Kekäläinen and Kalvero Järvelin. Using graded relevance assessments in IR evaluation. *Journal of the American Society for Information Science and Technology*, 53(13), September 2002.

[9] V. V. Raghavan, P. Bollmann, and G. S. Jung. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems*, 7(3):205–229, 1989.

[10] Thomas Schütz. Retrieval of complex objects, considering SGML documents as example (in German). Master's thesis, University of Dortmund, Computer Science Department, 1998.

[11] K. Sparck Jones and C. J. van Rijsbergen. Report on the need for and provision of an "ideal" information retrieval test collection. Technical report, British Library Research and Development Report 5266, Computer Laboratory, University of Cambridge, 1975.

[12] trec_eval. Evaluation techniques and measures. In Voorhees and Harman [13].

[13] E. M. Voorhees and D. K. Harman, editors. *The Tenth Text REtrieval Conference (TREC 2001)*, Gaithersburg, MD, USA, 2002. NIST.

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
email: 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 probabilistic methods, using a Logistic regression algorithm for estimation of collection relevance and element relevance, 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. The official INEX results are discussed, along with some analysis of subsequent trials, and some thoughts on future directions for XML retrieval approaches for INEX.

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. 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 XML 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 [10] 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. Any valid DTD or XMLSchema can be used for the the records in the database, and multiple record types can be combined on the same server.

2. The system uses an embedded database en-

gine (BerkeleyDB) for constructing and accessing indexes and for storage of component information. The user has the option of storing the SGML/XML records un-modified as files or in a parsed form in the database engine (with some added storage overhead for these pre-parsed records).

3. 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.

4. It provides flexible document retrieval, including the ability to request any individual XPATH specification from any document selected during searching.

5. 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 supports a variety of other protocols, including OAI, SDLIP, SOAP, and SRW.

6. The system includes multiple clients, all of which are scriptable using either Tcl/Tk[7] or the Python language. All of these client interfaces permit searches of the Cheshire II search engine as well as any other z39.50, SDLIP, SOAP, or SRW compatible search engine on the network.

7. It permits users to enter natural language queries that may be combined with Boolean logic. Indexing and searching can make use of the structure of the underlying documents to provide very complex searching. Multiple searches can be performed on different elements of a document collection and the results merged into a single ranked result set.

8. It uses probabilistic ranking methods based on the Logistic Regression research carried out at Berkeley to match the user's initial query with SGML/XML documents and document components in the database.

9. 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.

The original design rationale and features of the Cheshire II search engine have been discussed elsewhere [6, 5] 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 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[8]) and support for mapping database and query text words to single forms based on the WordNet dictionary and thesaurus using a adaption of the WordNet "Morphing" algorithm and exception dictionary.

The 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 $P(R \mid Q, D)$ 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 $P(R \mid Q, D)$ is calculated as the "log odds" of relevance $\log O(R \mid Q, D)$, where for any events A and B the odds $O(A \mid B)$ is a simple transformation of the probabilities $\frac{P(A|B)}{P(\overline{A}|B)}$. The Logistic Regression model provides estimates for a set of coefficients, c_i, associated with a set of S statistics, X_i, derived from the query and database, such that

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

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:

$X_1 = \frac{1}{M}\sum_{j=1}^{M} logQAF_{t_j}$. This is the log of the absolute frequency of occurrence for term t_j in the query averaged over the M terms in common between the query and the document or document component. The coefficient c_1 used in the current version of the Cheshire II system is 1.269.

$X_2 = \sqrt{QL}$. This is square root of the query length (i.e., the number of terms in the query disregarding stopwords). The c_2 coefficient used is -0.310.

$X_3 = \frac{1}{M}\sum_{j=1}^{M} logDAF_{t_j}$. This is is the log of the absolute frequency of occurrence for term t_j in the document (or component) averaged over the M common terms. The c_3 coefficient used is 0.679.

$X_4 = \sqrt{DL}$. This is square root of the document or component length. In Cheshire II the raw size of the document or component in bytes is used for the document length. The c_4 coefficient used is -0.0674.

$X_5 = \frac{1}{M}\sum_{j=1}^{M} logIDF_{t_j}$. This is is the log of the *inverse document frequency*(IDF) for term t_j in the document averaged over the M common terms. IDF is calculated as the total number of documents or components in the database, divided by the number of documents or components that contain term t_j The c_5 coefficient used is 0.223.

$X_6 = logM$. This is the log of the number of terms that are in both the query and in the document or component. The c_6 coefficient used in Cheshire II is 2.01.

These coefficients and elements of the ranking algorithm have proven to be quite robust and useful across a broad range of document and component types.

The system, as noted above, supports searches combining 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 in a single query, combined probabilistic and Boolean search results are evaluated using the assumption that the Boolean retrieved set has an estimated $P(R \mid Q_{bool}, D) = 1.0$ for each document in the set, and 0 for the rest of the collection. The final estimate for the probability of relevance used for ranking the results of a search combining Boolean and probabilistic strategies is simply:

$$P(R \mid Q, D) = P(R \mid Q_{bool}, D)P(R \mid Q_{prob}, D) \quad (2)$$

where $P(R \mid Q_{prob}, D)$ is the probability estimate from the probabilistic portion of the search, and $P(R \mid Q_{bool}, D)$ the estimate from the Boolean. This has the effect of restricting the results to those items that match the Boolean portion, with ordering based on the probabilistic portion.

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, we hope to leverage the strengths and to 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[9]. 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.

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 typically will 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 in the current implementation this final ranking value is not a an estimated probability but a combination of probabilistic weights and weighted Boolean values.

Relevance feedback is available 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. However it was not used in the INEX evaluation where the searches were done as a batch process.

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 INEX was to use all of the features of the cheshire system required to support the searches produced by the participants 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 described 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 INEX). Some more specialized extraction methods include DATE and DATETIME extraction, INTEGER, FLOAT and DECIMAL extraction, as well as extraction methods for geographic coordinates.

3. NORMAL: The type of normalization applied to the data extracted from the text nodes. The most commonly used are STEM and NONE. STEM uses an enhanced version of the Porter stemmer, and NONE (in spite of the name) performs case-folding. Specialized normalization routines for different date, datetime and geographic coordinate formats can also be specified.

Each index can have its own specialized stopword list, so that, for example, corporate names have a different set of stopwords from document titles or personal names.

Most of the indexes used in INEX used KEYWORD extraction and STEMming of the keyword tokens. Exceptions to this general rule were date elements (which were extracted using DATE extraction of the year only) and the names of authors which were extracted without stemming or stoplists to retain the full name.

Table 1 lists the document-level (//article) indexes created for INEX and the document elements from which the contents of those indexes were extracted. 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 reindexing 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.

As noted above the Cheshire system permits parts of the document subtree to be treated as

Name	Description	Contents
docno	Digital Object ID	//doi
pauthor	Author Names	//fm/au/snm //fm/au/fnm
title	Article Title	//fm/tig/atl

topic	Content Words	//fm/tig/atl
		//abs
		//bdy
		//bibl/bb/atl
		//app

date	Date of Publication	//hdr2/yr
journal	Journal Title	//hdr1/ti
kwd	Article Keywords	//kwd
abstract	Article Abstract	//abs
author_seq	Author Seq.	//fm/au @sequence
bib_author _fnm	Bib Author Forename	//bb/au/fnm
bib_author _snm	Bib Author Surname	//bb/au/snm
fig	Figure Contents	//fig
ack	Acknowledgements	//ack
alltitles	All Title Elements	//atl, //st
affil	Author Affiliations	//fm/aff
fno	IEEE Article ID	//fno

Table 1: Cheshire Article-Level Indexes for INEX

Name	Description	Contents
COMP_SECTION	Sections	//sec
COMP_BIB	Bib Entries	//bib/bibl/bb

COMP_PARAS	Paragraphs	//ilrj|//ip1|//ip2|
		//ip3|//ip4|//ip5|
		//item-none|//p|
		//p1|//p2|//p3|
		//tmath|//tf

COMP_FIG	Figures	//fig

Table 2: Cheshire Components for INEX

Component or Name	Description	Contents
COMP_SECTION		
sec_title	Section Title	//sec/st
sec_words	Section Words	//sec
COMP_BIB		
bib_author	Bib. Author	//au
bib_title	Bib. Title	//atl
bib_date	Bib. Date	//pdt/yr
COMP_PARAS		
para_words	Paragraph Words	*†
COMP_FIG		
fig_caption	Figure Caption	//fgc

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

separated documents with their own separate indexes. Tables 2 & 3 describe the XML components created for INEX 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.

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 INEX are described in the next section.

3.2 The INEX Search Approach

Berkeley submitted three retrieval runs for INEX. 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 Cheshire's Fusion Search facility in merging the result sets from multiple 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 retrieval 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/atl </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 Evaluation

INEX involved assessments of the submitted results of each participating group on two (separate though related) dimensions. The dimensions were relevance (assessed on an integer scale of 0-3 with 0 being nonrelevant, 1 being passing mention of the topic, 2 being partially relevant, and 3 being completely relevant) and coverage (with the four possible values: **E** for exact desired coverage by the retrieved element, **L** if a retrieved element was too large or included too much extraneous information, **S** if it was too small or incomplete desired coverage, and **N** for no coverage. Obviously not all combinations of these were sensible (or permitted).

For the calculation of the Recall and Precision analogs used for INEX, two different quantizations of these two dimensions were used:

$$f_{strict}(rel, cov) := \begin{cases} 1 & \text{if } rel = 3 \text{ and } cov = \text{E} \\ 0 & \text{otherwise} \end{cases}$$

and

$$f_{generalized}(rel, cov) := \begin{cases} 1.00 & if \quad \text{3E} \\ 0.75 & if \quad \text{2E,3L,3S} \\ 0.50 & if \quad \text{1E,2L,2S} \\ 0.25 & if \quad \text{1S,1L} \\ 0.00 & if \quad \text{0N} \end{cases}$$

The "strict" quantization is intended to be similar to the relevance assessments used in other IR evaluations. (One could argue, however, that a closer approximation to most relevance judgements might be to consider any full document containing a 3 as "relevant", and possibly some of the 2's).

Figure 1 and Figure 2 show, respectively the Recall/Precision curves for the CAS and CO results of the three submitted Berkeley runs, under both quantizations. The only run appearing among the top 10 (when compared to other

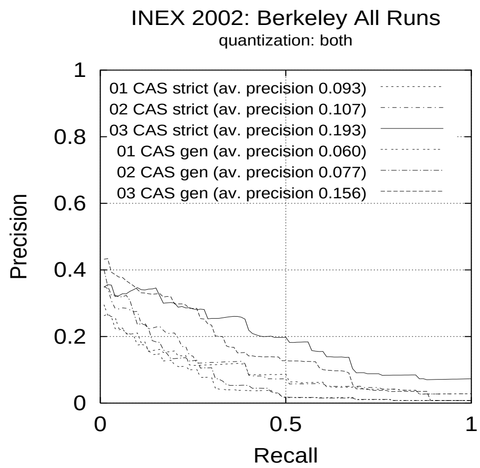


Figure 1: Berkeley CAS Runs

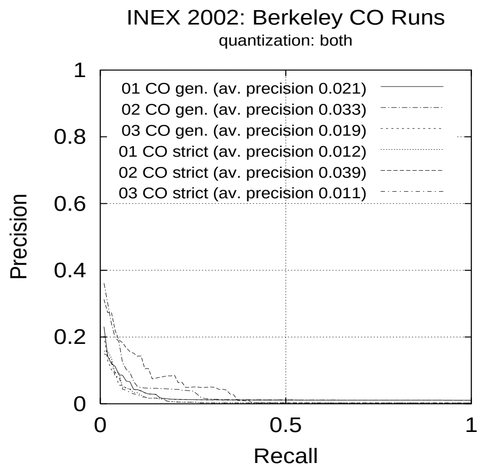


Figure 2: Berkeley CO Runs

participants) of the 4 quantization/type sets was Berkeley03 CAS run under strict quantization. The other runs seem to consistently fall near the median point for all of the participant runs. Curiously, and seemingly impossible from the definitions of "strict" and "generalized" quantization used in the official evaluation, in all of our runs the strict results were better than the generalized results for the same runs.

Needless to say, we had hoped for a better result and conducted some analyses to attempt to determine which factors of the current approach might yield better results. One of the obvious things to check was if errors had been made in the processing of queries, and this did turn out to be the case in some of queries. This bug was in the script that converted the results to the INEX submission format, not in retrieval itself, where only the first occurrence of component retrieved for some of the queries was converted to an entry for the submission (this was most signicant in one query where all of the relevant components were in a single article). It was also found that Fusion Searches were apparently not correctly accumulating scores for each component search in some cases (this is still being analyzed to determine exactly where it is failing). Another obvious failure was to submit only article-level results for CO searches, instead of a mixture of articles and components.

In the analysis we have found that in the CO queries (while maintaining article-level results only) any individual type of probabilistic search, as described above for each of the components of the Fusion Search, does not achieve the effectiveness of the Fusion approach. It was also found (probably due to the Fusion Search bug described above) that Fusion Searches with fewer searches than the submitted runs often could achieve higher effectiveness.

5 Conclusion

The INEX evaluation has proven very interesting, particularly as the first evaluation Cheshire's Fusion Search approach in a formal IR evaluation. As the above discussion shows, there remains considerable room for improvement of our results, but there also seems to be a fairly clear path for seeking those improvements. Specifically, we are planning to fix the identified bugs, to conduct further analyses to determine the optimal mixture of search elements to employ in Fusion Search, and to investigate some alternate approaches and implementation strategies for this retrieval method.

In addition we are planning to conduct experiments in adapting the regression approach to multivalued relevance criteria using the INEX test collection. The logistic regression equations that we used in this INEX evaluation were predicated on binary relevance judgements at the article level and not on component retrieval at the with relevance and coverage scales.

6 Acknowledgments

This work was supported in part by the National Science Foundation and Joint Information Systems Committee(U.K) under *NSF International Digital Libraries Program* award #IIS-9975164.

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 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.

[6] 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.

[7] J. K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, Mass., 1994.

[8] 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.

[9] 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.

[10] A. Z39.50-1995. *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification (ANSI/NISO Z39.50-1995).* NISO, Bethesda, MD, 1995.

Content-oriented XML retrieval with HyREX

Norbert Gövert
University of Dortmund
Germany

Mohammad Abolhassani
Norbert Fuhr Kai Großjohann
University of Duisburg-Essen
Germany

goevert@ls6.cs.uni-dortmund.de
{mohasani,fuhr,kai}@is.informatik.uni-duisburg.de

1 Introduction

The e*X*tensible *M*arkup *L*anguage (XML)[1] is the emerging standard for representing knowledge in almost arbitrary applications. At least almost every kind of knowledge can be represented in XML. The major purpose of XML markup is the explicit representation of the logical structure of a document. From an information retrieval (IR) point of view, users should benefit from the structural information inherent in XML documents. The *X*ML *I*nformation *R*etrieval *Q*uery *L*anguage (XIRQL) [Fuhr & Großjohann 01, Fuhr & Großjohann 02] has been developed to serve this purpose. XIRQL extends the XPath [Clark & DeRose 99] part of the (proposed standard) query language XQuery [Chamberlin et al. 01] by features 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. XIRQL allows for term weighting with regard to the components of the documents' logical structure. This is used for implementing the retrieval paradigm suggested by the FERMI multimedia model for IR [Chiaramella et al. 96]: Instead of treating documents as atomic units, we aim at retrieving those document *components* (elements) which answer a given information need in the *most specific* way. This strategy is used to process the *content-only (CO)* topics provided within the *IN*itiative for the *E*valuation of *X*ML retrieval (INEX)[2], where no structural conditions are used within the queries.

Given the logical structure inherent to XML documents, users want to pose queries not only on content but also on the structure of the documents. The INEX *content-and-structure (CAS)* topics reflect that. As an extension of XPath, the XIRQL query language is capable of processing these queries.

The *Hy*per-media *R*etrieval *E*ngine for *X*ML (HyREX)[3] [Abolhassani et al. 02] provides an implementation of the XIRQL query language. In the following we describe its implementation with regard to processing the INEX CO and CAS topics. In Section 2 we show how ranking of most specific document components is done in HyREX, thus serving for processing the content-only topics. Section 3 details the algorithms used to produce such a ranking of document components while Section 4 displays the evaluation results of our approach.

Section 5 shows how XIRQL concepts are used in order to process the CAS 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. 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]

[1] http://www.w3c.org/XML/
[2] http://qmir.dcs.qmw.ac.uk/INEX/
[3] http://www.is.informatik.uni-duisburg.de/projects/hyrex/

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:

- For relevance-oriented search, where no type of result element is specified, these units are the retrievable units. They provide a context within a document which can serve as a meaningful answer to a user's information need.

- Given these units, we can apply for example some kind of $\mathrm{tf} \cdot \mathrm{idf}$ formula for term weighting.

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.

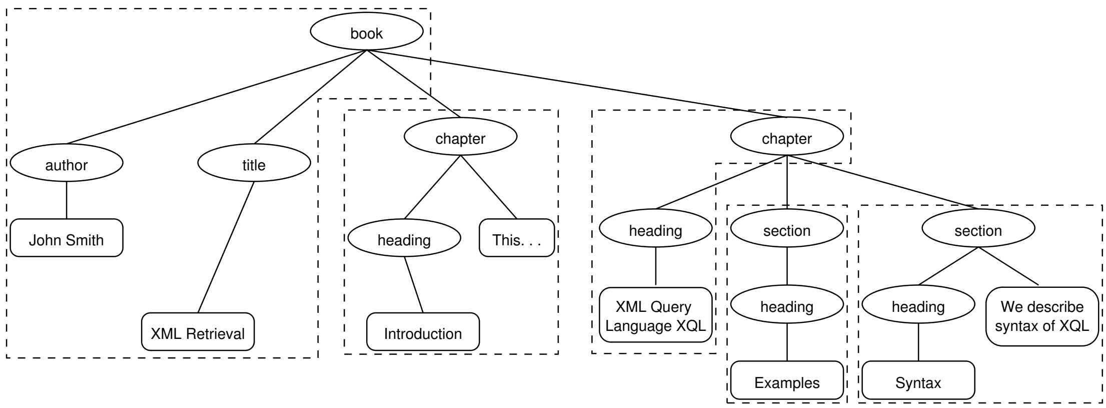


Figure 1: Example XML document tree with index nodes at the `boot`, `chapter`, and `section` levels.

Thus we have a method for computing term weights and we can do relevance-oriented search. For this, we must be able to retrieve index nodes at all levels. The indexing weights of terms within the most specific index nodes are given directly. For retrieval of the higher-level objects, we have to consider that their content is made up by the content of the index node under consideration plus the content of the descendent index nodes. Therefore, for a given index node its term weights have to be combined with the term weights of the descendant index nodes. 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 easily be 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 factor) when they are propagated upwards to the next index node. In our example, using an augmentation factor 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-oriented searches, the XIRQL query language defines the respective relevance selection operator ‘`inode()`’ and the relevance projection 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 factor.

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 retrieval status value (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 scheme 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 factors belonging to the index nodes represented by the `inode_path` array.

term Identifier of the inverted list from which the current occurrence description is read.

weight Term weight within the index node referenced by `inode_id`.

Within the outer loop of the algorithm occurrence descriptions for all of the query terms are read until all the respective inverted lists are processed:

```
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.

Before applying the retrieval function to an index node the contribution of the descendent index objects within the path represented by `lastnodes` to the term weights needs to be computed. The term weights are propagated beginning from the leaf in `lastnodes`; at each index node border they are reduced by means of an augmentation factor given for the specific index object. After an index object is processed this way the respective term weights in the `cumulated_weights` matrix is reset.

When the RSVs for the index nodes finished have been processed this way, the `lastnodes` vector is set to the path to the current index object under consideration. The weight of the term under consideration is stored within the `cumulated_weights` matrix.

```
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 = lastlevel downto j do
            // apply linear retrieval function (scalar value)
            rsv = cumulated_weights[i] * qterm_weights
            add_result(lastnode[i], rsv)
            // propagate term weights towards the root
            if i > 1 then
              cumulated_weights[i - 1] = cumulated_weights[i - 1]
                | augmentation[i] & cumulated_weights[i]
            fi
            // reset cumulated weights
            cumulated_weights[i] = 0
        od
        last // exit loop
    fi
od
lastnodes = inodes
lastlevel = level
// store weight of occurrence for current term
cumulated_weights[level, term] = weight
```

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 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 Evaluation of effectiveness

One of the results of the first INEX workshop 2002 has been the definition of a metrics for evaluation of the effectiveness of content-oriented XML retrieval approaches [Gövert & Kazai 03]. This metrics, based on the notion of recall and precision, has been used here for evaluation, together with the relevance assessments package version 1.7 (available from the INEX {down,up}load area[4]).

Our focus has been on experimenting with different augmentation factors when doing the relevance-oriented retrieval described in Section 2. Figure 3 show the recall / precision curves for six different augmentation factors from 0.0 to 1.0, step 0.2. For each plot the top 100 results from the rankings have been accounted for. From the graphs one can see that small augmentation factors in the range from 0.2 to 0.4 should be used for most effective content-oriented XML retrieval.

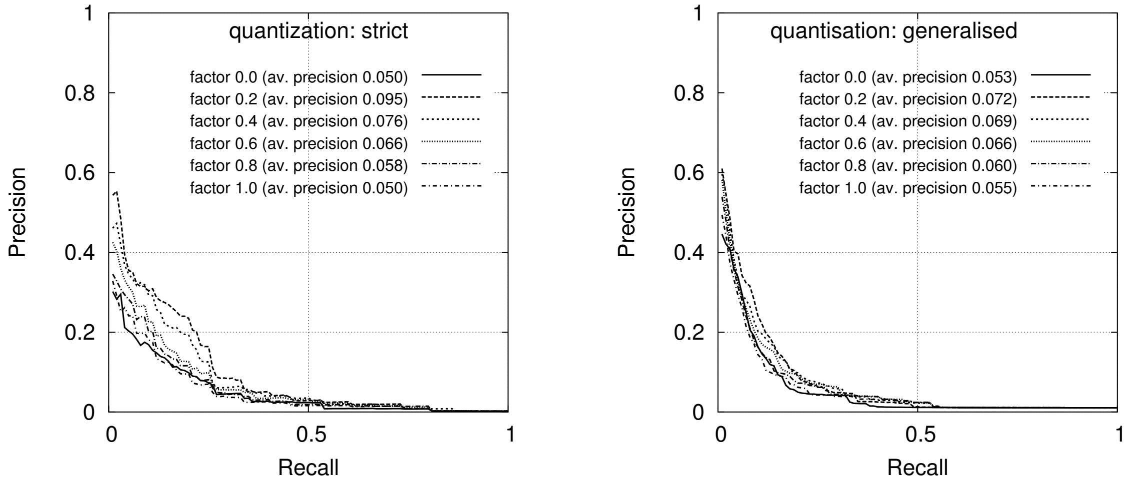


Figure 3: Recall / precision curves for different augmentation factors (content-only topics).

5 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.

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 4. Figure 5 shows the result of its conversion 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 filters which combine various conditions via weighted sum operators. The set of conditions in the first weighted sum results from 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 second set of conditions results from the query terms in the description and keywords section of

[4] `http://ls6-www.cs.uni-dortmund.de/ir/projects/inex/download/`

```
<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 4: CAS topic 24 in XML format

topic 24. We use relevance-oriented search for them, so that documents where all terms are in the same index node are boosted. The figures in front of the various conditions denote the (non-normalised) query term weights (the weighted sum operator normalises these weights internally). Some CAS topics include phrases which are emulated by requiring all terms to be in the same text node. For example, one component of the weighted sum could be as follows:

```
.//au//#PCDATA[. $soundex$ "John" $and$ . $soundex$ "Smith"]
```

The “`//#PCDATA`” part in the structural conditions is required for implementation-related reasons.

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

A first evaluation could show how index term weights can be augmented for effective content-oriented XML retrieval. For further improvements alternative approaches for selecting appropriate augmentation factors are to be tested. In principle, augmentation factors may need to 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 theoretical and empirical investigations. Another way to derive augmentation factors could be based on information about the size of index nodes and the number of siblings and children. 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.

Chamberlin, D.; Florescu, D.; Robie, J.; Siméon, J.; Stefanescu, M. (2001). *XQuery: A Query Language for XML*. Technical report, World Wide Web Consortium.

```
/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, ...             $stemen$  "Find",
    2, ...             $stemen$  "Jones",
    2, ...             $stemen$  "Smith",
    1, ...             $stemen$  "articles",
    1, ...             $stemen$  "author",
    1, ...             $stemen$  "believe",
    1, ...             $stemen$  "engineering",
    2, ...             $stemen$  "improvement",
    1, ...             $stemen$  "industry",
    1, ...             $stemen$  "named",
    2, ...             $stemen$  "process",
    2, ...             $stemen$  "programming",
    2, ...             $stemen$  "software",
    1, ...             $stemen$  "written" ) ]
```

Figure 5: CAS topic 24 in XIRQL syntax

Chiaramella, Y.; Mulhem, P.; Fourel, F. (1996). *A Model for Multimedia Information Retrieval*. Technical report, FERMI ESPRIT BRA 8134, University of Glasgow.

Clark, J.; DeRose, S. (1999). *XML Path Language (XPath) Version 1.0*. Technical report, World Wide Web Consortium.

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.

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, W. B.; Moffat, A.; van Rijsbergen, C. J.; Wilkinson, R.; Zobel, J. (eds.): *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 257–265. ACM, New York.

Gövert, N.; Kazai, G. (2003). Overview of the Initiative for the Evaluation of XML retrieval (INEX) 2002. In: Fuhr, N.; Gövert, N.; Kazai, G.; Lalmas, M. (eds.): *Initiative for the Evaluation of XML Retrieval (INEX). Proceedings of the First INEX Workshop. Dagstuhl, Germany, December 8-11, 2002*, ERCIM Workshop Proceedings. ERCIM, Sophia Antipolis, France.

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]:

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

where θ_D is the language model estimated from the document, θ_Q is the language model estimated from the query, and $P(w|\theta)$ estimates the probability of the word w given the language model θ. The $P(w|\theta_Q)$ 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]:

$$\mathrm{P}(\mathrm{Q}|\theta_{\mathrm{D}}) = \prod_{w \in (q_1, q_2, \ldots, q_n)} \mathrm{P}(w|\theta_{\mathrm{D}})$$

$$\propto \sum_{w \in (q_1, q_2, \ldots, q_n)} \log \mathrm{P}(w|\theta_{\mathrm{D}})$$

where Q = $(q_1,q_2,\ldots,q_n)$ 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 or the histogram distribution. An advantage of this estimate is that it is easy to compute. It is very good at estimating the probability distribution for the language model when the size of the observed text is very large. It is given by:

$$\mathrm{P}(w|\theta_{\mathrm{T}}) = \frac{count(w;\mathrm{T})}{|\mathrm{T}|}$$

where T is the observed text, $count(w;\mathrm{T})$ 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:

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

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

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

One use of linear interpolation is to smooth a document's language model with a collection language model. This new model would then be used as the smoothed document language model in either the generative or KL-divergence ranking approach. 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. θ_1 is the document language model, θ_2 is the collection language model, and the linear interpolation parameters are:

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

where the parameter μ 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

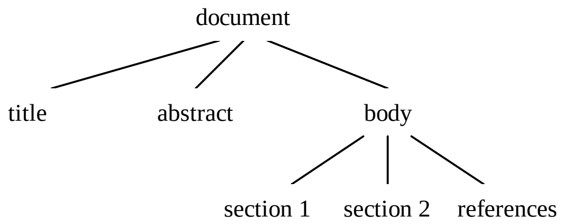
The previous section described how language modeling is used in unstructured document retrieval.

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 → 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.

However, in cases where the document structure is not known, the PCFG analogy is useful. Given a component recognizer and some training data, one could estimate a tree for documents. For example, one could use the existing INEX documents and corresponding flat text versions of the documents as a training set for an automatic tagger for computer science articles.

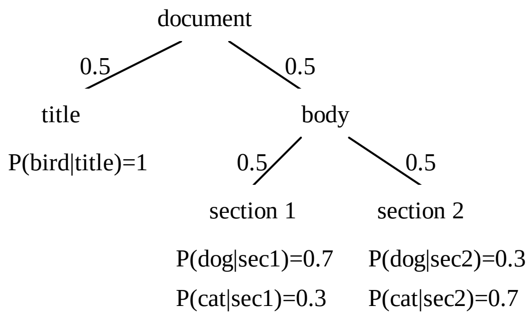
The example rule given above states that a document language model consists of a title, an abstract, and a body language model. We next must specify how to combine the children language models. We suggest that linear interpolation is an appropriate method of combining the children language models. We 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 θ generated either a or b (or both). This assumes independence of a and b. Allowing this only on individual query terms would fit within the unigram assumptions in the model. Alternatives here would be $P(a|\theta) + P(b|\theta) - P(a \text{ and } b|\theta)$, and $(1 - (1 - P(a|\theta))(1 - P(b|\theta)))$.

NOT a: Take $1 - P(a|\theta)$. This is the probability that the model θ 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

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

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

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

The multiplication of weights along the path to a node may seem like it places much more weight on nodes higher in the tree. This is only true under limited constraints. In general, as the model is multiplicative, the weights will factor out and be the same across documents. However, if there is an OR operation of two constraints, then this problem will happen. We do not expect this to be an issue for most queries.

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:

$$P(Q|\text{title}) = 0$$

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

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

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:

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

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

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

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

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

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:

$$\mathrm{P}(\mathrm{D}|\mathrm{Q}) = \mathrm{P}(\mathrm{Q}|\theta_\mathrm{D})\,\mathrm{P}(\mathrm{D})/\mathrm{P}(\mathrm{Q})$$

$$\propto \mathrm{P}(\mathrm{Q}|\theta_\mathrm{D})\,\mathrm{P}(\mathrm{D})$$

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:

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

   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^{th} iteration, and $(\mathrm{Q},\mathrm{D}) \in R$ 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 estimate 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), 2nd edition, Benjamin/Cummings Publishing.

[2] Chinenyanga, T.T. and N. Kushmerik. Expressive retrieval from XML documents. In *Proceedings of the 24th 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 24th 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 25th 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 8th 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 9th 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 24th 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. Zhoo. A flexible model for retrieval of SGML documents. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1998), ACM Press, 138-145.

[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 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1998), ACM Press, 275-281.

[14] Westerveld, T., W. Kraaij, and D. Hiemstra. Retrieving web pages using content, links, URLs, and anchors. In *Proceedings of the Tenth Text Retrieval Conference, TREC 2001*, NIST Special publication 500-250 (2002), 663-672.

[15] Zhai, C. and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 334-342.

[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 Importance of Morphological Normalization for XML Retrieval

Jaap Kamps, Maarten Marx, Maarten de Rijke, Börkur Sigurbjörnsson
Language and Inference Technology Group, ILLC, U. of Amsterdam
Nieuwe Achtergracht 166, 1018 WV Amsterdam, The Netherlands
{kamps,marx,mdr,borkur}@science.uva.nl
http://lit.science.uva.nl

Abstract

Current information retrieval systems typically ignore structural aspects of documents, solely focusing on the textual content instead. But documents containing additional structure in the form of HTML, XML, or SGML mark-up are pervasive on the Internet. The XML retrieval task presents a number of challenges for information retrieval, for we can no longer rely on the appropriate unit of retrieval to be fixed, or to be known beforehand. This implies that the effectiveness of standard IR techniques, such as morphological normalization methods, may not carry over to this particular task. This paper describes the fully automatic runs for the INEX 2002 task submitted by the Language and Inference Technology Group at the University of Amsterdam. We investigate the effectiveness of two standard approaches to morphological normalization, both a linguistically motivated stemming algorithm and a knowledge-poor character n-gramming technique. Our results show that morphological normalization is an important issue for XML retrieval. For all measurements, the combined run and the n-gram run perform better than the stemmed run.

1 Introduction

With recent advances in computer and Internet technology, people have access to more information than ever before. Much of the information is available in free text with little or no metadata, and there is a tremendous need for tools to help organize, classify, and store the information, and to allow better access to the stored information. Current information retrieval systems allow us to locate documents that might contain the pertinent information, but most of them leave it to the user to extract the useful information from a ranked list. This leaves the (often unwilling) user with a relatively large amount of text to consume.

To address these issues, a number of recent initiatives are aimed at providing highly focused information 'pinpointing.' For instance, in the TREC question-answering track [17] participants are given a large document set and a set of questions; for each question, the system has to return an exact answer to the question and a document that supports that answer. Another approach to providing highly focussed information access is to return only new *and* relevant sentences (within context) rather than whole documents containing duplicate and extraneous information, as is done within TREC's novelty track [5].

We view XML retrieval as yet another approach to providing more focused information access than traditionally offered by search engines. An XML document collection differs from a traditional IR document collection: in the latter, documents contain only plain text and they are the natural unit of retrieval. Documents in an XML collection are divided into a hierarchy of text objects. These text objects provide restricted and, we hope, semantically meaningful contexts for satisfying users' information needs. It is natural, therefore, to take advantage of this structural information and look below the document level for a suitable unit of retrieval. The main question then becomes: To which extent can XML document structure help improve retrieval effectiveness? Obviously, the creation of an XML test collection is a key resource for answering this question.

The INEX 2002 collection, 21 IEEE Computer Society journals from 1995–2002, consists of $12,135$ documents with extensive XML-markup (when ignoring the volume.xml files). The test collection contains two types of topics. Content-only topics (CO) ignore the structure of the documents and, hence, are nothing but traditional IR topics. Content-and-structure (CAS) topics are aware of the structure of the documents. They can include constraints on the type of elements that are to be retrieved as well as constraints on the context in which the search terms should appear. The main difference with traditional IR tasks is that we may retrieve any XML component in the collection.

The aim of our official runs was to experiment with the effectiveness of different types of morphological normalization for structured corpora. 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 (singular, plural, etc.) should share some of their relevance. Morphological normalization proved successful for plain text collections [8, 12]. In order to study the impact of morphological normalization in the setting of XML retrieval, we created stemmed and n-grammed indexes that preserve the XML-structure of the original documents. This allows for both the CO and CAS topics to be evaluated against both indexes.

Our strategy at INEX 2002 was to create a baseline system based on a traditional document index. That is, our index treats complete articles as the unit for retrieval. For the CO topics, the XML structure of the documents was not used, and we retrieve entire articles. For the CAS topics, we used a two step strategy. We first treated the topic as a CO topic and selected the 1000 highest ranking articles. Then we directly processed the (morphologically normalized) representation of these documents. All experiments were carried out with the FlexIR system developed at the University of Amsterdam [12].

The rest of this paper is organized as follows. We describe our experimental set-up in Section 2, and our official runs in Section 3. In Section 4 we present evaluation measures for XML retrieval and present our results. Section 5 provides a discussion of our results, and we end by drawing some conclusions.

2 Experimental Set-Up

2.1 The FlexIR information retrieval system

All submitted runs used FlexIR, an information retrieval system developed at the University of Amsterdam [12]. 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; 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 the 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; we fixed $slope$ at 0.2, while the pivot was set to the average number of unique words per document.

From both topics and documents we removed words occurring on a standard stop list with 391 words. Blind feedback was applied to expand the original query with related terms. Term weights were recomputed by using the standard Rocchio method [14], where we considered the top 10 documents to be relevant and the bottom 500 documents to be non-relevant. We allowed at most 20 terms to be added to the original query.

We experimented with two approaches to morphological normalization (discussed in Section 2.2 below). As a side issue, we 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. First, we normalized the retrieval status values (RSVs), since different runs may have radically different RSVs. Following [10], we mapped the values to $[0, 1]$ using $RSV_i' = (RSV_i - min_i)/(max_i - min_i)$. Next, we assigned new weights to the documents using a linear interpolation factor λ representing the relative weight of a run [15]: $RSV_{new} = \lambda \cdot RSV_1' + (1 - \lambda) \cdot RSV_2'$. For $\lambda = 0.5$ this is the combSUM function of [3].

2.2 Morphological normalization

As pointed out above, our overall aim was to study the effect of morphological normalization on the effectiveness of XML retrieval. One approach to morphological normalization is to use linguistically informed methods; we decided to use a stemming algorithm for the English language. Alternatively, there are knowledge-poor approaches to morphological normalization which do not require any knowledge of the particular source language; here, we decided to use an n-gramming method.

n-Grams Our n-gram-based approach was based on character n-grams, where the n-gram length was set to 5; this setting was motivated by the results of experiments on the CLEF [2] data sets. For each word we stored both the word itself and all possible character n-grams of length 5 that can be obtained from it without crossing word boundaries. As an example, Figure 1(a) shows the original Topic 31, and Figure 1(b) shows the (stopped and) n-grammed version of the topic.

Stemming For the linguistically informed method with which we wanted to contrast the effect of the n-gram method we used Porter stemming [13]. Figure 1(c) shows the (stopped and) stemmed version of Topic 31.

```
<INEX-Topic topic-id="31" query-type="CO" 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) The original version of Topic 31.

```
.i 31
computational compu omput mputa putat utati tatio ation tiona ional biology biolo iolog ology
challenges chall halle allen ... biology biolo iolog ology
```

(b) The n-grammed version of Topic 31.

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

(c) The stemmed version of Topic 31.

Figure 1: Topic 31.

3 Runs

We now describe how our runs were created. We built two base runs: one using the Porter stemmer and one in which we used n-grams in the manner described above. We then combined these two runs in the manner described in Section 2, thus producing a total of three official runs for INEX 2002:

Stemmed run We use a stemmed index and stemmed topics, the Lnu.ltc weighting scheme, and blind feedback.

n-Grammed run We use an n-grammed index and n-grammed topics, the Lnu.ltc weighting scheme, and blind feedback. We used n-gram-length 5, adding n-grams for words with length ≥ 4, while also keeping the originals words.

Combined run We combined the first two runs using an interpolation factor λ of 0.6 for the n-gram run. This higher weight for the n-gram run was motivated by the outcomes of experiments on the CLEF [2] data sets.

For both types of topics we wanted to use methods that were fully automatic and portable to other collections. In our retrieval we only used words from the title and description fields. In particular, we did not use the keywords provided with the topics: according to the topic development guidelines, keywords are supposed to be “good scan words that are used in the collection exploration phase of the topic development process” [7, p.107]. Furthermore, we did not use any information from the DTD either.

After the (document) pre-processing steps described in Section 2 were carried out, indexing of the collection was done at the article level, i.e., the indices were mappings from terms to articles in the collection. Since the topic processing and retrieval steps differ for the CO topics on the one hand and the CAS topics on the other, we describe them in separate subsections.

3.1 Content-only topics

For the CO topics, we automatically translated the topics into the FlexIR topic format, as illustrated in Figure 1, using only the words appearing in the title and description fields.

We ran the (stemmed or n-grammed) topics against the (stemmed or n-grammed) document index. The 100 documents with the highest RSVs were returned. The units of retrieval were articles. In other words, we always returned `/article[1]` in the path tag of the results.

```
<INEX-Topic topic-id="01" query-type="CAS" ct-no="010">
   <Title>
      <te>article/fm/au</te>
      <cw>description logics</cw><ce>abs, kwd</ce>
   </Title>
   <Description>
      Retrieve the names of authors of articles on description logic, in particular
      articles in which the abstract or the list of keywords contains a reference
      to description logic.
   </Description>
   ...
</INEX-Topic>
```

(a) The original version of topic 01.

```
.i 01
descript logic retriev author articl descript logic particular articl abstract list keyword
contain refer descript logic
```

(b) Stemmed version of the document retrieval translation.

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

(c) Stemmed version of the document filtering translation.

Figure 2: Topic 01.

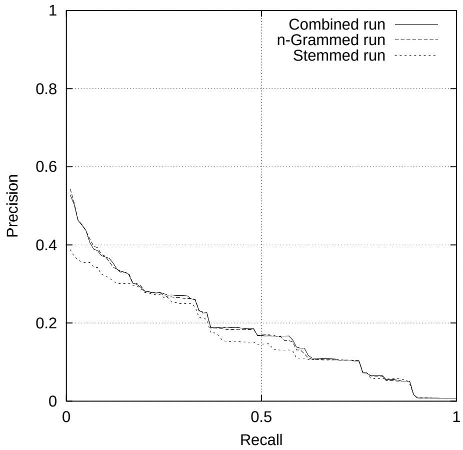
3.2 Content-and-structure topics

The CAS topics contain additional information in the `<ce>` and `<te>` tags; see Figure 2(a) for an example. For the CAS topics we divided the retrieval process into two subtasks: document retrieval and document filtering. This required two different topic translations, one for each subtask. For the document retrieval subtask, topics were processed similar to the CO topics: only the words in the title and description fields were selected, and from the title field we only selected the content of the `<cw>` field. For an example of this translation see Figure 2(b).

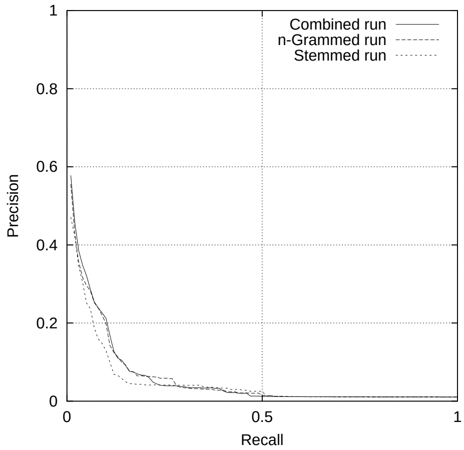
For the document filtering subtask, the `<Title>` field was processed to preserve the structural part of the query. For an example of this translation, see Figure 2(c). The first line contains the topic number, the second line gives the XML-field that is to be returned, the next line(s) give conditions for the document, consisting of a field name, and the words that are sought. This should be read as: retrieve the elements found by the XPath expression `//article/fm/au` in the documents whose elements found by the XPath expressions `//abs` or `//kwd` contain the words `descript` or `logic`. If no target element is specified in the topic title, we treat it as if the target element had been `<te>article</te>`. A connection between a disjunction of target elements and a disjunction of search criteria may lead to ambiguities. Hence we replaced disjunctions of target elements `<te>A,B,..</te>` by `<te>/article</te>`. Further motivation for this translation can be found in [11].

For the document retrieval subtask we ran the (stemmed or n-grammed) topics against the (stemmed or n-grammed) document index. The 1000 documents with the highest RSVs were returned. Our working assumption was that all relevant document were in this top 1000.

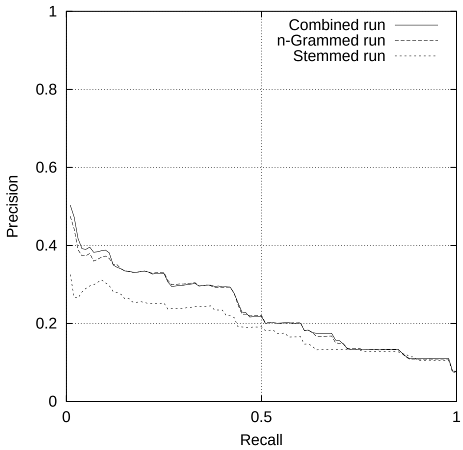
For the document filtering subtask, we created a special XML-file for each topic, containing these top 1000 documents. On these so-called doc-piles, we ran an XML-parser based on Perl's `XML::Twig` that handles XPath expressions. For each topic and for each context-element (`<ce>`) in its doc-pile, the XML-parser calculates a score for each context-element. This score is the count of how often a context-word (`<cw>`) appears in the context-element, divided by the number of words in the content-element. We sorted the documents in the doc-pile according to their highest scoring element. For each document in the doc-pile we extracted the target-elements (`<te>`), using the XML-parser. To each target-element we assign the score of the document that contains it. We select the 100 highest scoring target-elements. Those 100 elements are returned, sorted by RSV score of the document containing the element.



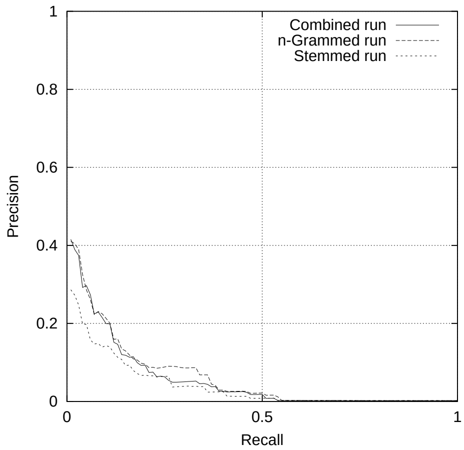
(a) **CAS** topics using the **generalized** measure.



(b) **CO** topics using the **generalized** measure.



(c) **CAS** topics using the **strict** measure.



(d) **CO** topics using the **strict** measure.

Figure 3: Precision recall graphs of our official runs for both topic types, using both evaluation measures.

4 Results

To evaluate our runs we used version 0.006 of the `inex_eval` program supplied by the organizers of INEX 2002. We used version 1.6 of the relevance assessments. The topics were assessed on a two dimensional graded relevance scale, one for topical relevance, with values taken from $\{0, 1, 2, 3\}$, and another for document coverage, with values taken from $\{$*exact*, *too_large*, *too_small*, *no_coverage*$\}$.

The evaluation software can create reports using two distinct measures, see [4] for details. The *strict* relevance measure considers only highly relevant items that have exact coverage. The strict relevance scores are calculated by means of the function f_s below.

$$f_s(e) := \begin{cases} 1 & \text{if } e = (3, exact) \\ 0 & \text{otherwise.} \end{cases}$$

$$f_g(e) := \begin{cases} 1 & \text{if} \quad e = (3, exact) \\ 0.75 & \text{if} \quad e = (2, exact) \text{ or} \\ & \phantom{\text{if}} \quad e = (3, too\_large) \text{ or} \\ & \phantom{\text{if}} \quad e = (3, too\_small) \\ 0.5 & \text{if} \quad e = (1, exact) \text{ or} \\ & \phantom{\text{if}} \quad e = (2, too\_large) \text{ or} \\ & \phantom{\text{if}} \quad e = (2, too\_small) \\ 0.25 & \text{if} \quad e = (1, too\_large) \text{ or} \\ & \phantom{\text{if}} \quad e = (1, too\_small) \\ 0 & \text{otherwise} \end{cases}$$

The *generalized* relevance measure considers all combinations of all values of relevance and coverage. The generalized relevance scores are calculated by means of the function f_g given above.

The strict and generalized measures defined above differ from the standard mean average precision scores. When ignoring the coverage dimension, the strict measure is similar to the work on judging by highly relevant document [16]. This strict measure is still a dichotomous measure. When ignoring coverage, the generalized measure is similar to the graded measures of relevance [9].

Generalized measure CAS				
Run	MAP	Impr.	P. at 0	Impr.
Combined run	**0.185**	+12%	0.528	+36%
n-Grammed run	0.183	+11%	**0.544**	+40%
Stemmed run	0.165	0%	0.388	0%

Generalized measure CO				
Run	MAP	Impr.	P. at 0	Impr.
Combined run	**0.0576**	+19%	**0.578**	+23%
n-Grammed run	0.0568	+17%	0.556	+18%
Stemmed run	0.0484	0%	0.471	0%

Strict measure CAS				
Run	MAP	Impr.	P. at 0	Impr.
Combined run	**0.234**	+23%	**0.503**	+55%
n-Grammed run	0.232	+21%	0.475	+46%
Stemmed run	0.191	0%	0.325	0%

Strict measure CO				
Run	MAP	Impr.	P. at 0	Impr.
Combined run	0.0553	+34%	**0.415**	+45%
n-Grammed run	**0.0618**	+55%	0.411	+44%
Stemmed run	0.0399	0%	0.286	0%

Table 1: The mean average precision results for our official runs. The precision at zero is the interpolated precision over the interval $(0, 0.1]$. Improvements are computed relative to the stemmed run.

The results for our official runs are displayed in Figure 3 and Table 1. Some obvious remarks can be made. First, compared to TREC-style document retrieval results, the mean average precision (MAP) scores are much lower (at TREC where one would expect a MAP of at least twice the best score in the table). Also, the scores for CO are much lower than for CAS topics. Second, we included the precision at 0 in Table 1 as an indication of the quality of the top ranked retrieved documents. These numbers are reassuring, and far less dramatic than the low MAP scores for, especially, CO would suggest. In fact, both CAS and CO topics have comparable p@0 scores. Third, the difference in performance of the three runs is a clear indication that morphological normalization is an important issue for XML retrieval. The relative results are in favor of the knowledge-poor approach: the n-grammed run is performing better than the stemmed run in all four cases. Fourth, the combined run is better than the best underlying baserun in three cases (CAS and CO generalized), although the improvement is unimpressive. This can be explained by the difference in score of the underlying baseruns: when the difference between stemmed and n-grammed runs peaks at over 50% (CO strict), the combined run is not better than the n-gram run! Fifth, when comparing the strict and generalized scores, the strict scores are almost always higher. This is somewhat counterintuitive, because the generalized score is a more liberal score that regards more retrieved elements as relevant.

5 Discussion and Conclusions

We entered the INEX initiative for the evaluation of XML retrieval with modest ambitions. We wanted to set up a baseline system based on a traditional document index where the unit of retrieval is an article. Only for the CAS topics did we attempt to retrieve the particular XML element requested by the target element field.

Our goal was to have a fully automatic XML retrieval system that can easily be ported to different topics, collections, and DTDs. All our runs are fully automatic TD-runs that ignore the keywords and the narrative fields of the topics (which are considered to be additional information for the relevance judgments). We did not correct misspellings or other errors in the topics, resulting in the retrieval of no results for two CAS topics. We use no manual query processing steps, nor human knowledge on the semantics of the tags.

We expected our system's performance to be just a baseline for 'proper' XML retrieval systems, i.e., for systems that return smaller XML components than articles. To our surprise, our runs turn out to be among the top scoring submissions on both CAS and CO tasks, and on both generalized and strict evaluation measures; this is even more surprising if we take into account that several teams submitted manual runs and runs using the narrative. How should we interpret this? On the one hand, the results show that a system returning entire articles is competitive to systems returning smaller units of text—our system, indeed, can function as the baseline performance we hoped to obtain. On the other hand, the results suggest that we do not yet fully understand how users (and assessors) perceive the coverage dimension of relevance. It is clear that more research is needed to better understand what users (and assessors) regard as meaningful units of retrieval.

There are a few things one needs to keep in mind when looking at the output of the `inex_eval` software. The software's definition of total recall does not take into account the graded relevance nor the limit on the number of elements retrieved. The total recall of the strict measure is defined as the number of highly relevant elements in the collection that have exact coverage. The total recall of the generalized measure is defined as the number of relevant elements in the collection. This puts an upperbound on the mean average precision scores that systems can achieve, as shown in Table 2; the upperbounds are calculated for 'perfect' run that return 100 relevant items.[1]

These upperbounds partly explain why the strict evaluation measure gives a higher average precision than the generalized measure. This is counter-intuitive as we would expect to do worse on the strict scale, having in mind that we do article retrieval for all the CO topics and approximately one-third of the CAS topics. Thus we would expect a too_large coverage, giving no score on the strict measure. When taking into account the maximally obtainable scores in Table 2, our generalized scores do outperform the strict scores. Added to that, whole articles seem to have been quite frequently judged highly relevant with exact coverage. This sheds some light on how exact coverage is perceived by users and assessors.

Topic type	Measure	Possible MAP
CAS	generalized	0.596
CO	generalized	0.332
CAS	strict	0.897
CO	strict	0.931

Table 2: Upper bounds on the average precision.

The official runs of INEX 2002 had a maximum number of retrieved elements set at 100 elements. A problem with this upperbound is that the number of relevant elements in the assessments can be much higher than 100, even on average. We modified our runs by allowing 1000 results to be returned (as is customary for CLEF and TREC ad-hoc retrieval experiments). A comparison of the MAP scores between runs with cut-off points at 100 and 1000 results is displayed in Table 3. Although the scores do improve, they remain low compared to MAP values

Generalized measure CAS			
	MAP		
Run	100	1000	Impr.
Combined run	**0.185**	**0.199**	+7.6%
n-Grammed run	0.183	0.196	+7.1%
Stemmed run	0.165	0.170	+3.0%

Generalized measure CO			
	MAP		
Run	100	1000	Impr.
Combined run	**0.0576**	**0.0677**	+18%
n-Grammed run	0.0568	0.0653	+15%
Stemmed run	0.0484	0.0551	+14%

Strict measure CAS			
	MAP		
Run	100	1000	Impr.
Combined run	**0.234**	**0.244**	+4.3%
n-Grammed run	0.232	0.240	+3.4%
Stemmed run	0.191	0.201	+5.2%

Strict measure CO			
	MAP		
Run	100	1000	Impr.
Combined run	0.0553	0.0609	+10%
n-Grammed run	**0.0618**	**0.0657**	+6.3%
Stemmed run	0.0399	0.0427	+7.0%

Table 3: Comparison of MAP scores for 100 and 1000 retrieved elements.

for unstructured documents. The improvement is higher for the generalized measure than for the strict measure. This may be due to the larger set of relevant items for the generalized measure. This may also explain why the improvement is greater for CO topics than for CAS topics, although this is partly caused by the lower score of the top-100 runs.

Our aim was to study the effect of morphological normalization for XML retrieval. We experimented with two distinct approaches to morphological normalization: by using linguistically informed methods and by using knowledge poor techniques. For the former we used the familiar Porter stemming algorithm for English. For the latter, we used character n-grams of length 5. Our results show a clear difference between the two approaches, which suggests that morphological normalization is an important issue for XML retrieval. Our results favor the knowledge-poor approach of n-gramming. For all measurements, the combined run and the n-gram run perform better than the stemmed run. This is consistent with results on plain text collections [6, 12]. We also experimented with the combination of the two approaches to morphological normalization. The combined runs score best in three out of four cases (CAS and CO generalized). Still, there is no remarkable difference between the combined run and the n-gram run; n-gramming seems to be the dominant factor of the combination, which, again, is consistent with the retrieval results for unstructured documents [8].

Using our INEX 2002 runs as a baseline, our future research focuses on how to retrieve smaller units of texts by

[1]For the strict measure, a perfect run without length restriction will score a MAP of 1.0; for the generalized measure, a perfect run cannot obtain the perfect score of 1.0. This is due to the definition of generalized recall [9, p.1123]. For example, if there are two relevant documents for a topic with relevance scores 1 and 0.5, respectively, then the generalized precision at generalized recall level 1 is only 0.75.

treating each tag occurring in the collection as a document by itself. Next to this, we are experimenting with ways of exploiting the collection's structure for improving retrieval on the article level, by considering the keywords assigned to documents, co-authors, citations, co-citations, etc. Finally, we are investigating efficient storage and processing architectures tailored to structured document collections.

Acknowledgments We want to thank Willem van Hage for his technical support and for assessing the Enigma topic 42. 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.

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, 2003. `http://www.clef-campaign.org/`.

[3] E. A. Fox and J. A. Shaw. Combination of multiple searches. In D. K. Harman, editor, *The Second Text REtrieval Conference (TREC-2)*, pages 243–252. National Institute for Standards and Technology. NIST Special Publication 500-215, 1994.

[4] N. Gövert and G. Kazai. Overview of the Initiative for the Evaluation of XML retrieval (INEX) 2002. In N. Fuhr, N. Goevert, G. Kazai, and M. Lalmas, editors, *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval* (INEX), Dagstuhl 9-11 Dec. 2002, ERCIM Workshop Proceedings, March 2003.

[5] D. Harman. Overview of the TREC 2002 novelty track. In E. M. Voorhees and D. K. Harman, editors, *The Eleventh Text REtrieval Conference (TREC 2002)*. National Institute for Standards and Technology, 2003.

[6] V. Hollink, J. Kamps, C. Monz, and M. de Rijke. Monolingual retrieval for European languages. *Information Retrieval*, 6, 2003.

[7] INEX guidelines for topic development. In N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas, editors, *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval* (INEX), Dagstuhl 9-11 Dec. 2002, ERCIM Workshop Proceedings, March 2003.

[8] J. Kamps, C. Monz, and M. de Rijke. Combining evidence for cross-language information retrieval. In C. Peters, M. Braschler, J. Gonzalo, and M. Kluck, editors, *Evaluation of Cross-Language Information Retrieval Systems, CLEF 2002*, Lecture Notes in Computer Science. Springer, 2003.

[9] J. Kekäläinen and K. Järvelin. Using graded relevance assessments in IR evaluation. *Journal of the American Society for Information Science and Technology*, 53:1120–1129, 2002.

[10] J. H. Lee. Combining multiple evidence from different properties of weighting schemes. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 180–188. ACM Press, New York NY, USA, 1995.

[11] M. Marx, J. Kamps, and M. de Rijke. The University of Amsterdam at INEX-2002. In N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas, editors, *INEX 2002 Workshop Proceedings*, pages 24–28, 2002.

[12] 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.

[13] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[14] J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System — Experiments in Automatic Document Processing*. Prentice Hall, 1971.

[15] C. C. Vogt and G. W. Cottrell. Predicting the performance of linearly combined IR systems. In W. B. Croft, A. Moffat, C. J. 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 190–196. ACM Press, New York NY, USA, 1998.

[16] E. M. Voorhees. Evaluation by highly relevant documents. In D. H. Kraft, W. B. Croft, D. J. Harper, and J. Zobel, editors, *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 74–82. ACM Press, New York NY, USA, 2001.

[17] E. M. Voorhees. Overview of the TREC 2002 question answering track. In E. M. Voorhees and D. K. Harman, editors, *The Eleventh Text REtrieval Conference (TREC 2002)*. National Institute for Standards and Technology, 2003.

A scalable architecture for XML retrieval

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

Abstract

While in classical text collections documents are regarded as atomic units, in XML collections nested elements of varying granularity are considered. This augmented view increases the number of potentially retrieved objects, e.g. documents, elements within documents, or aggregations of elements or of documents. The increase in the number of objects to be indexed and retrieved by XML retrieval systems leads, for XML collections of comparably small size (several 100 MB), already to the necessity to apply strategies for scalability, such as paralell and distributed processing, 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 indexing and retrieval architecture.

1 Introduction

With the growth of the amount of available digital data, aspects of efficiency, such as indexing speed, storage requirements and query response time, have been considered with increasing importance within information retrieval (IR) systems [2]. Although computer hardware are becoming faster, data and approaches require scalable strategies to support the increasing requirements on data processing. Issues related to efficiency gain further significance in the case of structured document retrieval (SDR) systems, which operate on large collections of structured documents, such as XML. These systems exploit both content and structure of XML documents and return elements of varying granularity to the user. This augmented view leads to additional resource requirements during the indexing and retrieval of structured documents, influencing the system's overall efficiency.

Work towards more efficient SDR systems has roots both in the IR and database (DB) communities. Several approaches to index structures and query optimization have been reported in the literature to improve the efficiency of IR systems [6, 17, 16, 8]. IR-based research into SDR focuses on the efficient extensions of conventional inverted index structures and retrieval functions to deal with XML documents. Methods include the use of specifically designed unique element identifiers [13, 21], path expressions [20] and separate text and structure indexes [14, 22]. On the other hand, database approaches aim to take advantage of existing database techniques and incorporate methods for dealing with textual data, uncertainty and ranking within database management systems [5]. Efforts have been invested in database schema designs for the efficient storage of XML data and query optimization techniques [1, 11, 12].

In this paper we describe a retrieval system for structured documents that employs a scalable architecture for collection indexing and retrieval based on strategies for efficient augmentation, distributed and paralell processing, term, document and database pre-selection. The retrieval system is implemented using HySpirit [19], a software development kit that provides a descriptional approach for modelling 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 indexing and retrieval efficiency of XML objects. We concentrate on the development of an architecture for the distributed indexing of a collection (section 2.1) and a strategy to "localise" the augmented representation of XML elements (section 2.5). 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 usually 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 a retrieval system's efficiency.

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

```
<collection>
 <journal>
  <year>
   <volume>
    <article> ... </article>
    <article> ... </article>
    ...
   </volume>
   <volume>
    ...
   </volume>
  </year>
  ...
 </journal>
 <journal>
   ...
 </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 the collection is divided at a given level of the hierarchy into a set of neighbour elements stored in different databases. Figures 1 and 2 illustrate the two architectures. Both architectures allow for the distributed and parallel processing of the source data.

As an example of the complex case, a journal in the above XML collection may be stored in the following databases:

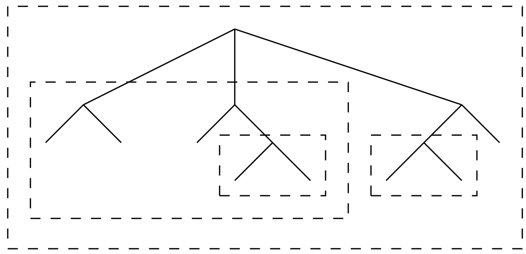


Figure 1: Complex (nested) distributed XML collection

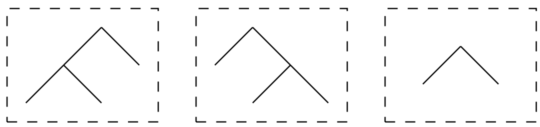


Figure 2: Flat (linear) distributed XML collection

(collection[1]/journal[1], db_1)
(collection[1]/journal[1]/year[1], db_2)
(collection[1]/journal[1]/year[5]/volume[1], db_3)

The relation above associates pathnames within the XML collection with database identifiers. It shows that while most sub-components of the journal are hosted in db_1, one of the year elements within the same journal (collection[1]/journal[1]/year[1]) is located in db_2, and a volume of another year element (collection[1]/journal[1]/year[5]/volume[1]) is stored in db_3.

From a practical point of view, we will often restrict ourselves to the flat architecture, where the design of the distribution structure is simplified. The following is an example of the linear case, where the data is distributed with respect to the sibling year elements.

(journal[1]/year[1], db_1)
(journal[1]/year[2], db_2)
(journal[2]/year[1], db_3)
...

In the realm of structured document retrieval, the processing of a collection, during indexing, involves the representation of both the content and structure of the XML elements. Representation along these two dimensions is necessary in order to support the content-oriented retrieval of XML documents, where elements of varying granularity can be returned to the user.

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

During this indexing process, 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 (or 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*.

For structured documents, we adapt *tf* and *idf* to the hierachical nature of the documents. In IR, *tf* is interpreted as the occurrence frequency of a given term in a given *document*. In XML retrieval, *tf* can be calculated relative to different container units, e.g. either as the number of term occurrences within the containing XML element, or within any ascendant node of that element. As for *idf*, the calculation of a terms' *idf* weight in IR is based on the number of *documents* in the collection that are indexed with that term. Again, since the concept of a document as a discernible retrieval unit is no longer valid in SDR, *idf* can be interpreted as a measure of a term's discriminative power among XML elements at different levels in the collection's hierarchy. Its value will depend on the chosen unit, and the collection (sub-collection) that is being considered.

When determining the local and global representations of a sub-collection, we also need to take into account the following two issues:

1. The resulting termspaces should support the selection of databases during retrieval.

2. In order to obtain an aggregated termspace for the whole collection we must be able to combine the local and global representations of the individual sub-collections that are considered for a retrieval run. Here, we could base our aggregation on the termspaces of the databases or on the termspaces of the atomic elements within the sub-collections (e.g. the union of XML documents within the databases).

For the first task we can use a probabilistic representation of the sub-collections' termspaces, where the probability of a given term can be estimated using the standard *tf* and *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 (section 2.2).

To address the second issue we maintain an occurrence value of the terms within the sub-collections. This is needed to overcome problems of information loss, which occurs when dealing with the probabilistic representations of termspaces. This problem can be demonstrated by the following simple example. Say that we have a sub-collection of 10000 documents and a term, "multimedia", which occurs in 1000 of these documents. The probabilistic representation of this term in the sub-collection's global termspace may be given as log(10000/1000), when estimated using a standard *idf* function. Similarly, in a sub-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 sub-collections based on the probabilistic (frequency-based) representation of their termspaces will obviously lead to incorrect weighting and hence retrieval results.

However, by maintaining the occurrence values of terms within the sub-collections, we can aggregate the termspaces without any misrepresentation. For example, the aggregated *idf*-value of the term "multimedia" from the above two sub-collections can be computed as follows:

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

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

$$P(\mathit{multimedia}) := \frac{\mathit{idf}(\mathit{multimedia})}{\mathit{idf}_{max}}$$

Thus, terms that occur infrequently in the collection have a high probability. The corresponding event in the event space would be: "term multimedia is informative/discriminative".

Aggregation based on the occurrence information, therefore, allows for transparent aggregation across heterogeneous 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 sub-collections. Using a cost function (for example, based on the expected number of retrieved documents), we access those databases that allow us to stay within a given time and resource limit. This approach could be extended using estimations for the probability of relevance [9, 7], however, often, retrieval quality data are not available, and therefore we apply content-based and quantity-based measures.

As an example for content-based measures, given the following representations, we can base the selection of promising databases on the *idf* values of the query terms, where low values would indicate higher concentration of relevant documents within a sub-collection.

db_1:
0.28 idf(multimedia)
0.34 idf(retrieval)
...
db_2:
0.78 idf(multimedia)
0.61 idf(retrieval)
...

In a collection of XML documents, each document can be viewed as a collection of XML elements, where each element can be regarded as a sub-collection in itself, the same way as we consider the hierarchy of a distributed collection. Based on this augmented view a hierarchy of representation layers, each with its own termspace, could be derived for a collection consisting of sub-collections, sub-sub-collections of XML elements, etc. The computational costs associated with the representations of the different layers, however, have to be balanced against the utility of such information. Depending on the size of the collection an appropriate hierarchy can support database selection strategies to zoom in on promising sub-collections and sub-sub-collections, etc.

2.3 Term and context selection

To further improve indexing and retrieval efficiency we reduce the number of terms and retrievable contexts. The removal of stopwords is the classical strategy in IR, and in the same manner, we consider some contexts (XML elements), 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 frequency information about the distribution of contexts within other contexts. A possible criteria for identifying stop-contexts (non-semantic contexts) is to classify contexts according to their occurrence within different super-context types and within the same actual context object. For example, we can detect a layout context-type, such as <bold>, based on the assumption that it is more likely to occur within a wide range of context types, e.g. title, paragraph, section, table, bibliography, etc., and hence follow a distribution that is closer to random across the different context-types, than the occurrence pattern of a semantic context type, such as <section>, which would usually occur only within a limited number of context-types, e.g. within article elements.

In addition to stopword and stop-context removal, we skip the indexing of further terms and contexts to reduce the use of resources. However, unlike 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 exists 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 both for document indexing, and query processing. Given this strategy, we view "intelligent" indexing as an indexing process that optimizes the retrieval quality for a given amount of resources (e.g. index what is needed not what is possible).

2.4 Parallel query processing

In a retrieval experiment, unlike in real life ad-hoc retrieval, we deal with many queries. Under such circumstances we need to decide about the strategy for combining the query and the database dimensions. 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 of a connection) to a database is very expensive. Therefore, it is often worthwhile to optimize with respect to database connectivity, e.g. we run the set of queries for a database. This strategy is based on the assumption that a query switch is less expensive than a database switch. The use of this strategy is further supported by the fact that parallel access to queries is usually less of a bottleneck than a parallel access to (possibly large) databases.

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 content of a context is made up of the contents of its sub-contexts. Augmentation is the underlying concept of aggregation-based structured document retrieval systems, which represent or estimate the relevance of document parts based on the aggregation of the representation or estimated relevance of their structurally related parts [15, 18, 4]

Computing the augmented (aggregated) content of each retrievable context is, however, 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 in queries).

In order to avoid this expensive use of resources, we restrict the aggregation to the query terms and the super-contexts of retrieved contexts only. 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 take place during indexing and would involve the augmentation of all retrievable contexts in the collection. Local augmentation puts emphasis on scalable strategies that reduce indexing resource usage.

We describe the augmentation process in a deductive database approach. Let the relation "acc(parent,child)" contain the parent-child relationships in an XML collection. The transitive closure over the collection is then 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, which allows for a stepwise computation of the transitive closure.

```
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))
...
```

Here, the relation "acc2" contains the (super-context, sub-context) relationships. Only these relationships are then used for computing the relationships with distance three in "acc3". 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 it becomes faster for high distance acc-relations as the number of tuples decreases. The repeated union and the comparison of acc-relations needed in the standard evaluation of a deductive formulation of augmentation is also omitted. We can further improve the efficiency of the algorithm by restricting the augmentation to a maximum 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.

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 12 107 articles. The articles 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 article 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.

We used HySpirit as the platform on which we implemented both the indexing and retrieval functions. We employed a probabilistic aggregation-based approach, which views a document (or collection) as a

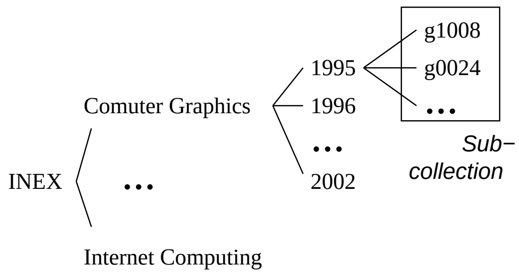


Figure 3: Collection tree

tree and defines the representation of a document component (or sub-collection) as the aggregated representation of its sub-components. The representation of a component includes aspects regarding both content and structure.

During the indexing process we derive a representation of the document's structure via the transitive closure of the document tree, and a representation of the content for each leaf node within the document tree. The content is then propagated up along the tree at retrieval time. The indexing process includes the modelling of the XML elements' contents as propositions in probabilistic object-oriented logic (POOL), which are then translated into tuples in probabilistic relational algebra (PRA). Finally, these are 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>
  <sec>
    Multimedia retrieval ...
  </sec>
</article>
```

Figure 4: XML

```
article(article_1)
article_1[sec_1[multimedia] ]
article_1[sec_1[retrieval] ]
...
```

Figure 5: POOL

In PRA, a document is represented using a number of relations, including "tf" and "acc". 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(term, path)
instance_of(article[1],
    article, cg/1995/g1008)
0.7 tf(multimedia,
    cg/1995/g1008/article[1]/sec[1])
0.8 tf(retrieval,
    cg/1995/g1008/article[1]/sec[1])
0.5 acc(cg/1995/g1008/article[1],
    cg/1995/g1008/article[1]/sec[1])
...
```

Figure 6: PRA

tf calculations applied within the container XML element. 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.

The global termspace of the collection is computed by aggregating the occurrence values of terms within the sub-collections using the augmentation method described in section 2. The following example shows the representation of the term "multimedia" in the termspace of the collection and a sub-collection.

```
0.2 idf(multimedia, INEX)
0.5 idf(multimedia, cg/1995)
```

As a result of our indexing process we created 125 relational databases, where each database contains the index of a sub-collection (the articles within a year of a journal). Each sub-collection maintains a local termspace and structure information, and an additional database contains the global termspace and information on the collection's overall structure.

During indexing we made use of distributed and parallel processing, although due to hardware limitations (we had the use of a non-dedicated dual AMD 800 MHz server with 256MB RAM), we only processed clusters of sub-collections in parallel (journals). Table 1 lists the indexing times for the 18 journals, calculated as the sum of the processing times of their respective journal/year subcollections. We indexed 4-6 journals in parallel, while other processes were also running on the server, which explains the big difference between the reported user and real times. Given a true parallel architecture of 18 processors, the total CPU time to index the INEX collection is 29.3 minutes. Parallel indexing of the whole collection at the sub-collection level (journal/year) would take 4.4 CPU minutes. Note that these times inlcude the creation of the different representations (e.g. POOL, PRA, MDS tuples, FREQ files etc.), the calculations of the different termspaces, and the generation of the SQL commands but not the actual population of the relational databases.

Journal id	Size (MB)	Real (min)	User (min)	CPU (min)
an	13.2	18.6	8.0	6.1
cg	19.1	41.3	12.9	9.6
co	40.4	125.5	27.7	21.0
cs	14.6	37.6	9.4	6.8
dt	13.6	32.3	9.2	6.8
ex	20.3	43.8	13.4	10.1
ic	12.2	17.8	8.3	6.3
it	4.7	8.0	3.2	2.4
mi	15.8	37.8	10.5	8.0
mu	11.3	30.1	7.6	5.7
pd	10.7	23.6	6.9	5.1
so	20.9	61.4	13.9	10.4
tc	66.1	92.1	43.3	29.3
td	58.8	71.2	39.5	26.8
tg	15.2	19.5	9.9	6.9
tk	48.1	55.9	31.6	21.51
tp	62.9	100.8	41.8	28.8
ts	46.1	54.0	29.1	20.2
Max.		125.5	43.3	29.3
Avg.		48.8	18.1	12.9
Per MB		1.7	0.6	0.4

Table 1: Indexing times

3.2 Query processing and retrieval

We used HySpirit and an additional perl script to automatically parse and process the title and keywords components of the INEX topics. The resulting PRA representation of a query contained the query terms with associated term weights and a PRA program implementing a retrieval strategy. For content-only topics the retrieval strategy was based on a simple content-retrieval approach, where the relevance status value of leaf elements were calculated using *tf* and *idf* estimations (section 2.1). For content-and-structure queries the retrieval strategy combined 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 (section 2.5) 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 the retrieved objects.

To implement parallel query processing we optimized 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 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 a local augmentation strategy, simply because a global augmentation would have led to huge resource requirements.

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

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[2] W. B. Croft. What do people want from information retrieval? *D-Lib Magazine*, 1(5), 1995.

[3] W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, editors. *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, 1998. ACM.

[4] Hang Cui, Ji-Rong Wen, and Tat-Seng Chua. Hierarchical indexing and flexible element retrieval for structured documents. In *25th European Conference on Information Retrieval Research (ECIR'03)*, 2003.

[5] D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into XML query processing. *Proceedings of the Ninth World Wide Web Conference*, 33(1–6):119–135, 2000.

[6] W.B. Frakes and R. Baeza-Yates. *Information Retrieval. Data Structures & Algorithms*. Prentice Hall, Englewood Cliffs, 1992.

[7] J.C. French, A.L. Powell, C.L. Viles, T. Emmitt, and K.J. Prey. Evaluating database selection techniques: A testbed and experiment. In Croft et al. [3], pages 121–129.

[8] O. Frieder, D. A. Grossman, A. Chowdhury, and G. Frieder. Efficiency considerations for scalable information retrieval servers. *Journal of Digital information*, 1(5), 2000.

[9] 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.

[10] N. Gövert and G. Kazai. Overview of the Initiative for the Evaluation of XML retrieval (INEX) 2002. In N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas, editors, *Initiative for the Evaluation of XML Retrieval (INEX). Proceedings of the First INEX Workshop. Dagstuhl, Germany, December 8-11, 2002*, ERCIM Workshop Proceedings, Sophia Antipolis, France, March 2003. ERCIM.

[11] T. Grabs, K. Böhm, and H-J. Schek. Scalable distributed query and update service implementations for XML document elements. In Karl Aberer and Ling Liu, editors, *Eleventh International Workshop on Research Issues in Data Engineering: Document Management for Data Intensive Business and Scientific Applications (RIDE-01)*, pages 142–152. IEEE Computer Society, 2001.

[12] T. Grabs, K. Böhm, and H-J. Schek. XMLTM: Efficient transaction management for XML documents. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM-02)*, pages 142–152, 2002.

[13] Yong Kyu Lee, Seong-Joon Yoo, Kyoungro Yoon, and P. Bruce Berra. Index structures for structured documents. In *Digital Libraries*, pages 91–99, 1996.

[14] H. Meuss and C. Strohmaier. Improving index structures for structured document retrieval. In *21st BCS IRSG Annual Colloquium on IR Research (IRSG'99)*, 1999.

[15] S.H. Myaeng, D.-H. Jang, M.-S. Kim, and Z.-C. Zhoo. A flexible model for retrieval of SGML documents. In Croft et al. [3], pages 138–145.

[16] B. Ribeiro-Neto, E. S. Moura, M. S. Neubert, and N. 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.

[17] W. Rogers, G. Candela, and D. Harman. Space and time improvements for indexing in information retrieval. In *Proceedings of 4th Annual Symposium on Document Analysis and Information Retrieval*, 1995.

[18] T. Rölleke, M. Lalmas, G. Kazai, I. Ruthven, and S. Quicker. The accessibility dimension for structured document retrieval. In *Proceedings of the BCS-IRSG, Glasgow*, March 2002.

[19] T. Rölleke, R. Lübeck, and G. Kazai. The HySpirit retrieval platform. In W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, USA*, New York, August 2001. ACM.

[20] R. Sacks-Davis, T. Dao, J. A. Thom, and J. Zobel. Indexing documents for queries on structure, content and attributes. In M. Yoshikawa and S Uernura, editors, *Proceedings of the International Symposium on Digital Media Information Base*, pages 236–245, 1997.

[21] D. Shin, H. Jang, and H. Jin. BUS: an effective indexing and retrieval scheme in structured documents. In *Proceedings of the third ACM Conference on Digital libraries*, pages 235–243, New York, 1998. ACM.

[22] J. E. Wolff, H. Flörke, and A. B. Cremers. Searching and browsing collections of structural information. In *Advances in Digital Libraries*, pages 141–150, 2000.

Determining the Unit of Retrieval Results for XML Documents

Kenji Hatano[†], Hiroko Kinutani[‡], Masahiro Watanabe[⋆], Masatoshi Yoshikawa[∗,†], and Shunsuke Uemura[†]
[†]Graduate School of Science and Technology, Nara Institute of Science and Technology, Japan
[‡]CREST Program, Japan Science and Technology Corporation, Japan
[⋆]Department of Educational and Information Technology, National Institute of Special Education, Japan
[∗]Information Technology Center, Nagoya Univeristy, Japan

Abstract

In the research field of document retrieval using several keywords as a query, retrieval results returned by information retrieval systems are whole documents or document fragments. However, these are not suitable for XML document retrieval since they do not correspond to the information which users are searching for. Therefore, we believe that retrieval results should be portions of XML documents, such as document chapters, sections, or subsections. That is, the most important concern in XML document retrieval is defining units for retrieval results. In this paper, we propose a method for determining a unit for retrieval results to be used in development of a keyword-based XML document retrieval system. Using our method, we can reduce the number of targeted portions of XML documents so that we can speed up searching retrieval results and enhance overall performance of XML document retrieval system.

1 Introduction

XML (Extensible Markup Language) [3] is becoming widely used as a standard document format in many application domains. In the near future, we believe that a great variety of documents will be produced in XML. Therefore, in a similar way to Web search engines, XML document retrieval systems will become very important tools for users wishing to explore XML documents.

In spite of the big demand for XML document retrieval systems, they are not yet available. It is true that many XML query languages have been proposed [2]; however, the XML query languages represent only one kind of retrieval method for XML documents. We believe that XML document retrieval systems should adopt a much simpler form of a query consisting of several keywords. This is because XML documents have various kinds of document structure, so it is hard for users to enter both keywords and document structures as a query into XML document retrieval systems. Therefore, specifying both keywords and document structures of XML documents, as is done with XML query languages, is clearly not suitable for a query to XML document retrieval systems. To cope with this problem, we have chosen development of a keyword-based XML document retrieval system as a research theme.

In order to develop a keyword-based XML document retrieval system, XML documents must first be divided into portions of XML documents. XML is a markup language, so it is easy to automatically divide XML documents into portions of XML documents using their markup [9]. However, if the XML documents are divided as far as possible using their markup, the number of resulting portions of XML documents will become huge. In other words, it takes a very long time to retrieve portions of XML documents related to a keyword-based query using our current XML document retrieval system [8]. For this reason, we have to determine meaningful portions of XML documents as retrieval results and reduce the number of targeted portions of XML documents.

In this paper, we propose a method for determining a unit of retrieval results in order to reduce the number of targeted portions of XML documents. If we reduce the number of targeted portions of XML documents, we can speed up searching retrieval results and enhance overall performance of XML document retrieval system. We think targeted portions of XML documents can be classified as either meaningful or meaningless for users. We call the meaningful portions *CPDs (Coherent Partial Documents)*. If we can eliminate the meaningless portions of XML documents, the number of targeted portions of XML documents will be reduced, with the result that we will be able to perform XML document retrieval more quickly and efficiently than with our current XML document retrieval system. For this purpose, it is important to decide how to define the CPDs of XML documents. This approach has also been adopted in other XML document retrieval systems. Kazai et. al. said that it was important to eliminate stop-contexts in order to enhance scalability of an XML document retrieval system [10]. Here “stop-contexts” has the same meaning as “meaningless portions of XML documents” in our own approach. Moreover, in research topics of Web information retrieval, some researchers have proposed a method for defining a meaningful set of Web pages [4, 12, 14]. Consequently, we believe that this research topic will be important for XML document retrieval in the near future.

The remainder of this paper is organized as follows.

First, we describe our data model of XML document retrieval in Section 2. Then, we explain how to determine meaningful portions of XML documents in Section 3, and report experimental results using information extracted from XML documents in Section 4. Finally, we conclude our paper in Section 5.

2 Our Data Model

In this section, we describe our data model following the notations and data model defined in XPath 1.0 [5].

In our data model, an XML document is modeled as a hierarchical tree. Figure 1 shows the logical structure of a sample XML document. The numbering of the nodes represent document IDs, which are derived using the document order defined in XPath 1.0. Although there are seven types of nodes in the XPath data model, for simplicity, we limit our attention to the root node, element nodes, attribute nodes and text nodes[1]. 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*. 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 the *string-value* of an element node is the concatenation of the string-values of all text-node descendants of the element 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 data 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.

Until now, two kinds of XML document retrieval model based on the XPath data model have been proposed [1]: one is the non-overlapping match [6] and the other is the proximal nodes [13]. Our retrieval model is similar to the model based on the proximal nodes. In other words, our logical model of portions of XML documents is a sub tree whose root node is an element node. Therefore, we can identify a portion of an XML document by the reference number *n* of the root node of the portion of XML document. We refer to such a portion of the XML document as "partial XML document #*n*."

We believe that retrieval results of XML document retrieval systems should be partial XML documents following the XPath data model if we adopt the INEX test collection[2]. For this reason, we divide XML documents of the INEX test collection into partial XML documents, and identify them by using their reference number in our proposed system.

[1] The remaining three types of nodes are namespace nodes, processing instruction nodes and comment nodes.

[2] The INEX test collection is constructed by INEX Project organized by the DELOS Network of Excellence for Digital Libraries.

3 Coherent Partial Document

In order to retrieve partial XML documents based on a keyword-based query, XML documents stored in an XML document retrieval system are required to be divided into partial XML documents. In such a situation, however, the number of divided partial XML documents may become huge[3]. As a result, it may be difficult to perform efficient XML document retrieval. To cope with such problem, it is important to determine CPDs of XML documents in order to reduce the number of targeted partial XML documents.

3.1 Concept of CPD

In our approach, we have to determine CPDs of XML documents. As mentioned previously, the CPD means coherent and meaningful portions of XML documents.

For example, let us consider the case when a user issues a single keyword query "Hatano." Which partial XML documents are relevant as retrieval results to this query? The minimum portion of XML document containing a character string "Hatano" is the partial XML document #25. A text representation of this partial XML document is shown as `<author>Hatano</author>`. However, we do not consider this partial XML document informative enough for the user, because the user cannot know what "Hatano" has authored. On the other hand, returning the whole document is not adequate either. This is because the XML document in Figure 1 has two chapters, and "Hatano" is the author of the second chapter. For this reason, we believe that partial XML document #20 will be the most relevant as a retrieval result of the query. That is, we regard partial XML document #20 as a semantically consolidated granule of documents.

In XML document retrieval, we believe that such semantically consolidated partial XML documents should be retrieved as retrieval results. We call this type of partial XML documents as *Coherent Partial Document (CPD)*. If we can determine CPDs as targeted partial XML documents, the number of targeted partial XML documents will be reduced. This is because the CPDs are not exactly congruent with partial XML documents divided as far as possible using their markup; in short, there may be some partial XML documents smaller or bigger than the CPDs. In order to determine the CPDs, we have already proposed context search approach in our previous paper [7]. *Context search* is used for representing a retrieval method which can return the CPDs as retrieval results of a keyword-based XML document retrieval system. 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.

[3] The number of divided partial XML documents is the same as that of intermediate nodes.

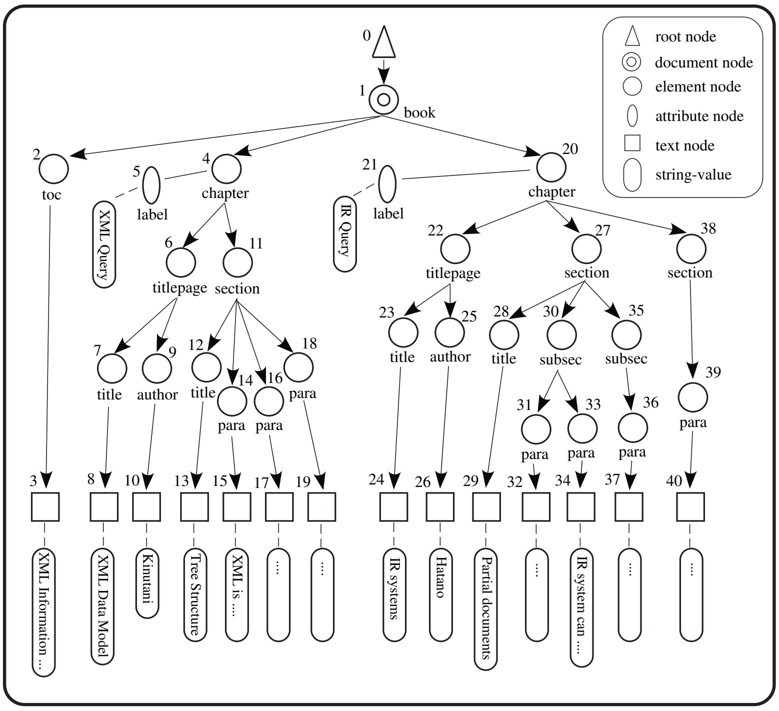


Figure 1: A Tree Representation of an XML Document.

3.2 Context Search Approach

In our context search approach, we are required to find context nodes. The context node of a node in an XML document is an element node which is an ancestor of the text node. Intuitively, the context node gives the boundary of the context of a text node or an attribute node. The context node is defined to be an ascendant node which does not have sibling nodes with same expanded-name. Thus, the element node plays a unique role in the partial XML document defined by the context node.

For example, in Figure 1, the context node of the text node #26 is the element node #20. This is because every node between the paths (#22, #25, #26) does not have a sibling node with same expanded-name, but the node #20 has (i.e. the node #4). This implies that the role of the text node #26 is unique within the partial XML document #20, but not unique within the partial XML document #1. In fact, we can observe the text node #26 represents the author of the second chapter (partial XML document #20), while the node #10 represents the author of the first chapter (partial XML document #4). As another example, let us consider the text node #32. The parent node #31 has the sibling node #33 with the same expanded-name. If we apply the rule explained in the above example, the context node of #32 would become #31. However, we consider the element node #31 does not give a proper boundary of the context of #32. To avoid such cases, we ignore the sibling nodes of the parent node. To find the context node of the node *n*, we start from the grandparent of *n*, and go up until we find a node having a sibling with the same expanded-name. Hence, the context node of #32 is defined to be #30. The formal definition of context nodes follows:

Definition 1 (Context node) *For a text node or an attribute node n in an XML document D, the context node of n in D is denoted by context*(n)*, and defined as follows:*

1. *For an attribute node n, context*(n) *is the parent element node of n.*
2. *For a text node n, context*(n) *is defined as follows: Let g*(n) *be the grandparent node of n. Then, context*(n) *is the lowest node m on the path between g*(n) *and the document node* n_d *such that m has a sibling node having the same expanded-name with m. If such a node m does not exist, context*(n) *is defined to be* n_d.

The definition of context node is based on the topology of document trees, especially on the number of siblings with same expanded-name. Identification of context nodes is easily done by scanning the XML document instances.

If we use the context search approach to determine CPDs of the example XML document in Figure 1, we can get partial XML document #4, #20, #27, #30, #35, and #38. However, we also want to get other partial XML documents as CPDs like partial XML document #11 in Figure 1. As the formal definition of context node shows, this type of partial XML documents cannot be derived as a CPD using the context search approach. As a result, we can use the context search approach to reduce the number of targeted partial XML documents, but cannot use it to strictly determine CPDs of XML documents.

3.3 Statistical Approach

In context search approach, we utilize only structural information of XML documents to determine CPDs. However, as it was shown, we cannot exactly determine the CPDs which we defined in Section 3.1 using only structural information of XML documents. Therefore, we have to consider not only structural information but also other information of XML documents for determining CPDs.

We distinguish three types of information in XML documents, such as *structural information*, *content information*, and *statistical information*. These types of information are extracted by structure analyzer and content analyzer in our XML document retrieval system.

- structure analyzer
  Using the structure analyzer, we can analyze structural information, such as element names, their path expressions, and element relationships in XML documents. The structure analyzer is composed of an XML parser, so it is easy to extract the structural information. Moreover, if we extract only structural information, we reconstruct original XML documents. Thus, the structure analyzer generates an index file based on structural 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 names of root node, their path expressions, IDs of targeted partial XML documents, and the number of words in the partial XML documents. Using this analysis, it becomes possible to derive CPDs statistically. For example, we can get 24 partial XML documents from the XML document shown in Figure 1, because the number of intermediate nodes of the XML document is 24. However, the size of some partial XML documents is too small, so that we believe that they are not adequate as CPDs, because they are not informative enough. Therefore, we utilize the number of words of targeted partial XML documents in order to eliminate small partial XML documents from targeted partial XML documents.

- content analyzer
  The content analyzer counts frequencies of words which are included in partial XML documents and calculates weights of words as feature vectors of each partial XML documents. The weights of words are calculated by using a keyword weighting strategy of having specialized in partial XML document retrieval.

  Table 2 shows a result of analyzing the XML document shown in Figure 1 using the content analyzer. If we use this analysis, we can retrieve partial XML documents related to a keyword-based query based on the vector space model because we can generate an inverted file for partial XML document retrieval. Moreover, we can find the number of tokens which are included in partial XML documents from the content information. We think the number of tokens is also statistical information, so that we can utilize it to determine CPDs of XML documents.

Eventually, we utilize these two analyses extracted by both structural analyzer and content analyzer of our XML document retrieval system, and generate a compound index file for efficient retrieval of partial XML documents using a keyword-based query. Needless to say, the partial XML documents contained in the compound index file are CPDs determined by analyzing the statistical information.

4 Experimental Evaluation

As we described in the previous section, the most important concern of XML document retrieval is to determine CPDs of XML documents using the statistical information. However, the size of partial XML documents differ, so that we cannot define an appropriate size of CPDs easily. Therefore, we perform many kinds of experiments and report the experimental results in order to determine threshold values of the statistical information.

4.1 Experimental Setup

Our prototype system for determining threshold values of the statistical information performs the following processes:

1. Our system analyzes XML documents using an XML parser called Apache Xerces[4], and constructs DOM trees of the XML documents. We use the XML documents included in the INEX test collection which consists of a set of journals of IEEE Computer Society. The size of the INEX test collection is about 500 MBytes and it contains 12,107 articles.

2. Our system divides the XML documents into partial XML documents as far as possible. The number of divided partial XML documents is about seven million, and the number of element types of partial XML documents is 181[5]. Moreover, it also carries out stemming and stopword removal to devided partial XML documents.

3. In order to determine CPDs of the XML documents, we investigate several statistical information, such as the number of words n^w and the number of tokens n^k, which were derived prior to and after stemming and stopword removal, respectively. Moreover, we also investigate the ratio of

[4] `http://xml.apache.org/xerces-j/index.html`

[5] In DTD of the INEX test collection, 192 element types of partial documents are defined. We think some element types of partial XML documents have no word in themselves.

Table 1: Structural analysis of an XML document shown in Figure 1.

partial doc. ID	element type	path expression	# of words
1	`book`	`/book[1]`	324
2	`toc`	`/book[1]/toc[1]`	47
4	`chapter`	`/book[1]/chapter[1]`	92
6	`titlepage`	`/book[1]/chapter[1]/titlepage[1]`	9
7	`title`	`/book[1]/chapter[1]/titlepage[1]/title[1]`	8
...	...	...	...
38	`section`	`/book[1]/chapter[2]/section[2]`	18
39	`para`	`/book[1]/chapter[1]/section[2]/para[1]`	18

Table 2: Content analysis of an XML document shown in Figure 1.

partial doc. ID	word					# of tokens
	data	hatano	information	···	xml	
1	0.435	0.123	0.231	···	0.645	245
2	0.241	0	0.728	···	0.824	5
4	0.781	0	0.765	···	0.645	183
···	···	···	···	···	···	···
39	0.303	0	0.116	···	0.183	2

tokens R defined as follows:

$$R = \frac{n^k}{n^w} \tag{1}$$

4. Using three types of statistical information of each partial XML document, we discuss which partial XML document is meaningful or not. If the XML document is meaningful portion of the XML documents, it is called CPD.

5. We also utilize the number of partial XML documents N as the statistical information, because N is useful for evaluating overall performance of XML document retrieval system. XML document retrieval system has to enhance overall performance for retrieving partial XML documents.

6. Finally, we determine the adequate size of partial XML documents as CPDs considering the statistical information.

4.2 Experimental Results

Table 3 shows the number of partial XML documents N, the number of words n^w, the number of tokens n^k, and average ratio of tokens R_{ave} in the partial XML documents. Here, R_{ave} is defined as follows:

$$R_{ave} = \frac{\sum_i n_i^k}{\sum_i n_i^w} \tag{2}$$

The elements in the table are sorted in descending order of average number of words n_{ave}^w. As Table 3 shows, the elements located at higher levels of the document structure of the INEX test collection, e.g. `books`, `journals`, `articles`, were ranked higher, because the size of the partial XML documents whose root node is a higher-level-element of the XML document are larger. We also

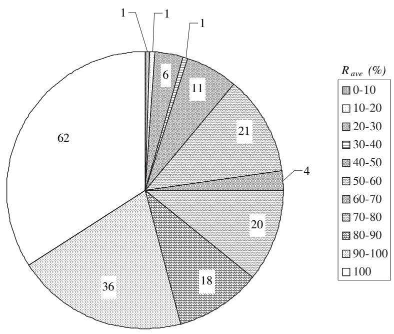


Figure 2: The number of element types of partial XML documents based on R_{ave}.

found that R_{ave} of the partial XML documents which contain many number of words and tokens in themselves is smaller than those of others. Moreover, the number of element types of the partial XML documents which have one hundred tokens or more is at most 20. In short, we can forecast that the size of almost all partial XML documents is small, so that they are not informative for users. Therefore, the partial XML documents whose R_{ave} is large may be not suitable for CPDs.

At the same time, we focus on the number of partial XML documents N (see Table 4), R_{ave} of the partial XML documents whose n_{ave}^w is small is approximately 100%, so that the partial XML documents may not be suitable for CPDs. From the above-discussed points, we think it is hard to determine CPDs based on element types of partial XML documents, because the number of words, n^w, (or the number of tokens, n^k) of each partial XML documents vary widely. Therefore, we need to analyze the statistical information in more detail.

Figure 2 shows the classification of partial XML documents based on average ratio of tokens R_{ave}. The values

Table 3: Statistical Analysis of Partial XML Documents (Top 20 in descending order of n^w_{ave})

element type	# of partial documents N	# of words n^w Ave. (n^w_{ave})	Max. (n^w_{max})	Min. (n^w_{min})	# of tokens n^k Ave. (n^k_{ave})	Max. (n^k_{max})	Min (n^k_{min})	R_{ave} (%)
books	125	337,099	894,853	42,734	28,897	64,181	6,341	8.57
journal	860	48,997	129,417	17,192	7,342	14,903	3,982	14.99
article	12,107	3,478	28,824	32	974	4,727	29	28.02
bdy	12,107	2,884	28,276	13	765	3,943	11	26.55
index	117	2,585	10,728	381	623	1,593	230	24.13
bm	10,060	604	10,074	2	310	2,863	2	51.40
sec	69,733	501	16,089	1	201	2,613	1	40.24
dialog	194	458	2,424	21	212	906	19	46.45
bib	8,543	350	5,690	8	194	1,959	8	55.48
bibl	8,551	350	5,690	8	194	1,959	8	55.48
tgroup	5,822	318	3,961	2	62	401	2	19.58
ss1	61,490	280	11,857	1	127	2,109	1	45.61
app	5,863	262	7,698	2	138	1,353	2	52.72
tbody	5,820	233	3,851	2	49	390	2	21.23
ss3	127	213	1,361	9	91	325	9	42.88
ss2	16,288	189	11,640	1	92	1,261	1	48.90
tbl	12,740	159	3,965	6	41	414	6	26.17
proof	3,765	122	3,815	5	60	801	5	49.71
dl	353	120	1,562	11	52	745	5	43.90
l4	117	92	794	6	37	231	6	40.83
	6,802,061	2,222	894,853	1	234	64,181	1	38.85

Table 4: Statistical Analysis of Partial XML Documents (Top 20 in descending order of N)

element type	# of partial documents N	# of words n^w Ave. (n^w_{ave})	Max. (n^w_{max})	Min. (n^w_{min})	# of tokens n^k Ave. (n^k_{ave})	Max. (n^k_{max})	Min (n^k_{min})	R_{ave} (%)
p	762,223	35	3,272	4	27	313	4	78.43
tmath	574,395	2	288	1	2	60	1	96.09
ref	395,933	5	15	3	5	15	3	100.00
it	394,549	2	149	1	2	96	1	97.21
au	317,457	2	28	1	2	26	1	99.96
entry	317,384	4	167	2	4	50	2	99.19
snm	311,257	1	15	1	1	15	1	100.00
ip1	178,788	32	1,529	1	24	400	1	74.69
obi	164,908	3	226	1	3	142	1	98.52
ti	159,565	4	65	1	4	48	1	99.13
pdt	154,978	4	7	1	1	7	1	100.00
yr	154,943	1	7	1	1	7	1	100.00
sub	154,324	1	18	1	1	15	1	99.82
bb	149,168	20	237	2	19	164	2	97.33
st	136,935	1	36	1	2	27	1	99.56
fnm	135,192	1	9	1	1	9	1	100.00
atl	134,247	5	70	1	5	54	1	99.35
b	123,463	2	273	1	2	86	1	98.54
pp	108,134	1	10	1	1	10	1	99.99
scp	107,544	1	18	1	1	14	1	99.99
	6,802,061	2,222	894,853	1	234	64,181	1	38.85

in the circle graph mean the number of element types of partial XML documents in each R_{ave} classified into eleven different classes. As in the figure, average ratio of tokens R_{ave} of 62 element types of partial XML documents is 100%, and that of 36 element types between 90% and 100%. Almost all partial XML documents classified into $90 \leq R_{ave} \leq 100\%$ lie at the end of XML tree which expresses an XML document of the INEX test collection[6], and have small number of words and tokens.

At the same time, we draw correlation between average ratio of tokens R_{ave} and average number of tokens n^k_{ave} as Figure 3. As in Figure 3, average number of tokens of partial XML documents whose ratio of tokens is more than 90% is less than 80, so that we can find that the size of partial XML document is small if R_{ave} of the partial XML document is large.

From the above-mentioned points, we believe that we can roughly determine CPDs of XML documents if we utilize the number of words n^w, the number of tokens n^k, and the ratio of tokens R. If we would like to strictly determine CPDs of XML documents, we may be able to utilize query/answer sets of a test collection. At the present stage, we summarize the definition of CPDs of XML documents as follows:

- In these experiments, we carried out stemming and stopword removal as pre-processing before analyzing the statistical information. On the other hand, we also analyzed the statistical information without pre-processing. Comparing these analyses, we cannot find any difference, so that we think that the statistical information is mostly unaffected by pre-processing.
- Almost all partial XML documents whose ratio of tokens R are less than 90% contain less than one thousand tokens. Therefore, we believe that the number of tokens of a CPD may be at most one thousand.

[6] XML documents of the INEX test collection can be expressed as one XML document.

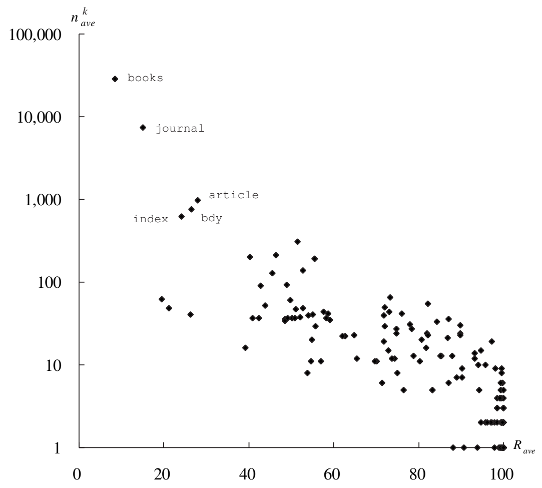


Figure 3: Correlation between R_{ave} and n^k_{ave}.

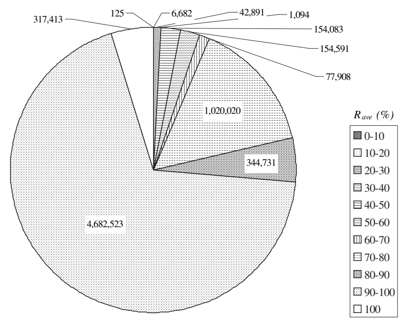


Figure 4: The number of partial XML documents based on R_{ave}.

- As we described in Section 3.2, meaningful partial XML documents appear repeatedly in XML documents. Consequently, the partial XML documents whose frequency N is large and whose ratio of tokens R is small are suitable for CPDs.

- We think that the partial XML documents whose ratios of tokens R are 100% must not be suitable for CPDs. Moreover, the partial XML documents whose ratio of tokens R is between 90% and 100% may be not suitable for CPDs. If we assume that the partial XML documents whose ratios of tokens R_{ave} are more than 90% are not CPDs of XML document of the INEX test collection, the number of partial XML documents which are indexed as a inverted list will be reduced to about one-third (see Figure 4). Furthermore, if we can utilize query/answer sets of the INEX test collection, we believe that we may be able to strictly determine CPDs of XML documents. We could utilize some query/answer sets of the INEX test collection[7], so that we also analyze the statistical information of the partial XML documents which participants of INEX project evaluated as answer documents to a query[8]. As in Table 5, average ratios of tokens of answer documents are less than 70%, so that we may be able to assume that the partial XML documents whose ratios of tokens R_{ave} are more than 70% are not CPDs. If we assume, the number of partial XML documents which should be CPDs will be reduced to about one-tenth (see Figure 4).

5 Conclusion

In this paper, we proposed a method for determining CPDs of XML documents in order to reduce the number of targeted partial XML documents. We only discussed a brief statement on the efficiency of our statistical approach, because we could not utilize query/answer sets of the INEX test collection, but then we could forecast that we will be able to reduce the number of targeted partial XML documents and perform efficient keyword-based XML document retrieval, so that overall performance of XML document retrieval system may be enhanced.

However, we cannot carry out in-depth experiments for verification of our statistical approach using the INEX test collection in this paper. Therefore, we have to verify the effectiveness of our approach as soon as possible. Moreover, if we can determine CPDs of XML documents, 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 partial XML documents, we believe. Lalmas and we have already studied solving this problem for semi-structured documents such as SGML and XML documents [8, 11], so that we will adopt these approaches to our XML document retrieval system. Furthermore, in this paper, we assumed that a query of XML document retrieval consists of several keywords in this paper; however, we have to consider queries specifying both contents and document structures as is done with XML query languages. Therefore, our next step will be developing an XML document retrieval system which can deal with such queries.

Acknowledgments

This work was partly supported by the Ministry of Education, Culture, Sports, Science and Technology, Japan, under grants #11480088, #14019064, and #14780325, and by CREST Program "Advanced Media Technology for Everyday Living" of Japan Science and Technology Corporation, Japan.

[7] In the INEX test collection, the query/answer sets are referred to as INEX relevance assessment.

[8] Answer documents are evaluated as "3E" based on the INEX relevance judgement by participants of INEX project.

Table 5: Statistical Analysis of Answer Partial XML Documents.

topic ID	# of answer doc.	sum of n^w	n^w_{ave}	sum of n^k	n^k_{ave}	R_{ave}(%)
31	4	5,333	1333.25	2,178	544.50	40.84
32	35	34,660	990.29	11,363	324.66	32.78
33	2	227	113.50	139	69.50	61.23
34	66	224,817	3406.32	50,624	767.03	22.52
36	31	5,868	189.29	3,065	98.87	52.23
37	138	35,051	253.99	14,833	107.49	42.32
38	111	102,736	925.55	29,932	269.66	29.13
39	48	90,561	1886.69	26,045	542.60	28.76
40	123	455,587	3703.96	120,760	981.79	26.51
41	57	3,526	61.86	2,216	38.88	62.85
42	91	25,043	275.20	11,778	129.43	47.03
43	15	58,971	3931.40	13,673	911.53	23.19
45	57	145,362	2550.21	47,449	832.44	32.64
46	26	15,674	602.85	5,591	215.04	35.67
47	22	177,377	8062.59	32,356	1470.73	18.24
48	65	117,851	1813.09	26,750	411.54	22.70
49	9	32,703	3633.67	7,149	794.33	21.86
51	26	36,592	1407.38	11,449	440.35	31.29
52	15	37,402	2493.47	11,551	770.07	30.88
53	34	73,217	2153.44	22,187	652.56	30.30
58	210	441,319	2101.52	125,981	599.91	28.55
60	174	46,235	265.72	20,957	120.44	45.33
Ave.	62	98,460	1916	2,7183	504	35

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.M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). `http://www.w3.org/TR/REC-xml`, Oct. 2000. W3C Recommendation 6 October 2000.

[4] S. Chakrabarti. Text Search for Fine-grained Semi-structured Data. In *Tutorial Notes of the 28th International Conference on Very Large Data Bases*, pages 115–135, 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] G. Kazai and T. Rölleke. A Scalable Architecture for XML Retrieval. In *Proc. of the First Workshop of the Initiative for the Evaluation of XML Retrieval*. ERCIM, Mar. 2003. (to appear).

[11] 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.

[12] W.-S. Li, K.S. 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.

[13] 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.

[14] 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[1] *James A. Thom*[2] *Alexander Krumpholz*[1] *Ian Mathieson*[1]

Peter Wilkins[1] *Mingfang Wu*[1] *Nick Craswell*[3]

David Hawking[3]

[1]CSIRO Mathematical and Information Sciences
Private Bag 10, South Clayton MDC, VIC 3169, Australia

[2]School of Computer Science and Information Technology, RMIT University
GPO Box 2476V, Melbourne 3001, Australia

[3]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 contraints for the results. Answers are extracted from ranked documents and document fragments based on the projection contraints 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 architecture 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 our 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

```
<te>fig|p</te>
```

```
<?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 Brocker Architecture
  interface invocation interoperability
  communication protocols IDL
</Keywords>
</INEX-Topic>
```

Figure 1: INEX topic 14

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 Enterprise Search Engine [2] and combines full-text

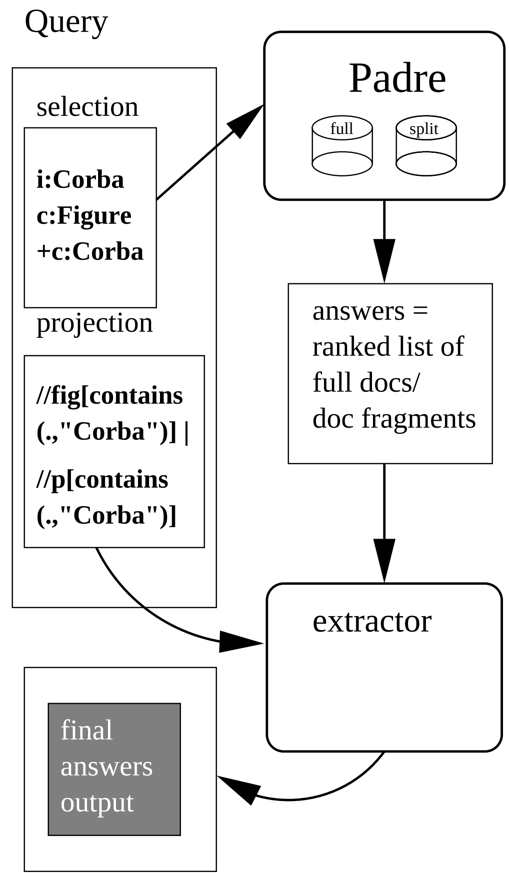


Figure 2: System architecture

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.

```
/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/atl → 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

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 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

```
/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

further extracting would be required. We describe this further in the next section.

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 of "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:

```
</query>
<query topic-id="14">
<selection>
  i:Corba
  i:architecture
  c:Figure
  c:Corba
  c:Architecture
  [CORBA ORB Object Request Brocker
   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

1. load the fragment, get the name of the embedding article, load the full article A.
2. apply the XPath projection to the article A; this returns $e_1, e_2, \ldots e_n$ 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 inital 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 lowercase 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 contraints should be added to the projection.

4 Experiments and Results

We submitted three official runs to INEX:

- queries on *full* articles (run 1)
- queries on *split* articles (run 2)
- *manually* contructed queries on split articles (run 3)

Subsequently we also explored:

- queries on split articles with *post-projection fragment reranking* (run 4)

and corrected a bug with run 1:

- queries on *full* articles – revised (run 5)

Results for runs 2, 3, and 5 on both the content-and-structure (CAS) and content-only (CO) topics are shown in Figures 8, 9, and 10 respectively. These figures show results for our runs (wide red line) with a comparison to other systems.

We also analysed results on topic 14 in more depth. Results for runs 2, 3, 4 and 5 on topic 14 are shown in Figures 11, 12, 13 and 14 respectively. These graphs show relevance judgements for the 100 highest ranked answers for each run. Each answer corresponds to a vertical bar of about 2mm

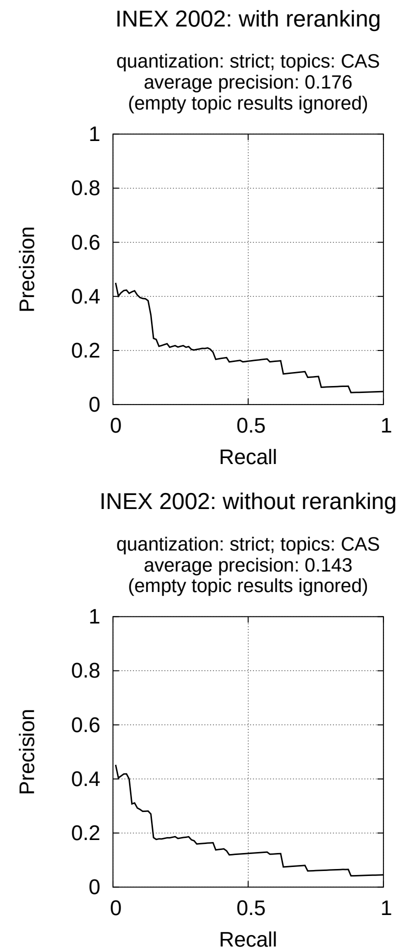


Figure 7: Nine queries on split articles (run 4) with and without post-projection reranking

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. For comparison we have also included the optimal ranking in Figure 15 which shows there is still considerable room for further improvement in XML retrieval.

Results for run 4 on a limited set of topics is shown in Figure 7. The reranking could only be applied to nine queries where the target elements also appear within the content word constraints. For such queries the post-projection reranking of fragments is effective as many unjudged elements were returned. This is very clearly borne out with topic 14, as can be seen from comparing Figure 14 with Figure 12. Overall the performance of the nine queries with reranking (top graph in Figure 7) is better than without reranking (bottom graph in Figure 7).

In topic 14 the manually constructed query performed worse than the automatically generated query using the query translator. However as shown in Figures 9 and 10 generally the manually constructed queries performed much better than the automatically generated queries for the CAS topics. But this was not the case for the CO topics as shown in Figures 9 and 10, perhaps because less effort was spent on improving these queries.

Our draft version of this paper presented at the INEX workshop as well as another of our papers [1] has a claim, based on the erroneous run 1, that using the collection containing documents and document fragments (run 2) was more effective than using just the full documents. However the new run for the full documents (run 5) invalidates this claim as shown by comparing Figure 10 and Figure 8, in fact the split performed worse.

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*, 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.

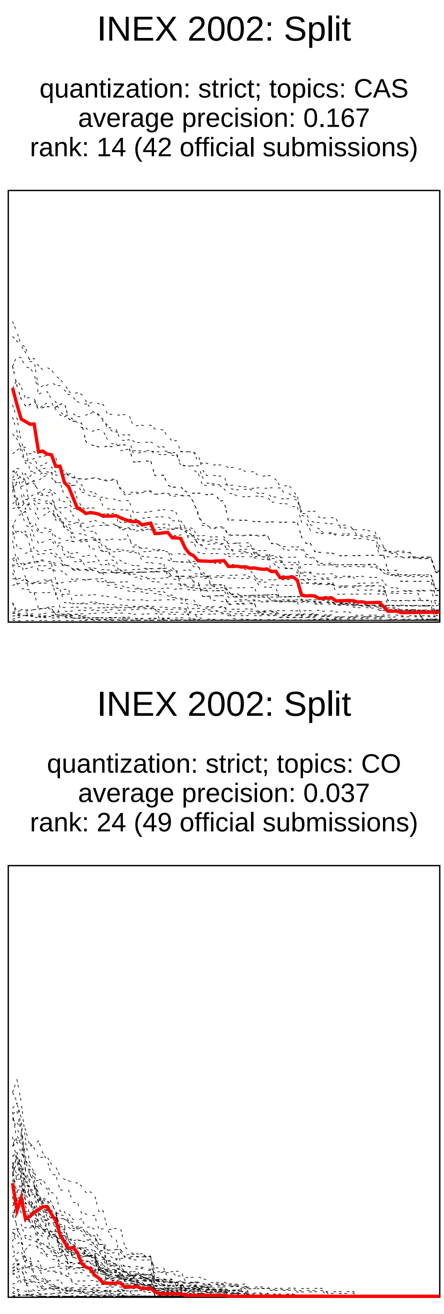


Figure 8: Queries on split articles (run 2)

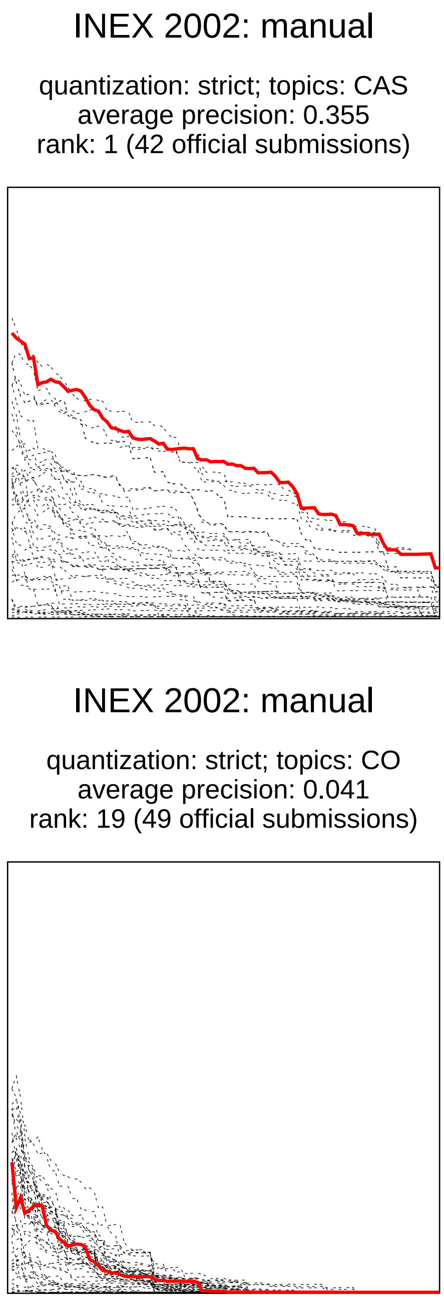


Figure 9: Manually improved queries (run 3)

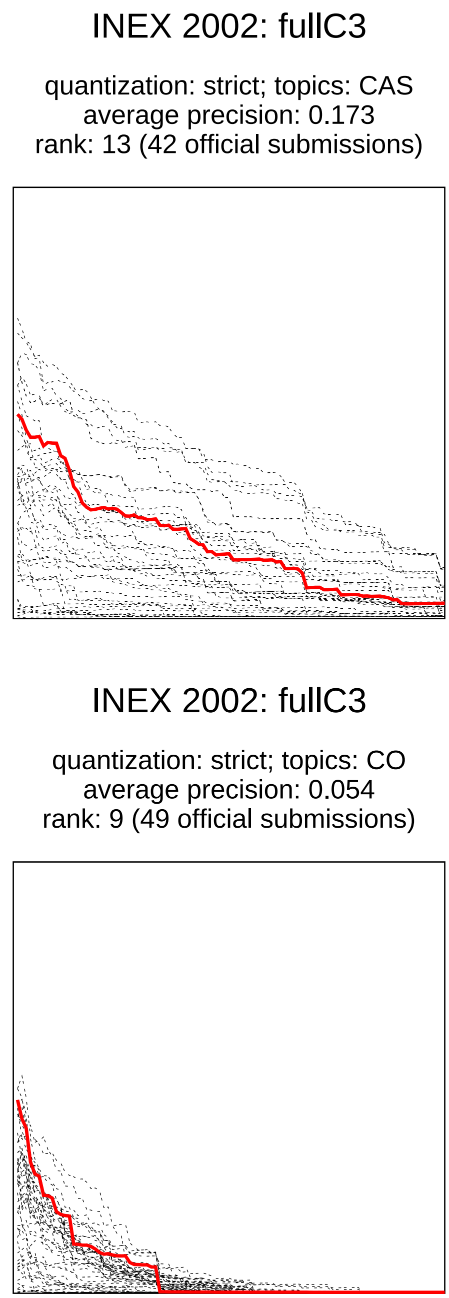


Figure 10: Queries on full articles (run 5)

Figure 11: Results for Topic 14 — query on full articles (run 5)

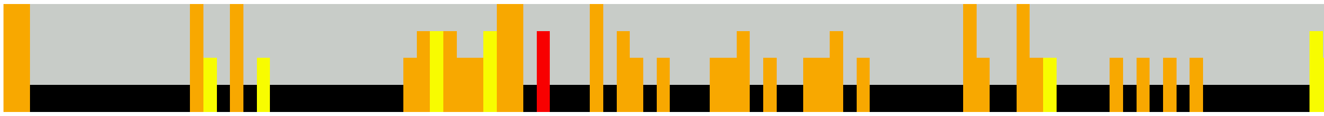


Figure 12: Results for Topic 14 — query on split articles (run 2)

Figure 13: Results for Topic 14 — manual queries (run 3)

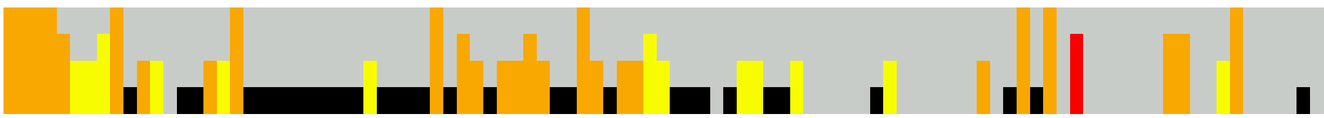


Figure 14: Results for Topic 14 - query on split articles with further reranking of final answers (run 4)

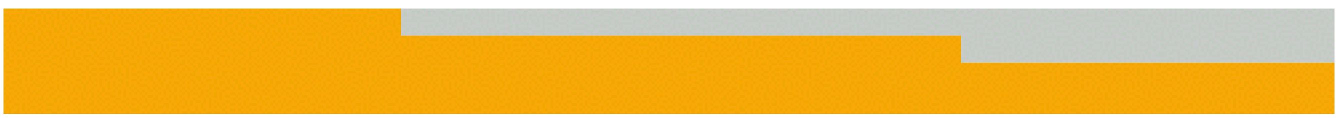
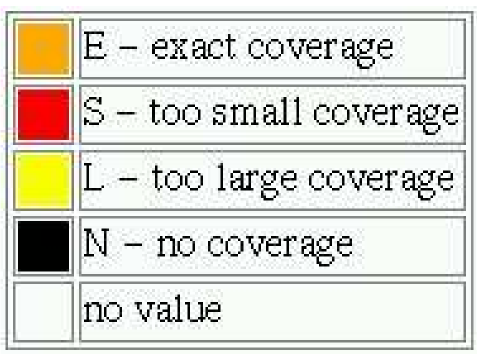


Figure 15: Results for Topic 14 - optimal ranking (relevance only) with first version of relevance judgements

In the above figures, the results are shown from left (highest ranked) to right. The height of the bar represents the relevance and the colour of the bar indicates the coverage as shown below:



JuruXML – an XML retrieval system at INEX'02

Yosi Mass, Matan Mandelbrod, Einat Amitay, David Carmel, Yoelle Maarek, Aya Soffer
IBM Research Lab
Haifa 31905, Israel
+972-3-6401627
{yosimass, matan, einat, carmel, 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 a generic 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 Search, Information Retrieval, Vector Space Model.

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 XPath[9] or XQuery [10], which are 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 [3], a full-text information retrieval system developed at the IBM Research Lab in Haifa, to handle XML documents. INEX provided a useful framework to evaluate the capabilities of our query format and ranking methods. 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. 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 sophisticated 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 [10]) 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[1]. Documents that contain the query or part of it as a subtree are returned as results. XML attributes are queried using the same syntax used in the XML documents[2].

[1] We add an artificial root node that encloses the whole query so as to make it a valid XML data

[2] 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 or 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)

2.1.1 Target elements

The user can accompany the query with an optional list of target elements *(te)* to be returned. If there are no defined *te*'s then 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.

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 <fnm>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

```
<bdy> <sec>+"nonmonotonic reasoning"</sec> </bdy>
<hdr>
    <yr>+1999</yr>
    <yr>+2000</yr>
</hdr>
<tig> <atl>-calendar –"call for papers"</atl> </tig>
```

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 below 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 gurantee 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 we apply the CO rules as described above.
- We 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:

```
<bibl>+QBIC</bibl>
+"image retrieval"
QBIC. IBM. image. video. "content query" . "retrieval
system"
```

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>
</article>
1996 1997 hypercube mesh torus toridal non-numerical
database
```

Note that according to our syntax, result candidates need to contain at least one of the years 1996 or 1997.

3.3 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 XML fragments 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 (journals are built from <sec1> nodes followed by <article> nodes that belong to that section). 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 [4]. 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 *cr* function in Section 6.1).

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 only 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 occurrences of t_i under 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 [3], 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 <*tag*> a pair *(_s_.tag, c)* for the tag start and *(_e_.tag, c)* for the tag end with *c* the path leading to the *tag*. 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 [1], 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 [3] stores all index terms (that form the lexicon of the system) in a *Trie* data structure (see for example [8]) 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[3] 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.

In future work we investigate how to support various levels of granularity in one index based on ideas taken form [5, 6]. 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 retrieval algorithm is described below:

1. Parse the query and create a list of terms of the form $t_i\#c_i$
2. Expand each term $(t_i\#c_i)$ to relevant terms $(t_i\#c_k)$ that resemble it from the index (see Section 5.2.1)
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 query tree structure (see section 5.2.2)

Figure 4: Retrieval 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:

<bibl>QBIC</bibl>

It is parsed into "qbic#/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 Result filtering

The retrieval process could potentially assign a non-zero score to any document containing parts of the query based on the selected scoring function. While we want such matches to contribute to the score, we also wish to assure that the documents conform to the well-specified parts of the query. This is achieved by post-filtering

This filtering is handled as follows. A "tree" representing the XML fragments associated with the query is created to represent the logical structure of the query. Each node in the tree corresponds to a single query term (either a content or context term). For each document that was assigned a non-zero score by our scoring model, we extract the query term's instances together with their offsets in the document (as stored in the index). We then confirm that the constraints imposed by the query tree hold in the specific document. This includes constraints imposed by +/- operators as well as instance level constraints. (For example for the query <+author>John Doe</author> the filtering verifies that only documents that contain both John and Doe under the same <author> instance are returned).

The filtering process is also responsible for filtering the required target elements (*te*) as defined by the user (see section 2.1.1 above). If there are no target element defined then the whole document is returned. Otherwise we return all *te*'s instances that satisfy the query constraints (or all *te* instances if there are no query constraints on the *te* – e.g. return all <author> of articles with <title>databases</title> from <yr>2002</yr>)

6. INEX RUNS

We conducted three INEX runs. For the first two runs, we applied the automatic query translation rules specified in section 3 above, while in the 3rd run we performed some manual editing of the query attempting to better fit the topic's <Description>.

6.1 First run – assigning weights to individual contexts

In the first run we employed the ranking method of formula (2) using the following context resemblance function

$$cr(c_i, c_k) = \begin{cases} \dfrac{1+|c_i|}{1+|c_k|} & c_i \textit{ subsequence of } c_k \\ 0 & \textit{otherwise} \end{cases}$$

where $|c_i|$ is the number of tags in the given query context and $|c_k|$ is number of tags in the expanded context. Thus, for example,

cr("/article/bibl", /article/bm/bib/bibl/bb") = 3/6 = 0.5

It is easy to see that $0 < \boldsymbol{cr} \leq 1$ and it is equal to 1 if and only if the query context is identical to the expanded context. For CAS topics this run was ranked 4th with Av. Precision 0.320 (see figure 5 below).

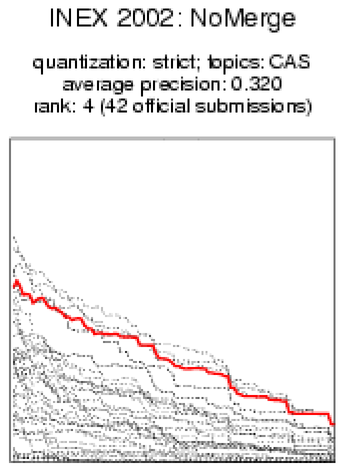


Figure 5 – individual weights

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

$$w(c_i) = (|c_i| + 1)$$

For example, the weight of the context in the query term "qbic#/bibl" is 2. For CAS topics this run was ranked 2nd with Av. Precision 0.352 (see figure 6 below)

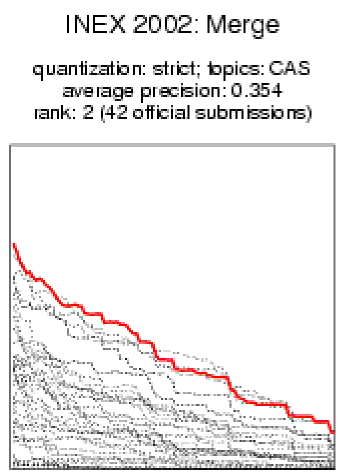


Figure 6 – CAS topics merge contexts

This result shows that merging contexts yields better results then the approach tested in the first run. In section 6.4 we analyze the reasons for this behavior.

For CO topics this run was ranked 10 with Av. Precision 0.053. As described above we didn't have dynamic component level statistics and for the CO topics we returned either the whole article or a sec. We expect that with dynamic component statistics we will achieve much better results.

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 7.

```
<Title>
   <te>article</te>
   <cw>Hypertext Information Retrieval</cw>
   <ce>article</ce>
   <cw>Hypertext Information Retrieval</cw>
   <ce>bib/bibl/bb/atl</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 7: INEX topic 18

This topic was translated for the first two runs into:

<article>
Hypertext Information Retrieval
</article>
<bib><bibl><bb><atl>
Hypertext Information Retrieval
</atl></bb></bibl></bib>

While it was expressed, in the third manual run as

<article>
Hypertext Information Retrieval
</article>
<+bib><bibl><bb><atl>
Hypertext Information Retrieval
</atl></bb></bibl></bib>

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. This run was ranked 3rd in the CAS topics.

6.4 Comparing the Runs

We compare here the first 2 runs ignoring the manual run. We achieved quite good results for the CAS topics and average results for the CO topics. Since for the INEX runs we didn't have dynamic component level statistics we didn't expect good results for CO topics. Instead we focus on the CAS topics and by looking at the first 2 runs it turned out that the approach that merges context gave better results then the approach that weights contexts by their resemblance to the user query context. This can be explained by looking at formula 2 where $W_X(t,c)$ is defined as -

$$W_X(t,c) = tf_X(t,c) * idf(t,c)$$

where x stands for either D or Q and

- $tf_x(t,c)$ is a monotonic function of the number of occurrences of (t,c) in x.
- $Idf(t,c) = log\ (|N|/|N_{(t,c)}|)$ with $|N|$ = total number of documents in the collection and $|N_{(t,c)}|$ = number of documents containing (t,c)

Since in formula 2 each term t is split into different contexts (t,c_k) it might happen that a given (t,c_k) would receive a very high *idf* value because (t,c_k) is very rare in spite of t being very common. In future work we investigate how to compensate for this behavior.

6.5 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 [7])

```
<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 AND FUTURE WORK

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 then presented two ranking methods that combine IR ranking for free text with XML structure ranking. One approach assigns different weights to term occurrences under different contexts and the other merges all occurrences of document terms that match a query term. We achieved very good results on the CAS topics where the first run was ranked 4th and the second run was ranked 2nd among all INEX submissions.

In a following work we further investigate more models of structure ranking by introducing different *Context Resemblance* functions. We also investigate different levels of context merging that cover the scale between no context merging at all to the full context merging models that were presented in this paper. For CO type topics we investigate a dynamic component level statistics that should allow to select the most relevant component when target elements are not defined.

8. REFERENCES

[1] 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.

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

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

[4] 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.

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

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

[7] Initiative for the evaluation of XML retrieval http://qmir.dcs.qmul.ac.uk/INEX/

[8] Donald E Knuth, The art of computer programming: sorting and searching (vol 3), Addison Wesley, 1973.

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

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

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

GREYC CNRS-UMR 6072
University of Caen, France

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-centric XML documents. We present some experiments we have led on clustering the INEX[1] 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 retrieved in answer 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 is data-centric, but text-centric 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 data-centric 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-centric XML documents (section 2) and experiment with them using a well-known partitional 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

*This work is supported by the Academy of Finland (project 50959; DoReMi - Document Management, Information Retrieval and Text Mining)

[1]Initiative for the evaluation of XML Retrieval (http://qmir.dcs.qmw.ac.uk/inex/)

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 discriminative 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., 'ss1' and 'ss2' for sub-section of level 1 and 2 are distinct). The only preprocessing we made was to prune 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 to many documents is less useful than a term common to only a few documents. This is the motive for combining a term's frequency with its inverse document frequency, which is the division of the total number of documents in the collection by the total number of documents where this 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 $cosine(\overrightarrow{d_1}, \overrightarrow{d_2})$ simplifies to the dot product $(d_1 \cdot d_2)$. Because their results are very similar in nature, cosine similarity can be prefered 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 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 hierachical 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.

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

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 dendrogram). Note from the nature of entropy that its optimal score is obtained with singleton clusters and therefore entropy 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 of the experiments, the only classification existing for the INEX collection were the year and journal volume in which an article was published. For more consistency, we have 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 the corresponding 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 inappropriate 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. Therefore, we plan in further work to use the manual assessments of the INEX evaluation, originally aimed at information retrieval systems, so as to evaluate the relevance consistency of documents clusters.

Finally, the clusterings have been evaluated according to the 18 journals where the documents were published, plus the additional volume class. The 12,232 documents of the INEX collection have thus been mapped to 19 classes.

Of course, in order to keep the experiments fair, we pruned all document elements containing the name of the journal where the document was published. 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[2].

We have run k-means with k $\in$ {5, 10, 15, 20, 25, 35} for text-only, tags-only and tags&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.

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 not valuable. 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 wih respect to its computational ef-

[2]CLUTO,
http://www-users.cs.umn.edu/~karypis/cluto/

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	0
Purity	0.295	0.992	1
Internal Similarity	0.094	0.900	0.912
Clustering Time	754s.	11s.	837s.

ficiency, 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).

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 publishing details (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.

Even though the "text+tags" performs slightly better, the efficiency/quality trade-off obviously plaids for prefering 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 (those that do not belong to preselected clusters).

With the INEX collection and k=15, 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 adressed the problem of clustering homogenous structured document collections. We experimented a common partitional 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

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%)

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-centric structured document collection. The INEX initiative has already provided such a collection, but meaningful classes were not yet available at the time of our experiments. The manual assessments of the INEX topics will permit us 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 specificities of the INEX collection.

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.

Tarragon Consulting at INEX 2002:

EXPERIMENTS USING THE K2 SEARCH ENGINE FROM VERITY, INC.

Richard M. Tong
rtong@tgncorp.com

TARRAGON CONSULTING CORPORATION
1563 Solano Avenue #350, Berkeley CA 94707, USA

1. Introduction

Tarragon Consulting Corporation (Tarragon) participated in INEX 2002 with the two main goals. First, we wanted to develop a performance baseline using the "out of the box" K2 search engine from Verity, Inc., and second, we wanted to test a range of techniques for search and retrieval of XML documents that we have been investigating as add-ons to K2. Unfortunately, time and resource constraints prevented us from experimenting with the planned extensions, but we did get valuable insight into the behavior of the K2 engine and the issues associated with performing a formal evaluation of XML document retrieval systems.

The remainder of this paper includes a brief introduction to the K2 search engine, a description of the techniques we used for constructing queries for each INEX topic type, a review of our official results and a more detailed analysis of performance on the Content and Structure topics. We conclude with general comments on the overall INEX experience.

2. The K2 Search Engine

K2 is an enterprise-class document retrieval platform from Verity, Inc. (http://www.verity.com/) that has a distributed, brokered architecture and that can access data from a wide range of sources with documents in multiple formats and languages. For the INEX experiments we made use of two key features of K2 — the ability to index XML-tagged documents, and the ability to create query expressions that define constraints on the content of tagged document elements.

2.1 Document and Zone Indexing

K2 has a built-in "zone indexing" mechanism that, in addition to creating a complete inverted keyword index, creates a set of auxiliary indexes that store positional information on the location and extent of each defined zone tag pair.

This is a very general mechanism that can be used with any form of document markup, and for the INEX experiments we chose to index the complete set of XML tags defined by the IEEE DTD. The K2 zone indexing mechanism also supports indexing of tag attributes, but for the INEX experiments the only tag for which we extracted attribute information was `<AU/>`, where we indexed the `SEQUENCE` attribute values.

All this additional zone information adds to the size of the basic index, of course, and increases the total time needed to index the INEX documents, but given the relatively small size of the INEX collection this was not a significant factor in our experiments. The total index size for the complete INEX collection was approximately 170Mb and the time needed to index the complete INEX collection was under an hour on a single-processor 1.5GHz Pentium-4 machine with 512Mb of RAM.

2.2 Verity Query Language

The Verity Query Language (VQL) is a rich and expressive language that supports a wide range of query constructs, including standard keyword and Boolean style operators, as well as sets of operators that specify the ways in which evidential strengths are to be combined. For the INEX experiments we were primarily concerned with the VQL constructs that support restrictions on zones so that we could capture the `<ce/><cw/>` constraints in the Content and Structure topics. The standard form of this in VQL is:

```
VQL-expression <IN> tag-name
```

where `<IN>` is a VQL operator that directs the K2 engine to search for the VQL expression in the named tag. So for example:

```
"QBIC" <IN> bbl
```

is a search request to look for the keyword `QBIC` in the zone (document element) defined by the pair of `<bbl/>` tags.

The 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. INEX Topics and Queries

In developing queries for each of the Content Only (CO) and Content and Structure (CAS) topics, we attempted to emphasize precision at the expense of recall. That is, we made no attempt to perform any kind of term expansion, using only those terms and phases found in the original topic specification.

The main issue for us however was that the standard Verity engine does not provide a mechanism for returning pointers to the specific document elements that match the search criteria. That being the case, we had to adopt a path reporting strategy that used either the first, or the smallest, unique element that contains the matched element(s) in those cases where the topic itself did not specify a unique target element. In general, of course, this has the effect of depressing both the recall and precision scores since we thereby artificially limit the number of elements returned and potentially report a path that has "larger" coverage than the actual matched element.

3.1 Content Only Queries

We used a semi-automatic technique for constructing queries from the CO topics. The first step was to run a 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" <IN> bdy
* 0.50 "bioinformatics" <IN> bdy
* 0.50 "genome" <IN> bdy
* 0.50 "genomics" <IN> bdy
* 0.50 "proteomics" <IN> bdy
* 0.50 "sequencing" <IN> bdy
* 0.50 "protein folding" <IN> bdy
```

where `<Accrue>` is the VQL operator that implements a basic evidence summation function, and the weights `0.50` define the relative 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. We also limited the search for terms and phases to just the `<bdy/>` elements as shown.

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]` even though our search was actually restricted to the `<bdy/>` elements.

3.2 Content and Structure Queries

We used a similar semi-automatic strategy for constructing queries from the 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 manually pre-selecting a path for each topic.

Of the 30 CAS topics, 7 have `<te/>` elements that are unique, so in these cases we used the `<te/>` element specified in the topic. In 8 additional 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 15 topics, we selected the smallest document element guaranteed to contain the element that matched the query. In many cases, of course, this was just the path `/article[1]/bdy[1]`.

We designated these three groups of CAS topics as "Actual Unique," "Assumed Unique," and "Default Unique," with corresponding topic IDs:

Actual Unique:	08, 09. 13, 18, 23, 24, 25
Assumed Unique:	01, 02, 05, 06, 16, 22, 26, 30
Default Unique:	03, 04, 07, 10, 11, 12, 14, 15, 17, 19, 20, 21, 27, 28, 29

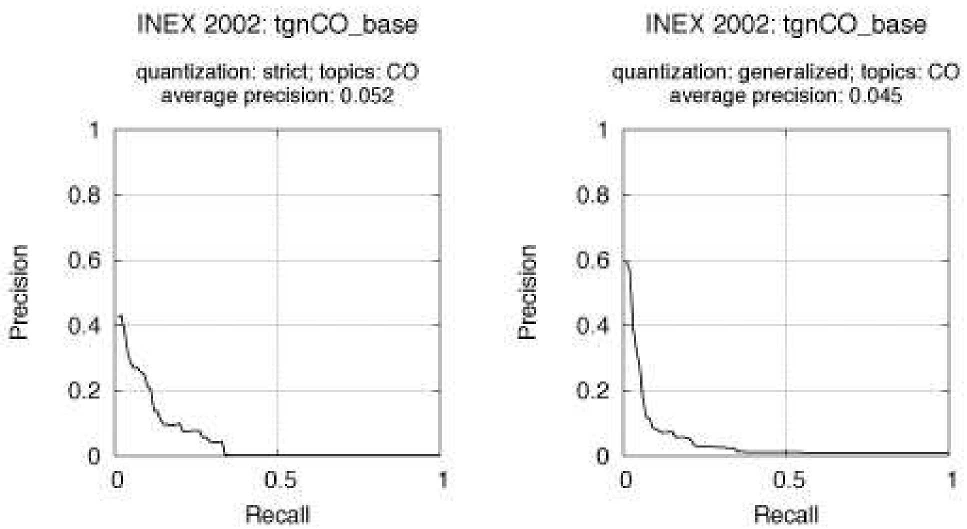
Note that in the Default Unique set, Topics 10 and 28 required the ability to extract information from a volume document and then use this information to identify specific articles. In the first case, we would have needed to identify that an article was a book review, in the second case that an article was published in a special features section. The basic K2 engine cannot do this, so in both these cases we created queries that located the appropriate volume and reported the path as `/books[1]`.

4. Results and Analysis

Since a key objective of our participation in INEX 2002 was to assess the ability of the K2 engine to capture the query structure, our focus in this section is on performance with respect to the CAS topics.

4.1 Content Only Topics

The precision-recall graphs for our official INEX CO queries (the run labeled tgnCO_base) are shown below:



The strict quantization results were ranked 10 of 49, and the generalized quantization results were ranked 17 of 49 (using the on-line evaluation tool on 2003-02-14).

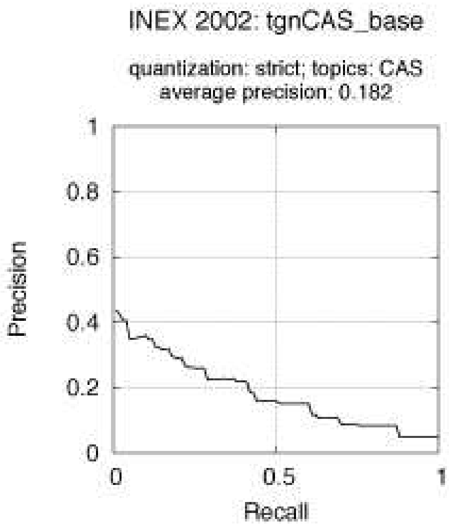
Inspection of individual topic runs shows that many of the topics had reasonable precision performance, but they universally failed on recall. This is in part because we only reported one path per relevant document, The table below reports the average precision scores (denoted AvP(S) and AvP(G) for the strict and generalized quantization respectively) for each topic as generated by the on-line evaluation tool. The blank entries correspond to those topics for which no assessments were available (as of 2003-02-14).

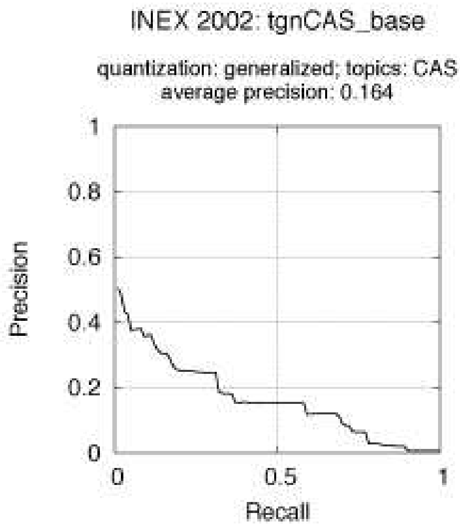
CO_ID	AvP(S)	AvP(G)	CO_ID	AvP(S)	AvP(G)	CO_ID	AvP(S)	AvP(G)
31	0.000	0.076	41	0.002	0.038	51	0.066	0.047
32	0.039	0.023	42	0.024	0.043	52	0.157	0.047
33	0.000	0.128	43	0.169	0.023	53	0.038	0.011
34	0.032	0.043	44	-	-	54	-	-
35	-	-	45	0.026	0.028	55	-	-
36	0.002	0.027	46	0.056	0.074	56	-	-
37	0.003	0.032	47	0.035	0.018	57	-	-
38	0.003	0.030	48	0.060	0.045	58	0.041	0.034

CO_ID	AvP(S)	AvP(G)	CO_ID	AvP(S)	AvP(G)	CO_ID	AvP(S)	AvP(G)
39	0.046	0.049	49	0.219	0.035	59	-	-
40	0.124	0.141	50	-	-	60	0.007	0.035

4.2 Content and Structure Topics

The precision-recall graphs for our official INEX CAS queries (the run labeled tgnCAS_base) are shown below:

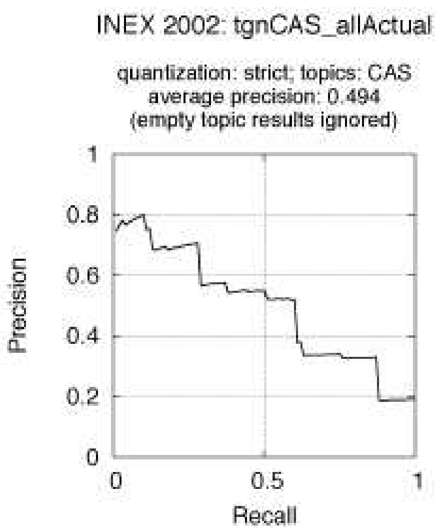


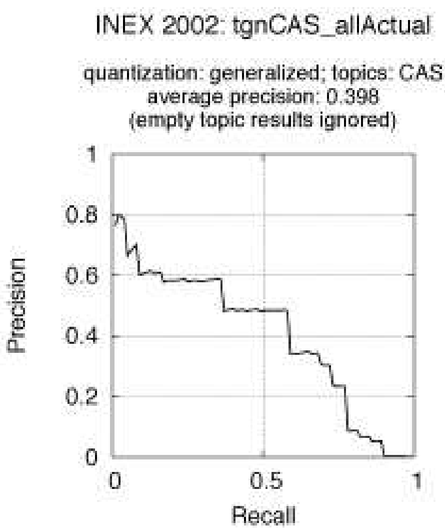


The strict quantization results were ranked 12 of 42, and the generalized quantization results were ranked 10 of 42 (using the on-line evaluation tool on 2002-02-14).

To see the effect of our inability to report all the paths, we used the INEX online evaluation tool to generate precision-recall graphs for each of the three sub-groups of CAS topics defined in Section 3.2. Those for which there was a unique <te/> element (the run denoted tgnCAS_allActual), those for which we used the first instance of the <te/> element (tgnCAS_allAssumed), and those for which we assigned the smallest unique element as the reported path (tgnCAS_allDefault).

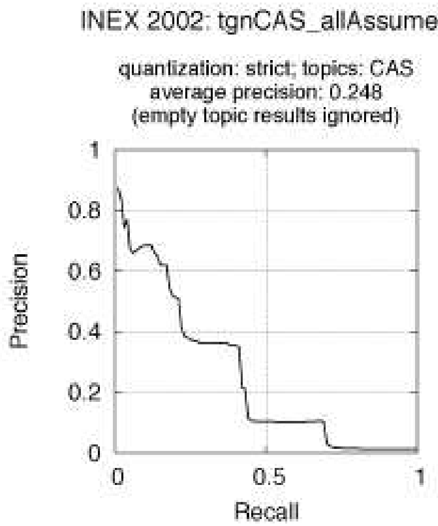
The "Actual Unique" results are:

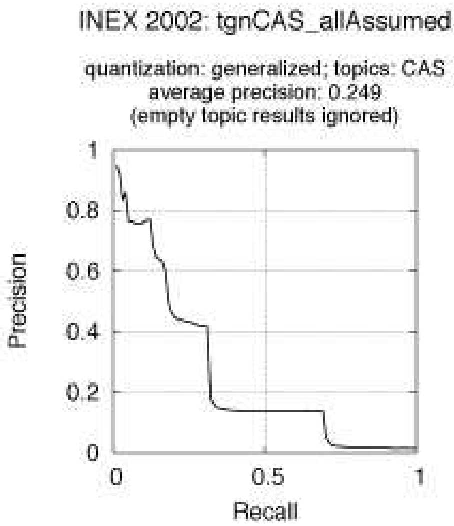




These clearly show that we can do a good job on both precision and recall in the case where there is a single unique element. The only "failure" we had was on Topic 24 where our policy of using a strict interpretation of the <ce/><cw/> constraints appears to have severely limited our recall numbers. We note, though, that as of 2003-01-29 the assessments for Topic 24 are marked as inconsistent.

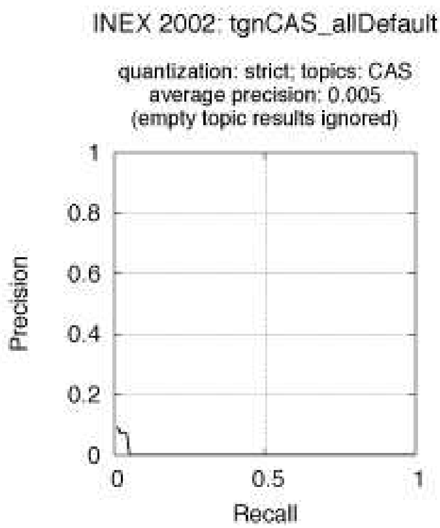
The results for those CAS topics for which we used the first instance of the target element, the "Assumed Unique" set, are:

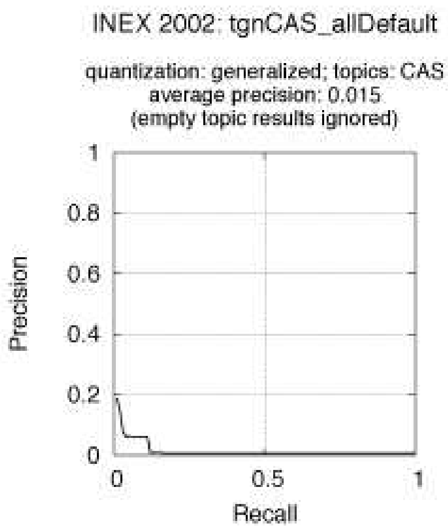




Here again precision performance is good, but the cost of limiting our results to just one element is clearly apparent in the lower recall values

Finally, the results for the "Default Unique" set are:





Obviously these all failed to produce significant results. As already noted, two of these (Topics 10 and 28) could not be expected to give any results, and of the remaining 13 topics , eight (Topics 07, 12, 14, 15, 17, 19, 20, and 27) failed because the scoring scheme does not allow partial credit for larger or smaller elements when a <te/> is in fact specified by the topic statement. The remaining five topics had no <te/> element specified (Topics 03, 04, 11, 21, and 29) so that we did get some credit for reporting a "large" path element This explains the slight positive spike in the allDefault precision-recall curves close to the origin.

The complete list of average precision scores generated by the on-line evaluation tool (denoted AvP(S) and AvP(G) for the strict and generalized quantization respectively) for each of the CAS topics are shown below. The blank entry correspond to the topic (Topic 28) for which no assessments were available (as of 2003-02-14).

CAS_ID	AvP(S)	AvP(G)	CAS_ID	AvP(S)	AvP(G)	CAS_ID	AvP(S)	AvP(G)
01	0.035	0.035	11	0.005	0.016	21	0.000	0.008
02	0.225	0.224	12	0.001	0.003	22	0.413	0.315
03	0.006	0.018	13	1.000	0.497	23	0.185	0.242
04	0.042	0.021	14	0.000	0.002	24	0.000	0.023
05	0.389	0.311	15	0.000	0.008	25	0.523	0.629
06	0.000	0.000	16	0.397	0.583	26	0.068	0.137
07	0.002	0.005	17	0.000	0.082	27	0.000	0.000
08	0.870	0.770	18	0.280	0.041	28	-	-
09	0.601	0.581	19	0.005	0.010	29	0.005	0.028
10	0.002	0.009	20	0.000	0.001	30	0.211	0.139

5. Overall Comments

Generally we were satisfied with the performance of the "out of the box" K2 engine. Although K2 does not have an explicit representation of XML document structure, we successfully exploited its generalized ability to search within "zones," so that, in all but two CAS topics, we were able to completely capture the `<ce/><cw/>` constraints. In addition, for those topics that did have a unique `<te/>` element we generally got good performance in both precision and recall.

Clearly though, the biggest issue for K2 with respect to the INEX experiments is its inability to report the actual path that matched the query constraint. This forced us to adopt a path reporting strategy that turned out to be ineffective in half the CAS topics, and significantly impacted recall in eight others. The same issue also limited our ability to do more than a traditional "ad hoc" retrieval with the CO topics.

As part of the "lessons learned" during the effort, we feel strongly that the assessment and results scoring procedures need further investigation and revision before the next INEX experiment. For example, it is not clear to us that it really is possible to treat relevance and coverage as independent concepts, or even that it is reasonable to apply these ideas to those elements in the IEEE DTD that deal with the "look and feel" of the document, as opposed to the substantive content. And we also believe that the different nature of the information needs expressed by the CO and CAS topics argues for the use of different evaluation methodologies for the two sets of results. The CO topics, it seems to us, are primarily about locating those thematic elements within a document that makes it relevant. Whereas, the focus of the CAS topics is on the nature of constraints over document elements.

Overall we found the INEX 2002 experiment cycle to be an extremely worthwhile exercise. It certainly helped us achieve our goal of establishing a performance baseline for the standard K2 engine, and gave us considerable insight into the challenges associated with evaluating XML document retrieval systems. We would like to thank all those at the University of Dortmund and at Queen Mary University of London responsible for organizing and managing the INEX 2002 effort.

Using the Extended Vector Model for XML Retrieval

Carolyn J. Crouch
Department of Computer Science
University of Minnesota Duluth
Duluth, MN 55812
(218) 726-7607
ccrouch@d.umn.edu

Sameer Apte
Department of Computer Science
University of Minnesota Duluth
Duluth, MN 55812
(218) 726-7607
apte0002@d.umn.edu

Harsh Bapat
Department of Computer Science
University of Minnesota Duluth
Duluth, MN 55812
(218) 726-7607
bapa0005@d.umn.edu

ABSTRACT
The authors describe an approach to XML retrieval based on Fox's extended vector space model [2]. The current implementation of their system and results to date are reported. (All results are based on retrieval at the article level since flexible retrieval is still being implemented.) The basic functions are performed using the Smart experimental retrieval system.

1. INTRODUCTION
With INEX, we have for the first time a large testbed—documents and topics, evaluation procedures—supporting experimentation in structured document retrieval. With the enormous influence of the web, it is not surprising that attention has been focused on XML and appropriate methods of retrieval in this environment.

Much investigation in information retrieval over the last 40 or so years has centered on the vector space model [8], developed by Salton and used as the basis for the Smart experimental retrieval system [7]. In the vector space model, each document (and query) is viewed as a set of word types 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. Very commonly, *tf-idf* weights [9] or some variation thereof [10] are used. The similarity between vectors (e.g., document and query) is represented by the mathematical similarity of their corresponding term vectors.

In 1983, Fox [2] proposed an extension of the traditional vector space model which he called the extended vector space model. This model allowed for the incorporation of objective identifiers along with the usual content identifiers in the storage and retrieval of documents. 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. In the extended vector model, 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.

Using this model for document retrieval normally presents at least two significant problems: (1) the construction of the extended search request and (2) the selection of the coefficients for combining subvector similarities. The generation of extended queries, in particular, has attracted some attention [3,1]. For XML retrieval, of course, this particular problem is no longer an issue because the query is already structured; i.e., it is given in a form that is easily translated into an extended vector. The second problem—the weighting of the subvectors themselves—remains open to investigation.

Smart is a powerful tool for experimentation. The extended vector capability which is a part of the Smart system would appear well suited for XML retrieval from the *retrieval* viewpoint. (It does not, of course, in its present state lend itself to flexible retrieval at various levels of granularity.) Since our interests lie in information retrieval, we chose this approach—using the extended vector facility of Smart to represent the structured documents and queries—for our initial investigations in XML retrieval. We seek to determine the feasibility of incorporating the functionality (e.g., flexibility and granularity) required for XML retrieval within the extended vector environment.

In traditional information retrieval, the system returns a set of documents, usually in rank order. 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, section, paragraph, etc.). That is, the granularity of the response varies depending on the relevance of the element. No target element is specified. For the CAS queries, the retrieval system should return a ranked list of elements as specified in the target element (<te>) field. Search terms themselves are specified in the <cw> element, and the context of the search terms is specified in the context element (<ce>) field. (In a relevant document, the search terms in the <cw> field should occur in the element specified in the <ce> field.) Otherwise (if no <ce> is specified), the search terms can occur anywhere in the document. For CAS queries, structure is used to limit the range of the search to a corresponding specified field in the document.

2. OUR APPROACH

In our approach, using Smart's extended vector capability, 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 or c-types (i.e., *article, ti, atl, pub_yr, sec, st, fgc, article_au_fnm, article_au_snm, abs, kwd, ack, tig, bibl_au_fnm, bibl_au_snm, bibl_ti, bibl_atl, p*) as defined in INEX guidelines.

2.1 Initial Runs

Our system lacks the capability for granular retrieval. With this in mind, we performed the following steps.

(1) The documents are parsed using a simple XML parser available on the web. This resulted in a parsing of the collection such that each of our 18 c-types is now identifiable in terms of its XML path.

(2) The documents and queries are translated into Smart format and indexed by Smart as extended vectors. The indexing was performed on both an *article* (i.e., document) and *paragraph* basis. (For the results reported here, we used only the *article*-based indexing.)

(3) Retrieval takes place by running the queries against the indexed collection. The result is a list of *articles* ordered by decreasing similarity to the query. (A variety of weighting schemes are available through Smart. *Lnu.ltu* [10] weighting is used here.)

(4) For each query, results are sorted by correlation and the top 100 elements are converted to INEX format and reported.

The retrieval itself is fairly straight-forward; the only variation from the normal vector processing at this point is the splitting of certain CAS queries into separate portions which are then run in parallel to ensure that the elements retrieved meet the specified criteria.

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. To guarantee that the system returns *only* those articles, we split the query into two parallel queries as follows:

Q1: <cw>ibm></cw><ce>fm/aff</ce>

Q2: <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 articles whose body or section(s) contain the term *certificate*. Smart returns a ranked list of documents for both queries. The intersection of these lists is the final, ranked list of documents returned for topic 8.

2.2 Results

Our system is still in a very rudimentary stage of development. The results reported here are all based on an *Lnu.ltu* [10] weighting of the collection indexed at the article level. We are not yet able to return the most relevant elements; we can report only what our system presents as the most highly correlated articles.

We participated in the early work of the INEX group (the initial submission of queries and relevance assessments). Those relevance assessments were based on results obtained by using Smart in extended vector form and the subsequent manual mapping of results to the format required by INEX. (In fact, the automatic mapping from INEX to Smart format and vice versa has consumed much of our time and effort to date.) The results reported here represent what is at this point a straight-forward search by Smart using the extended vector facility with results converted automatically to INEX reporting format. No attention has as yet been given to weighting within or among subvectors or to analyzing the queries with the aim of improving performance.

Mixing objective and content-based subvectors in a single query is interesting. The splitting of such queries into separate portions to be run in parallel, as described in the previous section, works well—in, for example, CAS topic 8, with an average precision of 0.801 under generalized quantization. It did not work (in the results shown here) for queries such as CAS topic 9, which seeks articles on nonmonotonic reasoning from 1999 or 2000. The reason is clear—there are hundreds of articles in the collection from these two years and the programming team (thinking in terms of content-based retrieval) initially decided to impose a limit (of 700) on the number of items returned by a subquery. Thus there may be many relevant articles retrieved by the content subvector which cannot be identified as meeting the second condition because they are not in the limited set of items retrieved by the objective subvector. (This error negatively impacts our results with respect to a number of queries, but we are unable to rerun these cases within the timeframe for reporting.) In all cases, we used only the title and keyword fields of the queries.

The recall-precision graphs for retrieval of the CAS and CO topics based on our current implementation are given below in Figures 1 and 2.

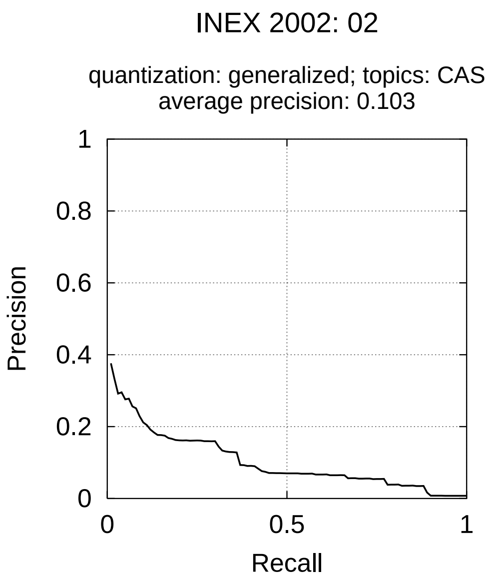


Figure 1. Recall-precision for CAS topics

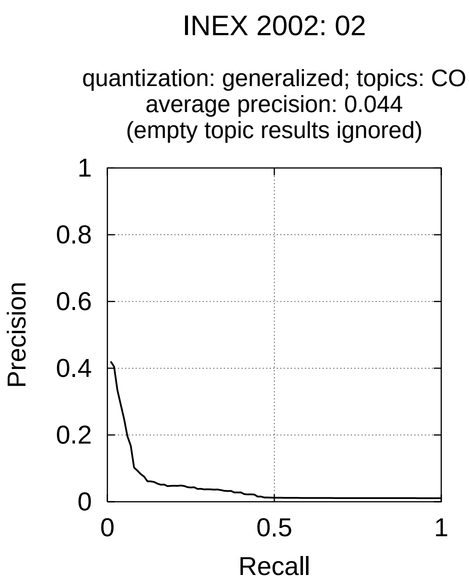


Figure 2. Recall-precision for CO topics

2.3 System Modification

Of course, the data in these figures are based on a flat or static indexing of the collection (at the article level). We now turn our attention to two important aspects of XML retrieval, namely, granularity and flexibility.

In XML retrieval, the system is supposed to return the most relevant element (as opposed to document). That element might be a paragraph, section, article, etc. The element may be specified (as it is for some CAS queries) or not specified (as is the case for CO queries). In either case, the requirement is to return those elements (at the proper level of granularity) that are most relevant to the query. To accomplish this goal in a realistic timeframe and manner, we need flexible retrieval—i.e., "the retrieval over arbitrary combinations and nestings of element types" as described by Grabs and Schek [5]. This means that we must decide, dynamically at execution time, which element(s) are most relevant to a particular query.

Toward this end, we will utilize only one indexing of the collection, namely, the indexing at the paragraph level. We now consider those basic indexing units (c-types) which serve as the root of a subtree in the document structure hierarchy--i.e., *article*, *section*, *acknowledgement*, and *abstract*. Articles contain sections; all contain paragraphs. These are the primary nodes of interest for flexible retrieval. (*Title-group* also serves as a root node but is of lesser interest in this context.)

Consider a query (in extended vector form) containing search terms in a non-objective subvector. Using an indexing of the collection at the paragraph level, a search of the corresponding document subvectors retrieves a ranked list of the most relevant (i.e., highly correlated) paragraphs. For each concept (or stemmed word type) in the collection, an inverted file specifies each paragraph in which that term is contained along with its weight in that paragraph (i.e., its term frequency).

To implement a version of flexible retrieval, we need additional information. A paragraph retrieved by the resolution of a content query subvector may in fact be the most relevant element associated with it—or not. The desired element may really be the section containing that paragraph (or the article containing that section). To decide which element to report, we need the appropriate statistics (term and element frequency) for each local environment (paragraph, section, and article). We have the data for each paragraph. In determining which element to report, we will calculate relevance at execution time for the subtrees rooted by the corresponding section and article.

How do we calculate the relevance of the parent element (e.g., section) of the current node (in this case, paragraph)? The nature of the vector space model suggests two approaches. We might choose, for example, to construct a vector for the section, using information available from its child nodes (the paragraphs contained in that section). Our retrieval system then correlates the section vector with the query, i.e., retrieves the section. This action is repeated first for all sections in the document and subsequently, in the same manner, for the article itself. The rank of the element in the final set of correlations (including paragraphs, sections, and article) determines whether one element is more relevant than another. The top n elements are reported.

Another approach to the problem (i.e., determining the relevance of the element rooted at the parent) might utilize the correlations of the child nodes. For example, at this point in the retrieval process, we already know the similarity of each paragraph with the query. We could then propose calculating the similarity of the parent (i.e., section) as a function of the similarities of its children (and likewise, using the section similarities, compute the similarity for the article).

In either case, it seems clear that we need particular information. From our viewpoint, the initial query retrieves a set of relevant paragraphs (i.e., those having a positive correlation with the query). For each of these paragraphs and for each query term, we need both term frequency (*tf*) and element frequency (*ef*) information. Suppose we want to use *tf-idf* weighting, the advantages of which are well known. To calculate the relevance of the section, we must know the frequency of each query term in each paragraph (*tf*) and the number of paragraphs in which that term occurs (*ef*). Almost all data is available in the inverted file.

One important aspect of this process relates to the position of a node in a subtree. As Grabs and Schek [5,6] indicate, information contained in a more distant node (the last paragraph in a section, for example) is often less important than that in a nearer node (e.g., the first paragraph). To deal with this issue, they adopt the augmentation weights of Fuhr and Broβjohann [4] wherein terms are downweighted when propagated upwards. [5,6] utilizes a vector space approach here which we find attractive. They claim that their model allows retrieval across the document hierarchy (i.e., using arbitrary combinations of element types) while at the same time dynamically performing flexible retrieval at desired levels of granularity. So does ours.

2.4 Current State

Our system is still in a very early stage of development. Weighting of terms within subvectors and the weighting of subvectors themselves are issues of concern which we have not yet had time to examine carefully. The next focus of development in our system is flexible retrieval (in particular, what [5] refers to as nested retrieval). We plan to implement a version using their method for calculating the weight of an interior element in nested retrieval. We will look at other approaches as well.

3. CONCLUSIONS

Given the small size of our team and the scheduling constraints, we are unable to report results attributable to flexible retrieval. However, the extended vector model would appear to provide a natural framework for structured retrieval. Except for the dynamic retrieval of elements, it provides the capabilities needed for the XML task. The dynamic aspects can be added. We do not expect that the additional costs at execution time will significantly impact retrieval. The major difficulties faced by our team to date pertain to the mapping from XML format to Smart and vice versa. These problems having now been solved, we anticipate more rapid progress.

4. REFERENCES

[1] Crouch, C., Crouch, D. and Nareddy, K. The automatic generation of extended queries. In Proc. of the 13th Annual International ACM SIGIR Conference, (Brussels, 1990), 369-383.

[2] 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).

[3] Fox, E., Nunn, G. and Lee, W. Coefficients for combining concept classes in a collection. In Proc. of the 11th Annual International ACM SIGIR Conference, (Grenoble, 1988), 291-307.

[4] Fuhr, N. and Broβjohann, K. XIRQL: a query language for information retrieval in XML documents. In Proc. of the 24th Annual International ACM SIGIR Conference, (New Orleans, 2001), 172-180.

[5] Grabs, T. and Schek, H. Generating vector spaces on-the-fly for flexible XML retrieval. In Proc of the ACM SIGIR Workshop on XML and Information Retrieval, (Tampere, Finland, 2002), 4-13.

[6] Grabs, T. and Schek, H. ETH Zurich at INEX: Flexible information retrieval from XML with PowerDB-XML. INEX 2002 Workshop Proceedings, (Dortland, 2002), 35-40.

[7] Salton, G. Automatic information organization and retrieval. Addison-Wesley, Reading PA (1968).

[8] Salton, G., Wong, A., and Yang, C. S. A vector space model for automatic indexing. Comm. ACM 18, 11 (1975), 613-620.

[9] Salton, G. and Buckley, C. Term weighting approaches in automatic text retrieval. In IP&M 24, 5 (1988), 513-523.

[10] Singhal, A. AT&T at TREC-6. In The Sixth Text REtrieval Conf (TREC-6), NIST SP 500-240 (1998), 215-225.

Compression and an IR Approach to XML Retrieval

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}`

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 Großjohamn [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.

```
<article>
    <atl> XML Retrieval </atl>
    <au sequence="first">
        <fnm> First N. </fnm>
        <ref> Surname </ref>
    </au>
    <sec> <st> Everything </st>
        <p>
            Everything about XML </it>
            and XML retrieval </it>.
        </p>
    </sec>
</article>
```

Figure 1: Example of XML document.

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 outlines the experiments we undertook.

2 Documents and queries

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*> 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 XML </*it*> and 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].

```
<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.

Note that normally in an XML tree, leaf nodes are textual, and internal nodes are skeletal, but this cannot be assumed. A counter-example 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 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* "non-monotonic 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 based on the *MG* system (see `http://www.cs.mu.oz.au/mg/`). The main feature of *MG* for text retrieval is that it applies compression to the documents as well as to the index. 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 (that is, number of nodes 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, that is, 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 (that is, 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

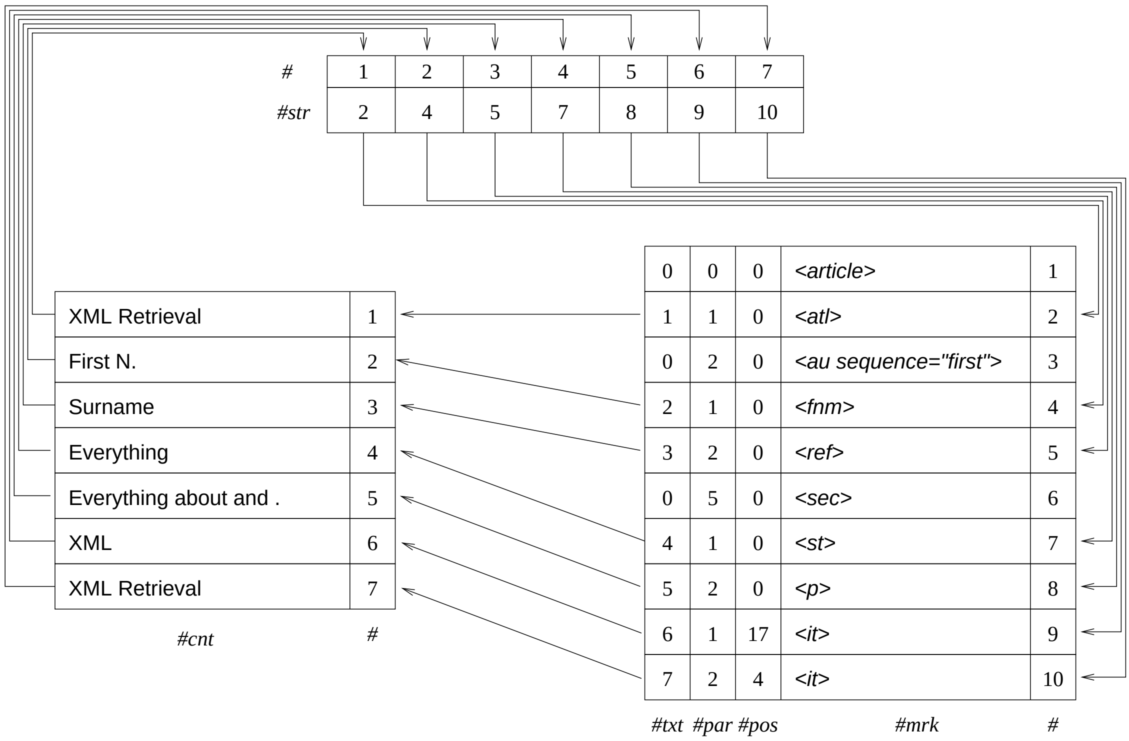


Figure 3: Example of textual, structural and their mapping files. The picture conceptually describes a database which has only one XML document, namely, the document presented in Figure 1. The arrows represent explicit or implicit links from a file to another. To each file, the field # is added to show 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, that is, 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 XML </*it*> and . </*p*>. Similarly, item number 10 should be inserted to this string at position 4 after </*it*>.

representation of the term itself, list of the query's <*ce*> 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 <*te*> 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

$$S_{d,q} = \sum\nolimits_{t \in q \cap d} p_{d,q,t} \cdot \omega_{d,t} \cdot \omega_{q,t}$$

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

$$p_{d,q,t} = c_w \cdot p^{w}_{d,q,t} + c_{\bar{w}} \cdot p^{\bar{w}}_{d,q,t} + c_e \cdot p^{e}_{d,q,t} + c_{\bar{e}} \cdot p^{\bar{e}}_{d,q,t} \,.$$

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:

- $p^{w}_{d,q,t}$ is set to 1 if t appears in either */query/cw* or */query/kw*, and d is any textual item,
- $p^{\bar{w}}_{d,q,t} = 1$ if t appears in */query/cw/nw*, and d is any textual item,
- $p^{e}_{d,q,t} = 1$ 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,
- $p^{\bar{e}}_{d,q,t} = 1$ 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.

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 $w^*_{q,t}$:

$$\begin{aligned}
w_{d,t} &= (1 + \log_e f_{d,t}) \\
W_d &= \sqrt{\sum_{t \in d} w^2_{d,t}} \\
W^*_d &= 1/((1-s) + s \cdot W_d / W^a) \\
w^*_{d,t} &= w_{d,t} / W^*_d \\
w^*_{q,t} &= \log_e \left(1 + \frac{f^m}{f_t}\right) \cdot (1 + \log_e f_{q,t})
\end{aligned}$$

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^m is the greatest value of f_t in the textual file; W_d is length of the textual item d; W^a 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:

$$\begin{aligned}
\omega_{d,t} &= \left\lfloor 2^b \cdot \frac{\log w^*_{d,t} - \log U}{\log U - \log L + \epsilon} \right\rfloor + 1 \\
\omega_{q,t} &= \left\lfloor 2^b \cdot \frac{\log w^*_{q,t} - \log U}{\log U - \log L + \epsilon} \right\rfloor + 1
\end{aligned}$$

in which $B = (U/L)^{L/(U-L)}$, and ϵ 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 $(s_1(b), s_2(b), \ldots, s_{|q|}(b))$, 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

$$s_i(a) = s_i(a) + \text{sign}\ s_i(b) \cdot \alpha \cdot \max_b |s_i(b)|,$$

where $|s_i(b)|$ is the absolute value of $s_i(b)$, α is a constant and is set to 0.8 in these experiments.

Label	Characteristics
um_mgx21_short	
	Queries: not having <*kw*> elements
	Type of transformation: global
	Propagation method: summation
um_mgx2_long	
	Queries: having <*kw*> elements
	Type of transformation: global
	Propagation method: maximum-by-category
um_mgx26_long	
	Queries: having <*kw*> elements
	Type of transformation: local
	Propagation method: maximum-by-category

Table 1: Settlement of the 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 β and γ are constants. Both of them are set to 0.5 in the experiments reported below.

6 Experiments

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.

Experiment parameters Three experiments were conducted. Their labels and settings are listed in Table 6.

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 extention 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. Großjohamn. 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.

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.robbert@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. IRStream is intended as a powerful framework to search for components of arbitrary granularity – ranging from single media objects to complete documents. IRstream combines traditional text retrieval techniques with content-based retrieval for other media types and fact retrieval on meta data. In contrast to other retrieval services which permit set-oriented or navigation-oriented access to the documents, we argue for a *stream-oriented* approach. 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 to 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 1 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 2. The concrete architecture of our IRstream implementation is presented in section 3. Section 4 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 5 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 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

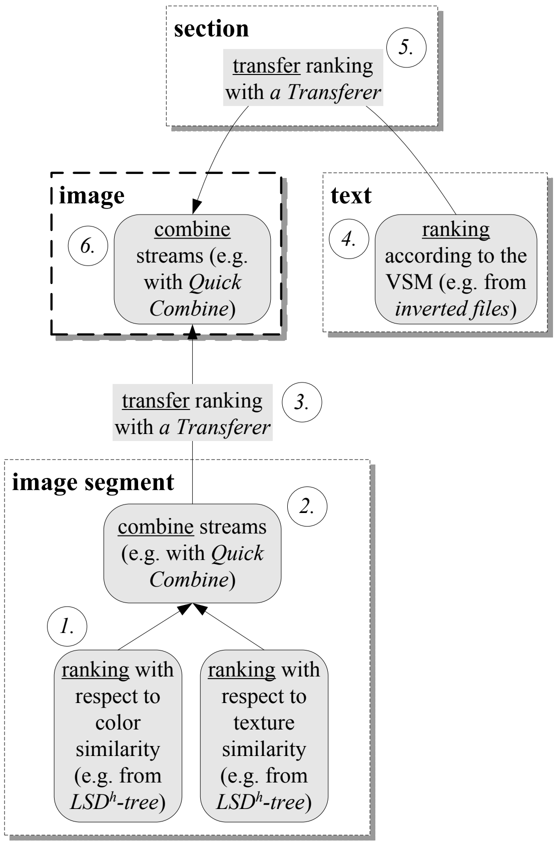


Figure 1: Stream-oriented processing of our example query

(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 an (ordered) 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 2.3.

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).

2. STREAM-ORIENTED QUERY PROCESSING

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

2.1 Rankers

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

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

2.2 Combiners

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

Since each element of each input stream is associated with some type of retrieval status value (RSV), a weighted average over the retrieval status values in the input streams can be used to derive the overall ranking [4]. Other approaches are based on the ranks of the objects with respect to the single criteria [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.

2.3 Transferers

With structured documents, ranking criteria are sometimes not defined for the required objects themselves but for their components or other related objects. An example arises when searching for images where the text in the "vicinity" (for example in the same section) should be similar to a given sample text. In such situations the ranking defined for the related objects has to be transferred to the desired result objects. 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 $\{ro_{i,1}, ro_{i,2}, \ldots, ro_{i,n_i}\}$ 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:

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

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 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: createAL() creates an empty auxiliary list. getObj(AL, i) yields the object with the i^{th} highest RSV_d value stored in AL. getRSV(AL, i) returns the RSV_d value for the object with the i^{th} highest RSV_d value stored in AL. contains(AL, do_j) checks whether there is an entry for object do_j in AL. insert$(AL, \langle do_l; RSV_d(do_l)\rangle)$ inserts the entry for do_l 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. 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 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 surly 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 insert $(AL, \langle o_d; RSV_d(o_d)\rangle)$ in the getNext method can be replaced by the more efficient operation insert $(AL, \langle o_d; RSV_r(o_r)\rangle)$.

2.4 Filters

Of course, it must be possible to define filter conditions for all types of objects. With our stream-oriented approach this means that filter components are needed. These filter components are initialized

```
Class Transferer {
  Stream : inputStream;
  RelationshipDef : rel_d; /* desired relationship */
  AuxiliaryList : AL;
  InputObject : o_r; /* next obj. 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_d → o_r)};
        /* all objects with the
           desired relationship to o_r */
      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

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.

3. THE IRSTREAM ARCHITECTURE

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

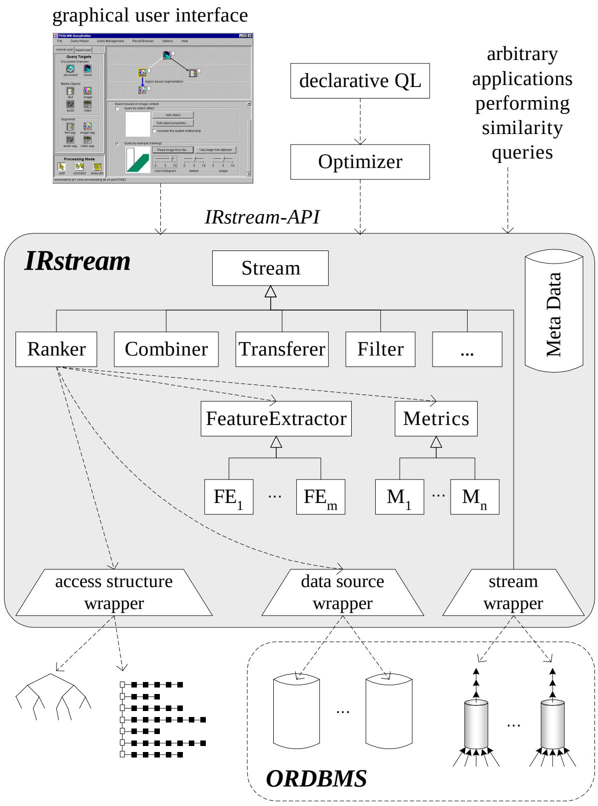


Figure 3: Architecture of the IRstream system

Feature Extractors and Similarity Measures

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

Ranker, Combiner, Transferer, Filter, ...

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

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

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

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

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

Meta Data

This component 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^h-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^{MM} query language [7, 11].

4. IRSTREAM IN THE CONTEXT OF INEX

To assess the applicability of our IRstream approach as a retrieval engine for XML documents, we performed one INEX retrieval run containing the top 100 results for all 60 topics. The INEX test collection consists of more than ten thousand documents and was inserted into the ORDBMS underlying our system. 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 document in order to support a search concerning the documents 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.

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

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):

```
<Title>
 <cw>information data visualization</cw>
 <ce>kwd</ce>
 <cw>large information hierarchies spaces
     multidimensional data databases</cw>
</Title>
<Description>
  I am looking for techniques for visualizing large
  information hierarchies or information spaces.
</Description>
<Narrative>
  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.
</Narrative>
```

To process topic 3 we used three rankers, three transferers 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 <keywd>) contains terms like *"information data visualization"*. In parallel, we employed two rankers to acquire a ranking for the document parts of type body (tag <bdy>) 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 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

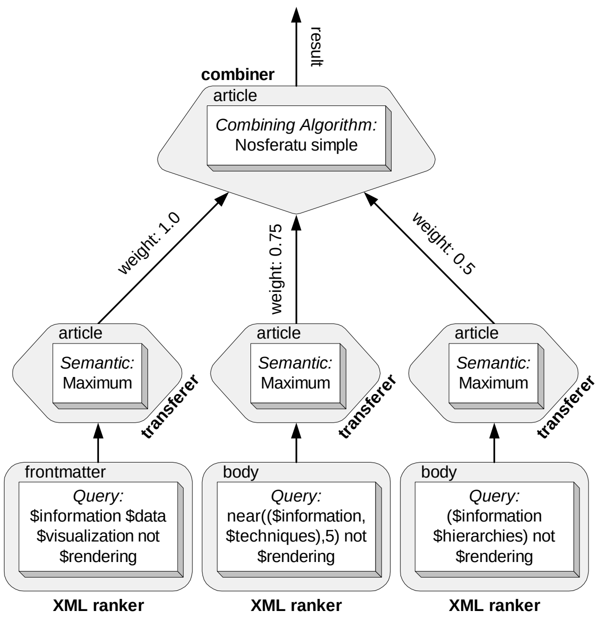


Figure 4: Stream-oriented processing of topic 3

weights are noted at the arrows leading from the tranferers 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.

5. CONCLUSION

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. The prototype directly applying the text retrieval facilities of the ORDBMS without a thesaurus or other enhancements obtained rank 16 among the 42 INEX participants with respect to the CAS topics.

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.

6. 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. Gövert. *Initiative for the Evaluation of XML retrieval (INEX)*. Online available: url: http://qmir.dcs.qmul.ac.uk/inex/, 2002.

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

[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^h-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. ERCIM Workshop Proceedings.

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

[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.

A database approach to content-based XML retrieval

Djoerd Hiemstra
University of Twente, Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE Enschede, The Netherlands
d.hiemstra@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 based on a language modelling approach to information retrieval. Evaluation using the INEX benchmark shows that it is beneficial if the system is biased to retrieve large XML fragments over small fragments.

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 support a small part of the functionality that we envision for future systems. In the upcoming years we will build a number of such prototype systems in the CIRQUID (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. Finally, Section 4 concludes this paper.

2 A multi-model 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 [26].

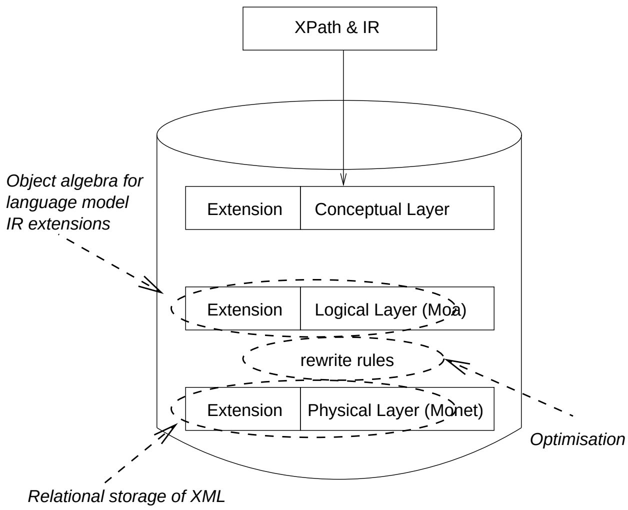


Figure 1: Database internals

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 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, we see that there is much overlap in the kind of functionality they provide.

```
1.  IT magazines
2.  +IT magazine* -MSDOS
3.  "IT magazines"
4.  IT NEAR magazines
5.  (IT | computer) (books | magazines | journals)
6.  XML[0.9] IR[0.1] title:INEX site:utwente.nl
```

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,[1] a wild card operator '`*`' stating that the document might match "magazine", but also "magazines" or "magazined" and 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 '`|`' operator (logical `OR`), 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.

These examples 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.

2.2 Moa and Language Models

Parts of a prototype multi-model database system have already been developed with the extensible object algebra Moa [14] as the logical layer. An open question in this set-up 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 [9, 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 $Q = q_1, \cdots, q_n$. Using Bayes' rule we can rewrite that as follows.

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

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 $P(q_1, q_2, \cdots, q_n|X)$. 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 [21], for which they are called "language models".

[1]Note that most retrieval systems do not distinguish upper case from lower case, and confuse the acronym "IT" with the very common word "it".

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:

$$P(q_1, \cdots, q_n|X) \;=\; \prod_{i=1}^{n}\Big((1{-}\lambda)P(q_i) + \lambda P(q_i|X)\Big) \quad (2)$$

Equation 2 defines our basic language model if we assume that each term is generated independently from previous terms given the relevant document. Here, λ 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 [10].

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 [22]? 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 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).

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

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, 13, 27].

In CIRQUID we will explore language modelling approaches that model all structured queries in Figure 2. The interested reader is referred to [18, 25] for so-called bigram models for proximity queries, and [12] for mandatory terms. A similar approach to querying XML data is proposed by List and De Vries [15], and Ogilvie and Callan [19].

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. [4, 24], and relational representations of information retrieval indexing structures as described by e.g. [3, 7, 26]. 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

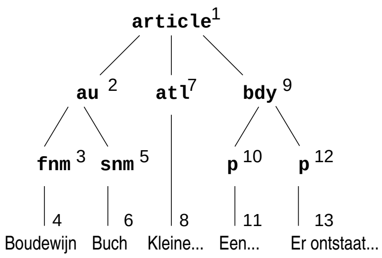


Figure 4: Tree representation of the data

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 [14] which provides efficient support for XML querying [24].

tags

id	parent	tag_name
1	0	article
2	1	au
3	2	fnm
5	2	snm
7	1	atl
9	1	bdy
10	9	p
12	9	p

pcdata

id	parent	string
4	3	Boudewijn
6	5	Büch
8	7	Kleine blonde...
11	10	Een schrijver...
13	12	Er ontstaat...

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.

In [3, 7, 26], `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

local_stats

word	id	P(word|id)
aardvark	43	0.007
after	3	0.09
after	42	0.11
after	78	0.015
after	980	0.067
affect	321	0.2
ambient	761	0.0001
:		
bekende	1	0.031
blonde	1	0.031
boudewijn	1	0.031
:		

global_stats

word	P(word)
aardvark	0.00001
after	0.0345
affect	0.00055
ambient	0.0000001
an	0.107
:	
:	
:	
:	
:	
:	
:	

Figure 6: Example relational storage of an IR index

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" [5, 28], 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 `pcdata` 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 [6]. In this case, 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 of 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.

```
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

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. 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 $\log(1 + P(q|X)/P(q))$ [10].

A first prototype

For our first prototype we implemented the XML storage scheme proposed by Grust [8]. 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.[2] The pre and post order assignment of XML element ids provides elegant support for processing XPath queries.

```
<article>[1]
 <au>[2]<fnm>[3]Boudewijn[4]</fnm>[5]<snm>[6]Büch[7]</snm>[8]</au>[9]
 <atl>[10]Kleine[11] blonde[12] dood[13]</atl>[14]
 <bdy>[15]
  <p>[16]Een[17]schrijver ontmoet een oude bekende.</p>
  <p>Er ontstaat een liefdesrelatie.</p>
 </bdy>
</article>
```

Figure 8: Example XML document: assigning ids

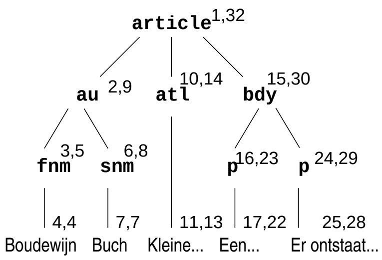


Figure 9: Tree representation: assigning ids

Note that pre and post order 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 data. These positions will be used in our term position index. This leads to the relational storage of XML data as shown in Figure 10 and the relational storage of the information retrieval positional index as shown in Figure 11.

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.

`tags2`

pre	post	tag_name
1	32	article
2	9	au
3	5	fnm
6	8	snm
10	14	atl
15	30	bdy
16	23	p
24	29	p

`pcdata2`

pre	post	string
4	4	Boudewijn
7	7	Büch
11	13	Kleine blonde...
17	22	Een schrijver...
25	28	Er ontstaat...

Figure 10: Relational storage of XML data

`position_index`

word	position
bekende	22
blonde	12
boudewijn	4
büch	7
een	17
een	20
een	27
er	25
kleine	11
:	

`global_stats`

word	P(word)
bekende	0.00321
blonde	0.00013
boudewijn	0.00004
büch	0.00001
een	0.0991
er	0.0145
:	

Figure 11: Relational storage of the IR positional index

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

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

Also note that, unlike the approaches in [6, 28], 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] Actually, [8] store the id of the parent as well. Similarly, in [24] a field is added to keep track of the order of XML elements; here we emphasise different view points.

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 [11] 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 [23] and fuzzy set models [20].

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 Experimental results

In this section we describe the experimental setup and the evaluation results of the system using the INEX testbed. We describe the tasks and evaluation procedure, the system setup and research questions, and finally the experimental results.

3.1 The INEX evaluation

INEX is the Initiative for the Evaluation of XML Retrieval. The initiative provides a large testbed, consisting of XML documents, retrieval tasks, and relevance judgements on the data. INEX identifies two tasks: the content-only task, and the content-and-structure task.

The content-only task provides queries of the form: `//*[. =~ "computational biology"]` ("XPath+IR" for: any element about "computational biology"). In this task, the system needs to identify the most appropriate XML element for retrieval. The task resembles users that want to search XML data without knowing the schema or DTD.

The content-and-structure task provides queries of the form: `//article[author =~ "Smith|Jones" and bdy =~ "software engineering"]` ("XPath+IR" for: retrieve articles written by either Smith or Jones about software engineering). This task resembles users or applications that *do* know the schema or DTD, and want to search some particular XML elements while formulating restrictions on some other elements.

For each query in both tasks, quality assessments are available. XML elements are assessed based on *relevance* and *coverage*. Relevance is judged on a four-point scale from 0 (irrelevant) to 3 (highly relevant). Coverage is judged by the following four categories: N (no coverage), E (exact coverage), L (the XML element is too large), and S (the XML element is too small).

In order to apply traditional evaluation metrics like precision and recall, the values for relevance and coverage must be quantised to a single quality value. INEX suggests the use of two quantisation functions: Strict and liberal quantisation. The strict quantisation function evaluates whether a given retrieval method is capable of retrieving highly relevant XML elements: it assigns 1 to elements that have a relevance value 3, and exact coverage. The liberal quantisation function assigns 1 to elements that have a relevance value of 2 and exact coverage, or, a relevance value of 3 and either exact, too small, or too big coverage.

3.2 Setup and research questions

We evaluate a system that only has limited functionality. First of all, we assume that $\lambda = 1$ in Equation 2, so we do not have to store the `global_stats` table of Figure 11. The system supports queries with a content restriction on only one XML element, so the example content-and-structure query in the previous section is not supported: Either the restriction on the `author` tag, or the restriction on the `bdy` tag has to be dropped. The system supports conjunction and disjunction operators, which are evaluated as defined in the example of Query 5 at the end of Section 2.2. All queries were manually formulated from the topic statements.

The experiments are designed to answer the following research question: Can we use the prior probability $P(X)$ (see Equation 1) to improve the retrieval quality of the system? We present three experiments using the system described in this paper, for which only the prior probabilities $P(X)$ differ. The baseline experiment uses a uniform prior $P(X) = c$, where c is some constant value, so each XML element will have the same a priori probability of being retrieved. A second experiment uses a length prior $P(X) =$ *number of tokens in the XML element*, where a token is either a word or a tag. This means that the system will prefer bigger elements, i.e. elements higher up the XML tree, over smaller elements. A third experiment uses a prior that is somewhere in between the two extremes. The prior is defined by $P(X) = 100 +$ *number of tokens in the XML element.* Of course, the priors should be properly scaled, but the exact scaling does not matter for the purpose of ranking. We hypothesise that the system using the length prior will outperform the baseline system

3.3 Evaluation results

This section presents the evaluation results of three retrieval approaches (no prior, 'half' prior, and length prior) on two query sets (content-only, and content-and-structure), following two evaluation methods (strict and liberal). We will report for each combination the precision at respectively 5, 10, 15, 20, 30 and 100 documents retrieved.

Table 1 shows the results of the three experiments on the content-only queries following the strict evaluation. The precision values are averages over 22 queries. The results show an impressive improvement of the length prior on all cut-off values. Apparently, if the elements that need to be retrieved are not specified in the query, users prefer larger elements over smaller elements.

precision	no prior	'half' prior	length prior
at 5	0.0455	0.0455	0.1909
at 10	0.0364	0.0455	0.1591
at 15	0.0303	0.0424	0.1394
at 20	0.0341	0.0364	0.1318
at 30	0.0364	0.0424	0.1318
at 100	0.0373	0.0559	0.1000

Table 1: Results of content-only (CO) runs with strict evaluation

Table 2 shows the results of the three experiments on the content-and-structure queries following the strict evaluation. The precision values are averages over 28 queries. The baseline system performs much better on the content-and-structure queries than on the content-only queries. Surprisingly, the length prior again leads to substantial improvement on all cut-off values in the ranked list.

precision	no prior	'half' prior	length prior
at 5	0.1929	0.2357	0.2857
at 10	0.1964	0.2321	0.2857
at 15	0.1976	0.2333	0.2714
at 20	0.1929	0.2232	0.2589
at 30	0.1786	0.2060	0.2607
at 100	0.0954	0.1107	0.1471

Table 2: Results of content-and-structure (CAS) runs with strict evaluation

Table 3 shows the results of the three experiments on the content-only queries using the liberal quantisation function defined above for evaluation. The precision values are averages over 23 queries. Again, the results show a significant improvement of the length prior on all cut-off values.

precision	no prior	'half' prior	length prior
at 5	0.1130	0.1391	0.4261
at 10	0.0957	0.1304	0.3609
at 15	0.0957	0.1333	0.3304
at 20	0.1000	0.1152	0.3000
at 30	0.1087	0.1232	0.2812
at 100	0.0896	0.1222	0.2065

Table 3: Results of content-only (CO) runs with liberal evaluation

Table 4 shows the results of the three experiments on the content-and-structure queries following the liberal evaluation. The precision values are averages over 28 queries. The length prior again shows better performance on all cut-off values. Note that the content-only task and the content-and-structure task show practically equal performance if the liberal evaluation procedure is followed.

precision	no prior	'half' prior	length prior
at 5	0.2429	0.2929	0.4000
at 10	0.2286	0.2823	0.3750
at 15	0.2262	0.2881	0.3738
at 20	0.2268	0.2821	0.3607
at 30	0.2179	0.2583	0.3595
at 100	0.1279	0.1571	0.2054

Table 4: Results of content-and-structure (CAS) runs with liberal evaluation

4 Discussion and future work

We presented an initial design and implementation of a system that supports XPath and complex information retrieval queries. In the CIRQUID project we will develop an algebra that allows us to define complex queries using language modelling primitives, like bigrams (proximity) conditional independence, and mixture models.

From the INEX experiments we conclude that it is beneficial to assign a higher prior probability of relevance to bigger fragments of XML data than to smaller XML fragments, that is, to users, more information seems to be better information.

Acknowledgements

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

References

[1] A. Berger and J. Lafferty. Information retrieval as statistical translation. In *Proceedings of 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 2.0. Technical report, World Wide Web Consortium, 2002.

[3] H.E. Blok. *Database Optimization Aspects for Information Retrieval*. PhD thesis, University of Twente, 2002.

[4] 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.

[5] N. Fuhr and K. Grossjohann. XIRQL: A query language for information retrieval in XML. In *Proceedings of SIGIR'01*, pages 172–180, 2001.

[6] 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.

[7] 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.

[8] T. Grust, Accelerating XPath location steps. In *Proceedings of ACM SIGMOD'02*, pages 109–120, 2002.

[9] D. Hiemstra. A linguistically motivated probabilistic model of information retrieval. In *Proceedings of the 2nd European Conference on Digital Libraries (ECDL)*, pages 569–584, 1998.

[10] D. Hiemstra. A probabilistic justification for using tf.idf term weighting in information retrieval. *International Journal on Digital Libraries*, 3(2):131–139, 2000.

[11] D. Hiemstra. *Using language models for information retrieval*. PhD thesis, University of Twente, 2001.

[12] D. Hiemstra. Term-specific smoothing for the language modeling approach to information retrieval: The importance of a query term. In *Proceedings of SIGIR'02*, pages 35–41, 2002.

[13] D. Hiemstra and F.M.G. de Jong. Disambiguation strategies for cross-language information retrieval. In *Proceedings of the 3rd European Conference on Digital Libraries (ECDL)*, pages 274–293, 1999.

[14] 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.

[15] J. List and A.P. de Vries. CWI at INEX. In *Proceedings of the first INEX workshop*, 2003. (in this volume)

[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 SIGIR'99*, pages 214–221, 1999.

[19] P. Ogilvie and J. Callan. Language Models and Structured Document Retrieval. In *Proceedings of the first INEX workshop*, 2003. (in this volume)

[20] C.P. Paice. Soft evaluation of Boolean search queries in information retrieval systems. *Information Technology: Research and Development*, 3(1):33–42, 1984.

[21] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Readings in speech recognition*, pages 267–296. Morgan Kaufmann, 1990.

[22] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[23] G. Salton, E.A. Fox, and H. Wu. Extended Boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.

[24] A. R. Schmidt, M. L. Kersten, M. A. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In *The World Wide Web and Databases – Selected Papers of WebDB 2000*, Springer-Verlag, pages 137–150, 2000.

[25] F. Song and W.B. Croft. A general language model for information retrieval. In *Proceedings of CIKM'99*, pages 316–321, 1999.

[26] A.P. de Vries. *Content and Multimedia Database Management Systems*. PhD thesis, University of Twente, 1999.

[27] J. Xu, R. Weischedel, and C. Nguyen. Evaluating a probabilistic model for cross-lingual information retrieval. In *Proceedings of SIGIR'01*, pages 105–110, 2001.

[28] R. van Zwol. *Modelling and searching web-based document collections*. PhD thesis, University of Twente, 2002.

The Xircus Search Engine*

Holger Meyer Ilvio Bruder Gunnar Weber
Andreas Heuer

University of Rostock
Database Research Group
18051 Rostock, Germany
{hme,ilr,weber,heuer}@informatik.uni-rostock.de

February 19, 2003

Abstract

Nowadays, XML is the document model in favour for both document- and data-centric web applications. Its influence in other, more traditional projects and applications 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 test bed for research in query processing for XML and semi-structured data in general.

1 Introduction

Traditional search engines are built 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 built XML-sensitive search engines should rely more on the XML structure and facilitate path expression and structured queries based on a type system.

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 fulltext 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 organised as follows. Primarily, we give an overview of the system architecture. Then, a brief discussion of query language issues 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.

2 Architectural Overview

Xircus has an component-based structure. Figure 1 depicts the distributed architecture of the search engine. The main components are the Xircus Agent, the Xircus Server and servlets in a web server.

The *Xircus Agent* gathers information from distributed XML-collections. It performs several document analysis and index preparation steps. Finally, it transmits the collected information to the server. The *Xircus Server* manages the basic data structures and performs the query evaluation. The *User-interface* is built by a set of servlets. These servlets communicate with the server using JDBC, as the agents do.

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

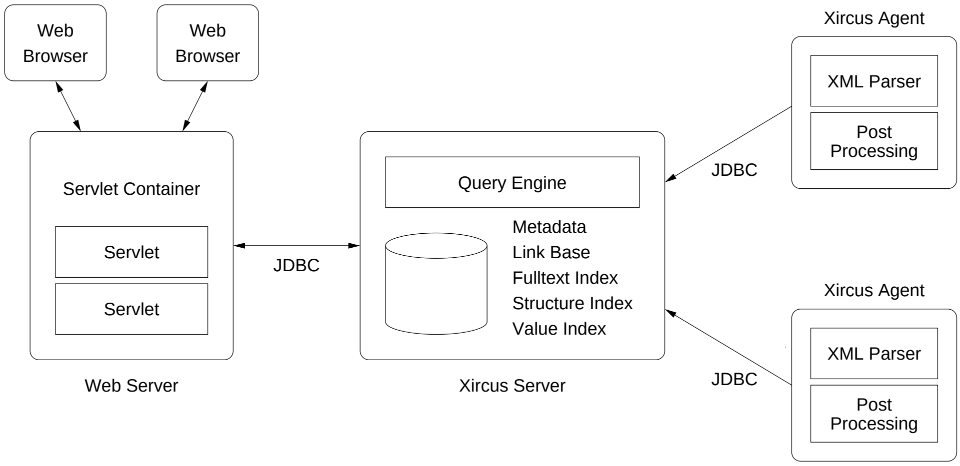


Figure 1: Xircus Architecture

Besides this component-based architecture, the process of indexing and querying can be illustrated by the processing steps necessary (Figure 2). After accessing the XML document collections the document analysis step starts in the Xircus agent. The extracted information is then handed over to the index preparation step that takes place in the Xircus server.

Document analysis First, some metadata on the document collections are collected by agents. This includes data like timestamps, document type, document length, checksum and other. The documents them-self are further analysed in two steps. At first, a structure analysis takes place which includes the extraction of the document structure tree and its relations to the content. Secondly, a content analysis is performed. There are a couple of analysis tools for the textual content analysis, e.g., linguistic tools, thesauri or ontologies.

There are some dependencies between analysis steps because some results of the one analysis is helpfully or even necessary for the other analysis. Term position must be determined before stopword elimination because some terms are not counted and some phrase search may fail. Stop-word elimination should be processed before stemming because stemming is expensive depending on the number of words. Generally, the metadata are collected first because some analysis are dependent on document or schema type.

The storage and index structures Several data and index structures are managed in the Xircus server:

- *Metadata storage*: collections, documents, statistics, schemes (DTD, XML Schema, index structures per collection)
- *Fulltext index*: words of the fulltext, sentences, phrases of a natural language, IR based querying
- *Structure- or path index* for querying the document structure, evaluating path expressions
- *Value index*, atomic element and attribute content, typed values (XPath 1.0 type system), for structured parts of a document, and SQL-like queries
- *Link-base*, outgoing and incoming edges per document, to analyse the hyperlink structure.

The *metadata* encompasses information on documents, e.g., checksum, timestamps, document type, collection affiliation and term statistics, and information on collections like document schema (DTD, XML Schema), main language, and other document statistics. Stopword lists or stop context can be defined on a per collection basis. A *stop context* is a XML fragment to be excluded from processing. It can be described by a path expression.

The data for the *fulltext index* consist of terms, their occurrences and the term position. The terms are processed from the document words by stopword elimination, stemming and possible usage of thesauri/ontology.

The term position are determined by sentence and word recognition. The *structure index* includes information on elements, element-subelement relationships, attributes, and paths. XML-elements are annotated by a position number too. Values, like author names or publication years, in the *value index* are extracted from XML-attributes or elements. They are associated with a data type as defined in XML Schema. Two kinds of *links*, references or citations are distinguished, in-collection and external hyperlinks and references. The links must be defined by structured elements in a known way, i.e using ID/IDREF in a DTD or with XLink/XPointer.

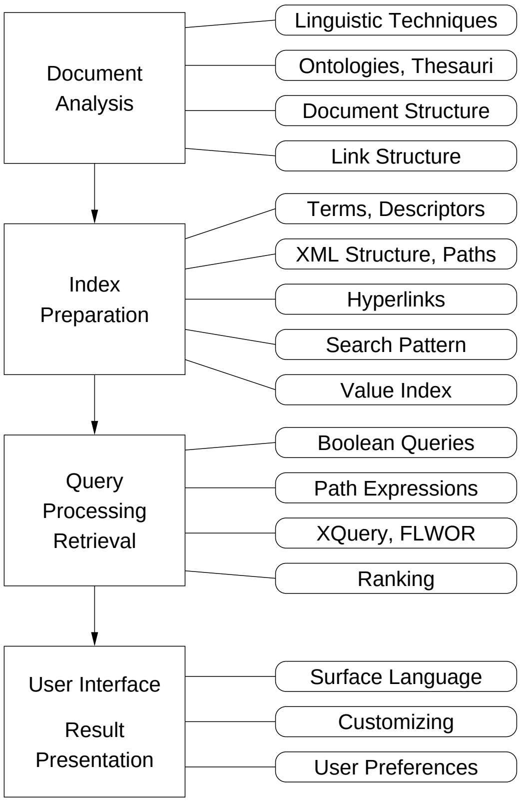


Figure 2: Xircus Processing Steps

3 Query Language Issues

A query language for an XML-sensitive search engine should support a combination of information retrieval (IR-like) and structured XML-queries (XML- or SQL-like), i.e.:

- Vague (IR-like) queries on the concatenated fulltext, regardless of the XML document structure.
- IR-like queries restricted to XML-fragments, which in turn can be described by path expressions.
- XML-like queries with a vague description of the content of desired elements or attributes.
- IR-like queries on the XML-structure which is basically IR on the XML-identifiers, e.g., pattern matching on element or attribute names. Here, the structure itself is a search term.
- Exact (SQL-like) queries on certain typed element or attribute values.
- Queries that allow for an exploration of the hyperlink structure.

Now we will have a short look at the retrieval language XircL, how combined queries can be expressed, and how the ranking mechanisms of Xircus works.

Table 1: Information retrieval like expressions

Expression types	**Meaning**
`term`	words
'$term_1$ $term_2$ ...'	phrases
{$term_1$ $term_2$ ...}	sentences
(`expr` ...)	grouping
$expr_1$ *op* $expr_2$	boolean operators *op*: **and**, **or**, **not**
`expr` * `factor`	weighted expressions to influence the ranking

Table 2: Query Expressions Involving Structure

Expression types	**Meaning**
path(*pexpr*)	embedded path expression, XPath 1.0
path(*pexpr*) **contains** *expr*	path restriction
pexpr comp const	value-based comparison, *comp*: =, <, ...
return *pexpr*	unit of interest described by *pexpr*, defaults to root-element, can be redefined on a per collection basis

3.1 The Query Language XircL

At the user level Xircus uses an information retrieval language (XircL). The user can pose boolean queries and use concepts like words, phrases, and sentences. With weighted expressions the ranking can be influenced. To query the structure of the XML documents path expression can be used in conjunction with fulltext operations. Tables 1 and 2 summarise the elements of XircL.

The IR-like part of XircL consists of simple keyword search, combining keywords within boolean expressions, or querying for phrases, sentences, and influencing the ranking of results by giving weightings for terms.

Path expression can be used in two ways:

(1) *Embedded paths*: expressions like "**path**(*expr*)" will qualify all documents containing the specified path expression (*pexpr*).

(2) *Path restrictions*: expressions like "**path**(*pexpr*) **contains**(*expr*)" will limit the search for certain concepts, terms or words *expr* to the XML document fragments described by *pexpr*.

For path expressions *pexpr* the *XML Path Language* is used. The XPath 1.0 implementation in Xircus comes with some restrictions: only a few of the built-in functions are implemented and solely navigations along the **ancestor** and **descendant** axis are permitted actually.

Often not the whole XML document should be referenced in the query result but only a certain fragment.

XircL offers a concept to influence the structure of the returned query result. "`return pexpr`" returns references to the document fragments matching `pexpr`. Per default, references to the root nodes of the matching documents are returned.

To illustrate querying with XircL we use the topic 21 from the INEX collection: "Which authors of articles cited recent work by Heikki Mannila?" The query is expressed in XircL this way:

```
path(//bm/bb/au)
  contains Heikki and Mannila
and
path(//bm/bb/pdt/yr) >= 1998
return /article/fm
```

The back matter of an article is searched for the author Heikki Mannila. The search is restricted by an exact query term, which selects references from 1998 up to now. Since we are interested in authors who cited Heikki Mannila, we just want to return the front matter stuff (author, title) of the article.

3.2 Ranking

The ranking mechanism of Xircus assigns relevance measures to documents or fragments based on the statistics stored in the database. The ranking value is calculated from four measures for similarity between documents and queries. These similarities are based on (1) terms, (2) the XML structure, (3) element and attribute values, and (4) the linking structure. These four measures can be combined in a ranking function. The combination is controlled by user ratings or by a user defined ranking function. Computing these similarities involves processing the related index structures: the fulltext/term index, the path index, the value index, and the hyperlink base.

- Ranking for term-based queries based on: $tf \cdot idf$ (term frequency and inverse document frequency).
- Similar ranking for embedded path expressions based on: $ef \cdot iff$. Element frequency ef: element occurrences divided by the number of elements in the XML fragment. Inverse fragment frequency iff: logarithm of number of fragments divided by number of fragments containing the element.
- Ranking of value-based comparison: is mapped to the boolean values {1, 0}.

Since XircL combines IR-like queries, which result in a ranking, with structured queries, the challenge is, how to integrate the result (ranking) of the different subquery types? We adopted a technique used in multimedia database systems [3]. Ranking values for different sub-queries are combined based on graded sets (Fuzzy sets).

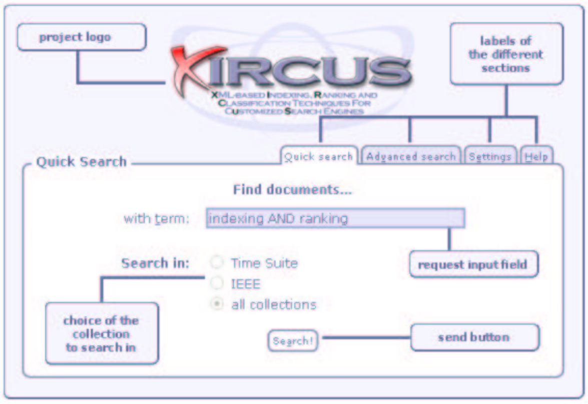


Figure 3: Xircus Search Interface

A graded set is a set of pairs (i, g): where i is an item (document, fragment, object) and g is a real number in the interval $[0, 1]$. The following rules hold for $rank_Q(i)$, grades/ranks for an item i under the query Q:

- conjuncts:

  $rank_{A \wedge B}(i) = min\{rank_A(i), rank_B(i)\}$
- disjuncts:

  $rank_{A \vee B}(i) = max\{rank_A(i), rank_B(i)\}$
- negations:

  $rank_{\neg A} = 1 - rank_A(i)$

Based on these rules the query evaluation will return a combined ranking for queries on both the fulltext and the structure of an XML document.

4 The Xircus prototype and user interface

The first Xircus prototype was implemented by students of the *Complex Software Systems* class at University of Rostock during the summer term 2002. The prototype realizes all major concepts except the index structures. By now, the functionality is provided by an object-relational database system (IBM DB2) and its extenders. Most index structures are implemented with user tables and indexes.

Xircus is implemented in the Java language. It makes heavy use of free software, e.g. for the checksum tool, based on the MD5 hash value (RFC 1321), the stemmer, based on the Porter stemming algorithm, and the synonym sets of Wordnet[1] (a project at University of Princeton).

The user interface (Figure 3 is realized as a set of servlets executed in the usual Apache/tomcat webserver. The servlets issue search queries in the Xircus surface language and inter-operate with the Xircus

[1] `http://www.cogsci.princeton.edu/~wn/`

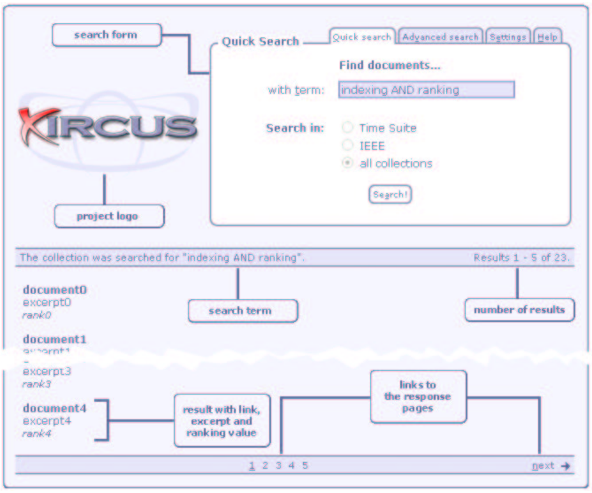


Figure 4: Xircus Result Presentation

search server using a standard JDBC-interface. This gives much freedom in independently changing the design of both components. Figure 4 exemplifies the search form and the search result presentation.

5 Related Work

We will have a short look at some related products, projects and research issues that are related to the Xircus project.

XML search engines. GoXML [10] provides the storage of XML Schema or DTD structure definitions. When XML content is inserted or updated in a database it is checked for compliance with a schema and the data types defined within that schema. The Index System creates and maintains indexes over attribute and element values. These are used by the XPath Query Engine, which supports XPath with proprietary extensions. GoXML DB includes also support for "... a major subset of XQuery (FLWR, SORTBY, distinct) as specified in the June 2001 public W3C drafts."

The TEXTML [7] Server processes any well-formed XML without being constrained by a particular schema or DTD. Indexes can be created to search for words (fulltext), dates, strings (whole content of an XML document), numerical values, and date and time values. The server offers fine granular indexes, which can account for the position of every occurrence of a word within a document, therefore allowing advanced search capabilities like proximity search. The query language is expressed as an XML document and provides Boolean search and fulltext search over whole documents or individual elements.

XYZFind [11] builds a search-able repository of all data from all XML documents, indexing values, numbers, structural names, namespaces, and content. It accepts any number of well-formed XML documents. XYZFind provides a powerful query language called XYZQL. XYZQL supports path-level queries, Boolean queries, keyword search, and numeric range queries. An XYZQL query is a filter specification that constrains which XML documents are returned as well as which parts of documents are returned.

Linguistic techniques. An overview on IR-related text analysis and processing gives [1]. It describes linguistic-related analysis with a focus on collecting statistic term information and term preparation for indexing. A robust and fast linguistic analysis tool is represented in [8] (SMES). SMES is a linguistic tool for the German language and consists of lexical, morphological and syntactical analysis. It can extract linguistic annotated word lists and also linguistic relations between words and word phrases.

Ranking. [6] gives an overview on ranking algorithms. It describes several ranking aspects in the IR research area including a guide to selecting ranking techniques. A survey on general combining ranking algorithms gives [2]. A ranking approach for structural data using the probabilistic model is XPRES [9] from the University of Bonn. XPRES describes extensions to the probabilistic ranking function using given structure information from XML documents. Another approach [5, 4] consists of an inference machine for probabilistic document weights combined with structural data. It defines different contexts for term weightings in different structural areas.

Using fuzzy sets for integrating scoring values into a structured query language like SQL was first introduced by Fagin [3].

6 Future Tasks

Based on the first prototype, future investigations will go into several directions. We will improve the path index structures especially if an XML schema for a document collection is present. A redesign of the distributed software architecture is needed to support better index preparation and distributed query processing. We plan for using the search engine in digital library projects in large scale, distributed environments where replication, caching and distributed query processing is important.

A recently started second student project will reimplement the fulltext index using compression algorithms and experiment with path index structures. The user interface will be extended and performance evaluation based on the INEX collection will take place.

Detailed information on the ongoing Xircus project can always be found on the project homepage[2].

[2] `http://www.xircus.de/`

7 Acknowledgements

The authors would like to thank the students of *Complex Software Systems* class at University of Rostock who implemented the first Xircus prototype. These are Ramona Bunk, Sebastian Dolke, Thomas Lange, Lars Milewski, Manja Nelius, Mathias Reusch, Gunnar Söllig, Sven Schattat, Matthias Schulz, and Ines Weber.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press & Addison Wesley, New York, USA, 1999.

[2] W. B. Croft. Combining Approaches to Information Retrieval. In W. B. Croft, editor, *Advances in Information Retrieval*. Kluwer Academic Publishers, Boston, 2000.

[3] R. Fagin. Combining Fuzzy Information from Multiple Systems. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, 1996, Montreal, Canada*, pages 216–226. ACM Press, 1996.

[4] N. Fuhr and K. Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proceedings of the 24th Annual International Conference on Research and Development in Information Retrieval*, pages 172–180. ACM, 2001.

[5] N. Fuhr and G. Weikum. Classification and Intelligent Search on Information in XML. *Bulletin of the IEEE Technical Committee on Data Engineering*, 25(1), 2002.

[6] D. Harman. Ranking Algorithms. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval — Data Structures & Algorithmns*, pages 363–392. Prentice Hall PTR, New Jersey, USA, 1992.

[7] IXIASOFT. *TEXTML-Server*, Nov. 2001.

[8] G. Neumann, R. Backofen, J. Baur, M. Becker, and C. Braun. An Information Extraction Core System for Real World German Text Processing. In *Proc. of the 5th International Conference of Applied Natural Language*, Washington, USA, 1997.

[9] J. E. Wolff, H. Floerke, and A. B. Cremers. XPRES: a Ranking Approach to Retrieval on Structured Documents. Technical report, University of Bonn, July 1999.

[10] XML Global Technologies, Inc. *Choosing The Correkt Database For XML Content*, 2002.

[11] XYZFind Corporation. *XYZFind Server User's Guide, Version 1.01*, Mar. 2001.

An XML Retrieval Model based on Structural Proximities

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

Abstract

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

1 Introduction

XML is a markup language for describing documents and for interchanging data among different systems. The application domain of XML is gradually growing as Web pages, e-catalogs, e-books, etc. increasingly employ XML for exchanging data. In particular, many XML systems have been or are being developed for storing, maintaining, and retrieving XML data or documents in companies or organizations.

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

1. An XML document has explicit structure.
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. Although some structured document retrieval models utilize structural information in ranking relevant document components [6], they do not explicitly consider proximities of the structural components in their weighting. Moreover, end-users in heterogeneous document collections may encounter problems in querying irregular structured documents because they do not know the exact structure or they have difficulty in composing structured queries.

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

- XML retrieval models can make the most of structured information of XML documents in ranking.
- The XML retrieval system should provide the user with an easy but descriptive structured query language.

In this paper, 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 treats a link just as an attribute.

We propose a model for XML document retrieval evaluated in a bottom-up way, as in [8,9,12]. Before a structural query condition is evaluated, content conditions in the query are evaluated in order to reduce the document search space. 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

We model XML documents as ordered trees, as shown in Figure 1. 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.

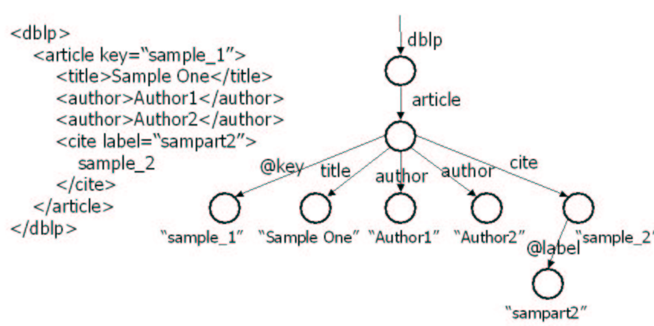
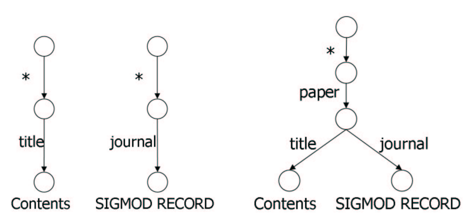


Figure 1: XML document and its document tree

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 PC-DATA value.

2.2 XML Querying Model

Because a document is 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.



(a) SRP query (b) query tree

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. [2–4, 7, 11, 14, 16] defined proximity operators for considering proximity in retrieval, or incorporated proximity in their weighting formulas. Among these, it was passage retrieval that showed the most possibility in ranking documents. Passage retrieval approaches retrieve relevant documents based on the combined weight of all relevant passages contained in a document or based on the highest ranking passage of the document. However, existing passage retrieval approaches have difficulty in combining the weights of passages when they were retrieving document which may include fixed size passages or retrieving logical units such as <section>, <chapter> or <book>. This problem is more serious in XML retrieval as a user may retrieve not only leaf nodes, but also nodes of varying granularity. To address this problem in the structured document retrieval, 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 nodes at higher levels in the structure.

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 sections 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:

$$\begin{aligned}
t_{ij} &: j^{th} \text{ word in the document } i \\
Nd(t_{ij}) &: \text{returns the leaf node containing } t_{ij} \\
N_{ik} &: k^{th} \text{ BFS(Breadth First Search)} \\
&\quad\ \text{numbered node in document} i \\
lev(N_{ik}) &: \text{return the level of } N_{ik} \\
Prnt(N_{ik}) &: \text{return the parent node of } N_{ik} \\
maxH(i) &: \text{maximum number of children of}
\end{aligned}$$

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 ∞).

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

$$V\text{–}dist(N_{ij}, N_{ik}) = |lev(N_{ij}) - lev(N_{ik})| \quad (2)$$
$$\text{if } N_{ij} \text{ is a descendant or ancestor of } N_{ik}$$

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

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

Basically, $T\text{–}dist$ is the same as word distance

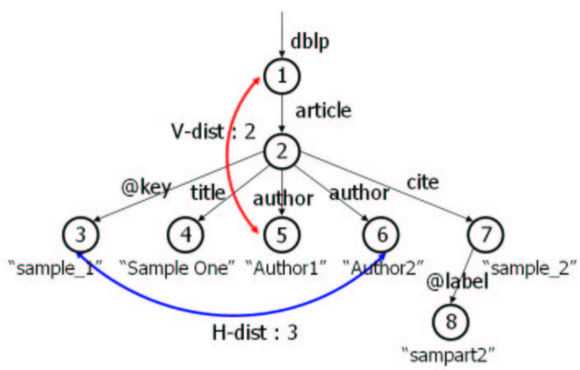


Figure 3: Distances

between words in an unstructured document, but here we apply it to the leaf node of XML documents. For example, $T\text{–}dist(\text{Sample}_{i6}, \text{One}_{i7})$ is 1 between 'Sample' and 'One' in a leaf node of the path "/dblp/article/title" on Figure 3. $H\text{–}dist$ compares sibling nodes and computes the of BFS numbers. For instance, $H\text{–}dist(N_{i3}, N_{i6})$ between '/dblp/article/@key' and '/dblp/article/author' is 3 in Figure 3. $\mathbb{H}\text{–}dist$ is a normalized version of $H\text{–}dist$. The reason for the normalization of $H\text{–}dist$ is to overcome the differences due to document length, especially in horizontal distance.

$V\text{–}dist$ is the difference between levels of nodes in a path. $V\text{–}dist(N_{i1}, N_{i5})$ between '/dblp' and '/dblp/article/title' is 2. Both $V\text{–}dist$ and $H\text{–}dist$ 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, $H\text{–}prox$(Horizontal proximity) and $V\text{–}prox$ (Vertical proximity), which relates to the corresponding distance measure.

With the following notations

$$D : H\text{–}dist(N_{ij}, N_{ik})$$
$$C : \frac{lev(N_{ik})}{maxV(i)}$$
$$p, v : \text{constant } (0 \le p, v \le 1)$$
$$maxV(i) : \text{the depth of document } i,$$

we define $H\text{–}prox$ as

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

$H\text{–}prox$ exponentially decreases with growing $H\text{–}dist$ D. This definition is based on the assumption that $H\text{–}prox$ should be close to 0 as the two nodes are far apart. We also considered the product and the sum of p and $\mathbb{H}\text{–}dist$. 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 $H\text{–}dist$ according to the level in the tree. v is the degree of the effect of C on p.

Using the notations

$$t : V\text{–}prox \text{ factor } (0 < t \le 1)$$
$$V : V\text{–}dist(N_{ij}, N_{ik})$$
$$N_{ij} : \text{an ancestor or descendant of } N_{ik},$$

we define

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

$$\mathbb{V}\text{–}prox(N_{ij}, N_{ik}) = min\left(\frac{lev(N_{ij})}{lev(N_{ik})}, \frac{lev(N_{ik})}{lev(N_{ij})}\right) \quad (7)$$

Like $H\text{–}prox$, $V\text{–}prox$ is a decreasing function according to $V\text{–}dist$. $V\text{–}prox$ is decreased by t ratio according to $V\text{–}dist$ V. The normalized version $\mathbb{V}\text{–}prox$ also considers the level of the nodes. $\mathbb{V}\text{–}prox$ 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 [8, 9, 12]. Firstly, content based queries are performed to reduce search space. Then, the nodes that satisfy content based queries are verified by the 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

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

The Equation 8 represents the weight of a leaf node N_{ik} computed by the naive 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 [10,13].

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, [5, 14] 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 only documents 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 in closer proximity, the document has more important meaning. On the other hand, if the matched nodes are further apart, then the document is less important. But [5,14] 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(Ξ) between two sibling nodes based on *H–prox*, *V–sum* operator(Υ) between ancestor-descendant nodes based on *V–prox* and the weight of an internal node using Ξ and Υ.

$$\begin{aligned} W_{iq} &= W_{ij} \Xi W_{ik} \text{ defined as} \\ &\quad W_{iq} = max(W_{ij}, W_{ik}) \\ &\quad + H\text{–}prox(N_{ij}, N_{ik}) \cdot min(W_{ij}, W_{ik}) && (9) \\ &\quad\quad \text{where } q = \frac{j * W_{ij}}{W_{ij} + W_{ik}} + \frac{k * W_{ik}}{W_{ij} + W_{ik}} \\ W_{ii} &= W_{ii} \Upsilon W_{ik} \text{ defined as} \\ &\quad W_{ii} = W_{ii} + V\text{–}prox(N_{ii}, N_{ik}) \cdot W_{ik} && (10) \\ &\quad\quad \text{where } N_{ii} \text{ is an ancestor of } N_{ik} \\ W_{ii} &= W_{ii} \Upsilon max\{W_{ij} \Xi W_{ik} \Xi \cdots \Xi W_{in}\} && (11) \\ &\quad\quad \text{where } N_{ii} \text{ is parent of } N_{ij}, N_{ik}, \cdots, N_{in} \end{aligned}$$

We design Ξ to preserve the weight of better one but decrease the weight of the other based on *H–prox*. When the weights of two nodes reside in a node, *H–sum* of these two node produce the sum of these two node with no loss; however, when the one is far apart from the other, *H–sum* of these two node 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–sum* is the centroid of these two node. On the contrary with Ξ, Υ use *V–prox* directly applied to only descendant node. Because associative law for Ξ is invalid, we employ $max\{\cdots\}$.

3.3 Heterogeneity

In 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 a 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 attribute for a leaf node – $H^S_{ik,j}$, the level of a leaf node to j^{th} query path for the later computation of *V–prox*; $H^T_{ik,j}$, the heterogeneity value to j^{th} query path. H^T_{ik}

is a vector for these two attribute of all kinds of query path. Our formal definition is as follows :

$$H^T_{ik,j} = \{ \begin{matrix} 1 \text{ if } j^{th} \text{ query path is matched} \\ 0 \text{ otherwise} \end{matrix} \quad (12)$$

$$H^S_{ik,j} = lev(N_{ik}) \quad (13)$$

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

$$H^T_{ik} = \begin{pmatrix} H^T_{ik,1} & H^T_{ik,2} & \cdots & H^T_{ik,|SRP|} \\ H^S_{ik,1} & H^S_{ik,2} & \cdots & H^S_{ik,|SRP|} \end{pmatrix} \quad (15)$$

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

For the heterogeneity of an internal node, we define the heterogeneity sum operator (Φ) between two sibling nodes N_{ik} and N_{iq} and define H^T_{ii}, the heterogeneity of an internal node(N_{ii}).

$$H^T_{it} = H^T_{ik} \Phi H^T_{iq} \qquad w.l.o.g.\ Let\ H_{ik} \geq H_{iq} \quad (16)$$

for each j

$$H^T_{it,j} = \{ \begin{matrix} \tau & \text{if } H^T_{ik,j} < \tau \\ H^T_{ik,j} & \text{otherwise} \end{matrix} \quad (17)$$

$$H^S_{it,j} = H^S_{iq,j}$$

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

$$\tau : \left(\frac{lev(N_{ik})}{H^S_{iq,j}} \right)^h$$

$$H^T_{ii} = max\{H^T_{ij} \Phi H^T_{ij} \Phi \cdots \Phi H^T_{in}\} \quad (18)$$

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

In Equation 16, τ adopts $V\text{–}prox$. 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, τ 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, τ will be nearly 1. Equation 18 computes the heterogeneity of an internal node. Since Φ is not associative, we also employ $max\{\cdots\}$. 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} ($\mathbb{W}_{ij}$) reflecting H_{ij} is as follows :

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

When $k = 0$, $\mathbb{W}_{ij}$ yields the same results of W_{ij}, which means that only reflects proximitis 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 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, 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

Since Ξ and Υ operators reduce the weight of sibling or child nodes, we utilized structural proximity operators as a our length normalization method. By using Ξ operator, we may sum up sibling nodes' degraded weights to one node's weight according to $H\text{–}prox$, which serve as a sibling node length normalization. With an iterated application of the Ξ operator from a leaf node level to the root node level, we can compute a normalized weight of the root node of a document. However, according to this, the level is closer to the root, the weight of an internal node will increase and not decrease. To solve this problem, we make use of Υ operator, which reduces the weight of an internal node for each level, to select best weight's level.

4 Experiments

We tested our proposed model on the INEX 2002 test collection. For the CO (Contents Only) topics, we generate CO queries automatically but for the CAS (Contents And Structure) topics, we slightly modified automatically generated queries.

4.1 Results

Figure 4-7 employed naive TF * IDF weighting scheme, p($H\text{–}prox$ factor) = 0.5 and $k = 100000000$ if

k is required [15]. For the Figure 4 and 5, we considered only title element of topics but we used title and keyword in the Figure 6 and 7

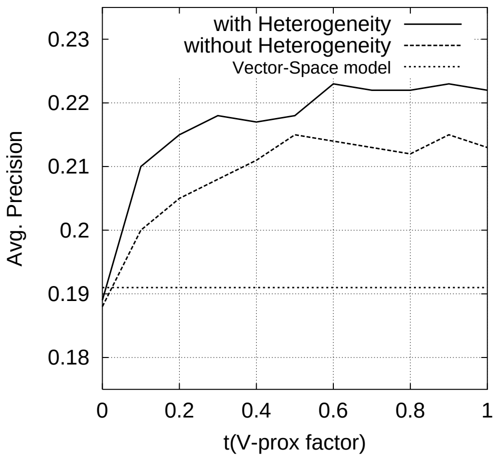


Figure 4: t variation graph for the CAS topics(strict)

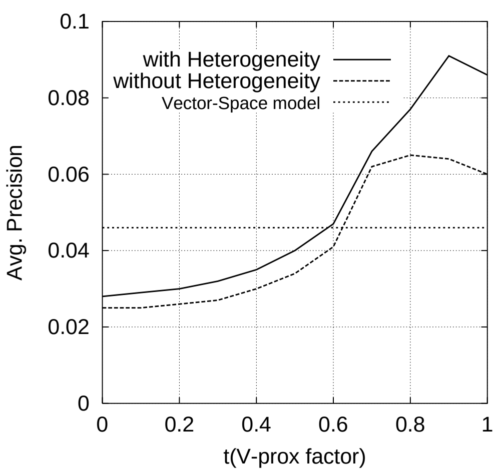


Figure 5: t variation graph for the CO topics(strict)

4.2 Result Analysis

Figure 4 and 5 showed average precision variations according to t. These two graph showed that our retrieval model outperformed Vector-Space model which used normalized TF*IDF when $t > 0.6$ for the CO topics and $t > 0$ for the CAS topics. With the Heterogeneity, we could obtain better average precisions, which may not changed when $t > 0$. We recommend $t = 0.9$ for the CAS topic and $t \geq 0.6$ for the CO topic.

Since our run produced 1500 maximum results for each topic, we could not directly compare our run with official runs. But compared with the other participants' official runs, our run, in the Figure 6 and 7, showed good retrieval performances on CO topics and reasonable retrieval performances on CAS topic.

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 proposed a new XML retrieval model, which considers proximities of query terms as well as heterogeneity of query paths, and we defined 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. Through the experiment we showed that our retrieval model has good retrieval performance on contents only topics and reasonable performance on contents and structure topics.

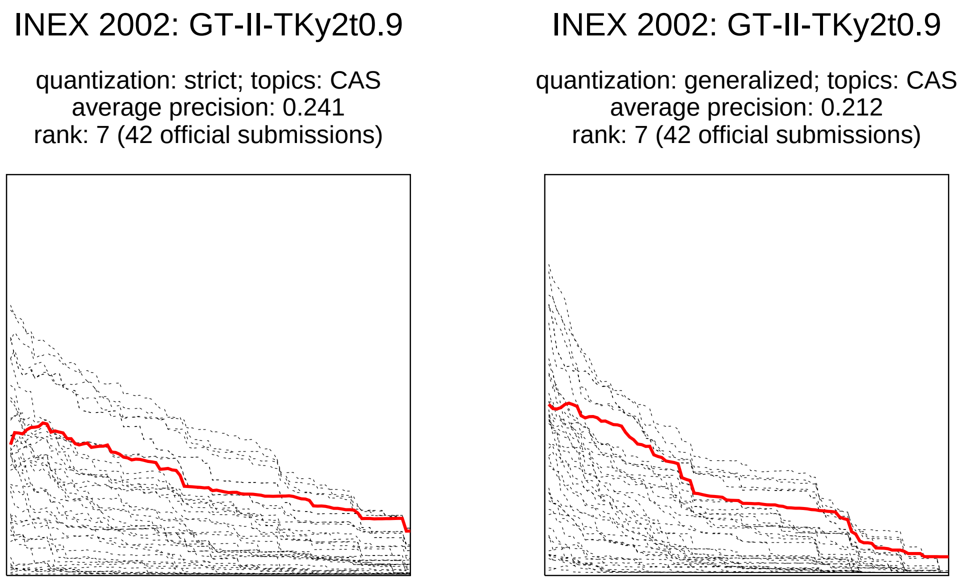
Our further work is the extension of our query model. So far, the logical query structure is linear but we may extend this structure, e.g. by grouping. In addition, we will also consider hyperlink structures or Boolean connectives, e.g. by applying the p-norm model.

References

[1] http://www.w3.org/DOM/.

[2] 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. Spring-Verlag.

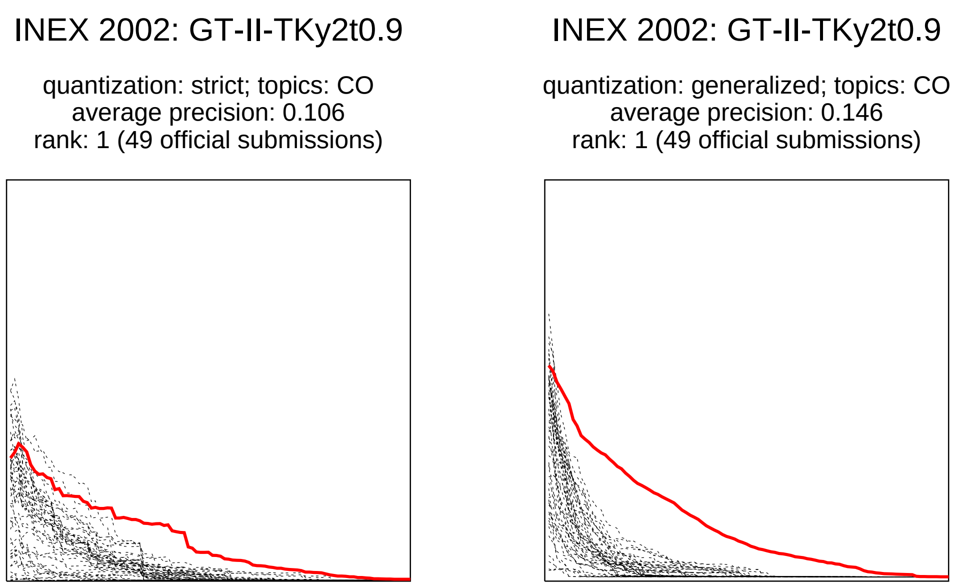
[3] C. Clarke, G. Cormack, and F. Burkowski. Shortest substring ranking (MultiText experiments for TREC-4). In D. K. Harman, editor, *Proceedings of the 4th Text Retrieval Conference(TREC-4, Washington, D.C., Nov.)*, pages 295–304, 1995.



(a) Strict quantization

(b) Generalized quatization

Figure 6: Precision/Recall graph for the CAS topics



(a) Strict quantization

(b) Genralized quatization

Figure 7: Precision/Recall graph for the CO topics

[4] Charles L. A. Clarke and Gordon V. Cormack. Shortest-substring retrieval and ranking. *TOIS*, 18(1):44–78, January 2000.

[5] 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.

[6] N. Fuhr, N. Goevert, G. Kazai, and M. Lalmas (eds). Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval(INEX), Dagstuhl 9-11 Dec. 2002, ERCIM Workshop Proceedings. March 2003.

[7] 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.

[8] S. H. Myaeng and et. al. A Flexible Model for Retrieval of SGML Documents. In *SIGIR*, pages 138–145, 1998.

[9] Gonzalo Navarro and Ricardo Baeza-Yates. Proximal nodes: a model to query document databases by content and structure. *TOIS*, 15(4):400–435, 1997.

[10] 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.

[11] G. Salton and C. Buckley. Automatic text structuring and retrieval: Experiments in automatic encyclopedia searching. In *ACM/SIGIR Conference*, pages 21–31, 1991.

[12] D. Shin, H. Jang, and H. Jin. BUS: An Effective Indexing and Retrieval Scheme in Structured Documents. In *Digital Libraries*, pages 235–243, 1998.

[13] Howard R. Turtle and W. Bruce Croft. Evaluation of an Inference Network-Based Retrieval Model. *TOIS*, 9(3):187–222, 1991.

[14] Ross Wilkinson. Effective retrieval of structured documents. In *SIGIR '94*, pages 311–317, 1994.

[15] Shinjae Yoo, Kyung-Sub Min, and Hyung-Joo Kim. XML Document Retrieval Models for Heterogeneous Data Set by using Independent Regular Paths. *JKISS(to be appear)*.

[16] Justin Zobel, Alistair Moffat, Ross Wilkinson, and Ron Sacks-Davis. Efficient retrieval of partial documents. *Information Processing and Management*, 31(3):361–377, 1995.

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
```

CWI at INEX2002

Johan List and Arjen P. de Vries

Center for Mathematics and Computer Science (CWI)
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 (the Initiative for the Evaluation of XML Retrieval) 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 *dimensions of relevance* to (possibly structural) properties of documents and use these dimensions of relevance for retrieval purposes. The study concentrates on *coverage*, a measure reflecting how focused the component is on the given topic while considering that it should serve as an informative unit to be retrieved by itself. 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.

1 Introduction

This paper describes our participation in INEX (the Initiative for the Evaluation of XML Retrieval). We participated with our XML retrieval system, built on top of 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 developed by Hiemstra [11].

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*, a measure describing how much of the document component is relevant to the topic of request while also considering that it should serve as an informative unit to be retrieved by itself.

2 The Retrieval Model

Research in the user modeling and concept of relevance areas (see e.g. [3, 4, 5, 2]) suggests that relevance is a multidimensional concept of which *topicality* (i.e. content-based relevance) is only a single one. Mizarro [16] 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; measured by e.g. the size of documents and the number of documents returned to the user).

Additional dimensions of relevance become more important for structured document retrieval. Retrieval units can vary in granularity and hence vary in the amount of information offered to the user. This varying amount of information highly likely causes a user to judge the relevance of document components on more properties besides topicality alone.

We model dimensions of relevance with a set of independent probabilities (assumed independent given a document instantiation) in a probabilistic retrieval model. The research question is whether we can effectively map dimensions of relevance to document properties (structural or otherwise) that in turn can be represented by (probabilistic) entities in the retrieval model. The results reported here investigate a combination of quantity and topicality, visualized in Figure 1; aiming to capture the notion of coverage used in the evaluation.

2.1 A Motivating Example

In INEX, retrieval results are judged on two aspects: relevance and coverage. Relevance is aimed to reflect how exhaustively a topic is discussed within a document component; coverage reflects how focused the component is on the given topic, considering that it also serves as an informative unit. The INEX relevance assessment guide [1] defines relevance and coverage on a four degree scale: relevance levels of 0 (irrelevant), 1 (marginally relevant), 2 (fairly relevant), and 3 (highly relevant), and coverage of N (no coverage), E (exact), S (too small) and L (too large). With the combination of these measures it is possible to identify document components that satisfy both topicality and quantity.

Consider the example document in Figure 2. Say that 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 (i.e. what is the retrieval unit?). The additional context

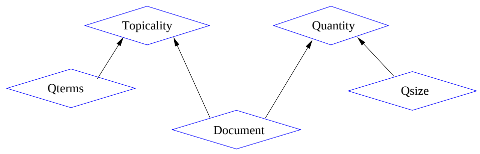


Figure 1: Encoding of additional relevance dimensions. Note that *Qterms* and *Qsize* denote information given by the query (query terms and preferred component size).

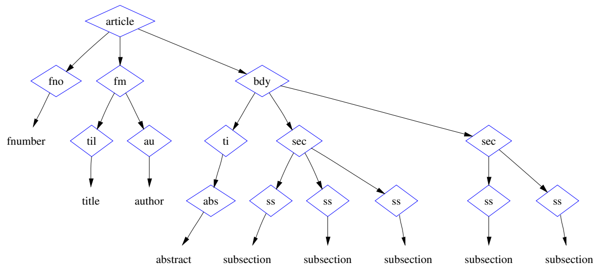


Figure 2: Running example XML syntax tree.

provided by the full sections or body may be more desirable for a user than the individual two subsections in isolation.

We approach the problem of chosing the best acceptable retrieval unit by optimizing on both topicality and size of document components:

- the shorter the document component, the more likely it will not contain enough information to fulfill the information need (the component may be less exhaustive, e.g. relevance level 1 or 2, and 'too small', coverage grade S);
- the longer the document component, the more likely that distilling the topically relevant information will take substantial more reader effort (the component may be more exhaustive, e.g. relevance level 3, but 'too large', grade L on the INEX coverage scale).

We therefore rank the documents in a collection against a combination of topicality and quantity (where the user uses document component size as a representation of quantity). In probabilistic terms, we calculate the probability of complete relevance of a document component, given its probability of relevance on both the topicality and the quantity dimensions.[1]

[1]Here, 'complete relevance' covers all dimensions of relevance, unlike the 'exhaustiveness only' notion of relevance used in INEX.

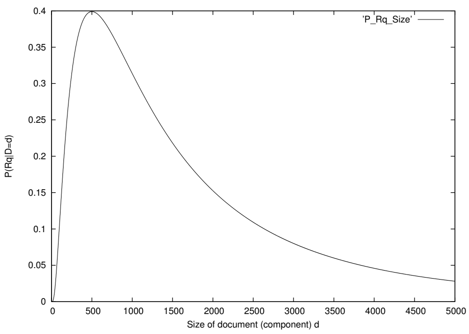


Figure 3: The log-normal distribution used for modeling the quantity dimension

2.2 Modeling Relevance Dimensions

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 subsection 2.1 and especially the user reasoning for modeling the quantity dimension, we decided to use a log-normal distribution as in Figure 3. The steep slope at the start reflects the pruning we want to model for (extremely) short document components since short components are unlikely to be good retrieval units. The long tail reflects that we do want to prune out very long document components, but not as rigorously as extremely short ones. Long components might be useful, even while taking more reader effort to distill the relevant information.

We also need a modeling parameter for the distribution itself. We have chosen 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 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.

2.3 Modeling Topicality

The model used for describing topicality of documents is a probabilistic model, the statistical language model described by Hiemstra [11]. 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 $\{T_1,\cdots,T_n\}$ is calculated with:

$$\begin{aligned} P(R_t|D_d,Q_{terms}) &= P(R_t|D_d,T_1,\cdots,T_n) \\ &= P(D_d)\prod_{i=1}^{n} P(I_i)P(T_i|I_i,D_d) \end{aligned}$$

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 [11] to relate the model to a weighting scheme (tf.idf-based). After some manipulation of the model we get:

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

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

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

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

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:

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

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 λ for all terms):

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

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 the storage of the edges present in the syntax tree, combined with storage of the text present. One of these approaches is described by Schmidt [17], 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:

```
<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 4a 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 4b. 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:

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

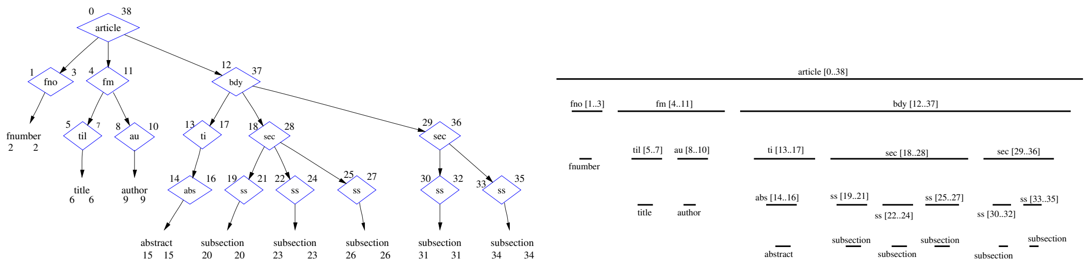
We also define the node index $\mathcal{N}$ as the projection of o_r over the set of all indexed regions.

3.2 Document Manipulation

The linearized string view enabled us to use theory and practice from the area of text region algebras [7, 8, 9, 13, 15, 14] for selection and manipulation of (sets of) text regions. Table 1 summarizes the operators in our system. The *containment* operation $a \supset b$ determines if the region a contains some other region b, $length$ gives the length of a region including markup and $textlength$ gives the length of a region excluding markup. Analogous join operators are defined on region sets (A and B).

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:



(a) Start point and endpoint assignment

(b) Region representation

Figure 4: Region indexing of XML documents

Table 1: Region and region set operators (the set operators are given in comprehension syntax [6]). Note that s_r and e_r denote the starting and ending positions of region r.

Operator	Definition
$a \supseteq b$	$true \iff s_b \geq s_a \wedge e_b \leq e_a$
$a \supset b$	$true \iff s_b > s_a \wedge e_b < e_a$
$length(a)$	$e_a - s_a + 1$
$textlength(a)$	$\vert\{a\} \bowtie_{\supset} \mathcal{W}\vert$
$A \bowtie_{\supseteq} B$	$\{(o_a, o_b) \vert\, a \leftarrow A,\ b \leftarrow B,\ a \supseteq b\}$
$A \bowtie_{\supset} B$	$\{(o_a, o_b) \vert\, a \leftarrow A,\ b \leftarrow B,\ a \supset b\}$
$length(A)$	$\{(o_a, length(a)) \vert a \leftarrow A\}$
$textlength(A)$	$\{(o_a, textlength(a)) \vert\, a \leftarrow A\}$

$desc := \{d\} \bowtie_{\supseteq} \mathcal{N}$. 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: $text := \{d\} \bowtie_{\supset} \mathcal{W}$;
- 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 post-ordering approach for acceleration of XPath queries, proposed by Grust [10] (we consider Grust's approach a specific instance of general text region algebras, as is ours).

4 Experiments

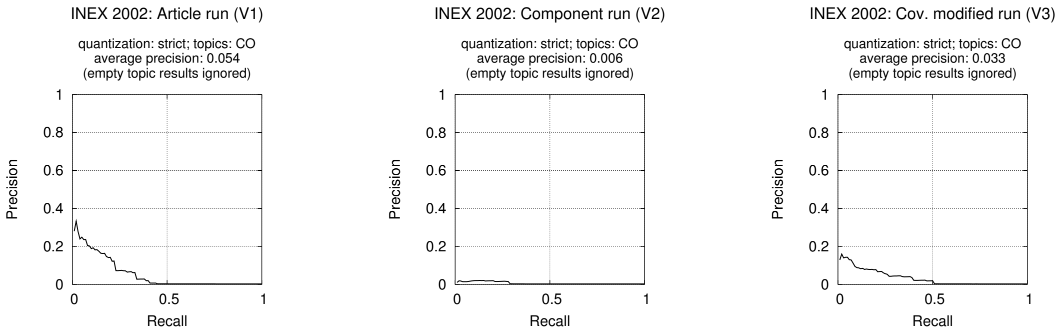
We designed three experimentation scenarios. The first scenario represents the baseline scenario of 'flat-document' retrieval, i.e. retrieval of documents which possess no structure. After examination of the document collection, we decided to perform retrieval of article-components. The second scenario regarded 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. Also note that we focused on content-only queries only (i.e. we used the same approach for content-and-structure queries).

Table 2: Experimentation scenarios

Scenario	Retr. Unit	Dimension(s)
V_1	$\{tr('article')\}$	$topicality$
V_2	$\{tr('*')\}$	$topicality$
V_3	$\{tr('*')\}$	$top., quant.(500)$
V_4	$\{tr('*')\}$	$top., quant.(2516)$
V_5	$\{tr('*')\}$	$top., quant.(5106)$

The official recall-precision graphs of our three submitted runs are presented in Figures 5a through 5f. The recall-precision graphs are constructed after mapping relevance/coverage combinations to a binary scale. The mapping function for strict evaluation is:

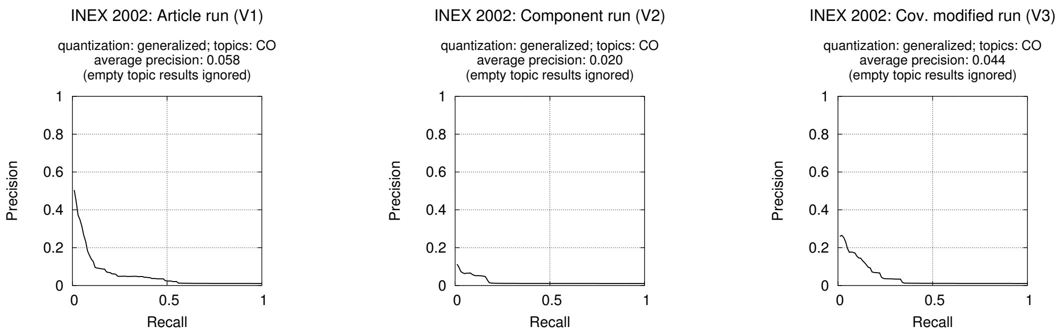
$$f_{strict}(r, c) = \begin{cases} 1 & \text{if 3E} \\ 0 & \text{otherwise} \end{cases}$$



(a) Scenario 1 (Article retrieval)

(b) Scenario 2 (Component retrieval)

(c) Scenario 3 (Cov. modified component retrieval)

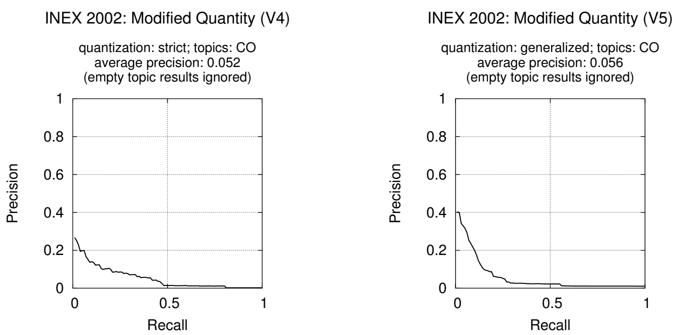


(d) Scenario 1 (Article retrieval)

(e) Scenario 2 (Component retrieval)

(f) Scenario 3 (Cov. modified component retrieval)

Figure 5: Recall - precision graphs for our experimentation scenarios, CO-topics only (first row: strict evaluation, second row: generalized evaluation).



(a) Additional run V_4 (strict), with preferred component size set at 2516 words

(b) Additional run V_5 (generalized), with preferred component size set at 5106 words

Figure 6: Recall-precision figures for additional runs 4 and 5.

The mapping function for generalized evaluation is:

$$f_{generalized}(r,c) = \begin{cases} 1.0 & \text{if 3E} \\ 0.75 & \text{if 3\{L,S\}, 2E} \\ 0.50 & \text{if 1E, 2L, 2S} \\ 0.25 & \text{if 1S, 1L} \\ 0.00 & \text{if 0N} \end{cases}$$

4.1 An Informal Analysis

A more detailed analysis of the evaluation results for all three runs showed us two observations that triggered our curiosity. The first observation was that 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 result set size). This small amount of relevant documents enables a 'perfect' retrieval system to retrieve all relevant documents in the result set, 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 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 with regard to the difference between 1) the number of relevant documents in the collection (for a given topic) and 2) the result set size. A possibility would be to use precision at various document cutoff levels, instead of precision at various levels of recall [12].

The second observation we made was the observation that, even with the strict evaluation that is most demanding coverage-wise, the article run (Figure 5a) still outperformed all other runs. We had expected that many article components would have been judged as too large. Examination of the judgements for the assessed CO-topics only[2] showed us the results in Tables 3 and 4. Note that the probability in the fourth column is not the probability of a node type being relevant *for all topics*, but the probability of a node type being relevant *for one of the assessed 25 CO-topics*. Both tables show that article-components have a much higher probability of being relevant for one of the CO-topics, when we would draw document components randomly from the collection. Knowing this, it is not surprising the article run performs very well.

Table 3: Top 5 of node types present in the judgements for the assessed 25 CO-topics only (strict evaluation function). The '*' denotes the any-element type.

Node type	**# relevant**	**# in collection**	**P(D)**
p	371	762.223	0.0004
article	**308**	**12.107**	**0.025**
sec	273	69.735	0.0039
ss1	111	61.492	0.0018
bdy	90	12.107	0.0074
*****	**1360**	**8239997**	**0.0001**

Table 4: Top 5 of node types present in the judgements for the assessed 25 CO-topics only (generalized evaluation function). The '*' denotes the any-element type.

Node type	**# relevant**	**# in collection**	**P(D)**
p	4198	762223	0.005
sec	2781	69735	0.039
article	**2606**	**12107**	**0.21**
bdy	1555	12107	0.12
ss1	1096	61492	0.017
*****	**18686**	**8239997**	**0.002**

We make one last remark regarding our second run, where each component was regarded as a document. The result sets of our second run were saturated with short document components. Looking at the language model used for estimating topical relevance, the cause of this saturation is clear: (query) terms occurring in short components will receive a higher weight than (query) terms occurring in longer components, resulting in higher overall rankings for short components. To remove this bias for short components, additional normalization will be necessary.

[2] At the time of writing this paper, 25 CO-topics had been assessed.

4.2 Preferred Component Length

In order to see whether our subjective guess of 500 words for acceptable document components was valid, we calculated the average length of relevant components (relevant according to the strict and generalized evaluation functions): 2516 terms (strict) and 5106 terms (generalized). We used these two means for updating the log-normal in two additional runs V_4 and V_5. The recall-precision graphs of these two additional runs are shown in Figures 6a and 6b, which also show that using the new averages does improve retrieval performance, but not radically. In short, using just document component length seems too naive for estimation of component coverage.

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. The experimental results and analysis of the assessments and additional fourth and fifth run showed us that using document component only is too naive an approach for estimation of component coverage.

Future work includes more extensive experimentation with the model described in this paper, especially in the area of relevance feedback and research into a fair normalization mechanism for removing the bias of the language model for short components.

Acknowledgements

Gabriella Kazai has greatly improved our understanding of the definitions of relevance and coverage in INEX.

References

[1] Inex relevance assessment guide. In N. Fuhr, N. Goevert, G. Kazai, and M. Lalmas, editors, *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX), Dagstuhl 9-11 Dec. 2002*, ERCIM Workshop Proceedings, March 2003.

[2] C.L. Barry. User-defined Relevance Criteria: An Exploratory Study. *Journal of the American Society for Information Science*, 45(3):149–159, 1994.

[3] 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.

[4] 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.

[5] 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.

[6] P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong. Comprehension Syntax. In *SIGMOD Record*, 1994.

[7] 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.

[8] 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.

[9] 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.

[10] T. Grust. Accelerating XPath Location Steps. In *Proceedings of the 21st ACM SIGMOD International Conference on Management of Data*, pages 109–120, 2002.

[11] D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, Twente, The Netherlands, 2000.

[12] 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.

[13] J. Jaakkola and P. Kilpelainen. Nested Text-Region Algebra. Technical Report C-1999-2, Department of Computer Science, University of Helsinki, 1999.

[14] 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.

[15] R.C. Miller. *Light-Weight Structured Text Processing*. PhD thesis, Computer Science Department, Carnegie-Mellon University, 2002.

[16] S. Mizarro. How Many Relevances in Information Retrieval? *Interacting With Computers*, 10(3):305–322, 1998.

[17] 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.

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 [12]) 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 [3]. 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 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.

With respect to 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. With weighted retrieval models, this may lead to inconsistent rankings 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/bdy/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,

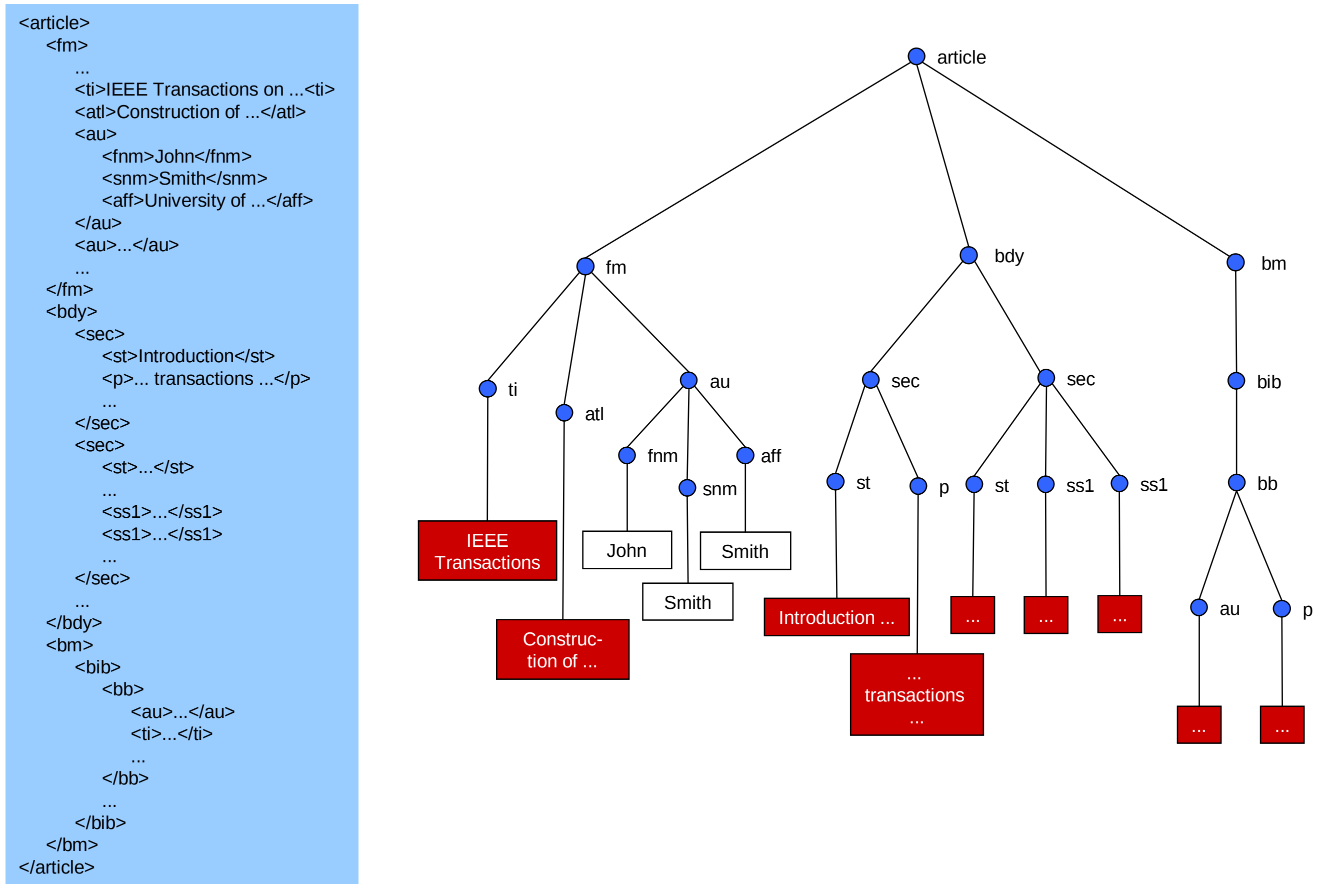


Figure 1: Sketch of an XML document from the INEX collection

the user expects the section paragraph to be ranked higher than the journal title element. However, conventional approaches to weighted and ranked 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 with 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 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 the experimental evaluation of PowerDB-XML within the INEX initiative. Section 6 covers related work, and Section 7 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 [13, 14]. 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 [5, 9]. PowerDB-XML takes over much of these ideas. We have also decided in favor of XPath because it is widely accepted 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, term weighting and relevance ranking are not available with XPath. Hence, our approach is to extend XPath with information retrieval functionality.

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 $<, \leq, =, \ldots$ 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 XML 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 $contains(expr, string) \rightarrow boolean$ 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 patameter.
- $contains(expr, query, irmodel, rsv, k) \rightarrow boolean$ 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 k 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) [3]. 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 with potentially different element types 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

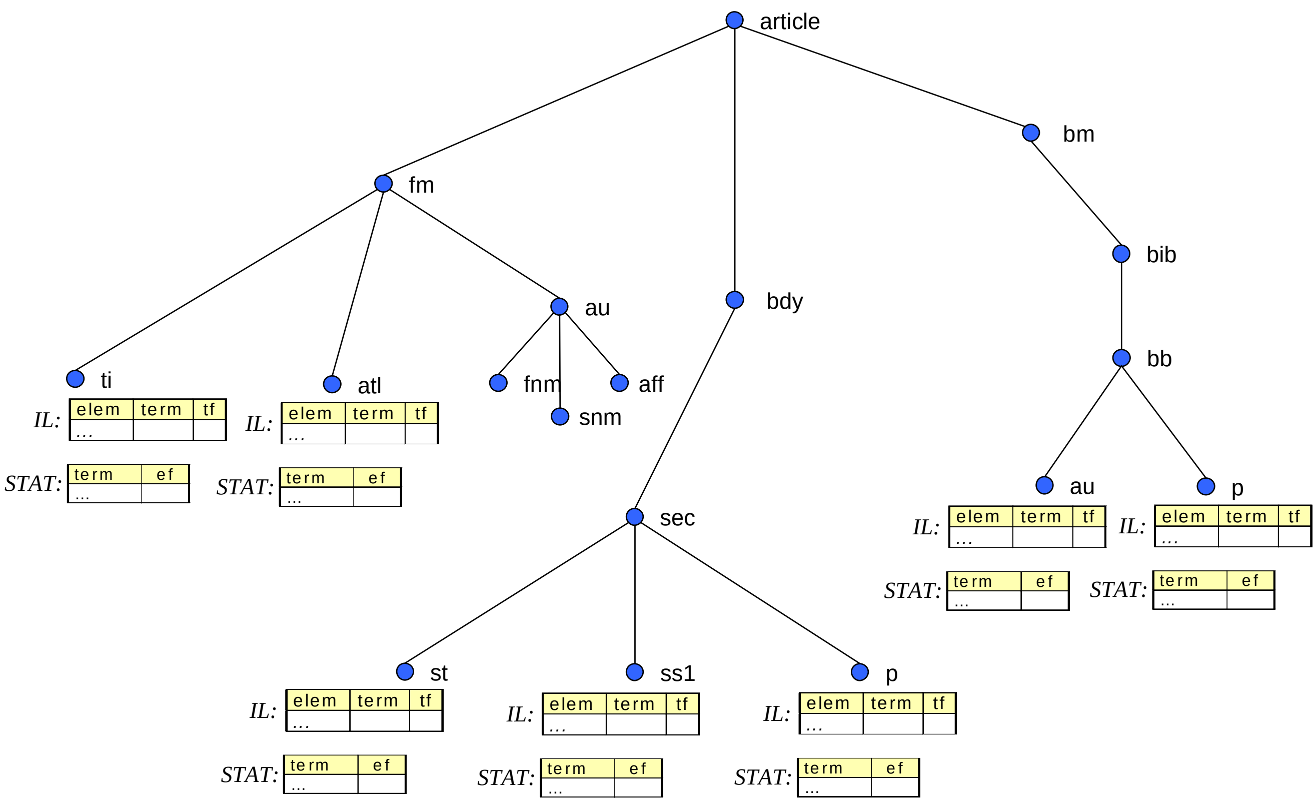


Figure 2: Example of basic indexing nodes for the INEX document collection

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 [8]. 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 which 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.

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

$$RSV(e,q) = \sum_{se \in SE(e)} \sum_{t \in terms(q)} tf(t,se) \Big(\prod_{l \in path(e,se)} aw_l \Big) ief_{cat(se)}(t)^2 \; tf(t,q)$$
$$= \sum_{se \in SE(e)} \Big(\Big(\prod_{l \in path(e,se)} aw_l \Big) \sum_{t \in terms(q)} tf(t,se) \; ief_{cat(se)}(t)^2 \; tf(t,q) \Big)$$

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

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 [6, 7]. We extend this notion for flexible XML retrieval 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. With multi-category retrieval, we define the

$$ief_{mcat}(t) = \log \frac{N_{mcat}}{\sum_{cat \in \mathcal{M}} ef_{cat}(t)}$$

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 subtress 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. [4, 5] tackle this issue by so-called *augmentation weights* which downweigh term weights when they are pushed upward in hierarchically structured documents such as XML documents.
- 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 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

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

  // 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

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

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 (cf. Figure 2). 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 the algorithm 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 dynamically 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. EVALUATION RESULTS

Experimental Setup. As outlined previously, our XML engine PowerDB-XML runs on top of a cluster of database systems. A cluster of database systems is a cluster of workstations interconnected by a standard network where each cluster node runs a commercially available database system. The PowerDB-XML middleware organizes distributed query processing over the nodes and integration of the results. Moreover, PowerDB-XML implements the different retrieval types discussed above, namely single-category, multi-category and nested retrieval for flexible retrieval from XML.

With INEX, we have used a cluster of 8 off-the-shelf PC nodes. Each node is equipped with one 400 MHz Pentium processor and an interface to switched duplex Ethernet with a data transmission rate of 100 Mbits/sec. Each

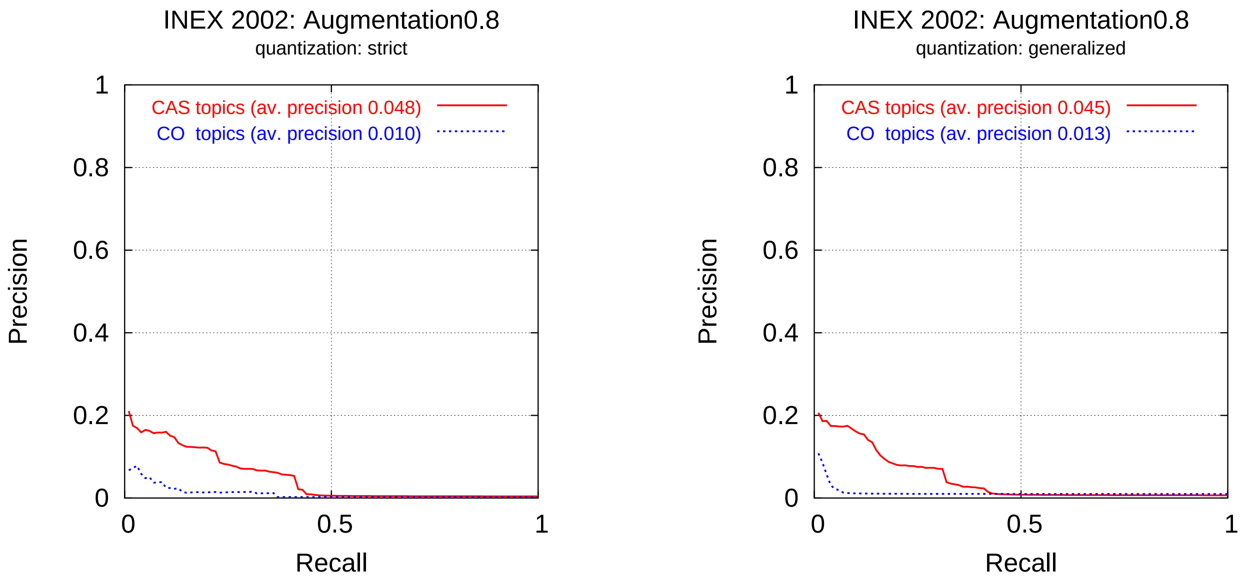


Figure 6: Evaluation results with PowerDB-XML: strict quantization (left) – generalized quantization (right)

node runs the Microsoft Windows 2000 Advanced Server operating system. The database system at each cluster node is Microsoft SQL Server 2000. The INEX document collection has been striped over all cluster nodes using hash partitioning over the (internal) document identifier. In other words, each cluster node i stores document texts, IR statistics, and index data of the XML documents assigned to node i. PowerDB-XML stores the original XML document text as a character-large-object, a model-mapping of the document using the EDGE approach [1], and the IR index and statistics data of the basic indexing nodes as described above using the relational database systems as storage managers. This yields a total database size of about 10 GB (accumulated over all cluster nodes) including database indexes. With the runs submitted to INEX, augmentation weights are 0.8.

Two different quantization functions have been applied to assess the retrieval results: *strict quantization* focuses on retrieving the highly relevant document components with exact coverage. In other words, the quantization of a document component is 1.0 for highly relevant components with exact coverage and 0.0 otherwise. *Generalized quantization* in turn also takes less relevant document components with less coverage into account and assigns them weights between 0.0 and 1.0.

Outcome and Discussion. Figure 6 shows precision/recall curves for the runs using the complete INEX XML document collection and all 60 INEX topics with the experimental setup of PowerDB-XML as outlined above. Figure 6 (left) shows the curves for strict quantization. Figure 6 (right) in turn graphs the outcome with the generalized quantization function. Both charts distinguish between CAS queries and CO queries. A first observation is that the level of the curves is less than with other text retrieval conferences such as TREC. This corresponds to a general result for all INEX participants and relates to the challenges of the semi-structured XML format which have not been considered by previous efforts on text retrieval. Another observation is that PowerDB-XML yields better retrieval quality with CAS queries than with CO queries. This corresponds to the observation for INEX results in general: retrieval performance of CAS queries is typically better than the one of CO queries. The reason for this is that the path expressions with CAS queries restrict the scope of retrieval to the target elements given by the query, and only target elements may qualify for the result. This is not the case with CO queries where an arbitrary document component may qualify as a result of a given query. The difficulty with CO queries therefore is to find the document component that is most specific and most relevant to the information need expressed by the query. This makes retrieval for CO queries more challenging than for CA queries, and the results with PowerDB-XML reflect this general difficulty.

6. 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 [2]. 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 [11]. 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 [5, 9].

Regarding IR statistics such as term frequencies (tf), Fuhr et al. have already argued in [4, 5] that treating documents as flat structures comes too short for XML. They propose to downweigh term weights by so-called augmentation weights when terms are propagated upwards in the document hierarchy. However, [5] derive IR statistics such as idf for the collection as a whole. But, retrieval in different contexts requires a more dynamic treatment of term weights. Hiemstra comes to a similar conclusion for query term weights used in different query contexts [10]. Therefore, our approach proposed in [8] 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.

7. 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 queries 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. The main objective of this future work is to compare retrieval quality with PowerDB-XML to other approaches which do not rely on computing IR statistics on-the-fly according to the scopes of the queries. Another important issue that warrants further investigation is retrieval quality on XML document collections with a semantically rich document structure.

8. REFERENCES

[1] D. Florescu and D. Kossmann. Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[2] D. Florescu, D. Kossmann, and I. Manolescu. Integrating Keyword Search into XML Query Processing. In *Proceedings of the International WWW Conference, Amsterdam, May 2000*. Elsevier, 2000.

[3] 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.

[4] 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.

[5] 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.

[6] T. Grabs, K. Böhm, and H.-J. Schek. PowerDB-IR - Scalable Information Retrieval and Storage with a Cluster of Databases. *To appear in: Knowledge and Information Systems*.

[7] T. Grabs, K. Böhm, and H.-J. Schek. PowerDB-IR – Information Retrieval on Top of a Database Cluster. In *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM2001), November 5-10, 2001 Atlanta, GA, USA*, pages 411–418. ACM Press, 2001.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. `http://www.w3.org/TR/1998/REC-xml-19980210`, Feb. 1998.

[13] World Wide Web Consortium. XML Path Language (XPath) Version 1.0. `http://www.w3.org/TR/xpath`, Nov. 1999.

[14] World Wide Web Consortium. XQuery 1.0: An XML Query Language. `http://www.w3.org/TR/xquery`, Nov. 2002.

Bayesian Networks and INEX

Benjamin Piwowarski
LIP 6, Paris, France
bpiwowar@poleia.lip6.fr

Georges-Etienne Faure
LIP 6, Paris, France
faure@poleia.lip6.fr

Patrick Gallinari
LIP 6, Paris, France
gallinar@poleia.lip6.fr

Abstract

We present a bayesian framework for XML document retrieval. This framework allows us to consider content only and content and structure queries. We perform the retrieval task using inference in our network. Our model can adapt to a specific corpora through 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* (CAS) queries. 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 as a *focused retrieval*, first described in [5, 13].

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

2 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 (figure 1), 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 "corpus" variable, its children the "journal collection" variables, etc. In this model, due to the conditional independence property of the BN variables, relevance is a local property in the following sense: if we know that the journal is (not) relevant, the relevance value of the journal collection will not bring any new information on the relevance of one article of this journal.

Three different models were considered.

Model I A simple model that computes a score for each element. Its only parameters are statistics on words contained in this element and in its parent.

The other two models correspond to two different sets of values $\mathcal{S}$ for the BN variables:

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 Chiaramella 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 a structural element (document, body, section, paragraph and so on), p its parent and q the query. 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 and b (among R, G, S, I according to model II or III) of the element e and of its parent: con

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

where $F_{rel}^{\diamondsuit}$ 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 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 a 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. αs, βs and γs are to be learnt by the BN.

This model operates in three modes, *training*, CO and CAS *retrieval*, which we now describe.

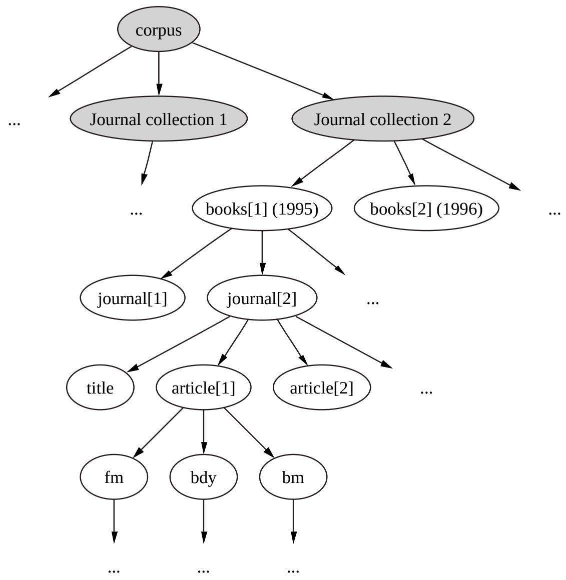


Figure 1: The document collection: each structured document is located in a specific part of the hierarchically organized collection. Here, each document is a collection of journals, each journal contains structured articles. The query q is added to this network while retrieving or learning. Below article[1], 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.

2.1 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 e_i. $P(e_i = R|q)$ can be computed using dynamic programming methods. We begin at the top of the hierarchy and use recursion to compute RSV (Retrieval Status Value) for each e_i:

$$P(e_i = .|q) = \sum_{p \in \{I,R,G[,S]\}} P(e_i = .|q, \text{parent}_i = p)$$

The score of one element is then given by $RSV(e_i, q) = P(e_i = R|q)$. Elements with highest values are then presented to the user.

2.2 Retrieval with CAS queries

INEX queries were composed of different parts (target element, relative context element and absolute context elements) or *subquery needs*. For CAS retrieval, we extend our bayesian network to handle multiple subqueries and use one sub-network for each one. Those networks are then connected in order to form one large network that represents the whole CAS query.

Example In order to describe CAS query processing, we make use of an example (figure 2). Each CAS query is first decomposed into elementary subqueries (here Q_0, Q_1 and Q_2). Each of those subqueries refers to a structural entity and an information need. Each information need is modeled by a BN constructed as for CO queries.

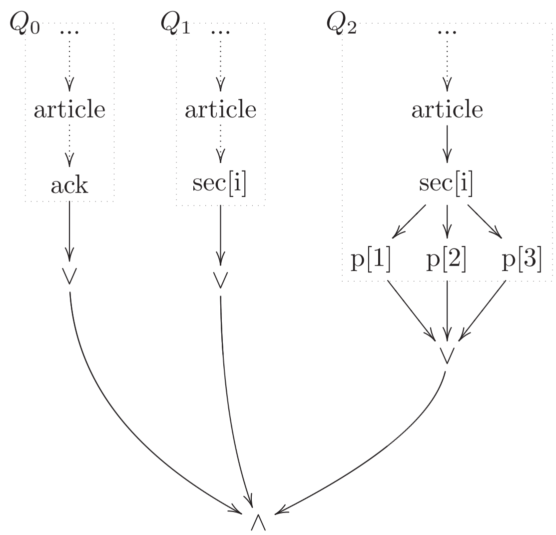


Figure 2: An example of BN for a CAS query: retrieval of *sections on information retrieval* (Q_1) in an article with an *acknowledgment referring to INEX* (Q_0). The section must have *paragraphs on XML retrieval*. The article must contain an acknowledgment (`ack`) relevant to query Q_0. This is an *absolute* context element, it does not depend on the section but on the document. The *retrieved* section (*target element*) must be relevant to query Q_1. This section has paragraphs relevant to query Q_2. Those paragraphs are *relative* context element as they change for every target element (for every section). Here only the network part involved in the relevance scoring of one `section` element is shown.

Those elementary BNs are then connected for each target element in order to give this element a global score. Two different subquery type were distinguished:

1. Absolute subqueries that were relative to the article element (Q_0);
2. Relative subqueries that were relative to the section element (Q_1 and Q_2).

Relative subqueries networks are constructed after finding a target element (here `sec[i]`) while absolute subqueries network are constructed for each document[1].

General algorithm 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:

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

and for $\vee$ nodes we have:

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

In order to compute the score for one target element e_i, we follow the following steps:

- For each target element e_i and for each subquery Q_j, let $ce(i,j,1),\ldots,ce(i,j,n_{i,j})$ be the context element fulfilling structural constraints (e.g. in figure 2, when e_i is `section[i]`, $ce(i,0,1)$ is `ack`, $ce(i,1,1)$ is `section[i]`, $ce(i,2,1)$ is `p[1]`, $ce(i,2,2)$ is `p[2]` and $ce(i,2,3)$ is `p[3]`).
- Compute the j^{th} subquery score $RSV_{q_j}(e_i,q)$ for the element e_i:

$$\begin{aligned} RSV_{q_j}(e_i,q) &= 1 - \\ &\quad (1 - RSV(ce(i,j,1),q_j)) \\ &\quad \times \ldots \\ &\quad \times (1 - RSV(ce(i,j,n_{i,j}),q_j)) \end{aligned}$$

  Note that when there is only one context element (like `ack` for subquery Q_0), this subquery score is reduced to $RSV_{ce(1,j,1),q_j}$.
- Compute the global score for element e_i: $RSV(e_i,q) = RSV_{q_1}(e_i,q) \times \ldots \times RSV_{q_n}(e_i,q)$.

2.3 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 Θ with respect to the likelihood $\mathcal{L}$ of the observed data :

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

[1] In INEX, documents were everything below the `article` tag

where $O = \{O^{(1)}, \ldots, O^{(|O|)}\}$ 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 into $E^{(i)}$ and $H^{(i)}$ where $E^{(i)}$ corresponds to structural entities for which we know whether they are relevant or not, i.e. structural parts for which we have a relevance assessment. $E^{(i)}$ is called the evidence. $H^{(i)}$ corresponds to hidden observations, i.e. all other nodes of the BN.

In our experiment, we used for learning about 200 assessments from CO queries that were obtained by taking only the browse keywords of CAS queries.

3 Related works

In this section, we make a short review of previous works in IR related structured retrieval and 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 Inquery [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 Inquery 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. 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.

4 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.

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. Zhoo. 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.

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

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 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. In section 7 we discuss the results and draw conclusions.

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 $\boldsymbol{k}$ characters is regarded as consisting of $\boldsymbol{k}$ 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 $\boldsymbol{n}$ characters the total number of inverted lists for the file is $\boldsymbol{k} * \boldsymbol{n}$. 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 at INEX 2002.

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) |
…
(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 it's 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*. Therefore, the size of the bitmaps file would be the same size as the original data file. This result is not surprising; the bitmaps represent a lossless transformation of the data file itself and contain exactly the same information.

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.

4.3 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 run-time 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 all EFI queries for a given topic correspond to raw XML text lines in the articles. It is necessary to combine all the topic's 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:

- Each query's score is computed as the inverse of the number of lines that it matches. More selective queries are thereby more heavily weighted.
- Query scores are totaled for each document to produce its rank. Note that documents may have many scored lines.
- Documents are ranked in descending order. The highest ranked document is that matching more of the (combined score) queries than any other document.
- Two variations to weighting were tested. One set of results was produced whereby queries that were generated from the <Title> element of a topic were weighted 100 times more heavily than queries generated from the <Keywords> element of a topic. This effectively implies that <Ttitle> terms dominate the ranking and <keywords> terms are only used to fine tune the ranking, or where less than 100 documents are selected by <title> queries. A second set of results was based on equal weight queries.
- Our system did not identify target elements. Retrieval was at the document level. It is possible of course to identify and extract target elements after document identification, but this functionality was not implemented. The system therefore always returned the **/article[1]** element, for both the CAS and CO topics.

6. Experimental Results

The EFI system was tested on both CAS and CO elements. Two results files were submitted. In the first, queries generated from the <Title> element were more heavily weighted. In the second, the <Title> queries and <Keywords> queries had equal weight.

Result files were generated for both the CAS and CO topics, but in both cases the returned elements were the /article[1] elements.

6.1 Content Only topics

The best results were obtained with the CO queries and with strict quantization. Assigning equal weight to queries from <Title> and <Keywords> of a topic produced better results (Figure 2). The EFI results were ranked **4th** (equal weight <Title> and <Keyword>) and **24th** (<Title> weighted 100 times more than <Keywords> (Figure 1). Clearly, the <Keywords> elements of topics were significant in selecting and ranking documents.

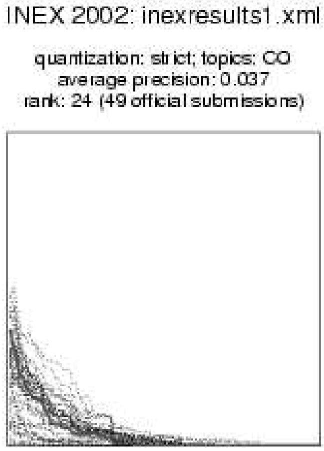


Figure 1: EFI retrieval for Content Only topics, higher weight to <Title> than to <Keywords> elements of topics.

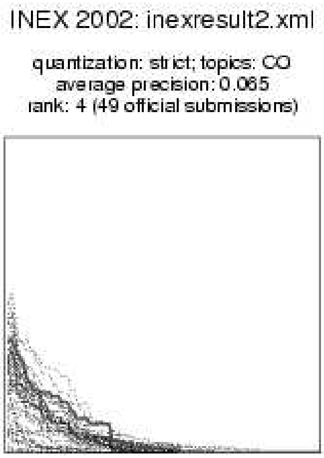


Figure 2: EFI retrieval for Content Only topics, equal weight to <Title> and <Keywords> elements of topics.

6.2 Content and Structure topics

CAS topics performance is difficult to judge from the Precision-Recall curves of the CAS topics because assessment was done against the <te> of topics while the EFI system returned entire documents, ignoring the <te> element.

Nevertheless, it is evident (from inspecting the actual relevance assessments), that the EFI system was just as effective with CAS topics in selecting the right documents. It is deficient in that it did not extract elements from within the selected articles.

7. Discussion

There are two aspects of performance which are note worthy. The first is the small size of the system and the other is its functionality. The size of the executable file inversion component of EFI is 45KB. The size of the EFI search engine is 62KB. These are extraordinary small files considering the magnitude of the task at hand. The index file (bitmaps) is only about the size of the data file itself (600MB) so the indexing overhead is about 1:1. A single query is usually evaluated within a few seconds, on a PC running at 1.2GHz clock speed. The topics, employing multiple queries, were evaluated in about 2 minutes each. The system is fast enough and small enough to run on a stand-alone PC as a console application (under either Windows or UNIX) and requires no database system to support it.

Future work will look at post-processing of selected documents to zoom in on the components which are most relevant to the topic, or are explicitly required in the <te> element of a topic.

In terms of functionality the EFI system was surprisingly effective in tackling the problem of document retrieval. It is surprising because of the brute-force approach that was adopted – the entire set of about 12,000 articles was converted (arbitrarily) into a set of about 11,000,000 lines of about 500 characters each. The XML tags were left intact, and indexing was performed at the single character level. There was no attempt at using XML knowledge (e.g the DTD) in the solution design process. Queries were constructed to search for both the keywords and the XML tags in the large set of text lines. Simple heuristics were used to rank documents.

The system performance could be improved if a more disciplined approach was taken to structuring document fragments. The arbitrary split of documents to lines of 500 characters was far from optimal and was merely imposed by resource constraints (mostly time) that we had to work with. Future work will also look at a more suitable representation to enable exact selection of XML elements.

REFERENCES

[1] Geva, S., "Implementing a Software Associative Memory", Thesis (1987), Queensland University of Technology (QUT), Australia

Integration of IR into an XML Database

Cong Yu
Department of EECS
University of Michigan
congy@eecs.umich.edu

Hong Qi
School of Information
University of Michigan
hqi@umich.edu

H. V. Jagadish
Department of EECS
University of Michigan
jag@eecs.umich.edu

ABSTRACT

Structure matching has been the focus and strength of standard XML querying. However, textual content is still an essential component of XML data. It is therefore important to extend the standard XML database engine to allow for "Information Retrieval" style queries, namely, "keyword" based retrieval and "result ranking". In this paper, we describe our effort in integrating information retrieval techniques into the Timber XML database system being developed at the University of Michigan, and our participation in the *IN*itiative for the *E*valuation of *X*ML Retrieval (INEX).

1 Introduction

With the growing popularity of XML, it is expected that more and more information will be stored and exchanged in XML format. Part of the information will be contained in the structure of the document. Another part, however, will be contained in a textual format within the elements (i.e, document components) of the XML documents. While boolean style querying is useful in some circumstances, there is a growing demand for querying both the textual information and the structure information in a non-boolean way. There are two general approaches to this problem. One is to start with a traditional IR system and augment it with the ability to recognize and extract the document structure. The other approach is to integrate IR facilities for querying textual content into a standard XML database engine, which handles structured queries well. We follow the second, database-oriented, approach, starting with Timber [13], a native XML database we have been developing.

There are four main challenges to this database-oriented approach. First, how to fit keyword based retrieval of the document components into the pipelined query evaluation of the database engine. Second, how to efficiently calculate the score of the matching elements to allow for future ranking of those elements. We developed `PhraseFinder` and `TermJoin` algorithms [3] to address both issues. The `PhraseFinder` algorithm uses a sort-merge based method to allow for pipelined retrieval of elements containing specified phrases (e.g., "information retrieval", instead of "information" and "retrieval"). The `TermJoin` algorithm is a stack-based algorithm that allows efficient retrieval of elements, at multiple granularities, that have a non-zero score according to a user-defined score function.

Third, how to aggregate the query results (i.e., a set of document components at different levels) such that users are not presented with redundant information, especially when they do not specify which type of elements to return (e.g., a content-only query). To address this so-called result redundancy issue, we have been working on the `Pick` algorithm which scrutinizes the result set according to a user-defined pick function and eliminates redundant elements using a stack based strategy.

Yet another main challenge in integrating IR into XML query is the specification of the query. We have devised a bulk algebra, TIX, for query *T*ext *I*n *X*ML, and several extensions to the XQuery language that give a framework on how IR style queries can be expressed at both the algebra level and the language level. Both TIX and the XQuery extension are out of the scope of this paper, interested readers are encouraged to take a look at [3].

The rest of the paper is organized as follows. Section 2 describes the Timber system and how it deals with structured queries. Section 3 describes the extensions to Timber that make the evaluation of IR style XML queries in Timber possible. In Section 4, we report our experience in using the Timber system to answer the set of INEX queries. Finally, we conclude in Section 5.

2 The Timber System

Timber [13] is a native database system currently being developed at the University of Michigan. The objective of the Timber system is to build an efficient database engine for storing and querying XML data. It is based upon the TAX (*T*ree *A*lgebra in *X*ML) algebra [14] as its theoretical foundation for manipulating tree structures. Several access methods have been developed to retrieve the natively stored XML elements and a comprehensive pipelined query processing engine is implemented in the system to evaluate queries in the XML context.

The overall system architecture of Timber is illustrated in Figure 1. The whole system is built on top of the Shore object-oriented storage manager [20] (we are also developing another version that is built on top of the Berkeley DB backend store [7]), which is responsible for buffer management and concurrency control. The rest of Timber is composed of several components. XML documents are first parsed by the Data Parser to produce parse trees as inputs to the Data Manager. The Data Manager then transforms the nodes of those parse trees into an internal representation and stores them into the Storage Manager. Index Manager and Metadata Manager, as their name suggest, builds indices on the data and stores statistics about the data, respectively. At the heart of Timber is the Query Evaluator. It executes *evaluation plan*, which is produced by the Query Parser and optimized by the Query Optimizer, by interacting with the Data Manager and Index Manager. The details of the Timber system can be found in [13]. In particular, the attributes of an element are combined together and stored as a child element to the original element. Similarly, the textual content of an element is also represented as a child element to the original element. Therefore, nodes stored in Timber can be mainly classified into three types: element, text, and attribute, each has a slightly different format.

Structural queries can be efficiently processed by Timber. Each node in an XML document is represented by a triple *(startkey, endkey, level)*, where the *startkey* uniquely identifies the node in the database. In the case of multiple documents, the *startkey* of nodes in subsequent documents are incremented by an *offset* to make them unique in the entire database. A very important property of this coding scheme is that all the descendent nodes of a particular node n will have a *startkey* larger than $n_{startkey}$ and an *endkey* smaller than n_{endkey}. With this property, whether two nodes fit the ancestor/descendent or parent/child relationship can be determined in constant time by examining the two triples. It allows for efficient processing of structural joins (i.e., containment queries) using a stack based algorithm [2].

The Query Evaluator currently is able to process most of the XQuery expressed in the format of

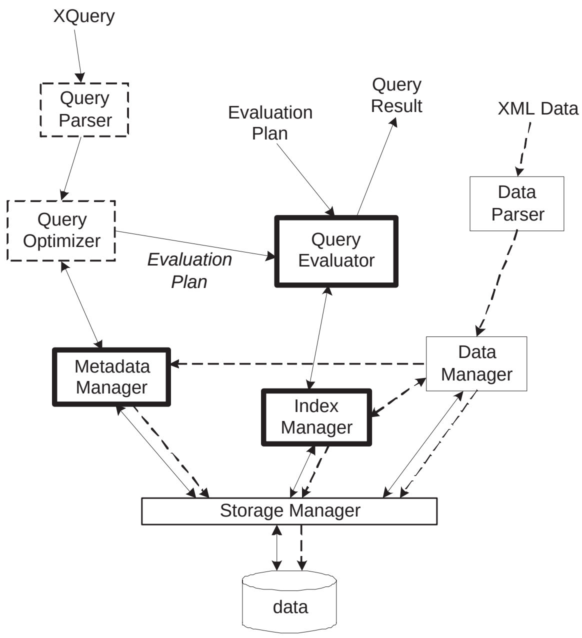


Figure 1: Timber System Architecture. Dashed lines represent loading data flow, solid lines represent retrieval data flow. Dashed rectangles represent components not used in INEX, solid rectangles represent components in Timber being used in INEX unmodified, bold solid rectangles represent components modified for integration of the IR extensions.

Timber Evaluation Plan, while the Query Parser and Query Optimizer can handle a smaller subset of the XQuery. In participating in the INEX, we have primarily used the *Evaluation Plan* interface instead of the XQuery interface because the XQuery interface was still under development at that time. However, in the future, we expect to be primarily using the XQuery interface so as to utilize the automatic query optimization provided by the Query Optimizer.

3 IR Extensions of Timber

To allow efficient processing of IR style query on the XML data, several components of the Timber system need to be extended. First, an IR-style inverted index is required to process keyword based search. Second, some extra information (e.g., how many words a text element contains) needs to be maintained by the Metadata Manager. Third, a score function needs to be integrated into the Query Evaluator to calculate relevance scores (i.e., return status value, *rsv*) to the matching elements. Fourth, an extra module is needed for eliminating redundant nodes in the final output set in the case when the user does not specify the type of elements to be returned. We describe the first two

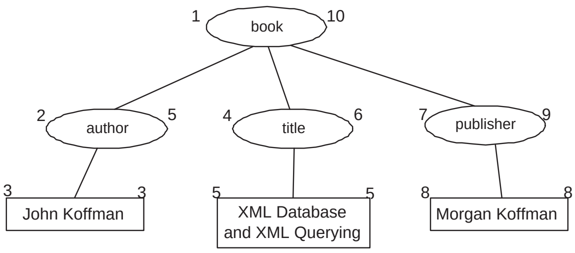


Figure 2: A Simple XML Document: ellipse indicate element nodes and rectangle indicate text nodes. The numbers on the shoulders of each node are the startkeys and endkeys respectively. The startkey and endkey for a text node are the same because it does not contain any child nodes.

keyword	(startkey, level, offset)
john	(3, 3, 0)
koffman	(3, 3, 1), (8, 3, 1)
xml	(5, 3, 0), (5, 3, 3)
database	(5, 3, 1)
query	(5, 3, 4)
morgan	(8, 3, 0)

Table 1: Sample Inverted Index Entries Based on Figure 2. Note that only startkey is needed since for text nodes, startkey is the same as the endkey.

extensions in this section and the latter two in Section 4.

3.1 Indexing INEX Data

Indices have been an integral part of Timber from the very beginning. Timber maintains several major types of indices. The most important ones include: 1) element tagname index, which maps a string s_1 to a set of element nodes (in the form of *(startkey, endkey, level)* triples) with tagname equal to s_1; 2) attribute name index, which maps a string s_2 to a set of element nodes that contain an attribute with the name equal to s_2; 3) attribute content index built on the element nodes with a specific attribute *attr*, which maps either a string s_3 or a number n_3 (floating-point number or integer number) to a set of element nodes that contain the *attr* attribute and have a value of s_3 or n_3 in *attr*; 4) element content index, which maps either a string s_4 or a number n_4 to a set of text nodes that have s_4 or n_4 as their value.

The aforementioned indices, however, are inadequate in supporting keyword based searches as required by IR style queries. An IR style query usually asks for document components related to a certain topic, which is frequently described as a set of terms (keywords) that, in the user's view, best capture the concept of the topic. Therefore, the results of an IR query are those elements that have the related terms in their textual content. The frequency and position of the term occurrences indicate the relevance of the element to the query.

To allow fast retrieval of elements that contain certain keywords, we extended the Timber Index Manager to include an inverted index on the text nodes. The index structure maps a word to the set of text nodes that contain the word. It also keeps track of the word offset in the textual content to allow matching of phrases (being used by `PhraseFinder`). Using the simple XML document in Figure 2 as an example, a total of six entries will be added to the inverted index, which are listed in Table 1.

A few strategies are employed to reduce the space requirement of the inverted index and to improve its accuracy. First, we compiled a list of most frequent used words in the INEX data set. Based on this list, we generated a list of 322 stop words for which we do not index, thereby reducing the index size significantly. Second, we do stemming of the words to index only the original form of the word (e.g., "query" instead of "querying"). The first stemming strategy we tried was the Porter's algorithm [16]. However, we found that Porter's algorithm is sometimes too aggressive (i.e., changing a word into its root instead of its original form). Instead, we decided to use WordNet's dictionary [22] to search for the original form of a word. This increases the time to build the inverted index, but it has the advantage of being more reliable than the Porter's algorithm.

3.2 Indexing Metadata

Another important extension is to the Timber Metadata Manager. As we will see in Section 4.2, for each node to be scored by the score function, not only the keyword occurrences in its textual content are needed, but also some extra statistics (metadata) related to the node in the context of the database. We have kept two main pieces of metadata information. First, the number of child nodes an element node has. Second, the total number of words a text node contains. Both are created by the Metadata Manager at the time the INEX data is loaded into the Timber. They can also be created after the loading in one pass over the entire database. We call them metadata indices to distinguish them from the normal indices that are maintained and accessed through the Index Manager. As a special variation to the first metadata index, we also maintain a separate index for <article> and <sec> element nodes that is tuned for the INEX data. For an <article> node, we index how many <sec> nodes in its entire subtree, and for <sec> nodes, we index how many <p> and <p1> nodes in its subtree. This special variation is employed because we have discovered that <article>s, <sec>s, and <p>s are frequently the most reasonable return units in response to an INEX query. Having this special index can significantly reduce the time it takes to perform the redundancy elimination as described in Section 4.3.

3.3 Integration of Scoring and Redundancy Elimination into Query Evaluator

Scoring of each matched node is integrated into the query evaluation engine. Adopting a tree structured view toward XML document, the score function in our framework maintains a *localization* property: i.e., all the information needed by the score function in order to determine the score of a particular node is contained in the subtree rooted at that node or can be obtained via the Metadata Manager using one of the metadata indices. This allows the scoring of each node to be pipelined using the stack based `TermJoin` algorithm and therefore integrated into the evaluation engine (discussed in Section 4).

The coverage issue as highlighted in the INEX result assessment documentation [12] has two important aspects. First, there will be nodes covering only a subset of the information content being queried, so called *small coverage* or *partial coverage*. Second, there will be nodes covering information content not related to the query, so called *large coverage*. It is notable that the two aspects are orthogonal, i.e., a node can be covering only a subset of the requested information while having some information not related to the query. A node with a *small coverage* can be penalized by engineering the score function so that nodes with *complete coverage* or *full coverage* are assigned a higher score even though the absolute volume of information they contain is less. A node with a *large coverage* can be penalized by taking the size of the node into consideration in the score function. However, the introduction of structures in XML poses problems to this approach of attacking the *large coverage* problem. Imagine a query that matches a node with five child nodes, four of them are relatively small but not related to the query, only the relatively large child is related. Both the parent node and the related child node are likely to be returned to the user. However, the parent node should not because all the information it can give to the user is present in its child node. There are also cases where the parent node instead of the child nodes should be returned (see Section 4.3). We call this the result redundancy issue.

We utilize the pick function, which implements the `Pick` algorithm [3], to address the result redundancy issue. It is added as a module at the end of each query evaluation. The input to this module is a set of scored nodes. User defined criteria are employed by the module to select those nodes at the appropriate granularity level. Metadata indices are also being used to help remove the redundancy. A default selection criterion is always provided in case user does not provide one. The output of the module is a set of nodes at, hopefully, the right granularity level.

The detailed description of both score function and pick function can be found in [3] and in the following section.

4 IR Query Evaluation

There are three phases in processing each INEX topic. First, the topic is translated into an *evaluation plan* that the Query Evaluator can understand. Second, the plan is executed to produce a set of nodes, each with a score indicating how relevant it is to the query. Third, in the case of a content-only query, a final result redundancy elimination procedure is performed. The final result can then be sorted and a certain number of the top results are returned to the user. Throughout this section, we will use the two query topics in Figure 3 as our running examples.

4.1 Topic Translation

The topic translation accomplishes two important tasks: one is translating the XPath expression in the original topic into *evaluation plan*; the other is categorizing keywords into several classes to be used by the score function. The first task can be accomplished relatively easily because Timber already supports most of the XQuery, a superset of XPath. Essentially, each "/" is translated into a parent/child join, each "//" is translated into an ancestor/descendent join, and each <cw>/<ce> pair is translated into a term join. Figure 4 shows the *evaluation plan* of Topic 12 (a content-and-structure query) in its tree format. The ellipse nodes labeled with tag names are retrieved via the *element tagname* index. The rectangle nodes represent text nodes retrieved via the *inverted index*. The content in those nodes dictates how they should be scored by the score function and is explained later in this section. Finally, the edges between two nodes indicate the join algorithms being used to fetch them, including parent/child join

```
<INEX-Topic topic-id="12" query-type="CAS" ct-no="069">
   <Title>
      <te>article/bdy/sec</te>
      <cw>2001 2002</cw><ce>article//pdt/yr</ce>
      <cw>internet search engine</cw>
      <ce>article/bdy/sec</ce>
   </Title>
   <Description>
      Retrieve sections of articles published in 2001
      or 2002 on the topic of internet search engines.
   </Description>
   <Narrative>
      To be relevant, the article should talk about
      the current status of internet search engines,
      problems associated with current technologies,
      and future developments.
   </Narrative>
   <Keywords>
      internet search engine information retrieval
   </Keywords>
</INEX-Topic>

<INEX-Topic topic-id="31" query-type="CO" 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>
   <Narrative>
      To be relevant, a document/component must either
      talk in general terms about the opportunities
      at the intersection of computer science and biology,
      or describe a particular problem and the ways it is
      being attacked.
   </Narrative>
   <Keywords>
      computational biology, bioinformatics, genome,
      genomics, proteomics, sequencing, protein folding
   </Keywords>
</INEX-Topic>
```

Figure 3: Two Example INEX Query Topics

(*PC Join*), ancestor/descendant join (*AD Join*), and the IR specific *Term Join*. The translation of a content-only query (e.g., Topic 31) to the *evaluation plan* is also easy. As shown in Figure 4, it involves term-joining matching text nodes with all their ancestor element nodes regardless of the tag name. The actual *evaluation plan* is in text format and is omitted here.

The second task is to separate the set of keywords provided in the <Keywords> part of the topic into three categories: REQ, HIGH, and LOW. A keyword appears in the REQ category if it is listed in the <Title>/<cw> part of the original topic. A keyword appears in the HIGH category if: 1) it is not in the REQ category and 2) it appears in the <Description> or <Narrative> part of the topic. Finally, a keyword is in the LOW category if it appears only in the <Keywords> part. Keywords falling into different categories will have different weights in contributing to the overall score of the element. More importantly, a node with all the REQ keywords is always assigned a higher score than one missing some of the REQ keywords, regardless of the other two categories. This means a node with *full coverage* of the information is always ranked higher than a node with *partial coverage*. In addition to categorizing keywords, we also try to identify keyword phrases by scanning through the <Description> and <Narrative> parts of the topic. A keyword phrase is identified if two or more consecutive words occurring in the <Keywords> part also occur in <Description> and <Narrative> parts in the same consecutive order. For example, the phrase “internet search engine” can be identified in Topic 12. It is worth mentioning that in some topics, the author specifies the phrases in certain format (e.g, Topic 31), which means this extra phrase identification step is not required in all topics. We do find that phrase identification has a very significant impact on the result accuracy, which suggests that some better mechanism for specifying or identifying phrases in the set of provided keywords is worth further investigation. The search engine *www.alltheweb.com* has made some efforts in this direction. It is also worth noting that some <cw>s are actually exact boolean matches rather than keyword based matches (e.g., Topic 12 specifies that the publication year of the article must be 2001 or 2002). However, there seems to be no easy way for the topic translation script to automatically recognize this. We therefore may have to manually notify the score function of this. The result of this keyword categorization is the content in the rectangle text nodes as shown in Figure 4.

The *evaluation plan* is then provided to the Query Evaluator for execution and the result of keyword categorization is supplied to the score function inside the Query Evaluator, which uses it to calculate scores for each matched node.

4.2 Score Generation

Score generation is accomplished by the score function inside the Query Evaluator. For INEX, we use a default score function based on the following formula:

$$score = \sum_{j=req,high,low} \left(\frac{W_j}{N_j} \sum_{i=1}^{N_j} \frac{\log N_{keyword_i}}{size_{node}}\right) \quad (1)$$

where W_j is the weight assigned to one of the three categories, N_j is the total number of keywords (a phrase is counted as one keyword) in that category, $N_{keyword_i}$ is the number of occurrences of a certain keyword in the current node, and finally, $size_{node}$ is the total number of words in the current node. For INEX, we have used

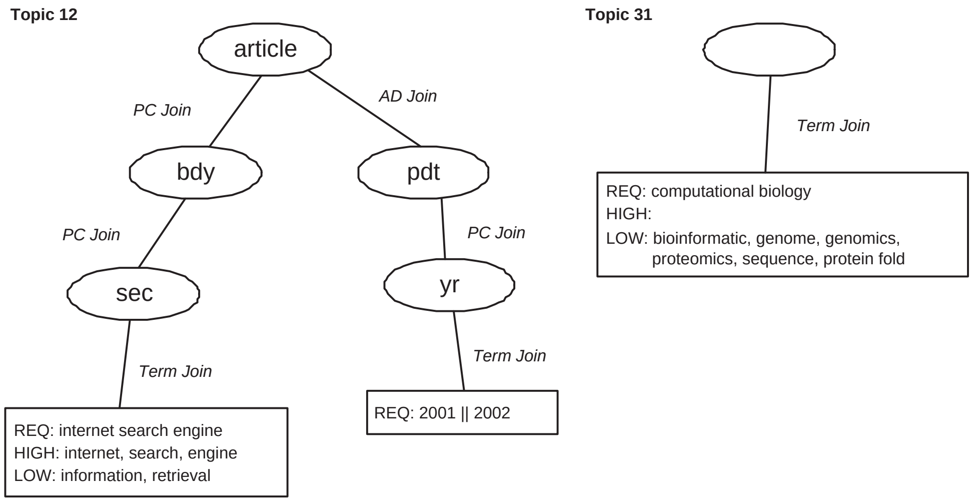


Figure 4: The *evaluation plan* of Topic 12 and Topic 31 in Tree Format.

$W_{req} = 0.6$, $W_{high} = 0.25$, and $W_{low} = 0.15$. As mentioned before, for nodes with all the REQ keywords, we add to it a constant that is large enough to make its score higher than any node without all the REQ keywords.

For a content-and-structure query, only the element nodes satisfying the structure requirement are processed by the score function. For a content-only query, all the element nodes that are ancestors of a text node, containing at least one of the keywords, are processed. For certain content-and-structure queries with, for example, <au> as the target element, we recognize that the real intention of the user is to rank <article>s while retrieving <au>s of the matched <article>s. In those cases, the <article> elements are processed by the score function instead of the <au> elements.

4.3 Redundancy Elimination

For content-only queries, the set of results produced by the score function can contain many redundant information. For example, it is possible that an <article> and all its five <sec>s have high scores. But the user should really be only presented with the <article> since that is the one that contains all the information. The example in Section 3.3 also illustrates the case where a child component, rather than the parent component, should be returned.

This redundancy elimination is accomplished by the pick function. For INEX, we have employed a special pick function that operates only on three major types of element nodes: <article>, <sec>, and <p>. The basic idea is as follow. To decide whether to return an <article> or a <sec> underneath it, we check how many <sec>s under that <article> are relevant (a node is considered relevant if it has a score that ranks it in top 500 when all the nodes being processed by the score function are considered). If above a certain percentage (we default it to 50%) of the <sec>s (among all the <sec>s underneath that <article>) are relevant, the <article> is picked and the <sec>s are discarded. Otherwise, the individual <sec>s are returned. Nodes of other types fall into two categories: one that is underneath an <article>, the other that is not. Nodes in the first category are discarded since the information contained in them is captured in one of the above three types. Nodes in the second category are kept because the information within them can not be captured in any of the above three types.

After redundancy elimination, all the remaining nodes are sorted and the top 100 are then returned back to the user.

4.4 Performance

We briefly discuss the performance of our system in terms of two measurements: storage space and querying time.

The entire INEX data occupies about 5GB (a roughly ten-time increase from the original data) disk space to accommodate both the raw data and the extra structural information needed (e.g., *startkey*, *endkey*, etc.). All the auxiliary indices (e.g., element tag index) are quite small. The only exception is the inverted index, which is considerably large compared to the other indices. The problem is made worse due to the fact that we are using GiST [10] as our physical level index manager, which leaves us no control over how things are organized on the disk. We are currently investigating ways to control the size of the inverted index without loss of efficiency.

Once the INEX topic gets translated into the *evaluation plan*, its execution time depends on how frequent the keywords are in the data set and how many structure constraints are in the query. The complexity of `TermJoin` is $O(\sum(T_i))$, where T_i is the number of how many times keyword T_i occurs in the data set. Therefore, the more frequent the keywords are, the longer it takes to evaluate the query. Structural constraints also plays an important role here because the Query Evaluator can quickly discard elements that do not satisfy the structural condition without trying to score them. On average, content-and-structure queries can be evaluated within a few seconds of CPU time. While content-only queries can take from several seconds to over a minute to finish.

Another component of querying time is the time it takes to translate the INEX topic into *evaluation plan*. As discussed in Section 4.1, although the topic translation is automated, the ambiguities in the topic specification mandate some manual work to ensure the resulting *evaluation plan* can be correctly executed by Timber. This step, which involves reading through and understanding both the topic and the plan, usually takes a few minutes.

5 Conclusion

In this paper, we described our participation in INEX. In particular, we described how we have extended Timber, a native XML database system, to query structured text in the format of XML.

The official assessment results from INEX indicate that, when equipped with IR extensions, Timber performs quite well in querying XML data (with regard to the topics whose assessments are finished). We believe the success comes from two aspects. First, as an XML database engine, Timber is able to handle structure constraints with ease. For content-and-structure queries, Timber can significantly reduce the number of document components to be scored based on the structure conditions. Second, the integration of the score function and pick function into the Query Evaluator allows Timber to efficiently assess a component based on keywords (score function) and structural containment (pick function), which makes it suitable to process IR style non-boolean queries.

ACKNOWLEDGEMENT

This work is supported in part by National Science Foundation under Grant No. IIS-0208852. We would like to thank members of the Timber research group at the University of Michigan for their technical assistance and helpful advice.

References

[1] Rakesh Agrawal and Edward L. Wimmers. A framework for expressing and combining preferences. In *International Conference on Management of Data (SIGMOD)*, 2000.

[2] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh Patel, Divesh Srivastava, and Yuqing Wu. Structural joins: A primitive for efficient XML query pattern matching. In *International Conference on Data Engineering (ICDE)*, Heidelberg, Germany, April 2001.

[3] Shurug Al-Khalifa, Cong Yu, and H. V. Jagadish. Querying structured text in an XML database. In *International Conference on Management of Data (SIGMOD)*, San Diego, CA, June 2003.

[4] Nicolas Bruno, Luis Gravano, and Amelie Marian. Evaluating top-k queries over web-accessible databases. In *International Conference on Data Engineering (ICDE)*, 2002.

[5] Kevin Chen-Chuan Chang and Seung won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *International Conference on Management of Data (SIGMOD)*, 2002.

[6] William Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *International Conference on Management of Data (SIGMOD)*, 1998.

[7] Berkeley DB. http://www.sleepycat.com/.

[8] Norbert Fuhr and Kai Großjohann. XIRQL: A query language for information retrieval in XML documents. In *International Conference on Information Retrieval (SIGIR)*, 2001.

[9] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database system. *ACM Transactions on Information Systems (TOIS)*, 15(1), January 1997.

[10] GiST. http://gist.cs.berkeley.edu/.

[11] Vagelis Hristidis, Nick Koudas, and Yannis Papakonstantinou. PREFER: A system for the efficient execution of multiparametric ranked queries. In *International Conference on Management of Data (SIGMOD)*, 2001.

[12] INEX: Initiative for the evaluation of XML retrieval. http://qmir.dcs.qmul.ac.uk/inex/.

[13] H. V. Jagadish, Shurug Al-Khalifa, Adriane Chapman, Laks V.S. Lakshmanan, Andrew Nierman, Stelios Paparizos, Jignesh M. Patel, Divesh Srivastava, NuweeWiwatwattana, Yuqing Wu, and Cong Yu. TIMBER: A native XML database. *The VLDB Journal*, 11(4):274–291, 2002.

[14] H. V. Jagadish, Laks V.S. Lakshmanan, Divesh Srivastava, and Keith Thompson. TAX: A tree algebra for XML. In *International Workshop on Database Programming Languages (DBPL)*, Marino, Italy, September 2001.

[15] Andrew Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *Very Large Data Bases (VLDB) Conference*, Hong Kong, China, August 2002.

[16] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3), 1980.

[17] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[18] Torsten Schlieder and Holger Meuss. Result ranking for structured queries against XML documents. In *DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries*, 2000.

[19] Albrecht Schmidt, Martin Kersten, and Menzo Windhouwer. Querying XML documents made easy: Nearest concept queries. In *International Conference on Extending Database Technology (EDBT)*, 2001.

[20] Shore. http://www.cs.wisc.edu/shore/.

[21] Anja Theobald and Gerhard Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *International Conference on Extending Database Technology (EDBT)*, 2002.

[22] WordNet. An electronic lexical database. http://www.cogsci.princeton.edu/ wn/.

[23] C. Zhang, J. Naughton, D. Dewitt, Q. Luo, and G. Lohman. On supporting containment queries in relational database management systems. In *International Conference on Management of Data (SIGMOD)*, 2001.

EXIMA™ Supply at INEX 2002: Using an Object-relational DBMS for XML Retrieval

Heesop KIM *, Daesik JANG **, Gi Chai HONG***, Jong Cheol SONG***, Seong Yong LEE***,
Hyun Soo CHUNG***, Jae Hwan LEE***, Byung Ju MOON***
* Department of Library and Information Science, Kyungpook National University, Daegu, 702-701, KOREA
heesop@knu.ac.kr
** INCOM I&C Co. Ltd. R&D Center, 996-1, Daechi-dong, KangNam-Gu, Seoul, 135-280, KOREA
dsjang@duli.incom.co.kr
*** IT Information Center, ETRI, 161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, KOREA
{gchong, jcsong, leesy5, hsjung, jeahlee, bjmoon}@etri.re.kr

Abstract

In this paper we report our approach using an object-relational DBMS for INEX collection. EXIMA™ Supply is a kind of native XML DB and supporting Xpath Standard to search elements in XML documents, however, it is not offer any functionality of intelligent searching techniques. We briefly describe the test collection preparation, indexing, retrieval processes, and the evaluation results. Although EXIMA™ Supply has many benefits, for example, no delay in storing and searching XML documents, it showed relatively poor performance in overall evaluation at INEX 2002. This result may be caused since the given topics had to be decomposed and modified to be processed by the Xpath processor in EXIMA™ Supply, and during this modification the original meaning of topics can be changed inevitably and some important information may missing. Furthermore, EXIMA™ Supply targets only for Korean documents, and we were not able to implement any aid tools for construction of indices, knowledge bases for INEX 2002 test collection.

Keywords

XML Retrieval; EXIMA Supply; Object-relational DBMS; UniSQL; IR Evaluation

1. Introduction

The topics provided by INEX (Initiative for the Evaluation of XML retrieval) were deployed and tested by the native XML DB named EXIMA™ Supply developed by Incom I&C Co. Ltd.

EXIMA™ Supply is a kind of native XML DB to store and manage XML documents effectively. It can store and retrieve XML and its related documents (e.g., DTD, XSL) fast enough to process XML information. EXIMA™ Supply is supporting Xpath Standard to search elements in XML documents. However, it is not provide any functionality of a searching engine. This means that it cannot search information as intelligently as most searching engines do. As a result, the given topics had to be decomposed and modified to be processed by the Xpath processor in EXIMA™ Supply. The modified topics were expressed in one or several Xpath queries. Some complicated topics had to be decomposed into several Xpath queries. During this process of modification, the original meanings of topics were changed inevitably and some information was lost.

2. System environments

2.1. Software

The topics provided by INEX were tested under the following software environment.

- OS: Windows 2000 Professional
- XML Server: EXIMA™ Supply 1.0
- DBMS: UniSQL 5.1
- Web Server: Tomcat
- Searching client: Web application developed with JSP,

2.2. Hardware

- Server
  : Machine - Pentium III PC
  : Memory – 256 MB
- Client
  : Machine - Pentium III PC
  : Memory – 256 MB

3. Experimental Design

3.1. Test collection preparation

3.1.1. Preparing of test collection

The XML documents in test collection are stored in EXIMA™ Supply. EXIMA™ Supply is a native XML DB based on object-relational DBMS technologies. Therefore, it can preserve the native features of XML documents by representing and storing them in object-oriented structures. This is one of the important features of EXIMA ™ Supply. Thanks to this feature, the data and hierarchical information of XML documents can be stored without modification or distortion.

Besides, EXIMA™ Supply helps manage and utilize XML documents with ease by providing the standard Xpath query language. With EXIMA™ Supply, there is no need to transform XML documents into other formats such as relational tables of commercial DBMS (many XML servers are using relation DBMS and therefore XML documents must be transformed into relational tables), because it can treat the hierarchical structures of XML documents as it is. As a result, there is no delay in storing and searching XML documents and it is possible to process XML data on the fly.

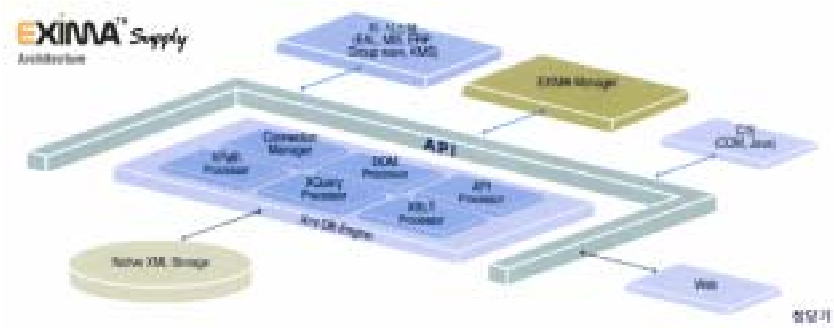


Figure 1: Architecture of EXIMA™ Supply

EXIMA™ Supply provides a logically hierarchical structure to manage the storage of XML documents. The logically hierarchical structure is the storage structure that is transparently accessible by users regardless of the internal physical storage structure. EXIMA™ Supply has two kinds of storage types, “Cabinet” and “Folder.” Cabinet is a logical storage that can contain cabinets and folders. Cabinet can be used to manage storage hierarchically.

Folder is the storage where XML and related documents are actually stored. A folder can contain one DTD and corresponding XML and XSL documents. On the other hand, XML documents correspond to a DTD can be stored in multiple folders if necessary.

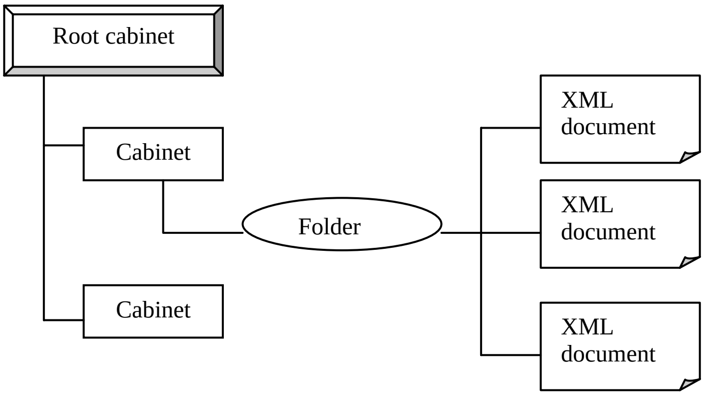


Figure 2: The Storage Types of EXIMA™ Supply

In set up the XML documents provided by INEX into EXIMA™ Supply, the directory structure of XML documents was mapped into the logical structure of EXIMA™ Supply. For example, XML documents in “E:\an\1995” directory are stored in the folder “1995” in the cabinet “an.”

The following picture shows the example storage structure of EXIMA™ Supply shown in EXIMA™ Manager.

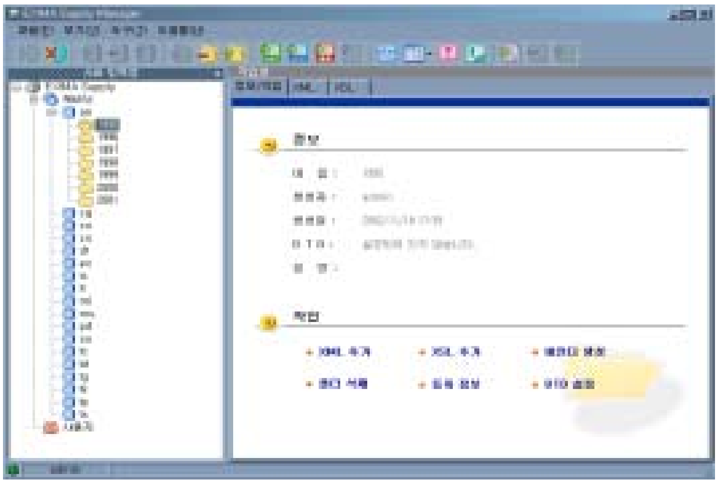


Figure 3: Example of the Storage Structure of EXIMA™ Supply

3.1.2. Indexing

EXIMA™ Supply has the functionality of indexing of elements of XML documents. EXIMA™ Supply makes indexes of elements when an XML document is stored. So it doesn’t need any extra indexing process. Elements in one folder are indexed together and the searching speed is almost same among elements in one folder. However, the indexing is done in each folders, the searching speed may be different from each folder.

3.1.3. Retrieval process

- Xpath query generation

EXIMA™ Supply is not equipped with any searching engine functionality and it just supports Xpath searching functionality. Therefore, searching topics from INEX has to be converted to Xpath queries for searching information. For instance, INEX topic 01 can be expressed in Xpath queries as follows:

Topic 01:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE INEX-Topic SYSTEM "inex-topics.dtd">
<INEX-Topic topic-id="01" query-type="CAS" ct-no="010">
    <Title>
        <te>article/fm/au</te>
        <cw>description logics</cw><ce>abs, kwd</ce>
    </Title>
    <Description>
        Retrieve the names of authors of articles on description logic, in particular articles in
        which the abstract or the list of keywords contains a reference to description logic.
    </Description>
    <Narrative>
        The rating should reflect the likeliness that a person is an expert on description logic.
    </Narrative>
    <Keywords>
        description logic DL ABox TBox reasoning
    </Keywords>
</INEX-Topic>
```

Xpath query:

```
"article/fm[abs//*/text('*')[contains('description logic')]]/au"
```

Complicated topics that can not be expressed in one Xpath query can be divided into several Xpath queries. For instance, topic 06 can be expressed in Xpath queries as follows:

Topic 06:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE INEX-Topic SYSTEM "inex-topics.dtd">
<INEX-Topic topic-id="06" query-type="CAS" ct-no="034">
    <Title>
        <te>tig</te>
        <cw>Survey on Software Engineering</cw>
        <cw>
            software engineering survey, programming survey, programming tutorial,
            software engineering tutorial
        </cw>
        <ce>tig</ce>
        <cw>programming languages</cw><ce>sec</ce>
    </Title>
    <Description>
        Retrieve the article title from all articles which are a tutorial or survey on software
        engineering or programming dealing with programming languages.
    </Description>
    <Narrative>
        To be relevant an article should offer a tutorial or survey on software
        engineering or programming containing sections dealing with programming languages.
    </Narrative>
    <Keywords>
        survey, tutorial software engineering, programming language
    </Keywords>
</INEX-Topic>
```

Xpath queries:
"article[//tig//*/text('*')[contains('Survey on Software Engineering')]]//tig"
"article[//tig//*/text('*')[contains('software')][contains('engineering')][contains('survey')][contains('tutorial')]]//tig"
"article[//sec//*/text('*')[contains('programming')][contains('languages')]]//tig"

If a topic can not be expressed in Xpath queries, just keywords can use for searching.

- Searching process of Xpath queries

In EXIMA™ Supply, the Xpath queries processed as the following Figure 4.

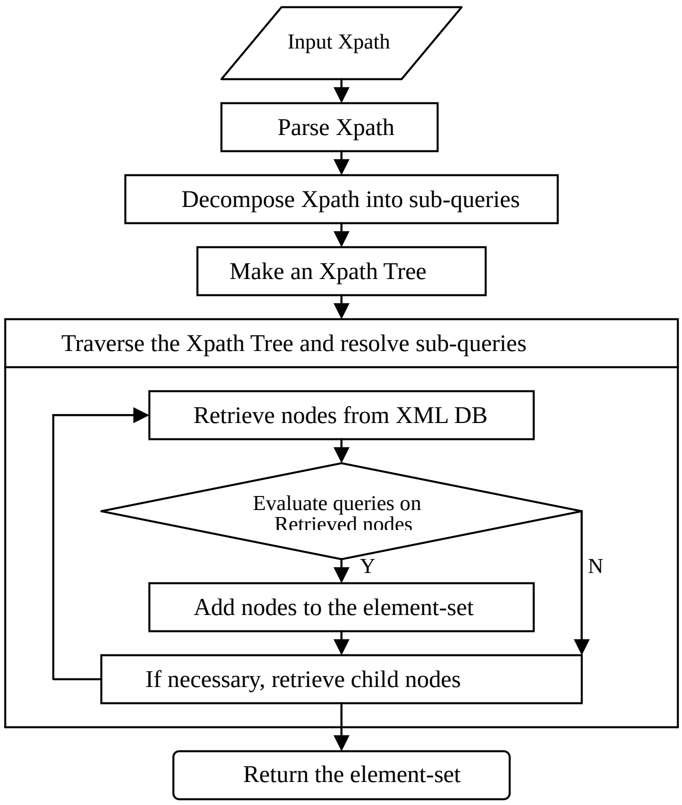
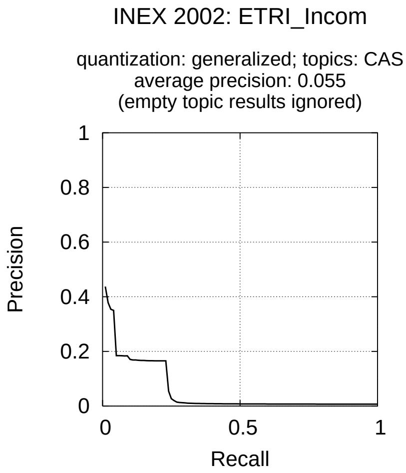


Figure 4: Flow of query processing in EXIMA™ Supply

As the above diagram illustrate, the given Xpath query is first parsed and then decomposed into several sub-queries. And based on these sub-queries, a query tree that represents the hierarchical relation of sub-queries is constructed. Once the query tree is constructed, the tree is traversed and evaluated to get the corresponding nodes. The traversing of query tree starts from the current context element. EXIMA™ Supply first retrieves the child elements of the current element as candidate elements from storage. And then the candidate elements are evaluated and elements that satisfy conditions are added to the element-set. The traversing is done recursively along to the child nodes of the query tree. If all nodes of the query tree are traversed and evaluated, the element-set is returned as the result of the search.

4. Results

We only submitted the results of CAS (content-and-structure) queries in INEX 2002. Figure 5 presents P-R graphs for the evaluations results of the subsets of CAS topics, i.e., #01, #04, #05, #06, #11, #21. Applying the strict evaluation gave slightly higher score (average precision: 0.077) than the generalized evaluation result (average precision: 0.055) which provided by the official INEX organizers.



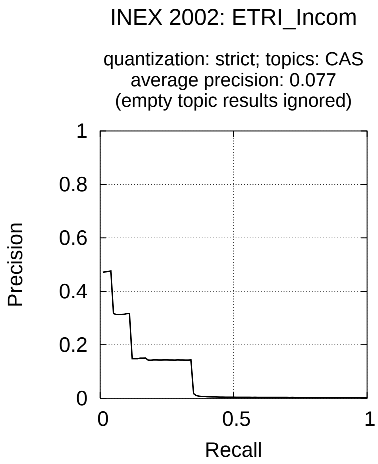


Figure 5: P-R Graph for (a) Generalized and (b) Strict CAS topic ignored empty results

Our overall, rather than empty topic results ignored, result showed relatively poor (average precision: 0.019). As shown in Figure 6 our results ranked with the 34th among 42 official submissions.

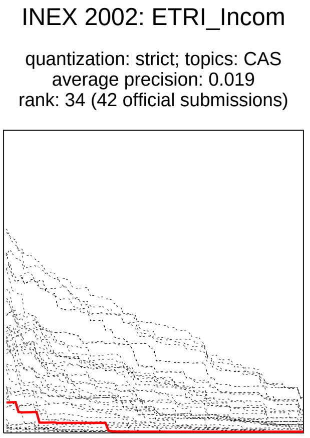


Figure 6: P-R Graph for Overall Results and Rank

5. Conclusion

In this paper, we described an approach of object-relational DBMS using EXIMA™ Supply for INEX test collections. Although EXIMA™ Supply has many benefits, for example, no delay in storing and searching XML documents, it showed relatively poor performance in overall evaluation at INEX 2002.

This result may be caused since the given topics had to be decomposed and modified to be processed by the Xpath processor in EXIMA™ Supply, and during this modification the original meaning of topics can be changed inevitably and some important information may missing. Some other possibilities are that because EXIMA™ Supply targets only for Korean, and we were not able to implement any aid tools of construction of indices, knowledge bases for INEX collection which will require to be investigating in the future study.

Acknowledgements

This work was supported in part by the Ministry of Information and Communication, Korea under the *Development of Information Distribution Framework Project.* Any opinions, findings, or conclusions expressed in this paper are those of the authors, and do not necessarily reflect those of the sponsor.

References

[1] Incom I&C Co. Ltd. EXIMA™ Supply. *Online available at*: http://www.incom.co.kr
[2] INEX homepage at: http://qmir.dcs.qmul.ac.uk/INEX/
[3] INEX {down, up} load area available at: http://ls6-www.cs.uni-dortmund.de/ir/projects/inex/download/
[4] Y. Despotopoulos, G. Patikis, J. Soldatos, L. Polymenakos, J. Kleindienst, and J. Geric. Accessing and transforming dynamic content based on XML: alternative techniques and a practical implementation. In W. Winiwarter, S. Bressan, and I.K. Ibrahim, editor, *Third International Conference on Information Integration and Web-based Applications and Services (IIWAS 2001).* Osterreichische Comput. Gesellschaft. 2001, pp. 95-105. Wien, Austria.
[5] T.T. Chinenyanga, and N. Kushmerick. Expressive retrieval from XML documents. *SIGIR Forum (ACM Special Interest Group on Information Retrieval)*, spec. issue., 2001, pp.163-71.
[6] S. Ha, and K. Kim. Mapping XML documents to the object-relational form. *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings. Part Vol. 3*, 2001, pp. 1757-61. Piscataway, NJ, USA.
[7] S.J. Lim, and Y. K. Ng. An automated integration approach for semi-structured and structured data. *Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications. CODAS 2001*. IEEE Comput. Soc. 2000, pp. 12-21. Los Alamitos, CA, USA
[8] C. Zhang, J. Naughton, D. DeWitt, O. Luo, and G. Lohman. On supporting containment queries in relational database management systems. *ACM. SIGMOD Record (ACM Special Interest Group on Management of Data)*, Vol. 30, No. 2, June 2001, pp. 425-36.
[9] D. Shin. XML indexing and retrieval with a hybrid storage model. *Knowledge & Information Systems*, Vol. 3, No. 2, May 2001, pp. 252-61
[10] J. A. Miller, and S. Sheth. Querying XML documents. *IEEE Potentials*, Vol. 19, No.1, Feb.-March 2000, pp. 24-6.
[11] C. Petrou, S. Hadjiefthymiades, and D. Martakos. An XML-based, 3-tier scheme for integrating heterogeneous information sources to the WWW. In A. Cammelli, A. Tjoa, R.R. Wagner, editors, *Proceedings of Tenth International Workshop on Database and Expert Systems Applications. DEXA 99*. IEEE Comput. Soc. 1999, pp. 706-10. Los Alamitos, CA, USA.
[12] M. M. David. SQL-based XML structured data access. *Web Techniques*, Vol. 4, No. 6, June 1999, pp. 67-8, 70, 72.
[13] O. Alonso. Generation of text search applications for databases. An exercise on domain engineering. In C. Gacek, editor, *Software Reuse: Methods, Techniques, and Tools. 7th International Conference, ICSR-7. Proceedings (Lecture Notes in Computer Science Vol. 2319).* Springer-Verlag. 2002, pp. 179-93.
[14] M. Papiani, J. L. Wason, A. N. Dunlop, and D. A. Nicole. A distributed scientific data archive using the Web, XML and SQL/MED. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, Vol. 28, No.3, Sept. 1999, pp. 56-62.
[15] N. Fuhr, N. Goevert, G. Kazai, and M. Lalmas. INEX: Initiative for the Evaluation of XML Retrieval, *ACM SIGIR Workshop on XML and Information Retrieval*, Tampere, Finland, August 2002.

Appendix

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) - <ce> - 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 <ce> 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 10th 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 it 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 then the 10th 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 and Procedure

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       (description?, topic+)>
<!ATTLIST inex-submission
     participant-id    CDATA    #REQUIRED
     run-id            CDATA    #REQUIRED
>
<!ELEMENT description  (#PCDATA)>
<!ELEMENT topic        (result*)>
<!ATTLIST topic
     topic-id          CDATA    #REQUIRED
>
<!ELEMENT result       (file, path, rank?, rsv?)>
<!ELEMENT file         (#PCDATA)>
<!ELEMENT path         (#PCDATA)>
<!ELEMENT rank         (#PCDATA)>
<!ELEMENT rsv          (#PCDATA)>
```

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 result 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 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[1]/bm[1]/ack[1]</path>
            <rsv>0.67</rsv>
        </result>
        <result>
            <file>an/1995/a1004</file>
            <path>/article[1]/bdy[1]/sec[1]/p[3]</path>
            <rsv>0.1</rsv>
        </result>
        [ ... ]
    </topic>
    <topic topic-id="02">
        [ ... ]
    </topic>
    [ ... ]
</inex-submission>
```

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 XPath syntax. To be more precise, only fully specified paths are allowed, as described by the following grammar:

Path	::=	'/' ElementNode Path | '/' ElementNode '/' AttributeNode | '/' ElementNode
ElementNode	::=	ElementName Index
AttributeNode	::=	'@' AttributeName
Index	::=	'[' integer ']'

An example path:

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

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 it can be seen, XPath counts elements starting with one and takes into account the element type, e.g. if a section had a title and 2 paragraphs then their paths would be ../title[1], ../p[1] and p[2].

As mentioned 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[1]/bdy[1]/sec[1]/p[3]</path>
</result>

<result>
    <file>an/1995/volume</file>
    <path>/books[1]/journal[1]/article[2]/bdy[1]/sec[1]/p[3]</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.

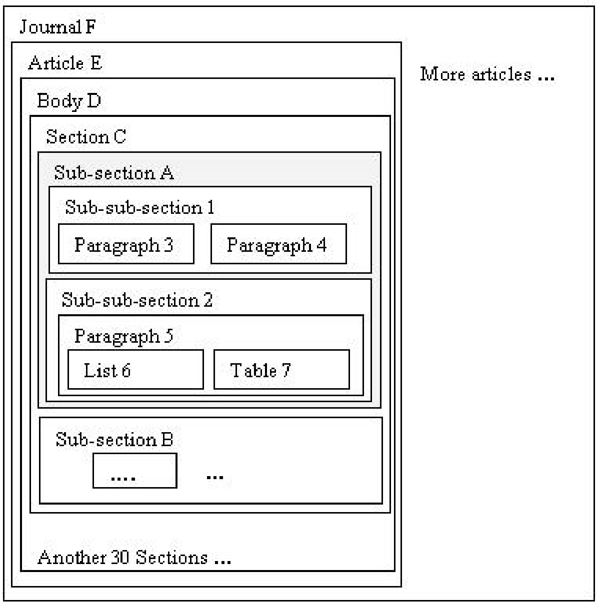


Figure 1. Example XML structure and result element

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 0N, 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.

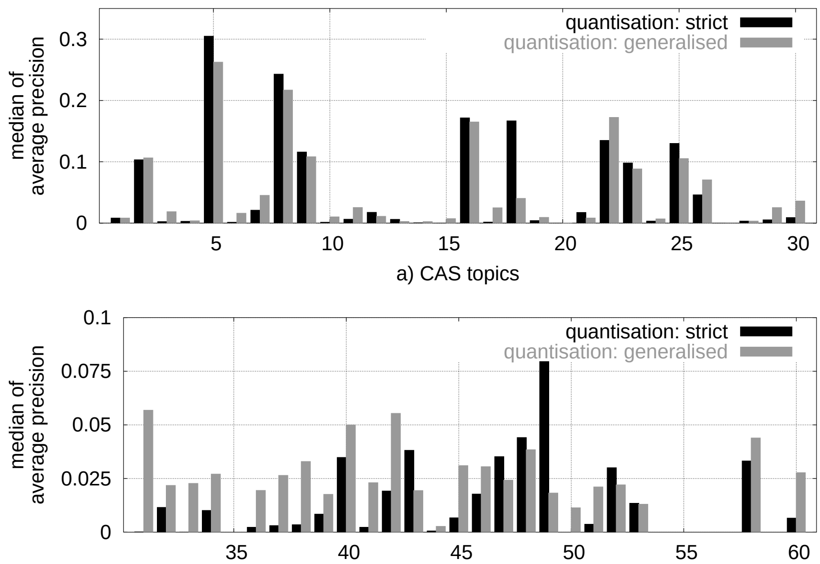
INEX 2002 Evaluation Results in Detail

The following pages contain the results for all submissions for INEX 2002. There were 42 submissions for the content-and-structure (CAS) topics and 49 submissions for the content-only (CO) topics.

The initial pages give a listing of all submissions for the CAS and CO tasks, respectively (identified by organisation name and run ID). The remainder of this report is made up by the detailed results for each submission. Each submission is presented on one page, with the following details (given for strict and generalised quantisation):

- A recall / precision graph, providing a plot of the precision values for 100 recall points. In a comparative diagram the recall / precision graph is plotted together with all the recall / precision graphs obtained from the other submissions.
- The overall average precision, computed over 100 recall points.
- A table displaying average precision values for each topic.
- A diagram which compares the evaluation results per topic to median performance in INEX 2002. For each topic, the difference in average precision, compared to the median average precision for that topic, is plotted.

The following figures contain an overview on the median average precision values per topic.



a) CAS topics

b) CO topics

All results presented in this report have been compiled using the assessment package version 1.8 and `inex_eval` version 0.007. A detailed description on the evaluation metrics used in INEX 2002 is provided in [1]. The result description is based on what has been done in TREC (see e. g. [2] for further details).

References

[1] Norbert Gövert and Gabriella Kazai. Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2002. In Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas, editors, *INitiative for the Evaluation of XML Retrieval (INEX). Proceedings of the First INEX Workshop. Dagstuhl, Germany, December 8–11, 2002*, ERCIM Workshop Proceedings, Sophia Antipolis, France, March 2003. ERCIM.

[2] E. M. Voorhees and D. K. Harman, editors. *The Tenth Text REtrieval Conference (TREC 2001)*, Gaithersburg, MD, USA, 2002. NIST.

Overview of content-and-structure submissions

Organisation	run ID	Page
Centrum voor Wiskunde en Informatica (CWI)	R_all	191
	R_article	192
	R_prel_length	193
CSIRO Mathematical and Information Sciences	full	194
	manual	195
	Split	196
doctronic GmbH & Co. KG	1	197
Electronics & Telecommunications Research Institute (ETRI)	ETRI_Incom	198
ETH Zurich	Augmentation0.8	199
IBM Haifa Labs	ManualNoMerge	200
	Merge	201
	NoMerge	202
Institut de Recherche en Informatique de Toulouse (IRIT)	Mercure1	203
Nara Institute of Science and Technology	20020824-article	204
Queen Mary University of London	QMUL1	205
	QMUL2	206
	QMUL3	207
Queensland University of Technology	inexresult2.xml	208
	inexresults1.xml	209
	inexresults3.xml	210
Salzburg Research Forschungsgesellschaft	1-corrected	211
Sejong Cyber University	TitleKeywordsWLErr	212
Tarragon Consulting Corporation	tgnCAS_base	213
Universität Bayreuth	IRStream	214
Universität Dortmund / Universität Duisburg-Essen	plain hyrex	215
Université Pierre et Marie Curie	bayes-3	216
	simple	217
University of Amsterdam	UAmsI02NGiSt	218
	UAmsI02NGram	219
	UAmsI02Stem	220
University of California, Berkeley	Berkeley01	221
	Berkeley02	222
	Berkeley03	223
University of Melbourne	um_mgx21_short	224
	um_mgx26_long	225
	um_mgx2_long	226
University of Michigan	allow-duplicate	227
	no-duplicate	228
University of Minnesota Duluth	01	229
University of Twente	utwente1h	230
	utwente1n	231
	utwente1pr	232

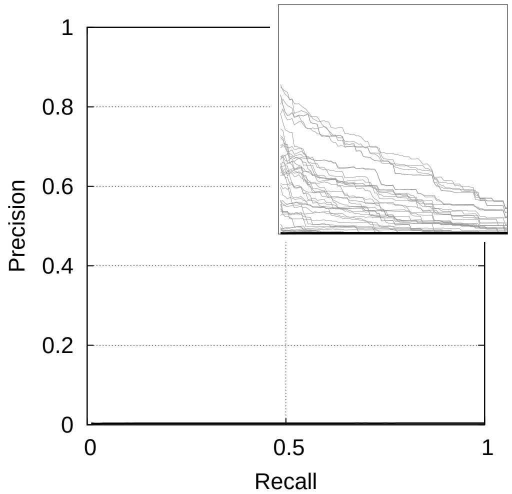
Overview of content-only submissions

Organisation	run ID	Page
Centrum voor Wiskunde en Informatica (CWI)	R_all	233
	R_article	234
	R_prel_length	235
CSIRO Mathematical and Information Sciences	full	236
	manual	237
	Split	238
doctronic GmbH & Co. KG	1	239
ETH Zurich	Augmentation0.8	240
IBM Haifa Labs	ManualNoMerge	241
	Merge	242
	NoMerge	243
Institut de Recherche en Informatique de Toulouse (IRIT)	Mercure1	244
Nara Institute of Science and Technology	20020824-article	245
Queen Mary University of London	QMUL1	246
	QMUL2	247
	QMUL3	248
Queensland University of Technology	inexresult2.xml	249
	inexresults1.xml	250
	inexresults3.xml	251
Royal School of Library and Information Science	bag-of-words	252
	boomerang	253
	polyrepresentation	254
Salzburg Research Forschungsgesellschaft	1-corrected	255
Sejong Cyber University	TitleKeywordsWLErr	256
Tarragon Consulting Corporation	tgnCO_base	257
Universität Bayreuth	IRStream	258
Universität Dortmund / Universität Duisburg-Essen	Epros03	259
	Epros06	260
	plain hyrex	261
Université Pierre et Marie Curie	bayes-2	262
	bayes-3	263
	simple	264
University of Amsterdam	UAmsI02NGiSt	265
	UAmsI02NGram	266
	UAmsI02Stem	267
University of California, Berkeley	Berkeley01	268
	Berkeley02	269
	Berkeley03	270
University of California, Los Angeles	CorrectedFormat	271
University of Melbourne	um_mgx21_short	272
	um_mgx26_long	273
	um_mgx2_long	274
University of Michigan	allow-duplicate	275
	no-duplicate	276
University of Minnesota Duluth	01	277
University of North Carolina at Chapel Hill	irt	278
University of Twente	utwente1h	279
	utwente1n	280
	utwente1pr	281

Centrum voor Wiskunde en Informatica (CWI)
R_all (CAS)

Quantisation: strict

Recall / precision graph:

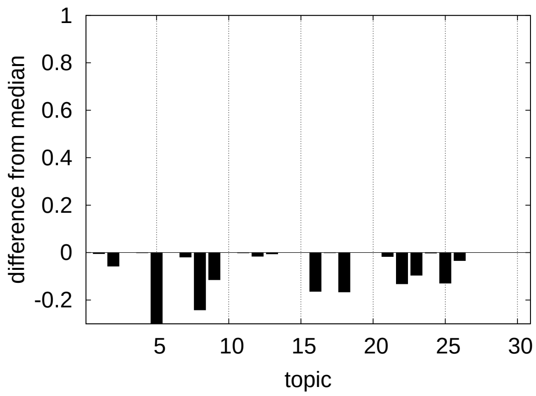


Overall average precision: 0.0039

Average precision per topic:

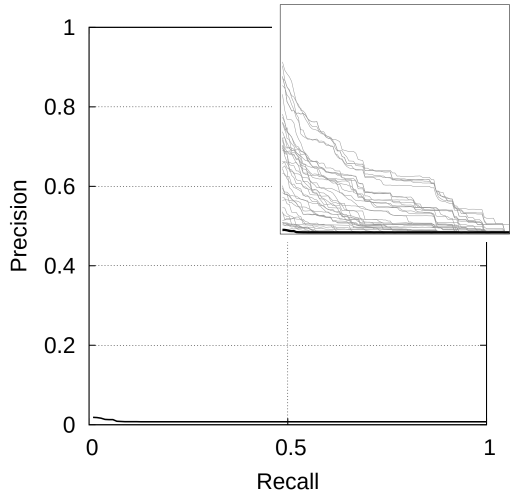
01	0.0030	11	0.0045	21	0.0002
02	0.0455	12	0.0013	22	0.0024
03	0.0025	13	0.0001	23	0.0019
04	0.0015	14	0.0004	24	0.0005
05	0.0053	15	0.0004	25	0.0005
06	0.0013	16	0.0073	26	0.0117
07	0.0015	17	0.0002	27	0.0001
08	0.0006	18	0.0002	28	0.0037
09	0.0008	19	0.0047	29	0.0050
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

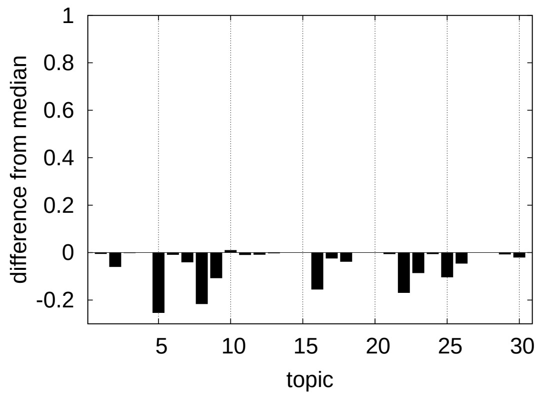


Overall average precision: 0.0080

Average precision per topic:

01	0.0030	11	0.0158	21	0.0025
02	0.0463	12	0.0025	22	0.0032
03	0.0170	13	0.0001	23	0.0024
04	0.0040	14	0.0024	24	0.0009
05	0.0087	15	0.0076	25	0.0015
06	0.0072	16	0.0097	26	0.0247
07	0.0049	17	0.0008	27	0.0001
08	0.0007	18	0.0021	28	0.0037
09	0.0008	19	0.0104	29	0.0182
10	0.0211	20	0.0006	30	0.0158

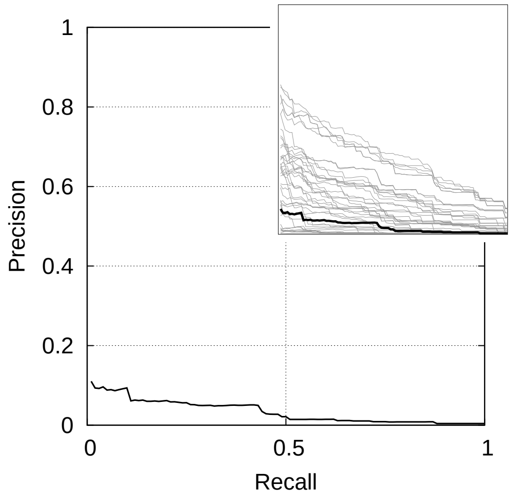
Difference from median in average precision per topic:



Centrum voor Wiskunde en Informatica (CWI)
R_article (CAS)

Quantisation: strict

Recall / precision graph:

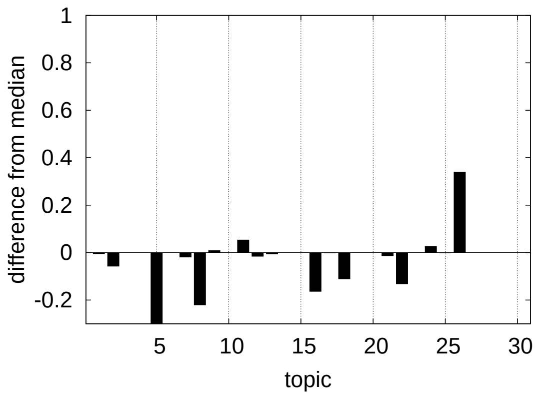


Overall average precision: 0.0338

Average precision per topic:

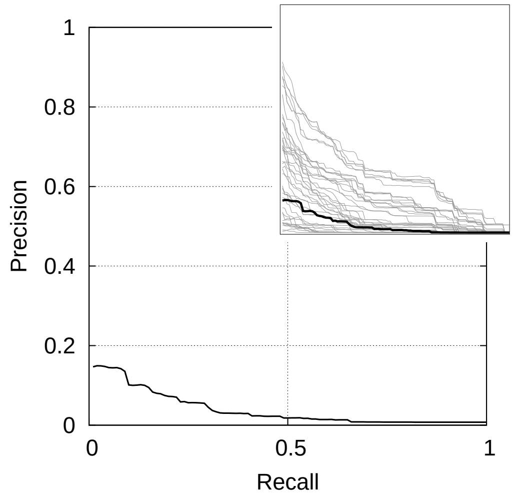
01	0.0030	11	0.0612	21	0.0033
02	0.0452	12	0.0013	22	0.0024
03	0.0039	13	0.0001	23	0.0994
04	0.0036	14	0.0004	24	0.0311
05	0.0053	15	0.0004	25	0.1284
06	0.0013	16	0.0073	26	0.3875
07	0.0015	17	0.0002	27	0.0001
08	0.0219	18	0.0552	28	0.0037
09	0.1259	19	0.0045	29	0.0058
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

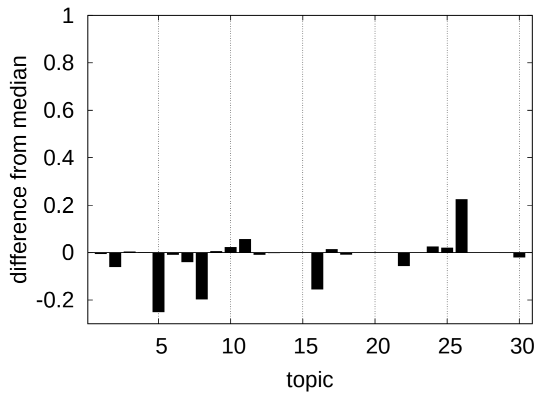


Overall average precision: 0.0391

Average precision per topic:

01	0.0030	11	0.0833	21	0.0081
02	0.0461	12	0.0025	22	0.1162
03	0.0236	13	0.0001	23	0.0889
04	0.0065	14	0.0024	24	0.0331
05	0.0118	15	0.0076	25	0.1268
06	0.0074	16	0.0097	26	0.2955
07	0.0049	17	0.0398	27	0.0001
08	0.0198	18	0.0322	28	0.0037
09	0.1147	19	0.0096	29	0.0245
10	0.0341	20	0.0006	30	0.0158

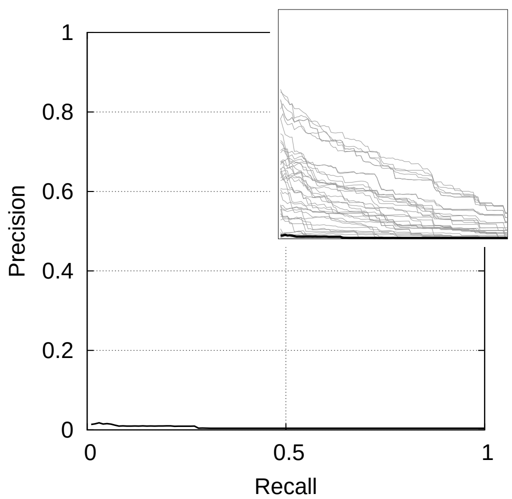
Difference from median in average precision per topic:



Centrum voor Wiskunde en Informatica (CWI)
R_prel_length (CAS)

Quantisation: strict

Recall / precision graph:

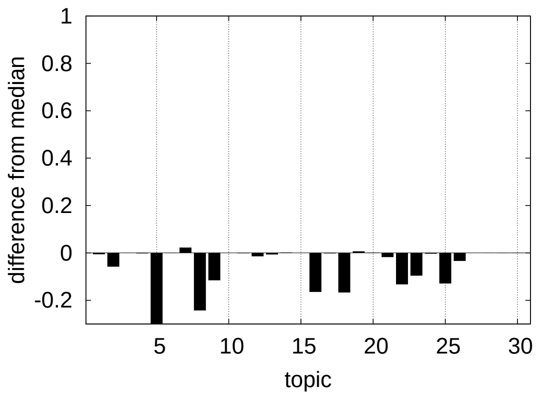


Overall average precision: 0.0059

Average precision per topic:

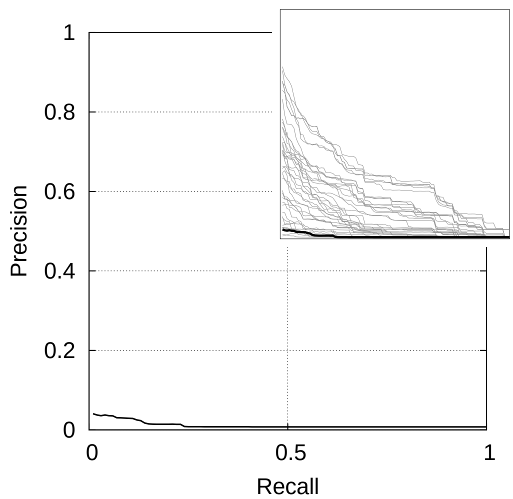
01	0.0030	11	0.0054	21	0.0002
02	0.0457	12	0.0036	22	0.0024
03	0.0026	13	0.0001	23	0.0023
04	0.0015	14	0.0030	24	0.0005
05	0.0053	15	0.0004	25	0.0014
06	0.0013	16	0.0073	26	0.0128
07	0.0442	17	0.0002	27	0.0001
08	0.0006	18	0.0002	28	0.0037
09	0.0008	19	0.0115	29	0.0052
10	0.0018	20	0.0021	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

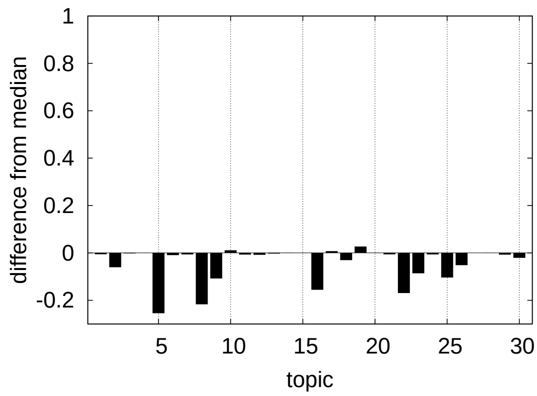


Overall average precision: 0.0115

Average precision per topic:

01	0.0030	11	0.0190	21	0.0028
02	0.0466	12	0.0037	22	0.0034
03	0.0172	13	0.0001	23	0.0028
04	0.0047	14	0.0036	24	0.0011
05	0.0085	15	0.0076	25	0.0019
06	0.0073	16	0.0097	26	0.0193
07	0.0395	17	0.0332	27	0.0001
08	0.0007	18	0.0102	28	0.0037
09	0.0008	19	0.0370	29	0.0186
10	0.0217	20	0.0014	30	0.0158

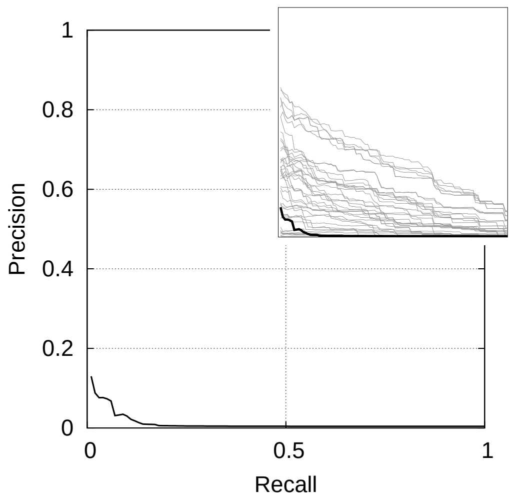
Difference from median in average precision per topic:



CSIRO Mathematical and Information Sciences
full (CAS)

Quantisation: strict

Recall / precision graph:

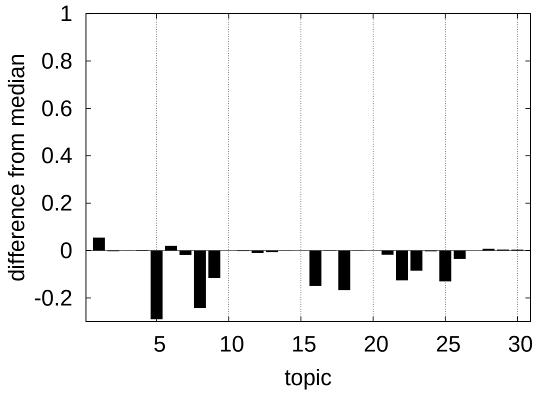


Overall average precision: 0.0109

Average precision per topic:

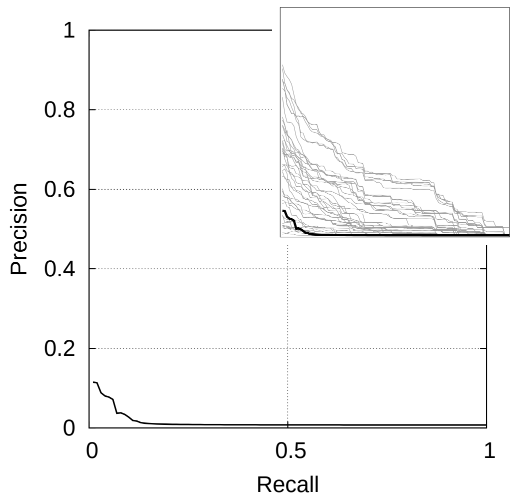
01	0.0634	11	0.0045	21	0.0002
02	0.1008	12	0.0082	22	0.0095
03	0.0025	13	0.0001	23	0.0137
04	0.0015	14	0.0004	24	0.0005
05	0.0155	15	0.0004	25	0.0005
06	0.0218	16	0.0228	26	0.0115
07	0.0032	17	0.0034	27	0.0001
08	0.0006	18	0.0002	28	0.0112
09	0.0008	19	0.0053	29	0.0098
10	0.0018	20	0.0002	30	0.0133

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

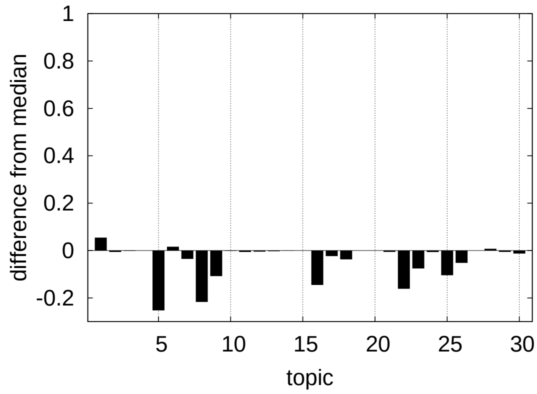


Overall average precision: 0.0143

Average precision per topic:

01	0.0634	11	0.0201	21	0.0033
02	0.1009	12	0.0070	22	0.0115
03	0.0173	13	0.0001	23	0.0133
04	0.0044	14	0.0023	24	0.0009
05	0.0106	15	0.0080	25	0.0015
06	0.0327	16	0.0201	26	0.0188
07	0.0105	17	0.0020	27	0.0001
08	0.0007	18	0.0037	28	0.0112
09	0.0008	19	0.0098	29	0.0203
10	0.0091	20	0.0006	30	0.0240

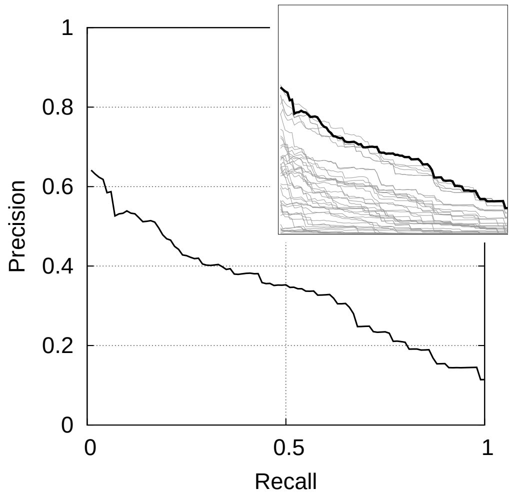
Difference from median in average precision per topic:



CSIRO Mathematical and Information Sciences
manual (CAS)

Quantisation: strict

Recall / precision graph:

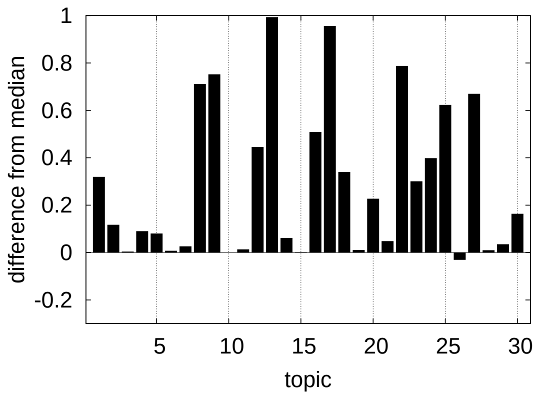


Overall average precision: 0.3438

Average precision per topic:

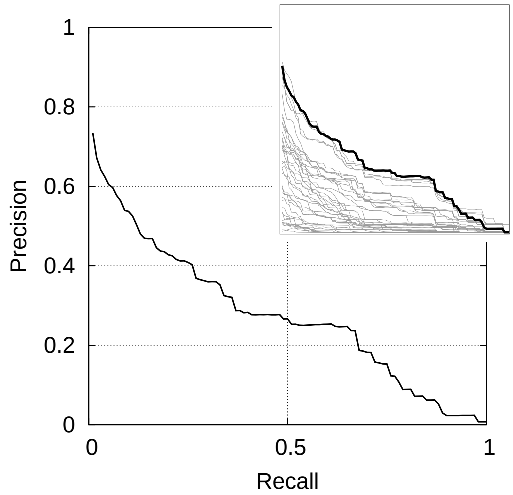
01	0.3282	11	0.0203	21	0.0659
02	0.2209	12	0.4636	22	0.9232
03	0.0070	13	1.0000	23	0.3993
04	0.0935	14	0.0625	24	0.4021
05	0.3858	15	0.0018	25	0.7535
06	0.0095	16	0.6810	26	0.0156
07	0.0479	17	0.9583	27	0.6701
08	0.9548	18	0.5076	28	0.0138
09	0.8688	19	0.0151	29	0.0408
10	0.0018	20	0.2276	30	0.1729

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

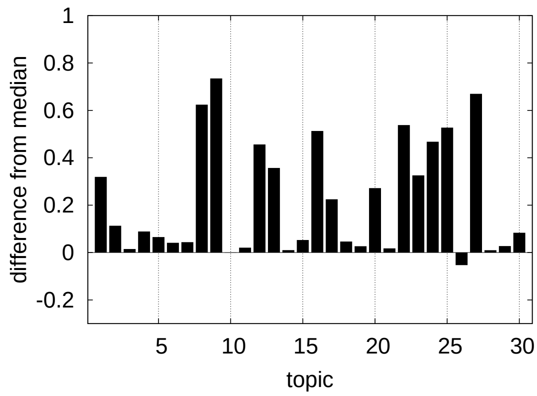


Overall average precision: 0.2752

Average precision per topic:

01	0.3282	11	0.0466	21	0.0263
02	0.2202	12	0.4679	22	0.7108
03	0.0338	13	0.3601	23	0.4148
04	0.0932	14	0.0129	24	0.4753
05	0.3283	15	0.0607	25	0.6331
06	0.0578	16	0.6787	26	0.0179
07	0.0895	17	0.2505	27	0.6701
08	0.8418	18	0.0875	28	0.0138
09	0.8437	19	0.0363	29	0.0533
10	0.0100	20	0.2724	30	0.1201

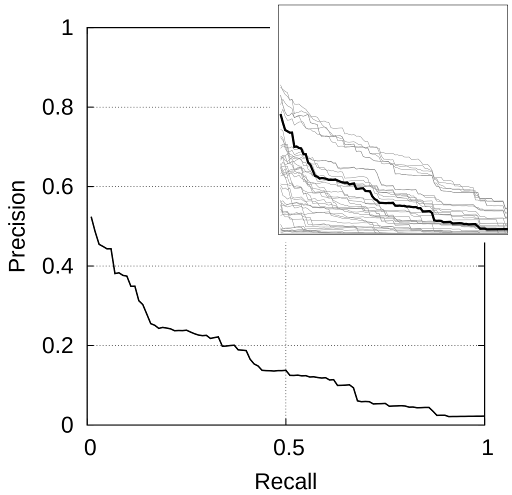
Difference from median in average precision per topic:



CSIRO Mathematical and Information Sciences
Split (CAS)

Quantisation: strict

Recall / precision graph:

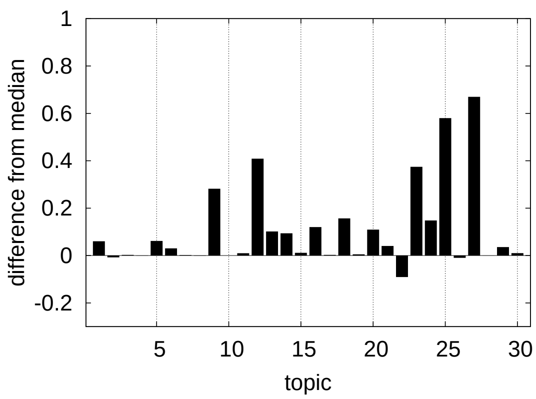


Overall average precision: 0.1616

Average precision per topic:

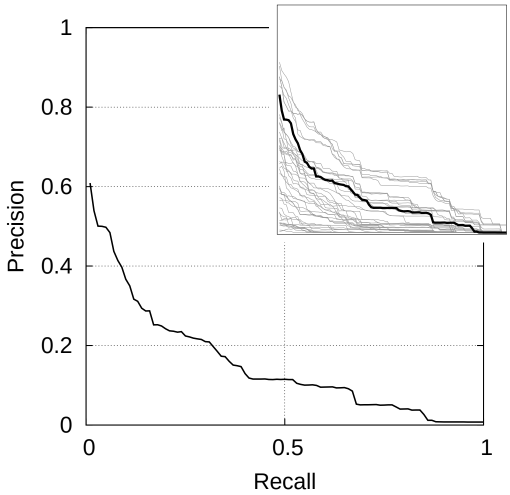
01	0.0691	11	0.0166	21	0.0583
02	0.0963	12	0.4271	22	0.0445
03	0.0057	13	0.1084	23	0.4731
04	0.0028	14	0.0950	24	0.1518
05	0.3672	15	0.0119	25	0.7105
06	0.0322	16	0.2922	26	0.0368
07	0.0238	17	0.0048	27	0.6701
08	0.2425	18	0.3241	28	0.0037
09	0.3983	19	0.0097	29	0.0416
10	0.0018	20	0.1098	30	0.0196

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

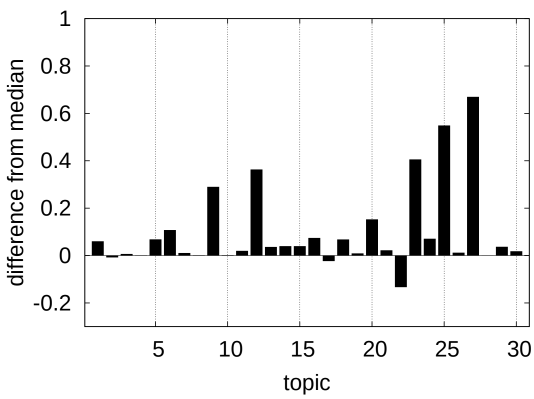


Overall average precision: 0.1528

Average precision per topic:

01	0.0691	11	0.0458	21	0.0312
02	0.0988	12	0.3750	22	0.0394
03	0.0261	13	0.0394	23	0.4948
04	0.0051	14	0.0427	24	0.0785
05	0.3315	15	0.0475	25	0.6548
06	0.1245	16	0.2400	26	0.0833
07	0.0567	17	0.0019	27	0.6701
08	0.2187	18	0.1089	28	0.0037
09	0.3989	19	0.0189	29	0.0630
10	0.0091	20	0.1533	30	0.0549

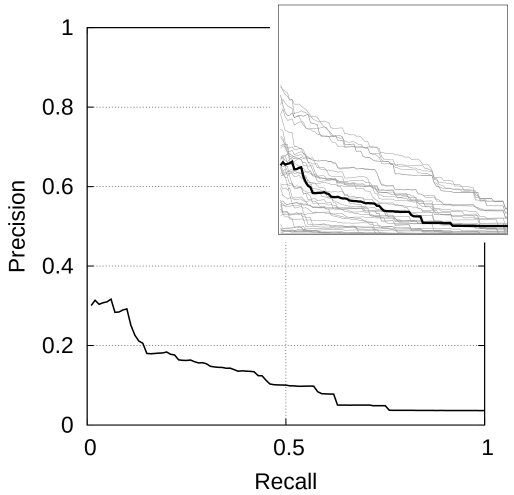
Difference from median in average precision per topic:



doctronic GmbH & Co. KG
1 (CAS)

Quantisation: strict

Recall / precision graph:

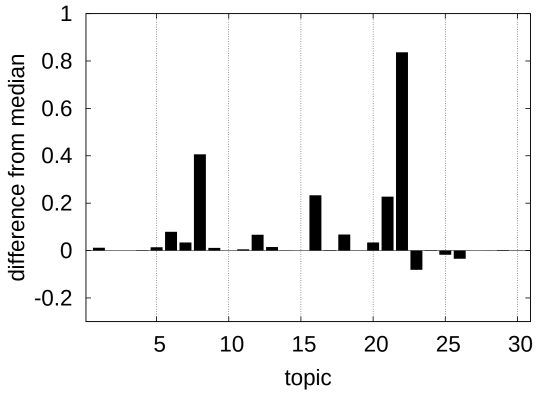


Overall average precision: 0.1182

Average precision per topic:

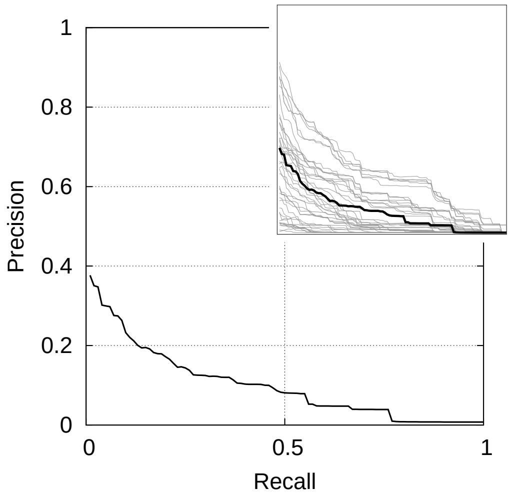
01	0.0205	11	0.0118	21	0.2453
02	0.1039	12	0.0849	22	0.9723
03	0.0031	13	0.0213	23	0.0174
04	0.0017	14	0.0004	24	0.0019
05	0.3192	15	0.0004	25	0.1129
06	0.0809	16	0.4054	26	0.0120
07	0.0556	17	0.0002	27	0.0001
08	0.6496	18	0.2349	28	0.0037
09	0.1277	19	0.0045	29	0.0079
10	0.0020	20	0.0343	30	0.0093

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

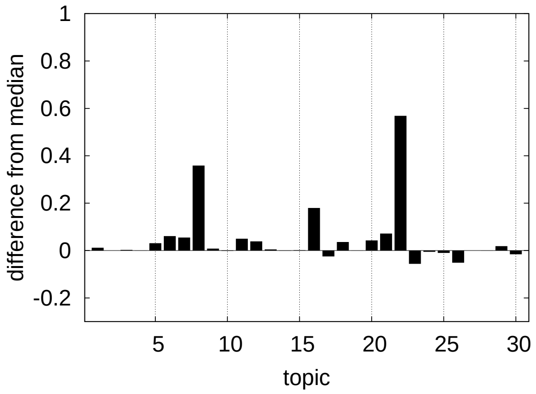


Overall average precision: 0.0997

Average precision per topic:

01	0.0205	11	0.0758	21	0.0804
02	0.1069	12	0.0504	22	0.7416
03	0.0219	13	0.0079	23	0.0327
04	0.0040	14	0.0024	24	0.0021
05	0.2942	15	0.0090	25	0.0956
06	0.0774	16	0.3451	26	0.0197
07	0.1005	17	0.0008	27	0.0001
08	0.5761	18	0.0768	28	0.0037
09	0.1170	19	0.0096	29	0.0444
10	0.0090	20	0.0434	30	0.0209

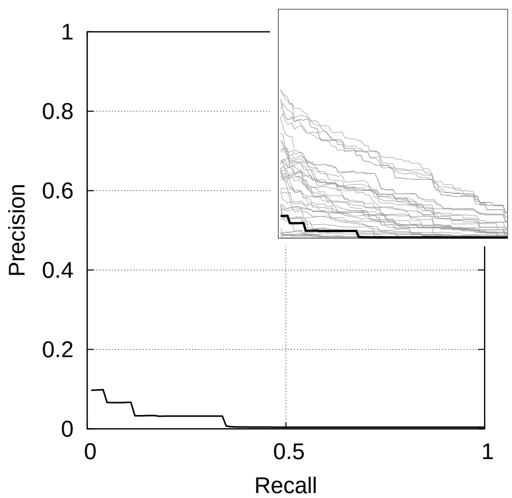
Difference from median in average precision per topic:



Electronics & Telecommunications Research Institute (ETRI)
ETRI_Incom (CAS)

Quantisation: strict

Recall / precision graph:

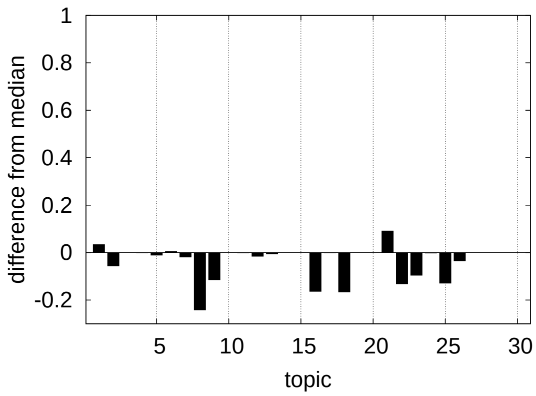


Overall average precision: 0.0188

Average precision per topic:

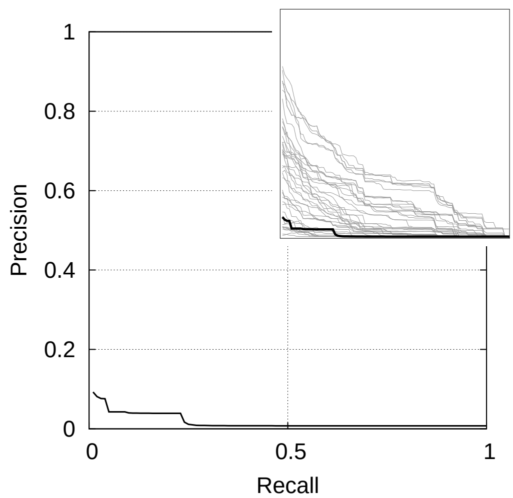
01	0.0433	11	0.0046	21	0.1102
02	0.0465	12	0.0013	22	0.0024
03	0.0026	13	0.0001	23	0.0019
04	0.0015	14	0.0004	24	0.0005
05	0.2932	15	0.0004	25	0.0005
06	0.0076	16	0.0074	26	0.0108
07	0.0015	17	0.0002	27	0.0001
08	0.0006	18	0.0002	28	0.0038
09	0.0008	19	0.0045	29	0.0051
10	0.0018	20	0.0002	30	0.0089

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

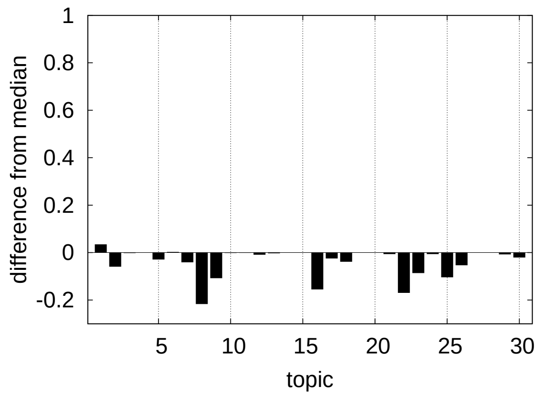


Overall average precision: 0.0169

Average precision per topic:

01	0.0433	11	0.0248	21	0.0026
02	0.0474	12	0.0025	22	0.0032
03	0.0172	13	0.0001	23	0.0025
04	0.0039	14	0.0024	24	0.0010
05	0.2339	15	0.0076	25	0.0015
06	0.0195	16	0.0100	26	0.0175
07	0.0049	17	0.0008	27	0.0001
08	0.0007	18	0.0021	28	0.0038
09	0.0008	19	0.0098	29	0.0183
10	0.0090	20	0.0006	30	0.0159

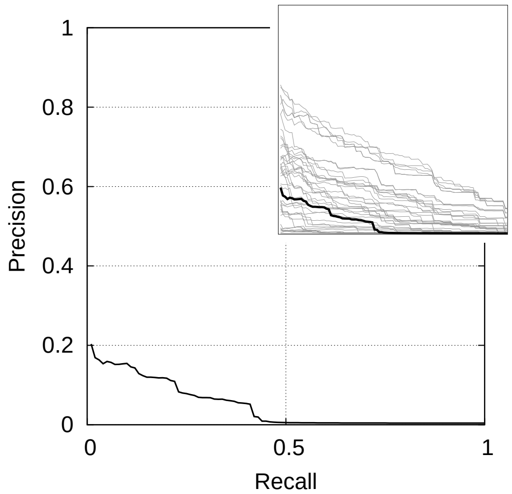
Difference from median in average precision per topic:



ETH Zurich
Augmentation0.8 (CAS)

Quantisation: strict

Recall / precision graph:

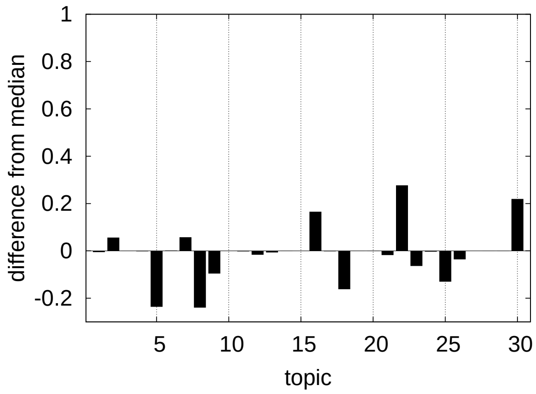


Overall average precision: 0.0466

Average precision per topic:

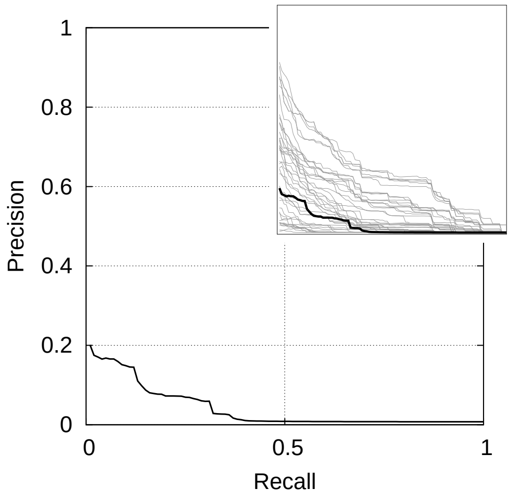
01	0.0033	11	0.0045	21	0.0002
02	0.1603	12	0.0016	22	0.4123
03	0.0025	13	0.0001	23	0.0348
04	0.0015	14	0.0004	24	0.0005
05	0.0689	15	0.0004	25	0.0005
06	0.0033	16	0.3376	26	0.0106
07	0.0791	17	0.0002	27	0.0001
08	0.0038	18	0.0051	28	0.0037
09	0.0204	19	0.0045	29	0.0050
10	0.0018	20	0.0009	30	0.2288

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

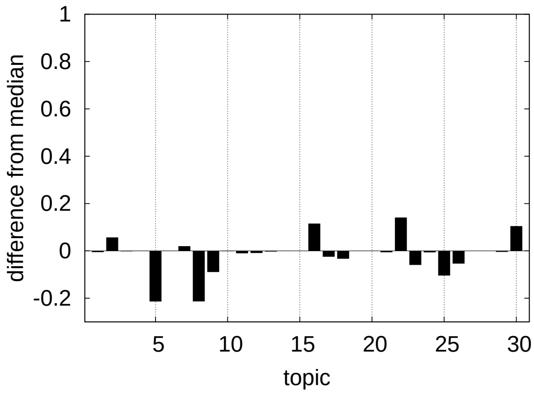


Overall average precision: 0.0404

Average precision per topic:

01	0.0033	11	0.0157	21	0.0028
02	0.1638	12	0.0027	22	0.3141
03	0.0170	13	0.0001	23	0.0295
04	0.0039	14	0.0023	24	0.0009
05	0.0492	15	0.0077	25	0.0015
06	0.0172	16	0.2808	26	0.0171
07	0.0656	17	0.0008	27	0.0001
08	0.0040	18	0.0075	28	0.0037
09	0.0194	19	0.0096	29	0.0216
10	0.0090	20	0.0009	30	0.1414

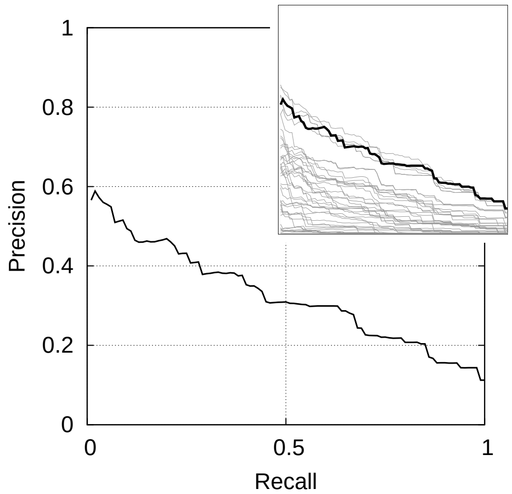
Difference from median in average precision per topic:



IBM Haifa Labs
ManualNoMerge (CAS)

Quantisation: strict	Quantisation: generalised

Recall / precision graph:

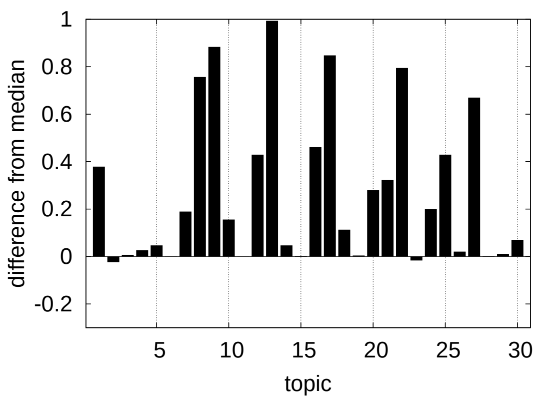


Overall average precision: 0.3248

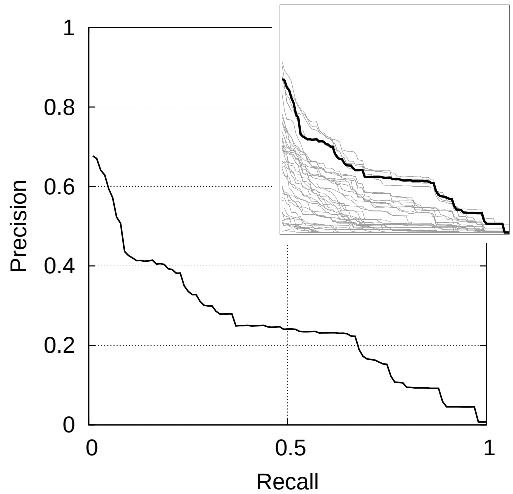
Average precision per topic:

01	0.3876	11	0.0067	21	0.3402
02	0.0801	12	0.4475	22	0.9302
03	0.0102	13	1.0000	23	0.0819
04	0.0299	14	0.0478	24	0.2036
05	0.3522	15	0.0032	25	0.5600
06	0.0013	16	0.6332	26	0.0673
07	0.2111	17	0.8501	27	0.6701
08	1.0000	18	0.2802	28	0.0058
09	1.0000	19	0.0086	29	0.0171
10	0.1577	20	0.2796	30	0.0797

Difference from median in average precision per topic:



Recall / precision graph:

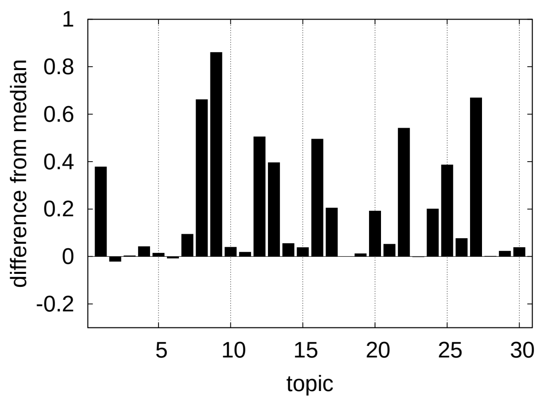


Overall average precision: 0.2535

Average precision per topic:

01	0.3876	11	0.0453	21	0.0618
02	0.0855	12	0.5175	22	0.7151
03	0.0231	13	0.3999	23	0.0865
04	0.0470	14	0.0589	24	0.2089
05	0.2782	15	0.0465	25	0.4930
06	0.0086	16	0.6615	26	0.1482
07	0.1405	17	0.2311	27	0.6701
08	0.8802	18	0.0407	28	0.0058
09	0.9700	19	0.0227	29	0.0496
10	0.0509	20	0.1936	30	0.0757

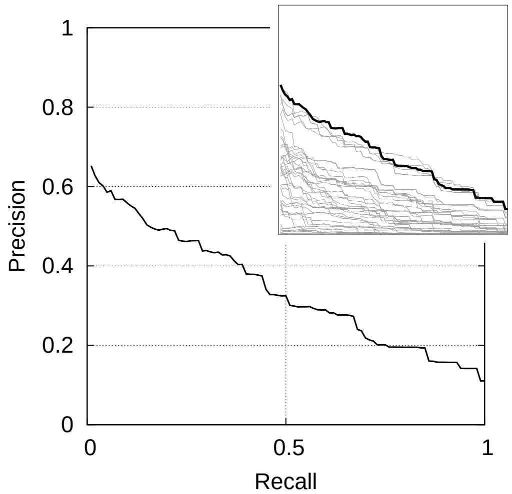
Difference from median in average precision per topic:



IBM Haifa Labs
Merge (CAS)

Quantisation: strict

Recall / precision graph:

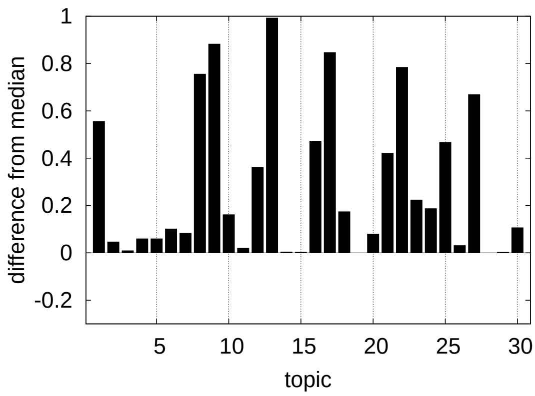


Overall average precision: 0.3411

Average precision per topic:

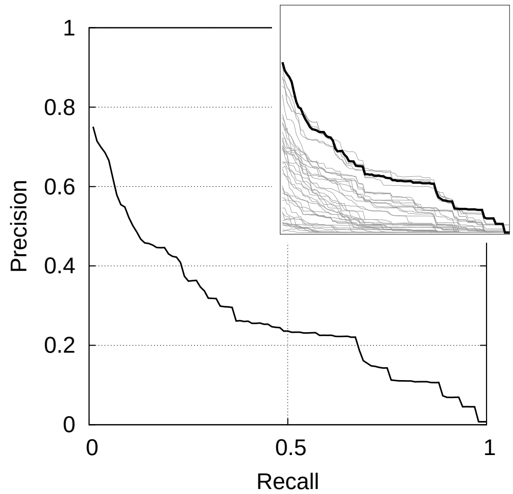
01	0.5655	11	0.0278	21	0.4402
02	0.1514	12	0.3810	22	0.9207
03	0.0130	13	1.0000	23	0.3234
04	0.0642	14	0.0061	24	0.1919
05	0.3659	15	0.0047	25	0.5988
06	0.1042	16	0.6457	26	0.0785
07	0.1055	17	0.8501	27	0.6701
08	1.0000	18	0.3423	28	0.0047
09	1.0000	19	0.0045	29	0.0094
10	0.1644	20	0.0808	30	0.1168

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

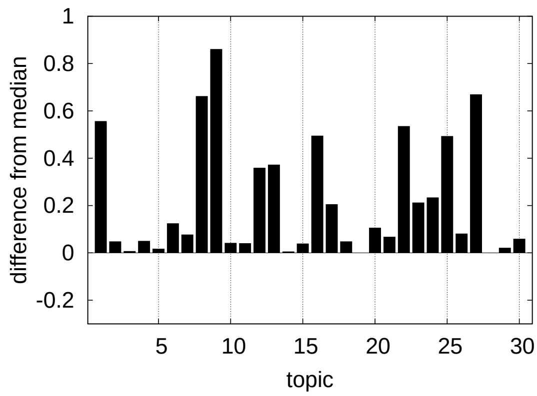


Overall average precision: 0.2706

Average precision per topic:

01	0.5655	11	0.0669	21	0.0769
02	0.1556	12	0.3714	22	0.7084
03	0.0266	13	0.3761	23	0.3016
04	0.0550	14	0.0085	24	0.2415
05	0.2805	15	0.0471	25	0.5994
06	0.1416	16	0.6606	26	0.1525
07	0.1232	17	0.2311	27	0.6701
08	0.8802	18	0.0895	28	0.0047
09	0.9700	19	0.0096	29	0.0475
10	0.0529	20	0.1071	30	0.0963

Difference from median in average precision per topic:

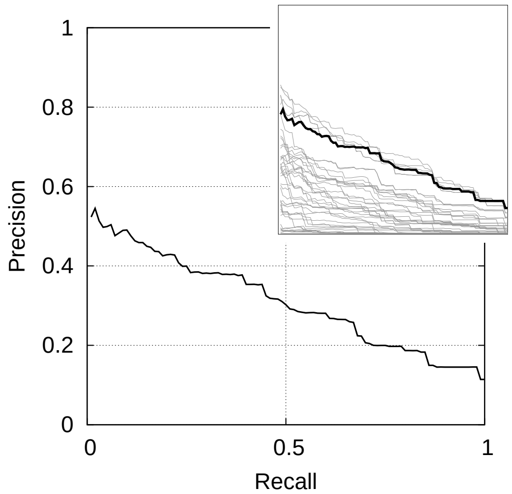


IBM Haifa Labs
NoMerge (CAS)

Quantisation: strict	Quantisation: generalised

Quantisation: strict

Recall / precision graph:

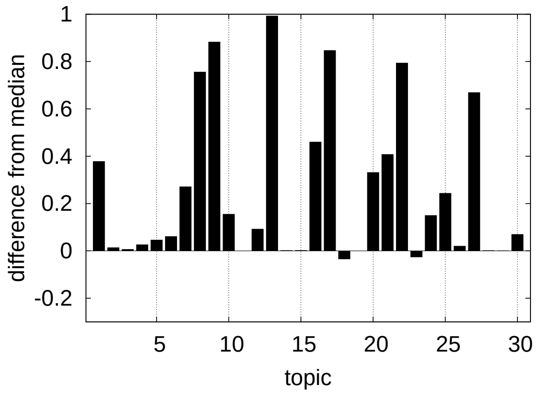


Overall average precision: 0.3093

Average precision per topic:

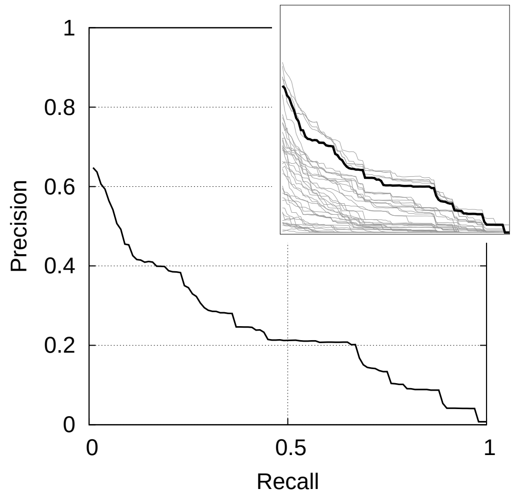
01	0.3876	11	0.0067	21	0.4264
02	0.1184	12	0.1110	22	0.9302
03	0.0102	13	1.0000	23	0.0719
04	0.0304	14	0.0033	24	0.1542
05	0.3522	15	0.0032	25	0.3749
06	0.0637	16	0.6332	26	0.0676
07	0.2933	17	0.8501	27	0.6701
08	1.0000	18	0.1322	28	0.0058
09	1.0000	19	0.0045	29	0.0073
10	0.1577	20	0.3325	30	0.0797

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

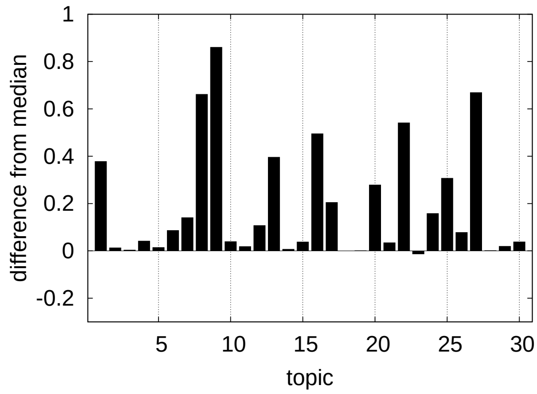


Overall average precision: 0.2419

Average precision per topic:

01	0.3876	11	0.0453	21	0.0441
02	0.1208	12	0.1196	22	0.7151
03	0.0236	13	0.3999	23	0.0750
04	0.0467	14	0.0108	24	0.1662
05	0.2782	15	0.0465	25	0.4136
06	0.1039	16	0.6615	26	0.1499
07	0.1871	17	0.2311	27	0.6701
08	0.8802	18	0.0411	28	0.0058
09	0.9700	19	0.0116	29	0.0460
10	0.0509	20	0.2803	30	0.0757

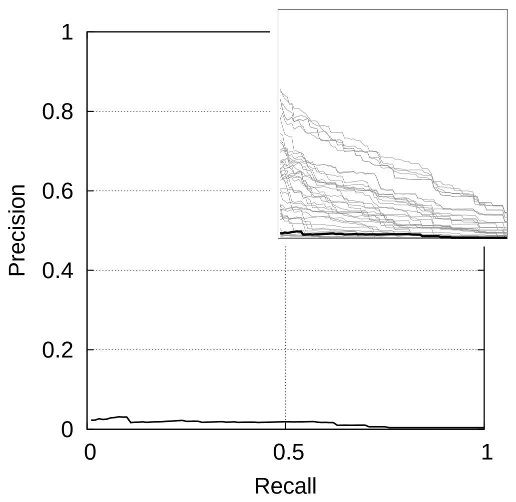
Difference from median in average precision per topic:



Institut de Recherche en Informatique de Toulouse (IRIT)
Mercure1 (CAS)

Quantisation: strict

Recall / precision graph:

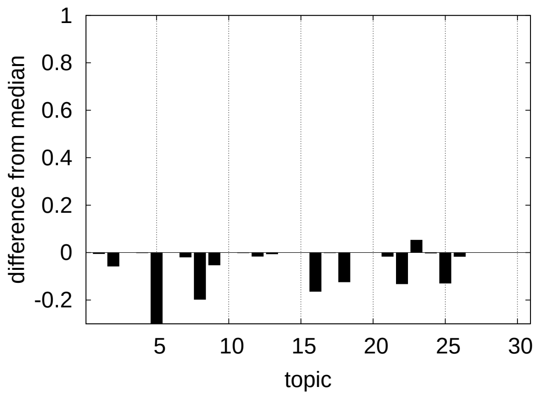


Overall average precision: 0.0145

Average precision per topic:

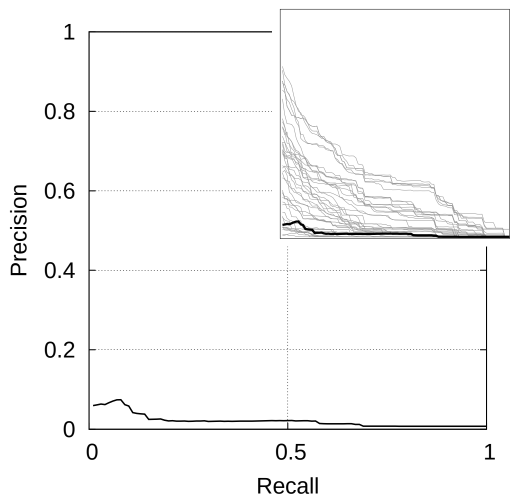
01	0.0030	11	0.0048	21	0.0008
02	0.0452	12	0.0013	22	0.0024
03	0.0025	13	0.0001	23	0.1523
04	0.0015	14	0.0004	24	0.0005
05	0.0053	15	0.0004	25	0.0005
06	0.0013	16	0.0073	26	0.0291
07	0.0015	17	0.0002	27	0.0001
08	0.0449	18	0.0427	28	0.0037
09	0.0630	19	0.0045	29	0.0050
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

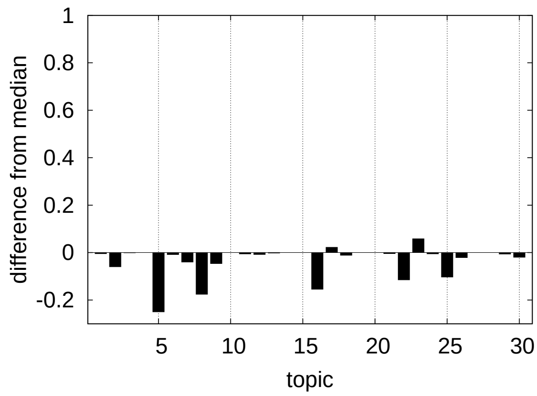


Overall average precision: 0.0212

Average precision per topic:

01	0.0030	11	0.0193	21	0.0036
02	0.0461	12	0.0025	22	0.0571
03	0.0171	13	0.0001	23	0.1482
04	0.0039	14	0.0024	24	0.0009
05	0.0122	15	0.0076	25	0.0015
06	0.0073	16	0.0097	26	0.0489
07	0.0049	17	0.0487	27	0.0001
08	0.0407	18	0.0286	28	0.0037
09	0.0613	19	0.0096	29	0.0184
10	0.0114	20	0.0006	30	0.0158

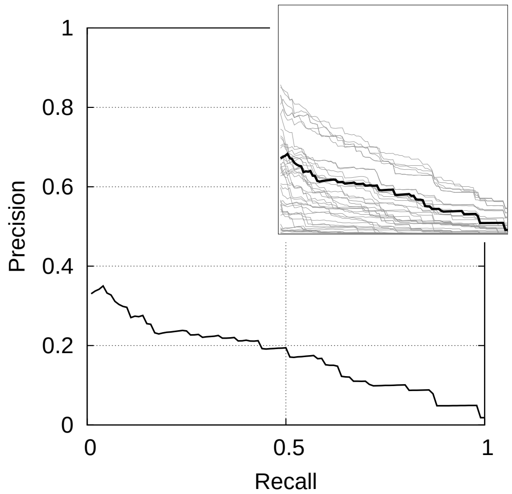
Difference from median in average precision per topic:



Nara Institute of Science and Technology
20020824-article (CAS)

Quantisation: strict

Recall / precision graph:

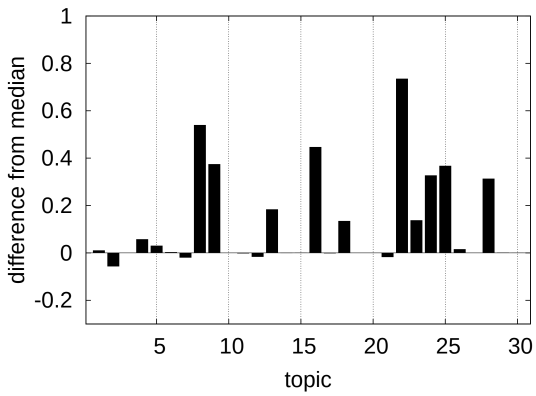


Overall average precision: 0.1736

Average precision per topic:

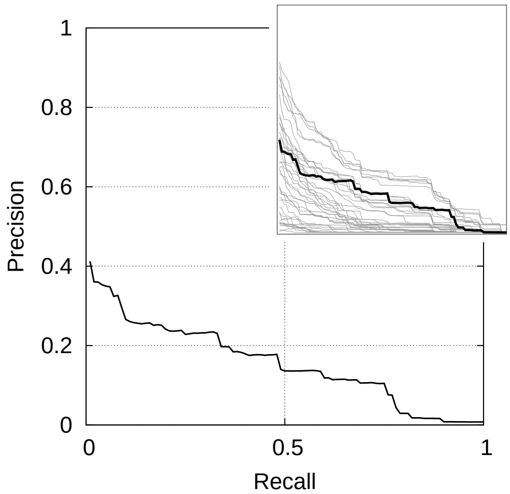
01	0.0194	11	0.0046	21	0.0002
02	0.0468	12	0.0013	22	0.8708
03	0.0029	13	0.1907	23	0.2365
04	0.0615	14	0.0004	24	0.3313
05	0.3362	15	0.0012	25	0.4982
06	0.0053	16	0.6195	26	0.0624
07	0.0015	17	0.0002	27	0.0001
08	0.7835	18	0.3022	28	0.3173
09	0.4912	19	0.0045	29	0.0071
10	0.0018	20	0.0002	30	0.0092

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

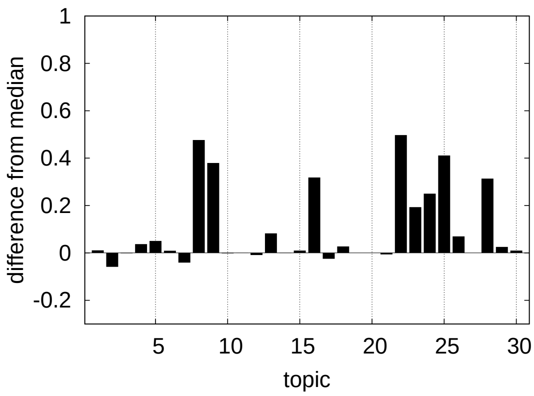


Overall average precision: 0.1548

Average precision per topic:

01	0.0194	11	0.0255	21	0.0025
02	0.0480	12	0.0025	22	0.6702
03	0.0179	13	0.0857	23	0.2820
04	0.0413	14	0.0024	24	0.2576
05	0.3135	15	0.0177	25	0.5171
06	0.0256	16	0.4837	26	0.1406
07	0.0049	17	0.0008	27	0.0001
08	0.6941	18	0.0681	28	0.3173
09	0.4884	19	0.0098	29	0.0510
10	0.0090	20	0.0006	30	0.0465

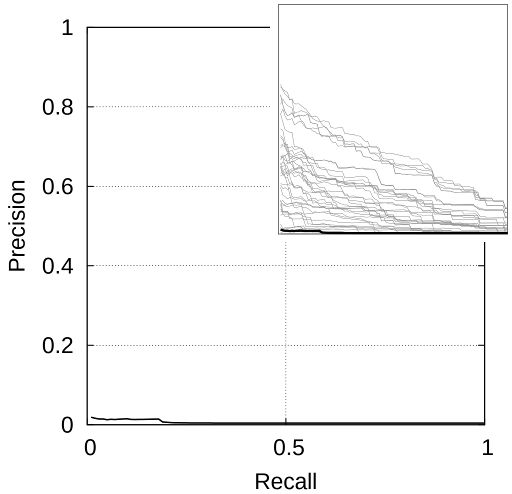
Difference from median in average precision per topic:



Queen Mary University of London
QMUL1 (CAS)

Quantisation: strict

Recall / precision graph:

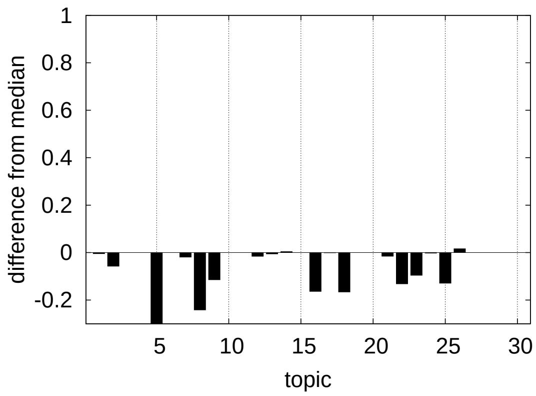


Overall average precision: 0.0060

Average precision per topic:

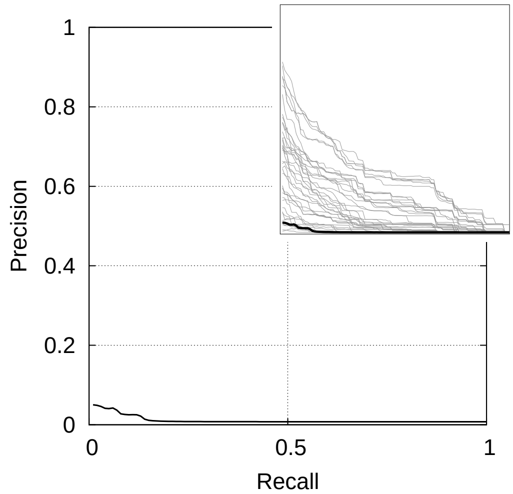
01	0.0030	11	0.0063	21	0.0015
02	0.0452	12	0.0013	22	0.0024
03	0.0025	13	0.0001	23	0.0019
04	0.0030	14	0.0061	24	0.0005
05	0.0053	15	0.0004	25	0.0005
06	0.0013	16	0.0073	26	0.0637
07	0.0015	17	0.0002	27	0.0001
08	0.0006	18	0.0002	28	0.0037
09	0.0008	19	0.0054	29	0.0052
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

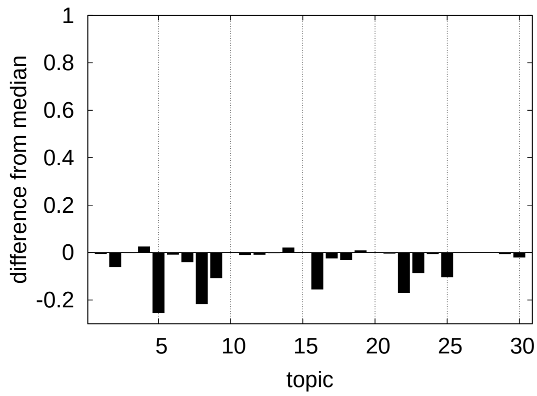


Overall average precision: 0.0114

Average precision per topic:

01	0.0030	11	0.0160	21	0.0045
02	0.0461	12	0.0025	22	0.0032
03	0.0170	13	0.0001	23	0.0024
04	0.0300	14	0.0242	24	0.0009
05	0.0085	15	0.0076	25	0.0015
06	0.0081	16	0.0097	26	0.0696
07	0.0049	17	0.0008	27	0.0001
08	0.0007	18	0.0103	28	0.0037
09	0.0008	19	0.0191	29	0.0190
10	0.0104	20	0.0006	30	0.0158

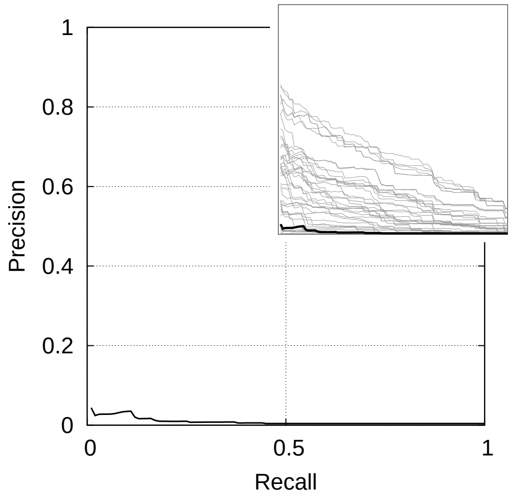
Difference from median in average precision per topic:



Queen Mary University of London
QMUL2 (CAS)

Quantisation: strict

Recall / precision graph:

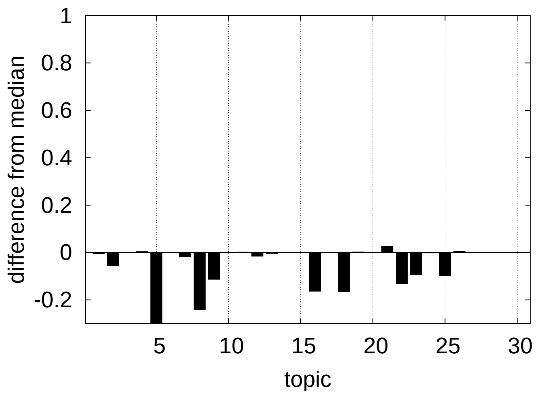


Overall average precision: 0.0088

Average precision per topic:

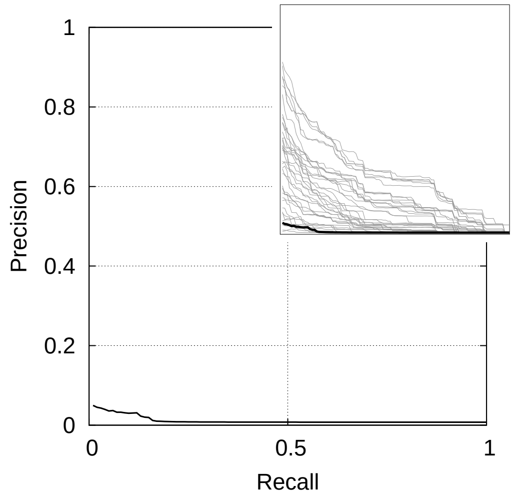
01	0.0030	11	0.0103	21	0.0462
02	0.0477	12	0.0013	22	0.0024
03	0.0049	13	0.0001	23	0.0035
04	0.0086	14	0.0014	24	0.0005
05	0.0053	15	0.0004	25	0.0317
06	0.0013	16	0.0073	26	0.0533
07	0.0030	17	0.0002	27	0.0001
08	0.0006	18	0.0011	28	0.0037
09	0.0022	19	0.0083	29	0.0052
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

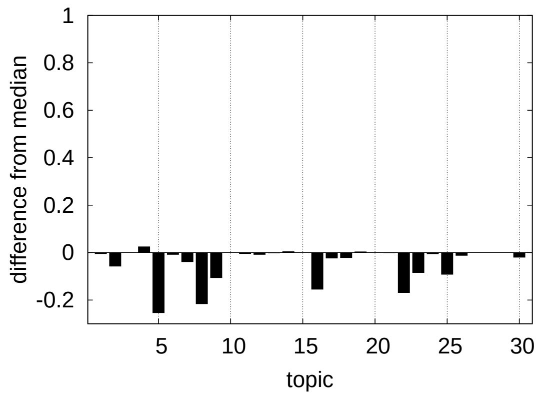


Overall average precision: 0.0117

Average precision per topic:

01	0.0030	11	0.0207	21	0.0070
02	0.0486	12	0.0025	22	0.0032
03	0.0202	13	0.0001	23	0.0037
04	0.0298	14	0.0081	24	0.0009
05	0.0085	15	0.0076	25	0.0131
06	0.0074	16	0.0097	26	0.0579
07	0.0063	17	0.0010	27	0.0001
08	0.0007	18	0.0183	28	0.0037
09	0.0020	19	0.0141	29	0.0253
10	0.0098	20	0.0006	30	0.0158

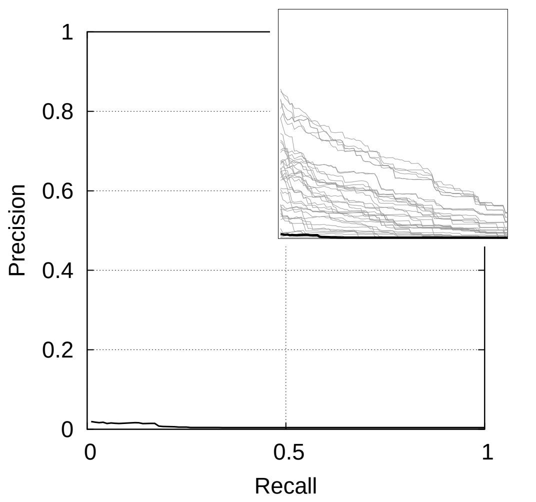
Difference from median in average precision per topic:



Queen Mary University of London
QMUL3 (CAS)

Quantisation: strict

Recall / precision graph:

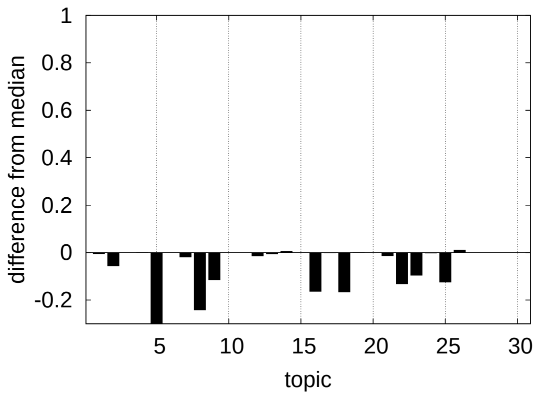


Overall average precision: 0.0063

Average precision per topic:

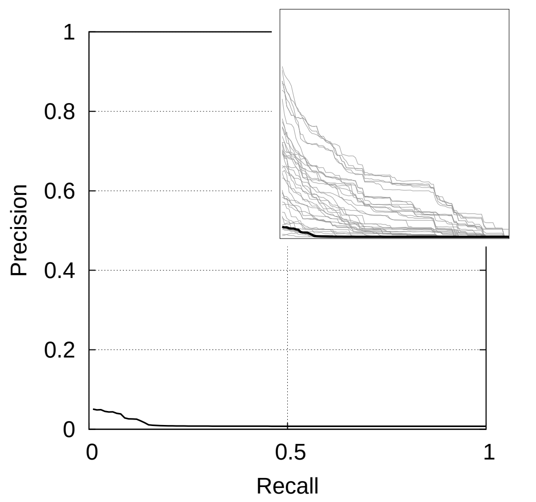
01	0.0030	11	0.0061	21	0.0034
02	0.0466	12	0.0023	22	0.0024
03	0.0025	13	0.0001	23	0.0019
04	0.0052	14	0.0078	24	0.0005
05	0.0053	15	0.0004	25	0.0047
06	0.0013	16	0.0073	26	0.0583
07	0.0015	17	0.0002	27	0.0001
08	0.0006	18	0.0002	28	0.0037
09	0.0008	19	0.0065	29	0.0052
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

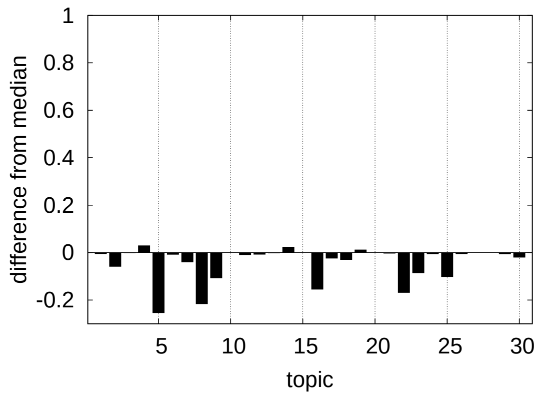


Overall average precision: 0.0117

Average precision per topic:

01	0.0030	11	0.0160	21	0.0050
02	0.0475	12	0.0031	22	0.0033
03	0.0170	13	0.0001	23	0.0024
04	0.0344	14	0.0273	24	0.0009
05	0.0085	15	0.0076	25	0.0032
06	0.0080	16	0.0097	26	0.0651
07	0.0049	17	0.0008	27	0.0001
08	0.0007	18	0.0103	28	0.0037
09	0.0008	19	0.0224	29	0.0190
10	0.0101	20	0.0006	30	0.0158

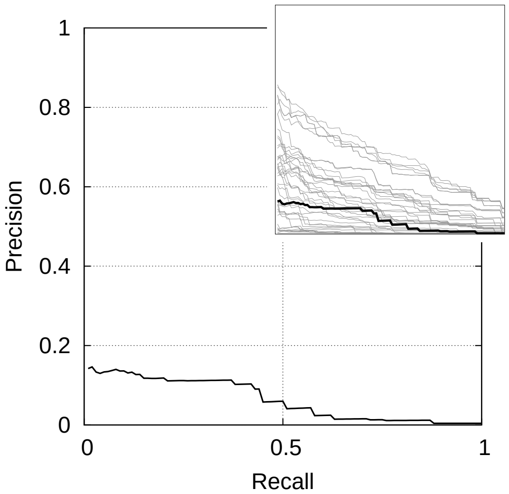
Difference from median in average precision per topic:



Queensland University of Technology
inexresult2.xml (CAS)

Quantisation: strict

Recall / precision graph:

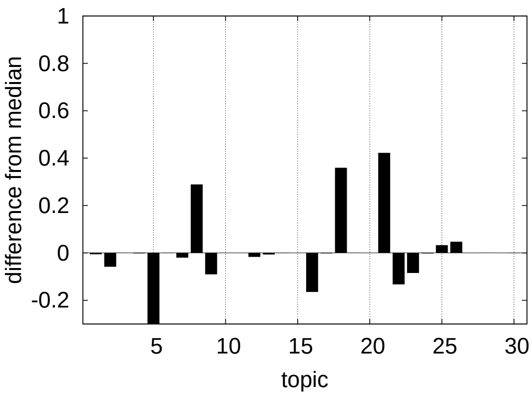


Overall average precision: 0.0634

Average precision per topic:

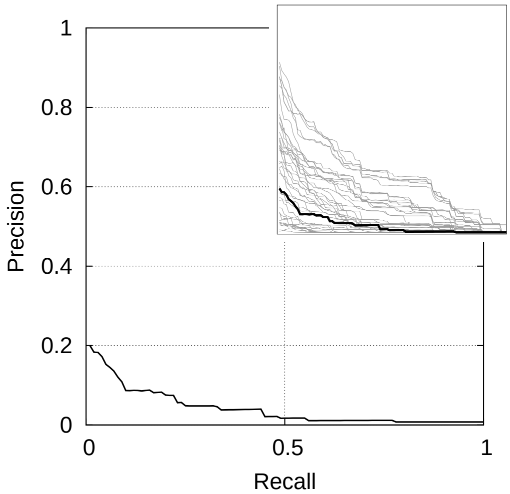
01	0.0030	11	0.0068	21	0.4402
02	0.0452	12	0.0013	22	0.0024
03	0.0026	13	0.0001	23	0.0139
04	0.0015	14	0.0004	24	0.0017
05	0.0053	15	0.0004	25	0.1631
06	0.0013	16	0.0073	26	0.0937
07	0.0015	17	0.0002	27	0.0001
08	0.5324	18	0.5266	28	0.0037
09	0.0260	19	0.0045	29	0.0058
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

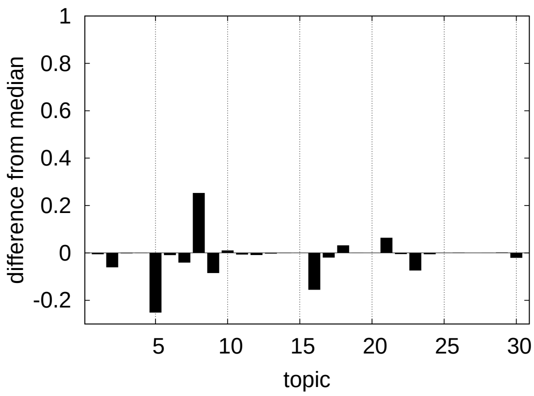


Overall average precision: 0.0407

Average precision per topic:

01	0.0030	11	0.0191	21	0.0730
02	0.0461	12	0.0025	22	0.1679
03	0.0174	13	0.0001	23	0.0147
04	0.0039	14	0.0024	24	0.0017
05	0.0113	15	0.0076	25	0.1056
06	0.0072	16	0.0097	26	0.0724
07	0.0049	17	0.0057	27	0.0001
08	0.4706	18	0.0727	28	0.0037
09	0.0237	19	0.0096	29	0.0278
10	0.0212	20	0.0006	30	0.0158

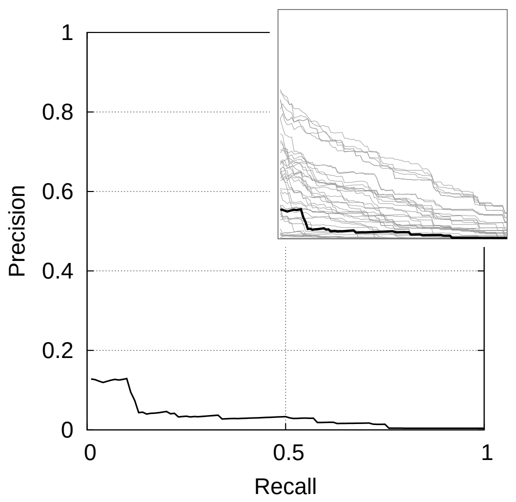
Difference from median in average precision per topic:



Queensland University of Technology
inexresults1.xml (CAS)

Quantisation: strict

Recall / precision graph:

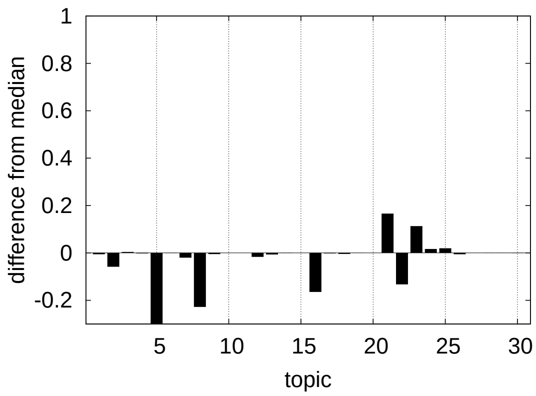


Overall average precision: 0.0335

Average precision per topic:

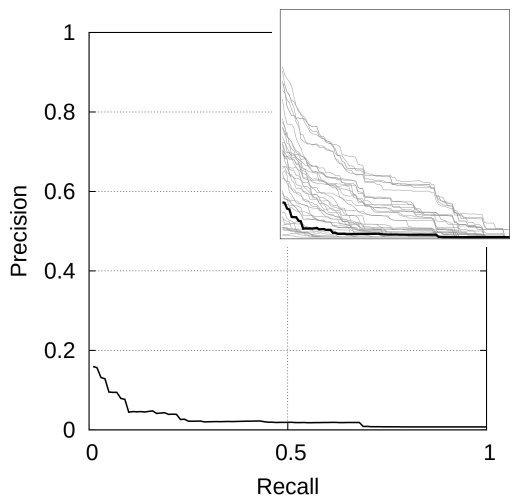
01	0.0030	11	0.0068	21	0.1838
02	0.0452	12	0.0013	22	0.0024
03	0.0072	13	0.0001	23	0.2117
04	0.0015	14	0.0004	24	0.0205
05	0.0053	15	0.0004	25	0.1501
06	0.0013	16	0.0073	26	0.0408
07	0.0015	17	0.0002	27	0.0001
08	0.0155	18	0.1630	28	0.0037
09	0.1115	19	0.0045	29	0.0055
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

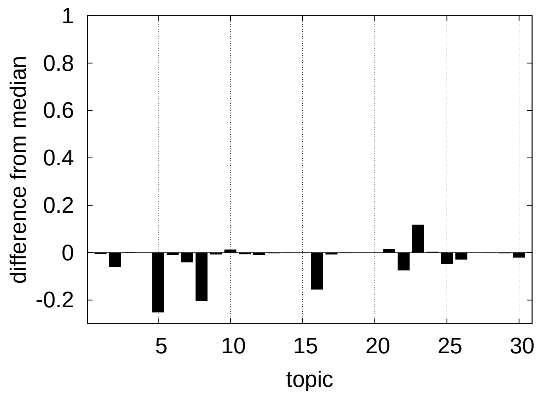


Overall average precision: 0.0276

Average precision per topic:

01	0.0030	11	0.0191	21	0.0246
02	0.0461	12	0.0025	22	0.0981
03	0.0206	13	0.0001	23	0.2069
04	0.0039	14	0.0024	24	0.0116
05	0.0107	15	0.0076	25	0.0587
06	0.0072	16	0.0097	26	0.0418
07	0.0049	17	0.0179	27	0.0001
08	0.0138	18	0.0381	28	0.0037
09	0.1011	19	0.0096	29	0.0230
10	0.0239	20	0.0006	30	0.0158

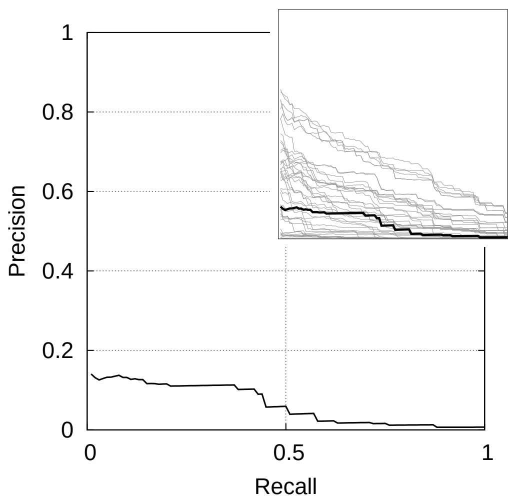
Difference from median in average precision per topic:



Queensland University of Technology
inexresults3.xml (CAS)

Quantisation: strict

Recall / precision graph:

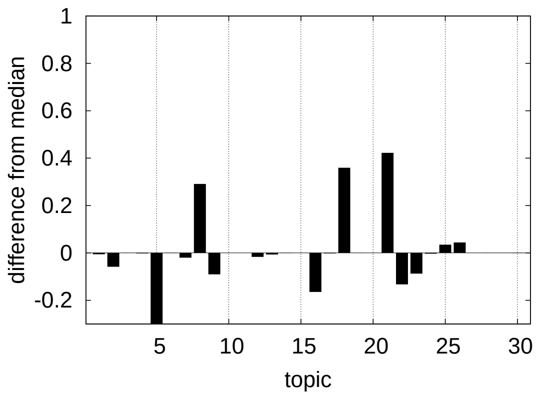


Overall average precision: 0.0633

Average precision per topic:

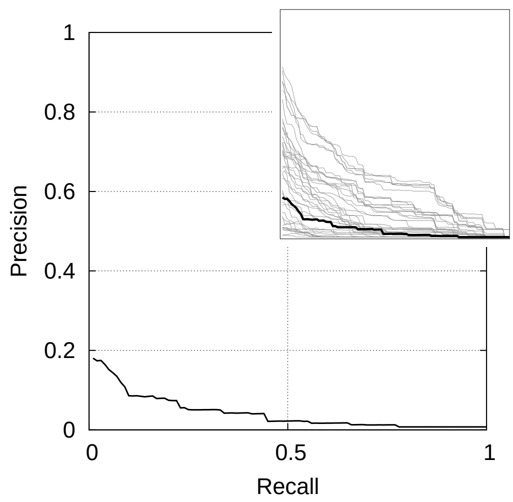
01	0.0030	11	0.0068	21	0.4402
02	0.0452	12	0.0013	22	0.0024
03	0.0026	13	0.0001	23	0.0111
04	0.0015	14	0.0004	24	0.0005
05	0.0053	15	0.0004	25	0.1653
06	0.0013	16	0.0073	26	0.0909
07	0.0015	17	0.0002	27	0.0001
08	0.5346	18	0.5266	28	0.0037
09	0.0260	19	0.0045	29	0.0057
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

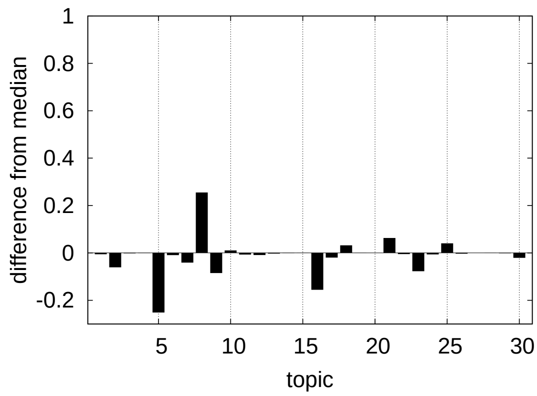


Overall average precision: 0.0417

Average precision per topic:

01	0.0030	11	0.0191	21	0.0719
02	0.0461	12	0.0025	22	0.1679
03	0.0174	13	0.0001	23	0.0117
04	0.0039	14	0.0024	24	0.0009
05	0.0115	15	0.0076	25	0.1463
06	0.0072	16	0.0097	26	0.0676
07	0.0049	17	0.0057	27	0.0001
08	0.4725	18	0.0727	28	0.0037
09	0.0237	19	0.0096	29	0.0246
10	0.0212	20	0.0006	30	0.0158

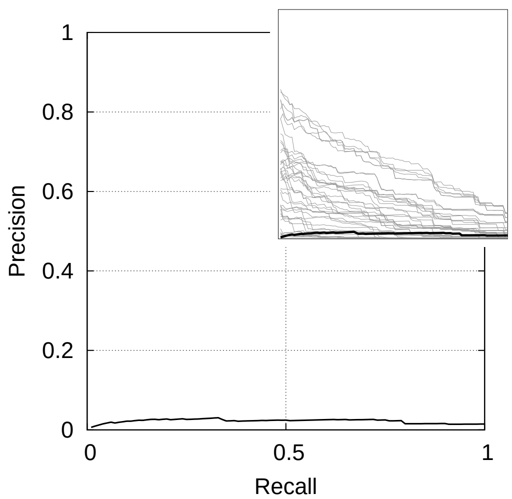
Difference from median in average precision per topic:



Salzburg Research Forschungsgesellschaft
1-corrected (CAS)

Quantisation: strict

Recall / precision graph:

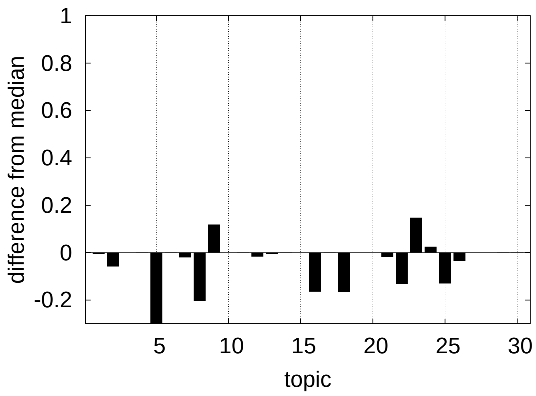


Overall average precision: 0.0221

Average precision per topic:

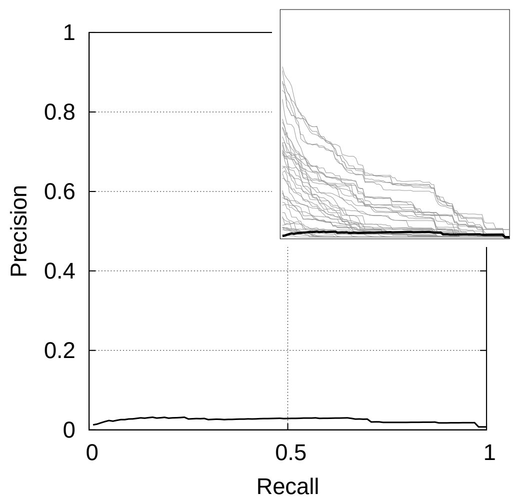
01	0.0030	11	0.0045	21	0.0002
02	0.0452	12	0.0013	22	0.0024
03	0.0030	13	0.0001	23	0.2465
04	0.0015	14	0.0004	24	0.0293
05	0.0053	15	0.0004	25	0.0005
06	0.0013	16	0.0073	26	0.0106
07	0.0015	17	0.0002	27	0.0001
08	0.0386	18	0.0002	28	0.0038
09	0.2351	19	0.0045	29	0.0050
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

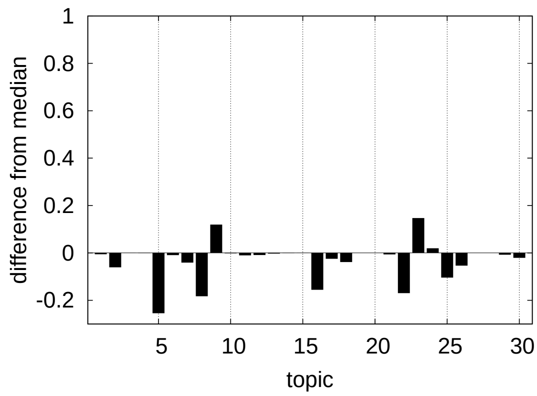


Overall average precision: 0.0247

Average precision per topic:

01	0.0030	11	0.0156	21	0.0025
02	0.0461	12	0.0025	22	0.0032
03	0.0189	13	0.0001	23	0.2360
04	0.0039	14	0.0024	24	0.0270
05	0.0085	15	0.0076	25	0.0015
06	0.0072	16	0.0097	26	0.0171
07	0.0049	17	0.0008	27	0.0001
08	0.0344	18	0.0021	28	0.0038
09	0.2282	19	0.0098	29	0.0181
10	0.0090	20	0.0006	30	0.0158

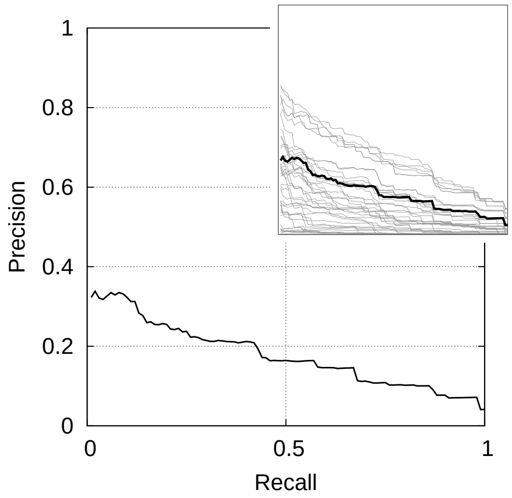
Difference from median in average precision per topic:



Sejong Cyber University
TitleKeywordsWLErr (CAS)

Quantisation: strict

Recall / precision graph:

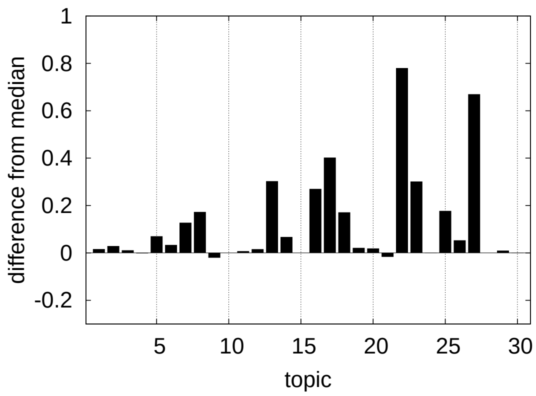


Overall average precision: 0.1777

Average precision per topic:

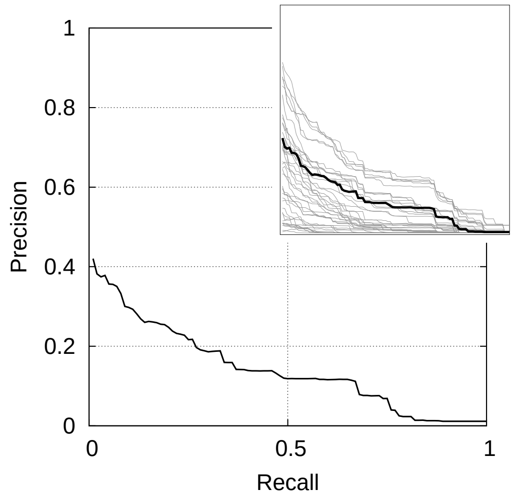
01	0.0250	11	0.0142	21	0.0010
02	0.1330	12	0.0339	22	0.9158
03	0.0140	13	0.3093	23	0.4003
04	0.0017	14	0.0685	24	0.0031
05	0.3757	15	0.0004	25	0.3079
06	0.0356	16	0.4425	26	0.0997
07	0.1492	17	0.4047	27	0.6701
08	0.4163	18	0.3387	28	0.0040
09	0.0960	19	0.0259	29	0.0157
10	0.0018	20	0.0193	30	0.0089

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

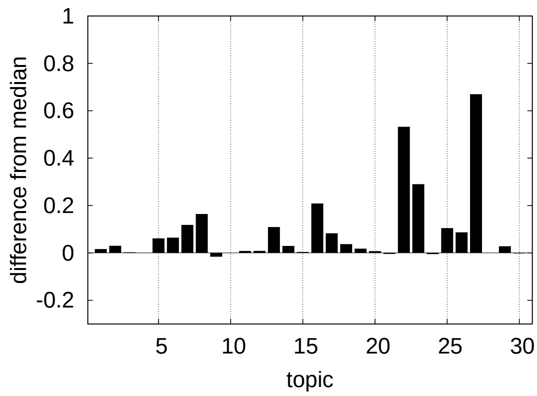


Overall average precision: 0.1424

Average precision per topic:

01	0.0250	11	0.0341	21	0.0044
02	0.1368	12	0.0202	22	0.7052
03	0.0218	13	0.1124	23	0.3791
04	0.0040	14	0.0322	24	0.0024
05	0.3244	15	0.0118	25	0.2106
06	0.0808	16	0.3742	26	0.1577
07	0.1639	17	0.1085	27	0.6701
08	0.3817	18	0.0780	28	0.0040
09	0.0926	19	0.0277	29	0.0536
10	0.0107	20	0.0080	30	0.0347

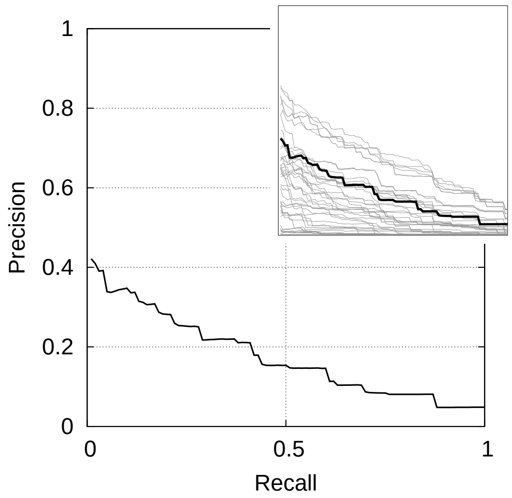
Difference from median in average precision per topic:



Tarragon Consulting Corporation
tgnCAS_base (CAS)

Quantisation: strict

Recall / precision graph:

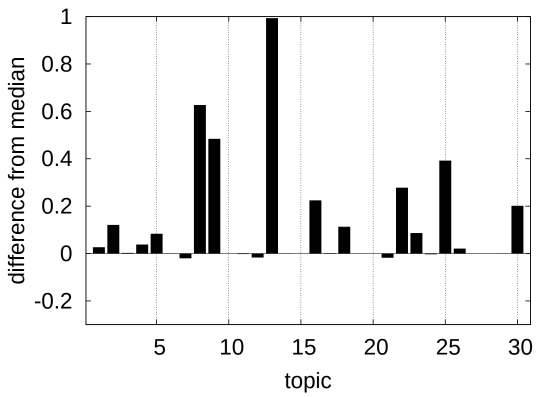


Overall average precision: 0.1757

Average precision per topic:

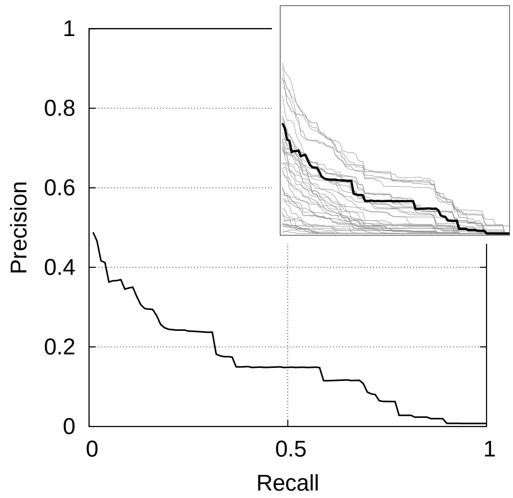
01	0.0354	11	0.0046	21	0.0002
02	0.2248	12	0.0013	22	0.4133
03	0.0057	13	1.0000	23	0.1852
04	0.0417	14	0.0004	24	0.0005
05	0.3889	15	0.0004	25	0.5229
06	0.0013	16	0.3967	26	0.0676
07	0.0015	17	0.0002	27	0.0001
08	0.8702	18	0.2802	28	0.0038
09	0.6006	19	0.0045	29	0.0051
10	0.0018	20	0.0002	30	0.2112

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

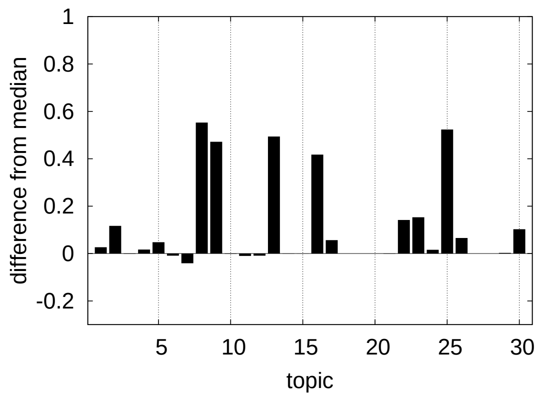


Overall average precision: 0.1583

Average precision per topic:

01	0.0354	11	0.0159	21	0.0080
02	0.2238	12	0.0025	22	0.3148
03	0.0179	13	0.4972	23	0.2422
04	0.0214	14	0.0024	24	0.0232
05	0.3110	15	0.0076	25	0.6294
06	0.0074	16	0.5829	26	0.1366
07	0.0049	17	0.0823	27	0.0001
08	0.7703	18	0.0409	28	0.0038
09	0.5807	19	0.0098	29	0.0285
10	0.0095	20	0.0006	30	0.1394

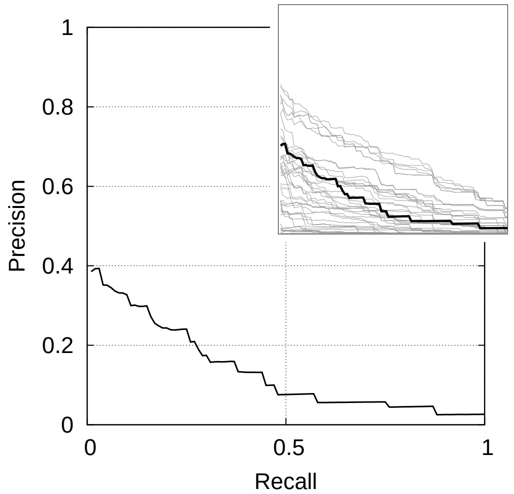
Difference from median in average precision per topic:



Universität Bayreuth
IRStream (CAS)

Quantisation: strict

Recall / precision graph:

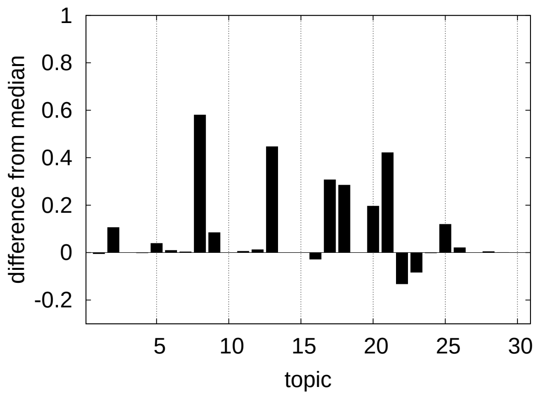


Overall average precision: 0.1346

Average precision per topic:

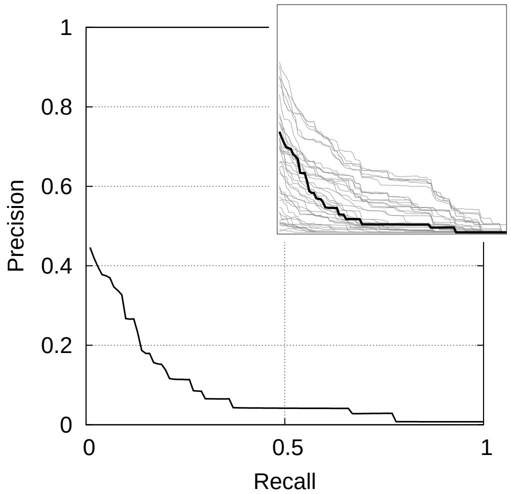
01	0.0030	11	0.0133	21	0.4402
02	0.2105	12	0.0314	22	0.0024
03	0.0026	13	0.4540	23	0.0147
04	0.0015	14	0.0004	24	0.0014
05	0.3453	15	0.0004	25	0.2506
06	0.0122	16	0.1431	26	0.0682
07	0.0256	17	0.3100	27	0.0001
08	0.8245	18	0.4528	28	0.0091
09	0.2016	19	0.0045	29	0.0072
10	0.0018	20	0.1972	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

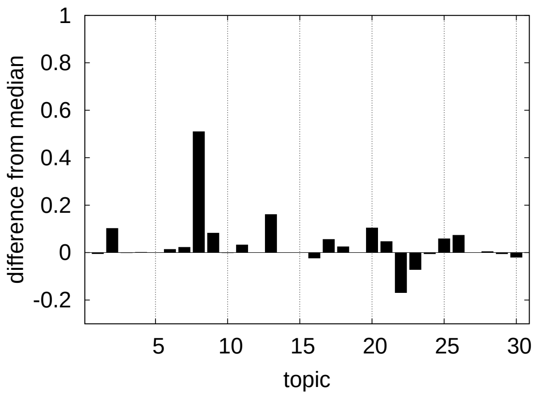


Overall average precision: 0.0871

Average precision per topic:

01	0.0030	11	0.0595	21	0.0563
02	0.2098	12	0.0127	22	0.0032
03	0.0176	13	0.1649	23	0.0162
04	0.0065	14	0.0024	24	0.0015
05	0.2627	15	0.0076	25	0.1651
06	0.0310	16	0.1413	26	0.1449
07	0.0689	17	0.0820	27	0.0001
08	0.7283	18	0.0664	28	0.0091
09	0.1920	19	0.0097	29	0.0199
10	0.0090	20	0.1057	30	0.0158

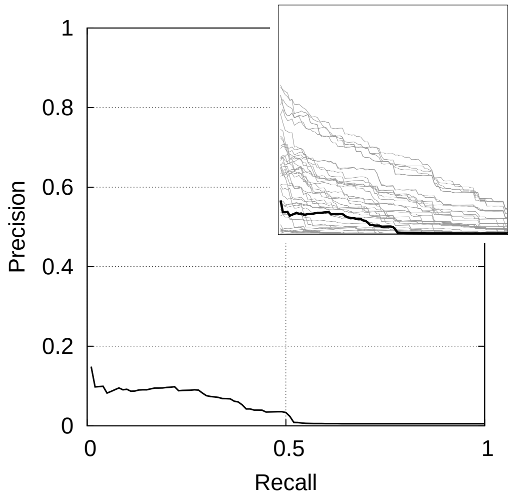
Difference from median in average precision per topic:



Universität Dortmund / Universität Duisburg-Essen
plain hyrex (CAS)

Quantisation: strict

Recall / precision graph:

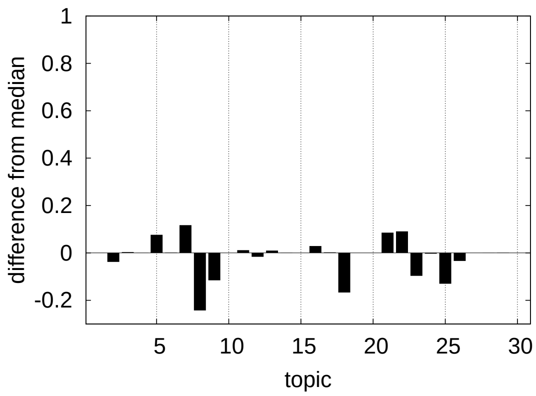


Overall average precision: 0.0409

Average precision per topic:

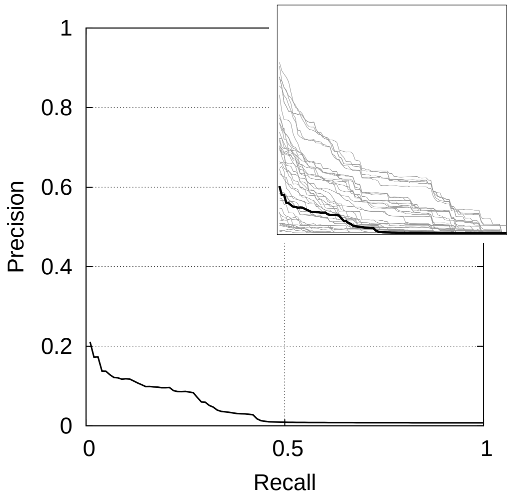
01	0.0090	11	0.0182	21	0.1033
02	0.0660	12	0.0018	22	0.2262
03	0.0065	13	0.0165	23	0.0019
04	0.0032	14	0.0017	24	0.0005
05	0.3818	15	0.0004	25	0.0005
06	0.0035	16	0.2011	26	0.0127
07	0.1387	17	0.0045	27	0.0001
08	0.0006	18	0.0002	28	0.0037
09	0.0008	19	0.0047	29	0.0068
10	0.0018	20	0.0002	30	0.0096

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

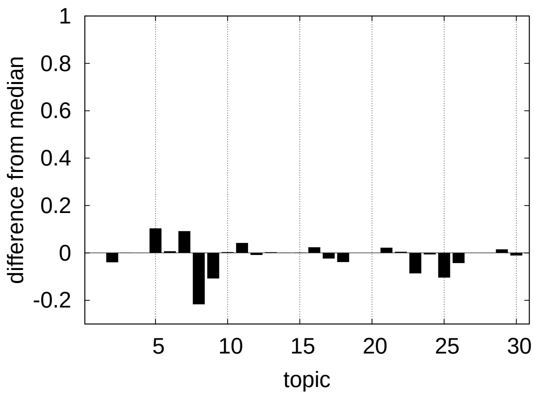


Overall average precision: 0.0417

Average precision per topic:

01	0.0090	11	0.0682	21	0.0308
02	0.0674	12	0.0029	22	0.1778
03	0.0200	13	0.0061	23	0.0025
04	0.0042	14	0.0040	24	0.0010
05	0.3669	15	0.0094	25	0.0015
06	0.0237	16	0.1895	26	0.0279
07	0.1377	17	0.0018	27	0.0001
08	0.0007	18	0.0021	28	0.0037
09	0.0008	19	0.0098	29	0.0410
10	0.0141	20	0.0006	30	0.0257

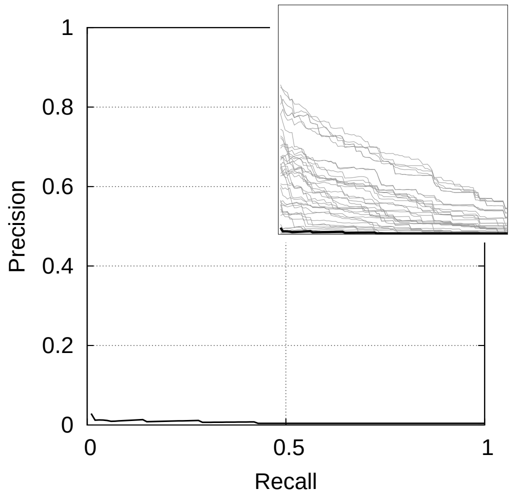
Difference from median in average precision per topic:



Université Pierre et Marie Curie
bayes-3 (CAS)

Quantisation: strict

Recall / precision graph:

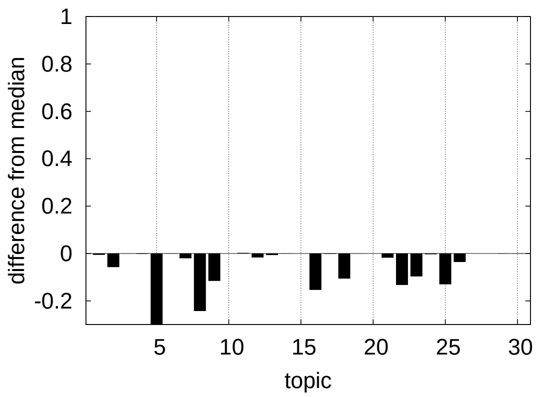


Overall average precision: 0.0065

Average precision per topic:

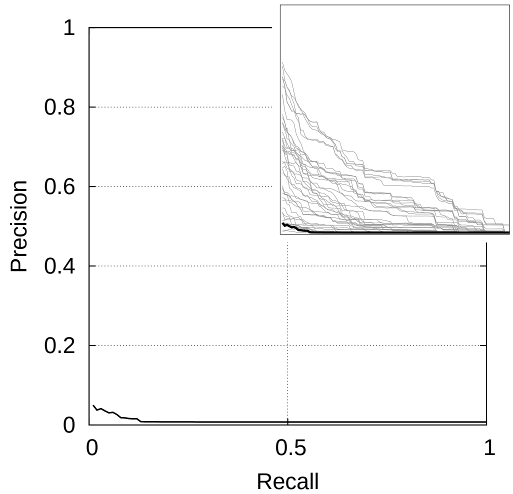
01	0.0030	11	0.0096	21	0.0002
02	0.0465	12	0.0013	22	0.0024
03	0.0026	13	0.0001	23	0.0019
04	0.0015	14	0.0004	24	0.0005
05	0.0054	15	0.0004	25	0.0005
06	0.0013	16	0.0185	26	0.0108
07	0.0015	17	0.0002	27	0.0001
08	0.0006	18	0.0615	28	0.0038
09	0.0008	19	0.0045	29	0.0050
10	0.0018	20	0.0002	30	0.0089

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

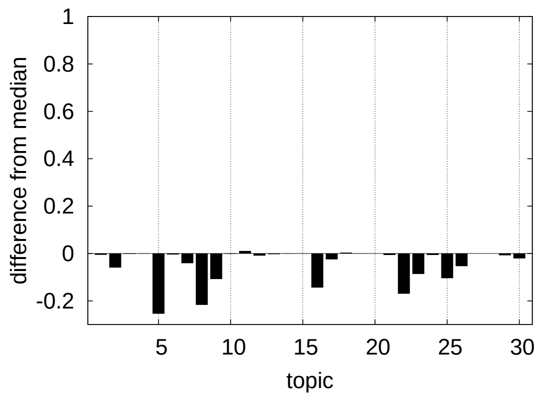


Overall average precision: 0.0100

Average precision per topic:

01	0.0030	11	0.0369	21	0.0025
02	0.0474	12	0.0025	22	0.0032
03	0.0171	13	0.0001	23	0.0025
04	0.0039	14	0.0023	24	0.0010
05	0.0087	15	0.0076	25	0.0015
06	0.0124	16	0.0214	26	0.0175
07	0.0049	17	0.0008	27	0.0001
08	0.0007	18	0.0445	28	0.0038
09	0.0008	19	0.0096	29	0.0181
10	0.0090	20	0.0006	30	0.0159

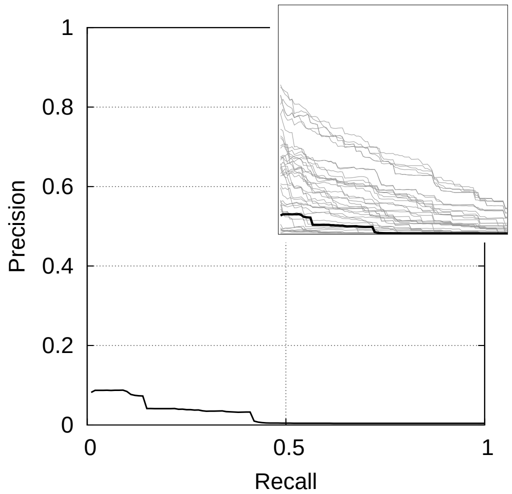
Difference from median in average precision per topic:



Université Pierre et Marie Curie
simple (CAS)

Quantisation: strict

Recall / precision graph:

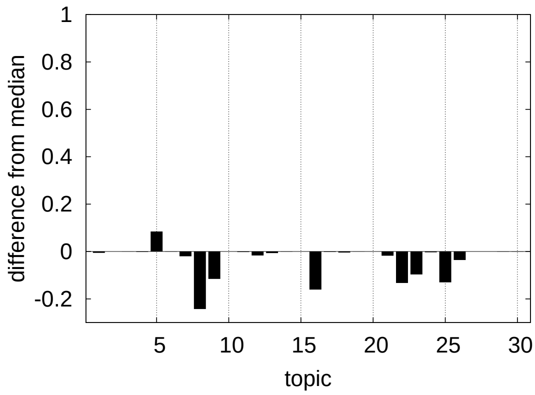


Overall average precision: 0.0243

Average precision per topic:

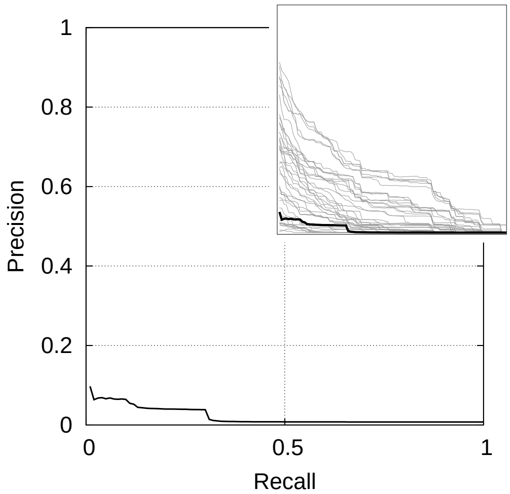
01	0.0030	11	0.0046	21	0.0002
02	0.1041	12	0.0013	22	0.0024
03	0.0026	13	0.0001	23	0.0019
04	0.0015	14	0.0004	24	0.0005
05	0.3900	15	0.0004	25	0.0005
06	0.0013	16	0.0116	26	0.0108
07	0.0015	17	0.0002	27	0.0001
08	0.0006	18	0.1630	28	0.0038
09	0.0008	19	0.0045	29	0.0050
10	0.0018	20	0.0002	30	0.0089

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

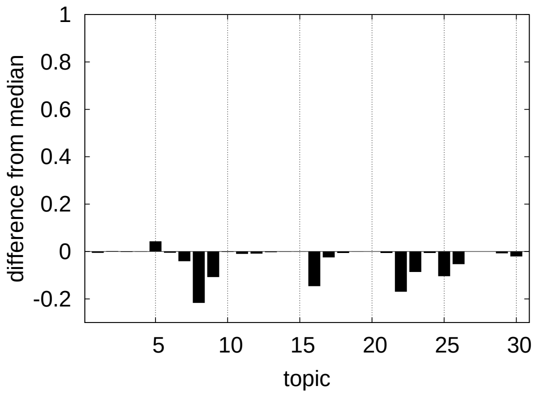


Overall average precision: 0.0208

Average precision per topic:

01	0.0030	11	0.0159	21	0.0025
02	0.1089	12	0.0025	22	0.0032
03	0.0171	13	0.0001	23	0.0025
04	0.0039	14	0.0023	24	0.0010
05	0.3060	15	0.0076	25	0.0015
06	0.0112	16	0.0191	26	0.0175
07	0.0049	17	0.0008	27	0.0001
08	0.0007	18	0.0346	28	0.0038
09	0.0008	19	0.0096	29	0.0181
10	0.0090	20	0.0006	30	0.0159

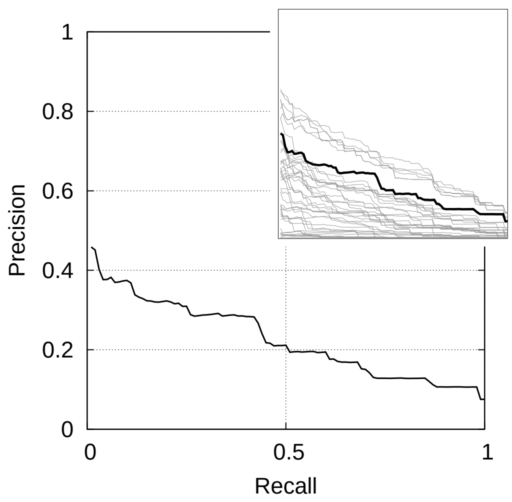
Difference from median in average precision per topic:



University of Amsterdam
UAmsI02NGiSt (CAS)

Quantisation: strict

Recall / precision graph:

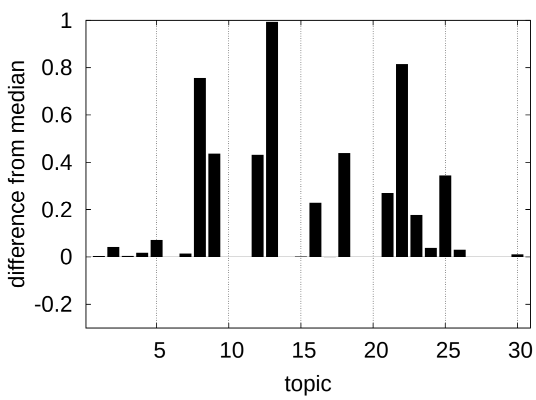


Overall average precision: 0.2257

Average precision per topic:

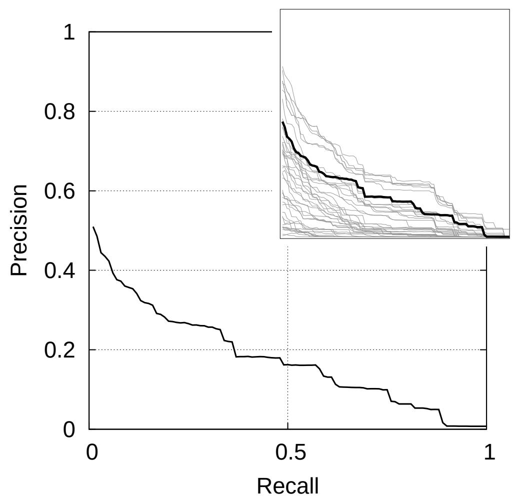
01	0.0121	11	0.0067	21	0.2886
02	0.1456	12	0.4503	22	0.9505
03	0.0074	13	1.0000	23	0.2769
04	0.0216	14	0.0004	24	0.0426
05	0.3767	15	0.0026	25	0.4748
06	0.0013	16	0.4016	26	0.0776
07	0.0360	17	0.0009	27	0.0001
08	1.0000	18	0.6062	28	0.0038
09	0.5532	19	0.0045	29	0.0065
10	0.0018	20	0.0002	30	0.0202

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

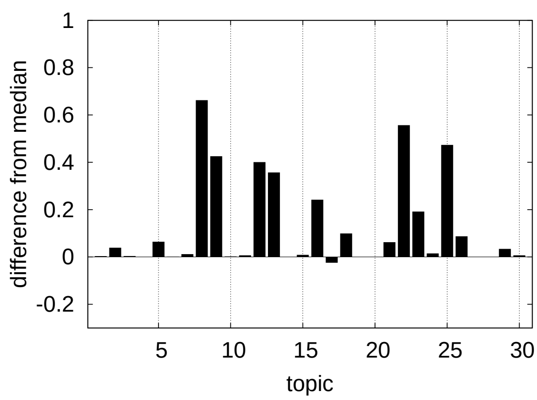


Overall average precision: 0.1782

Average precision per topic:

01	0.0121	11	0.0329	21	0.0711
02	0.1460	12	0.4124	22	0.7298
03	0.0228	13	0.3601	23	0.2806
04	0.0042	14	0.0024	24	0.0224
05	0.3273	15	0.0166	25	0.5792
06	0.0180	16	0.4073	26	0.1581
07	0.0575	17	0.0011	27	0.0001
08	0.8802	18	0.1402	28	0.0038
09	0.5343	19	0.0096	29	0.0597
10	0.0129	20	0.0006	30	0.0434

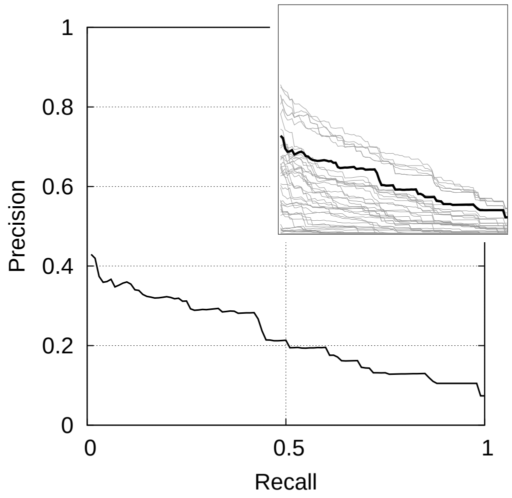
Difference from median in average precision per topic:



University of Amsterdam
UAmsI02NGram (CAS)

Quantisation: strict

Recall / precision graph:

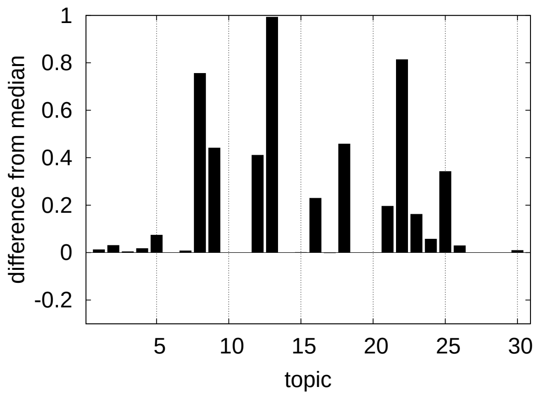


Overall average precision: 0.2233

Average precision per topic:

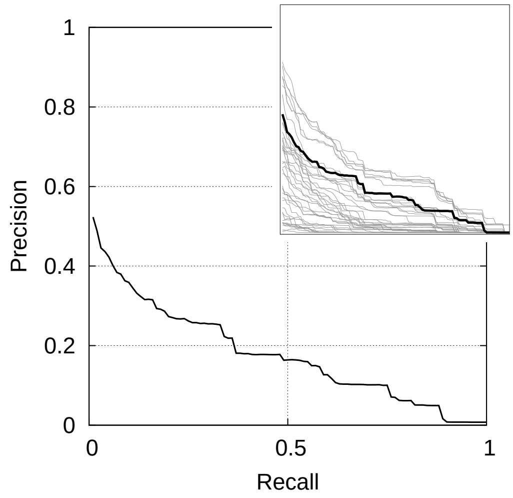
01	0.0220	11	0.0074	21	0.2145
02	0.1352	12	0.4296	22	0.9500
03	0.0077	13	1.0000	23	0.2611
04	0.0216	14	0.0004	24	0.0618
05	0.3802	15	0.0022	25	0.4733
06	0.0013	16	0.4022	26	0.0766
07	0.0300	17	0.0002	27	0.0001
08	1.0000	18	0.6263	28	0.0038
09	0.5587	19	0.0045	29	0.0063
10	0.0018	20	0.0002	30	0.0197

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

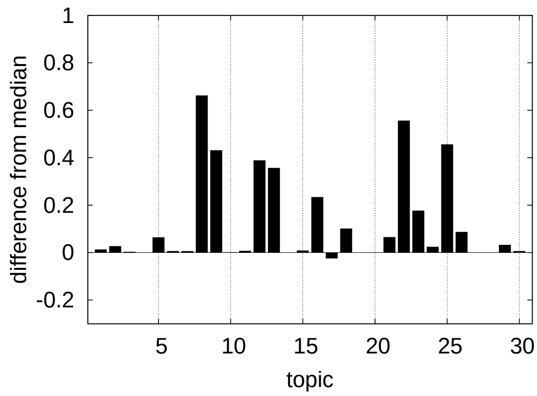


Overall average precision: 0.1770

Average precision per topic:

01	0.0220	11	0.0334	21	0.0744
02	0.1342	12	0.4005	22	0.7293
03	0.0225	13	0.3601	23	0.2660
04	0.0043	14	0.0024	24	0.0322
05	0.3275	15	0.0165	25	0.5619
06	0.0232	16	0.3996	26	0.1583
07	0.0519	17	0.0008	27	0.0001
08	0.8802	18	0.1421	28	0.0038
09	0.5401	19	0.0096	29	0.0584
10	0.0124	20	0.0006	30	0.0430

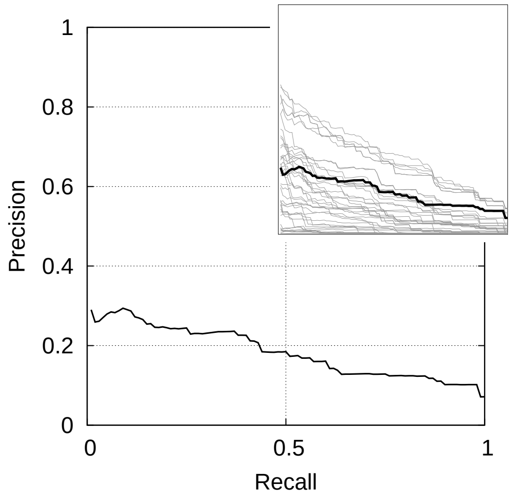
Difference from median in average precision per topic:



University of Amsterdam
UAmsI02Stem (CAS)

Quantisation: strict

Recall / precision graph:

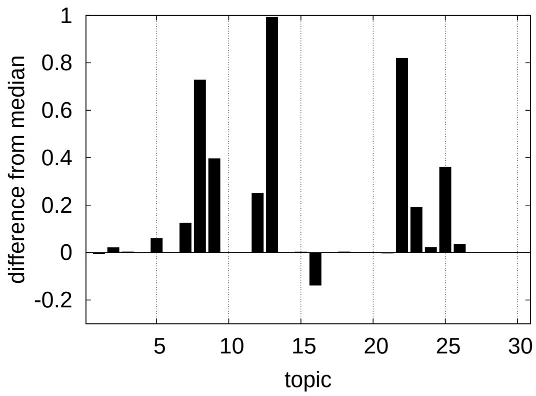


Overall average precision: 0.1839

Average precision per topic:

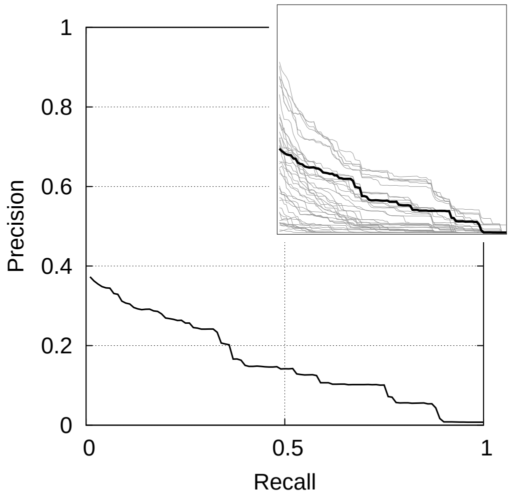
01	0.0033	11	0.0067	21	0.0140
02	0.1258	12	0.2686	22	0.9557
03	0.0069	13	1.0000	23	0.2913
04	0.0020	14	0.0004	24	0.0263
05	0.3660	15	0.0044	25	0.4920
06	0.0022	16	0.0333	26	0.0831
07	0.1474	17	0.0034	27	0.0001
08	0.9727	18	0.1715	28	0.0038
09	0.5134	19	0.0045	29	0.0070
10	0.0018	20	0.0002	30	0.0088

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

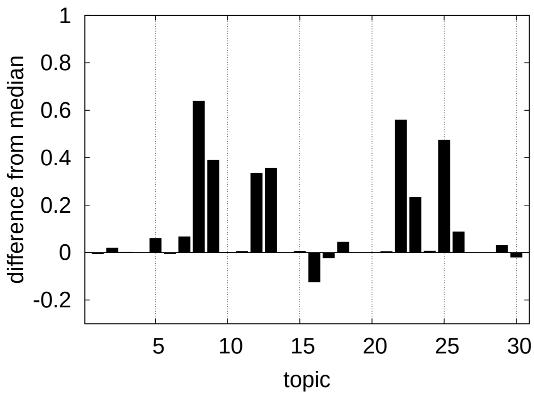


Overall average precision: 0.1592

Average precision per topic:

01	0.0033	11	0.0315	21	0.0141
02	0.1276	12	0.3474	22	0.7336
03	0.0226	13	0.3601	23	0.3220
04	0.0041	14	0.0024	24	0.0152
05	0.3234	15	0.0150	25	0.5813
06	0.0117	16	0.0403	26	0.1595
07	0.1134	17	0.0016	27	0.0001
08	0.8569	18	0.0865	28	0.0038
09	0.5002	19	0.0096	29	0.0579
10	0.0133	20	0.0006	30	0.0158

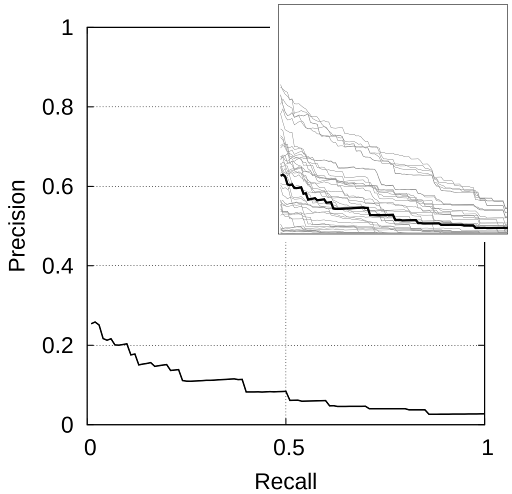
Difference from median in average precision per topic:



University of California, Berkeley
Berkeley01 (CAS)

Quantisation: strict

Recall / precision graph:

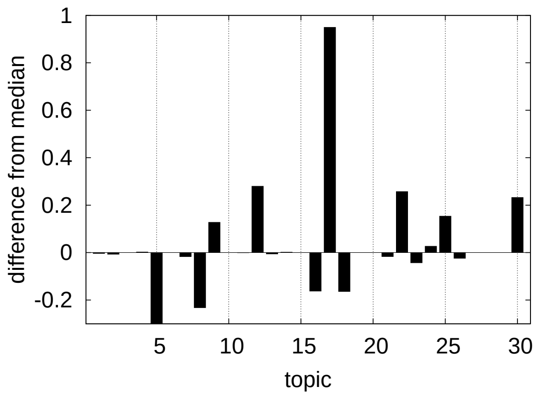


Overall average precision: 0.0897

Average precision per topic:

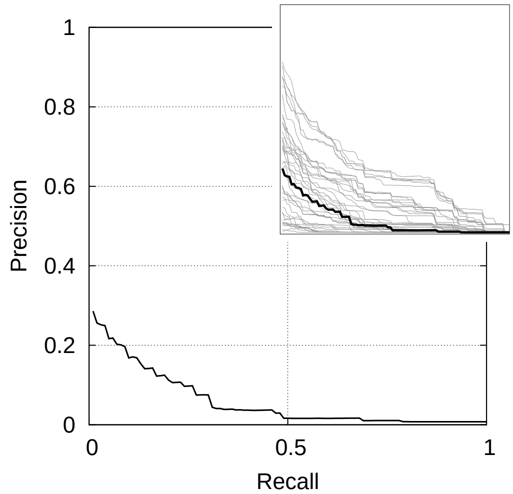
01	0.0040	11	0.0054	21	0.0002
02	0.0958	12	0.2988	22	0.3936
03	0.0025	13	0.0001	23	0.0545
04	0.0068	14	0.0036	24	0.0314
05	0.0053	15	0.0004	25	0.2849
06	0.0017	16	0.0088	26	0.0214
07	0.0033	17	0.9526	27	0.0001
08	0.0100	18	0.0020	28	0.0041
09	0.2450	19	0.0045	29	0.0052
10	0.0018	20	0.0002	30	0.2428

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

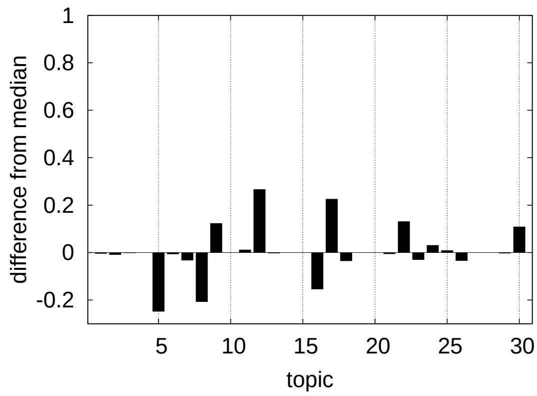


Overall average precision: 0.0583

Average precision per topic:

01	0.0040	11	0.0382	21	0.0028
02	0.0977	12	0.2784	22	0.3046
03	0.0171	13	0.0001	23	0.0584
04	0.0043	14	0.0036	24	0.0387
05	0.0146	15	0.0076	25	0.1153
06	0.0100	16	0.0109	26	0.0364
07	0.0127	17	0.2518	27	0.0001
08	0.0098	18	0.0052	28	0.0041
09	0.2327	19	0.0096	29	0.0226
10	0.0099	20	0.0006	30	0.1461

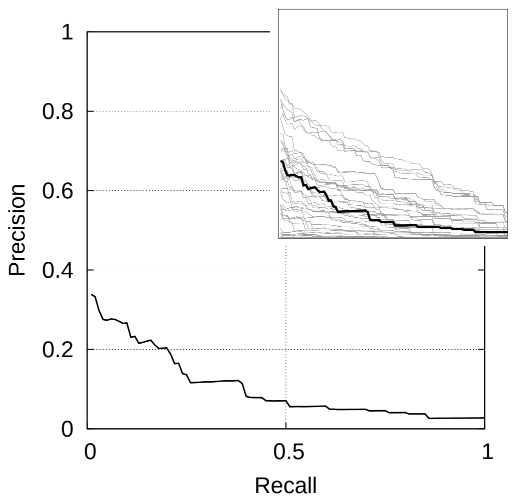
Difference from median in average precision per topic:



University of California, Berkeley
Berkeley02 (CAS)

Quantisation: strict

Recall / precision graph:

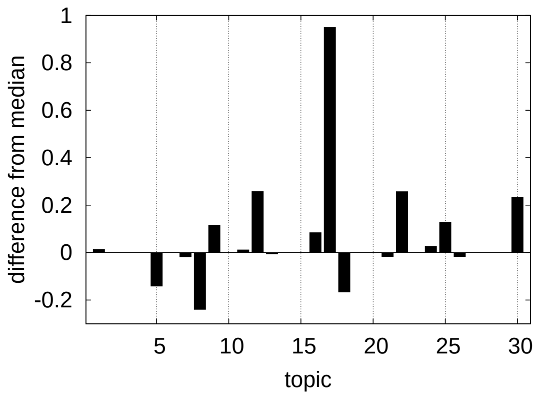


Overall average precision: 0.1038

Average precision per topic:

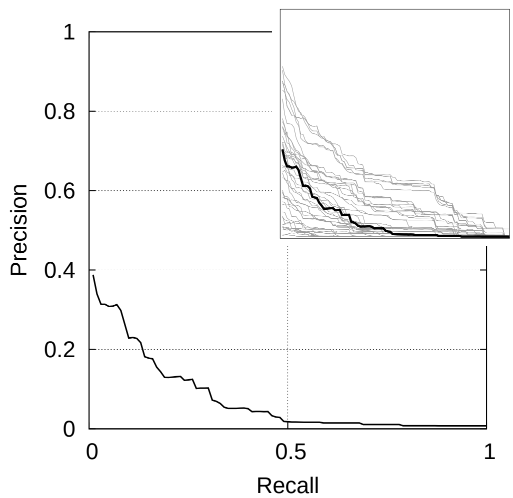
01	0.0231	11	0.0192	21	0.0002
02	0.1037	12	0.2764	22	0.3936
03	0.0025	13	0.0001	23	0.0978
04	0.0042	14	0.0004	24	0.0314
05	0.1628	15	0.0004	25	0.2597
06	0.0019	16	0.2573	26	0.0289
07	0.0029	17	0.9526	27	0.0001
08	0.0026	18	0.0002	28	0.0040
09	0.2331	19	0.0045	29	0.0052
10	0.0021	20	0.0002	30	0.2435

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

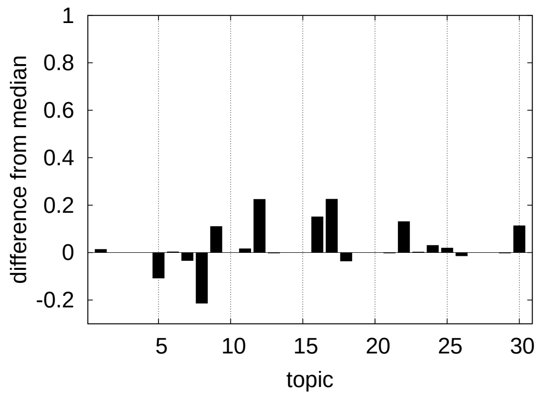


Overall average precision: 0.0749

Average precision per topic:

01	0.0231	11	0.0431	21	0.0060
02	0.1069	12	0.2369	22	0.3046
03	0.0193	13	0.0001	23	0.0922
04	0.0040	14	0.0024	24	0.0387
05	0.1544	15	0.0076	25	0.1261
06	0.0206	16	0.3172	26	0.0562
07	0.0113	17	0.2518	27	0.0001
08	0.0030	18	0.0044	28	0.0040
09	0.2198	19	0.0096	29	0.0228
10	0.0097	20	0.0006	30	0.1506

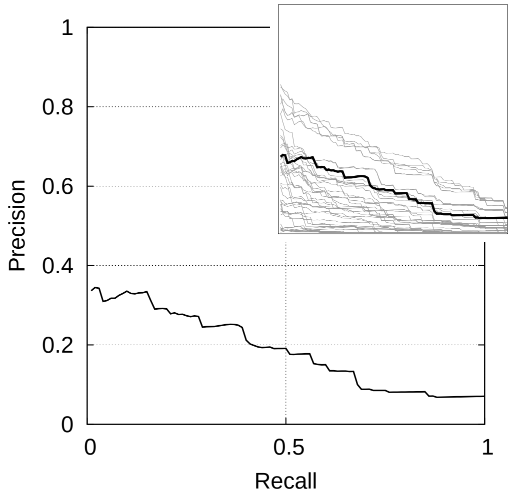
Difference from median in average precision per topic:



University of California, Berkeley
Berkeley03 (CAS)

Quantisation: strict

Recall / precision graph:

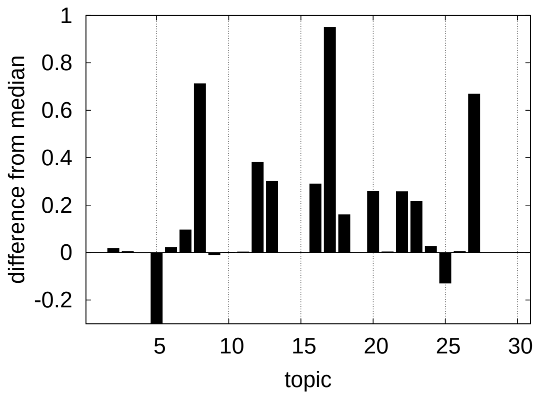


Overall average precision: 0.1865

Average precision per topic:

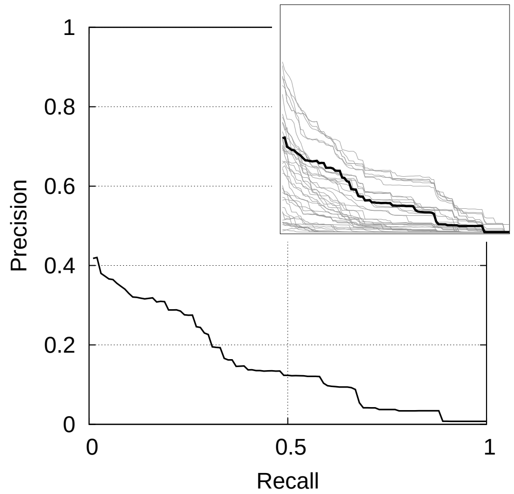
01	0.0083	11	0.0109	21	0.0223
02	0.1227	12	0.3999	22	0.3936
03	0.0084	13	0.3093	23	0.3164
04	0.0019	14	0.0019	24	0.0314
05	0.0054	15	0.0004	25	0.0005
06	0.0249	16	0.4628	26	0.0523
07	0.1184	17	0.9526	27	0.6701
08	0.9567	18	0.3283	28	0.0041
09	0.1063	19	0.0045	29	0.0062
10	0.0049	20	0.2601	30	0.0108

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

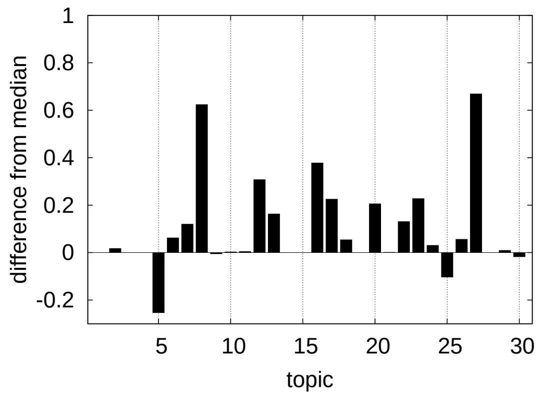


Overall average precision: 0.1513

Average precision per topic:

01	0.0083	11	0.0313	21	0.0109
02	0.1252	12	0.3201	22	0.3046
03	0.0188	13	0.1672	23	0.3176
04	0.0040	14	0.0033	24	0.0387
05	0.0087	15	0.0076	25	0.0015
06	0.0797	16	0.5443	26	0.1281
07	0.1667	17	0.2518	27	0.6701
08	0.8422	18	0.0956	28	0.0041
09	0.1028	19	0.0096	29	0.0361
10	0.0146	20	0.2071	30	0.0181

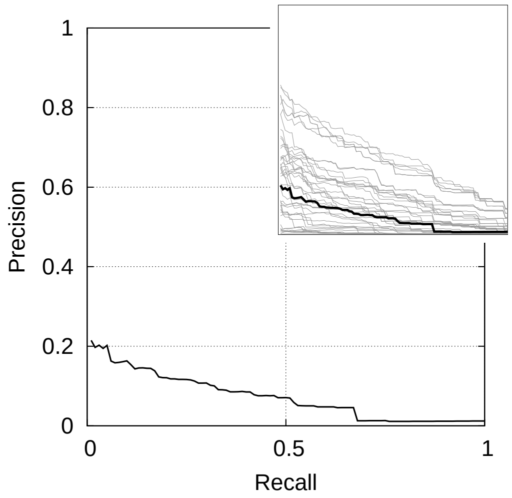
Difference from median in average precision per topic:



University of Melbourne
um_mgx21_short (CAS)

Quantisation: strict

Recall / precision graph:

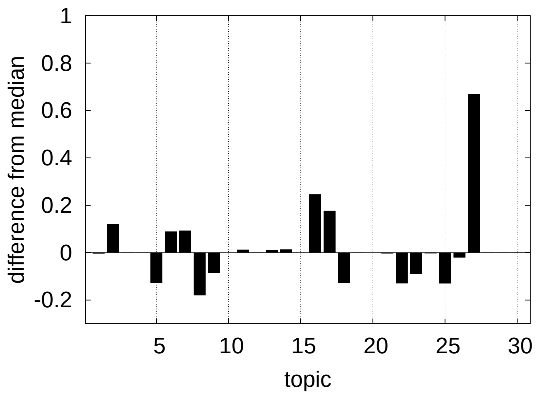


Overall average precision: 0.0723

Average precision per topic:

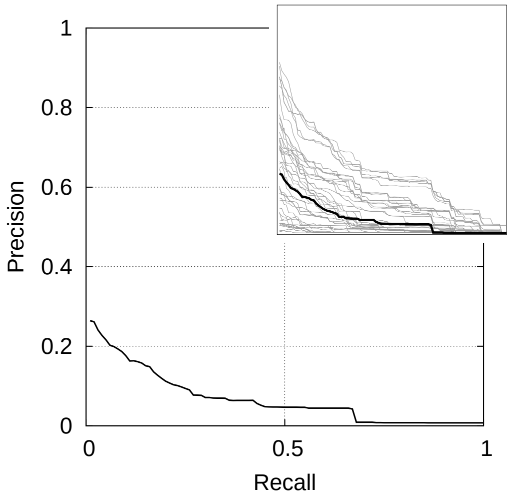
01	0.0047	11	0.0198	21	0.0141
02	0.2240	12	0.0162	22	0.0057
03	0.0026	13	0.0176	23	0.0082
04	0.0039	14	0.0150	24	0.0005
05	0.1775	15	0.0004	25	0.0005
06	0.0914	16	0.4186	26	0.0258
07	0.1148	17	0.1792	27	0.6701
08	0.0632	18	0.0386	28	0.0037
09	0.0310	19	0.0045	29	0.0061
10	0.0018	20	0.0002	30	0.0093

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

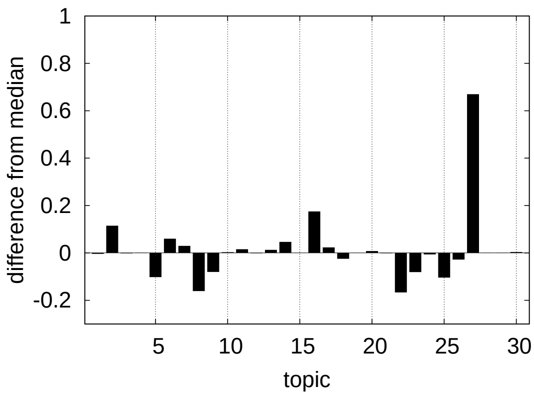


Overall average precision: 0.0672

Average precision per topic:

01	0.0047	11	0.0414	21	0.0075
02	0.2218	12	0.0104	22	0.0062
03	0.0175	13	0.0159	23	0.0079
04	0.0055	14	0.0495	24	0.0009
05	0.1608	15	0.0076	25	0.0015
06	0.0764	16	0.3406	26	0.0428
07	0.0755	17	0.0489	27	0.6701
08	0.0563	18	0.0161	28	0.0037
09	0.0284	19	0.0096	29	0.0267
10	0.0137	20	0.0083	30	0.0403

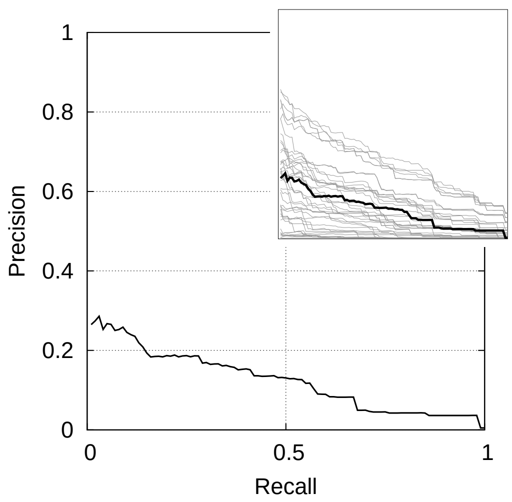
Difference from median in average precision per topic:



University of Melbourne
um_mgx26_long (CAS)

Quantisation: strict

Recall / precision graph:

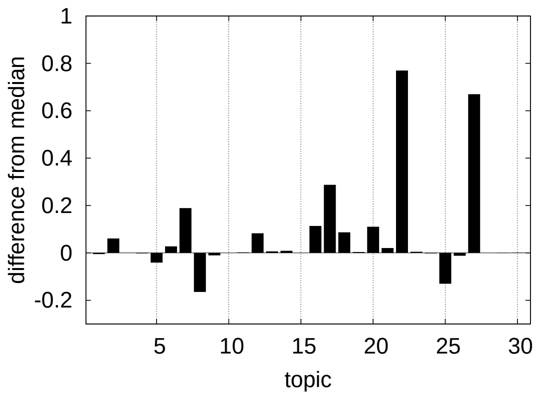


Overall average precision: 0.1240

Average precision per topic:

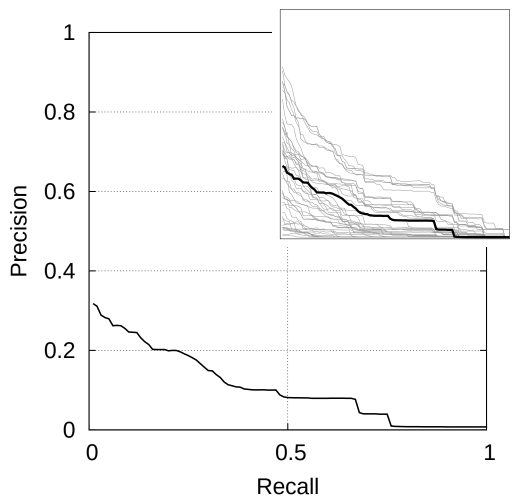
01	0.0036	11	0.0092	21	0.0384
02	0.1647	12	0.1009	22	0.9052
03	0.0025	13	0.0131	23	0.1034
04	0.0015	14	0.0098	24	0.0018
05	0.2643	15	0.0008	25	0.0005
06	0.0294	16	0.2861	26	0.0345
07	0.2107	17	0.2896	27	0.6701
08	0.0787	18	0.2540	28	0.0040
09	0.1060	19	0.0084	29	0.0050
10	0.0018	20	0.1108	30	0.0108

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

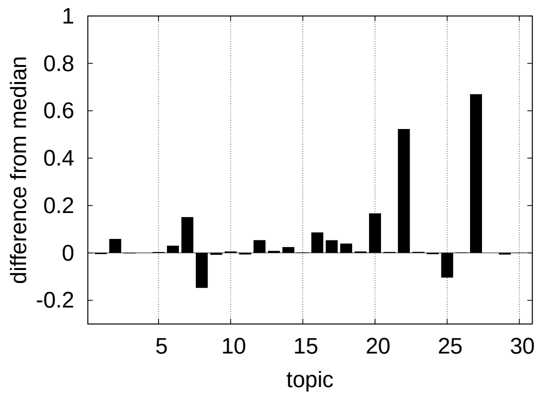


Overall average precision: 0.1076

Average precision per topic:

01	0.0036	11	0.0194	21	0.0130
02	0.1658	12	0.0656	22	0.6959
03	0.0170	13	0.0118	23	0.0933
04	0.0040	14	0.0276	24	0.0021
05	0.2668	15	0.0100	25	0.0015
06	0.0468	16	0.2518	26	0.0734
07	0.1969	17	0.0792	27	0.6701
08	0.0699	18	0.0804	28	0.0040
09	0.1007	19	0.0160	29	0.0187
10	0.0167	20	0.1675	30	0.0385

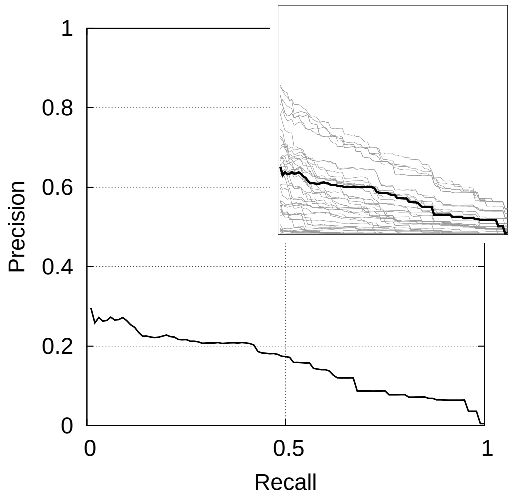
Difference from median in average precision per topic:



University of Melbourne
um_mgx2_long (CAS)

Quantisation: strict

Recall / precision graph:

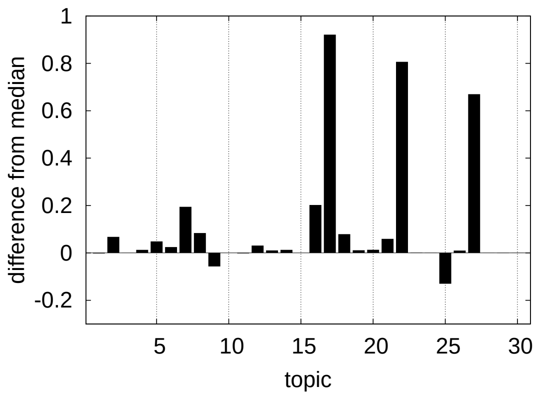


Overall average precision: 0.1570

Average precision per topic:

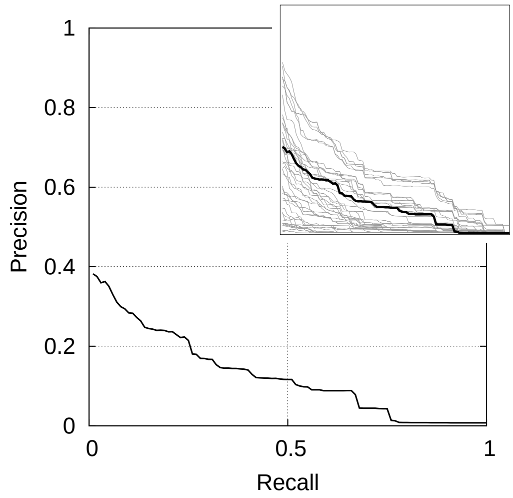
01	0.0066	11	0.0048	21	0.0773
02	0.1715	12	0.0493	22	0.9419
03	0.0025	13	0.0170	23	0.0999
04	0.0164	14	0.0138	24	0.0045
05	0.3540	15	0.0006	25	0.0005
06	0.0267	16	0.3745	26	0.0563
07	0.2159	17	0.9233	27	0.6701
08	0.3274	18	0.2465	28	0.0037
09	0.0592	19	0.0156	29	0.0052
10	0.0018	20	0.0134	30	0.0091

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

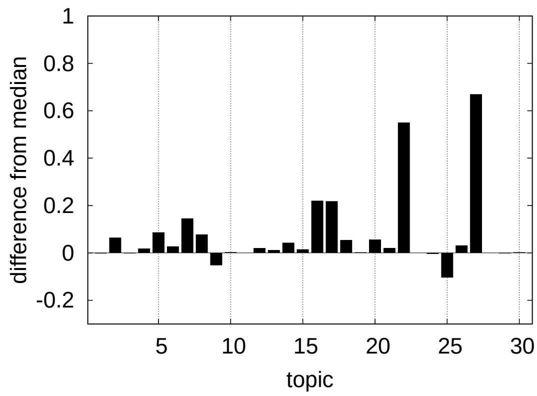


Overall average precision: 0.1265

Average precision per topic:

01	0.0066	11	0.0263	21	0.0298
02	0.1715	12	0.0323	22	0.7234
03	0.0170	13	0.0154	23	0.0889
04	0.0230	14	0.0462	24	0.0032
05	0.3501	15	0.0230	25	0.0015
06	0.0442	16	0.3857	26	0.1028
07	0.1915	17	0.2440	27	0.6701
08	0.2955	18	0.0954	28	0.0037
09	0.0564	19	0.0127	29	0.0239
10	0.0141	20	0.0573	30	0.0396

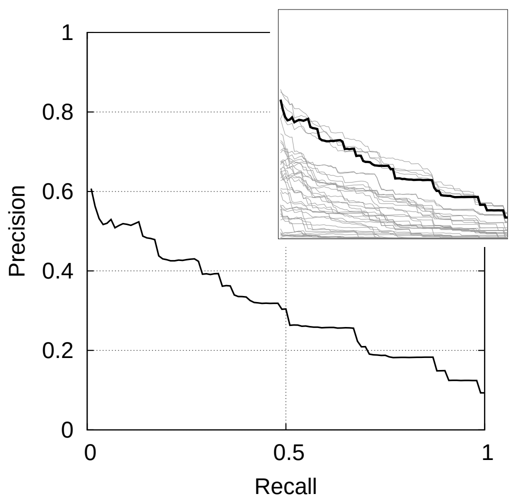
Difference from median in average precision per topic:



University of Michigan
allow-duplicate (CAS)

Quantisation: strict

Recall / precision graph:

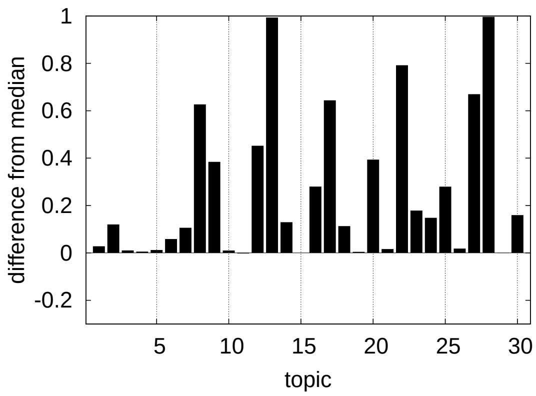


Overall average precision: 0.3090

Average precision per topic:

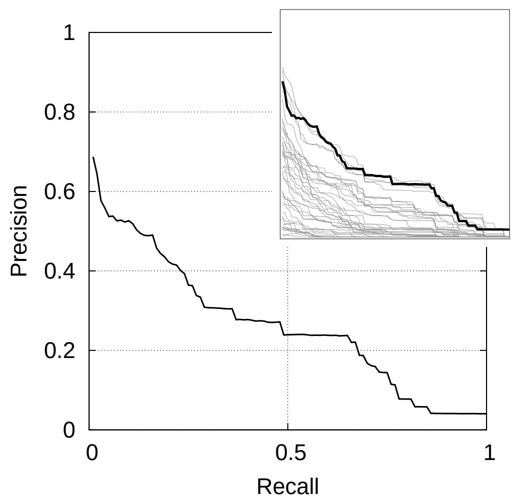
01	0.0366	11	0.0046	21	0.0340
02	0.2241	12	0.4702	22	0.9276
03	0.0134	13	1.0000	23	0.2772
04	0.0090	14	0.1305	24	0.1518
05	0.3175	15	0.0004	25	0.4102
06	0.0606	16	0.4523	26	0.0648
07	0.1275	17	0.6462	27	0.6701
08	0.8702	18	0.2802	28	1.0000
09	0.5008	19	0.0089	29	0.0052
10	0.0119	20	0.3941	30	0.1690

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

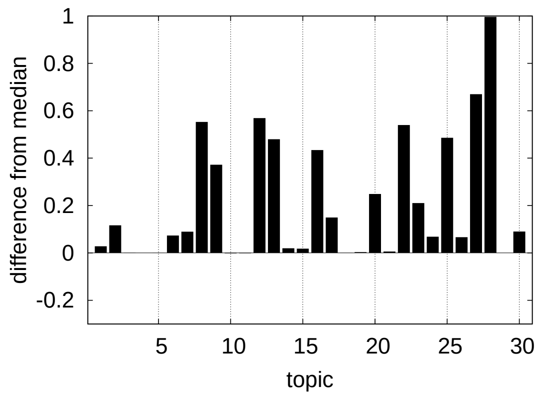


Overall average precision: 0.2634

Average precision per topic:

01	0.0366	11	0.0248	21	0.0145
02	0.2234	12	0.5806	22	0.7127
03	0.0198	13	0.4829	23	0.2992
04	0.0043	14	0.0225	24	0.0757
05	0.2633	15	0.0256	25	0.5916
06	0.0899	16	0.5996	26	0.1373
07	0.1354	17	0.1751	27	0.6701
08	0.7703	18	0.0407	28	1.0000
09	0.4808	19	0.0133	29	0.0258
10	0.0091	20	0.2491	30	0.1268

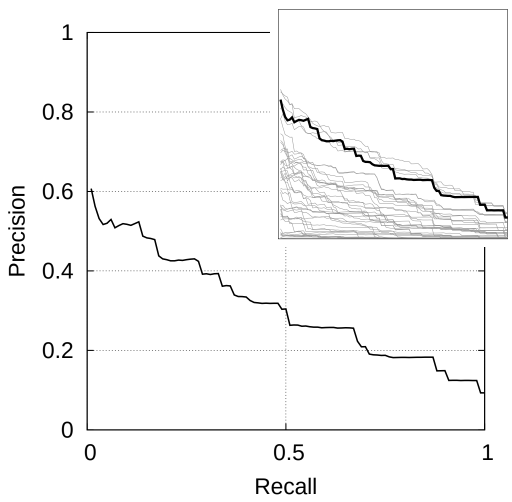
Difference from median in average precision per topic:



University of Michigan
no-duplicate (CAS)

Quantisation: strict

Recall / precision graph:

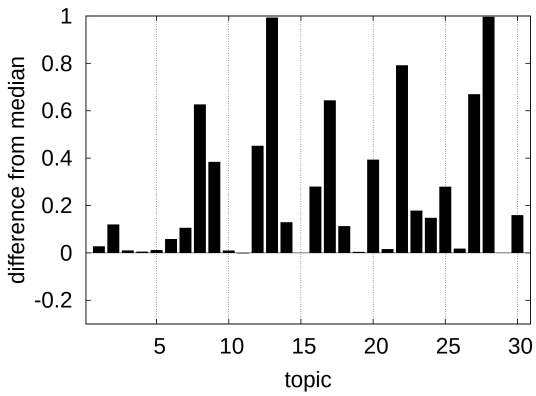


Overall average precision: 0.3090

Average precision per topic:

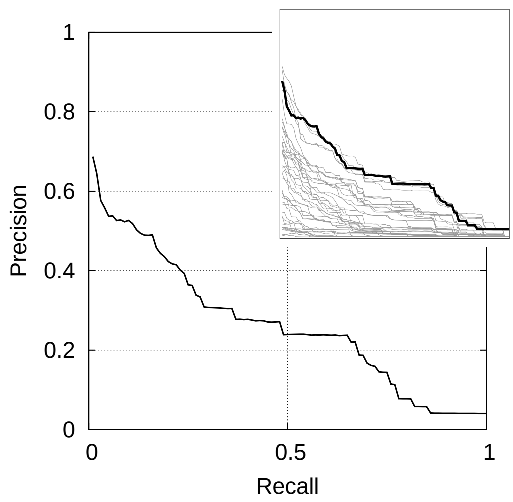
01	0.0366	11	0.0046	21	0.0340
02	0.2241	12	0.4702	22	0.9276
03	0.0134	13	1.0000	23	0.2772
04	0.0090	14	0.1305	24	0.1518
05	0.3175	15	0.0004	25	0.4102
06	0.0606	16	0.4523	26	0.0648
07	0.1275	17	0.6462	27	0.6701
08	0.8702	18	0.2802	28	1.0000
09	0.5008	19	0.0089	29	0.0052
10	0.0119	20	0.3941	30	0.1690

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

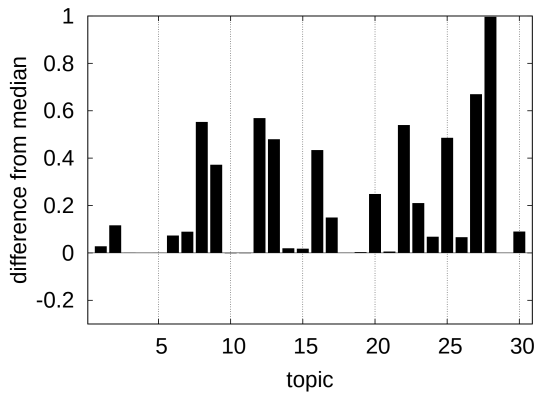


Overall average precision: 0.2634

Average precision per topic:

01	0.0366	11	0.0248	21	0.0145
02	0.2234	12	0.5806	22	0.7127
03	0.0198	13	0.4829	23	0.2992
04	0.0043	14	0.0225	24	0.0757
05	0.2633	15	0.0256	25	0.5916
06	0.0899	16	0.5996	26	0.1373
07	0.1354	17	0.1751	27	0.6701
08	0.7703	18	0.0407	28	1.0000
09	0.4808	19	0.0133	29	0.0258
10	0.0091	20	0.2491	30	0.1268

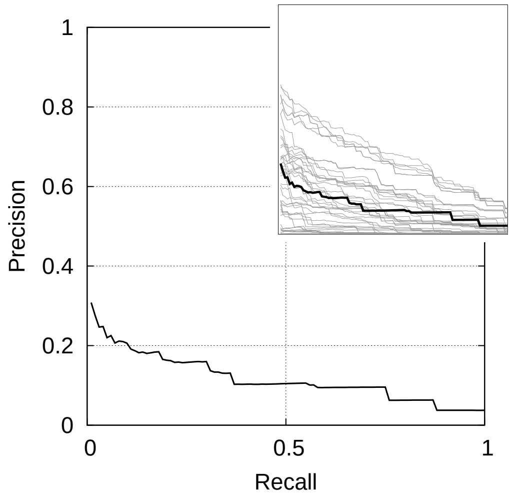
Difference from median in average precision per topic:



University of Minnesota Duluth
01 (CAS)

Quantisation: strict

Recall / precision graph:

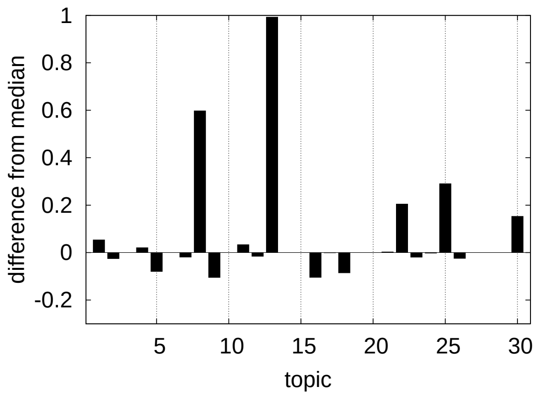


Overall average precision: 0.1168

Average precision per topic:

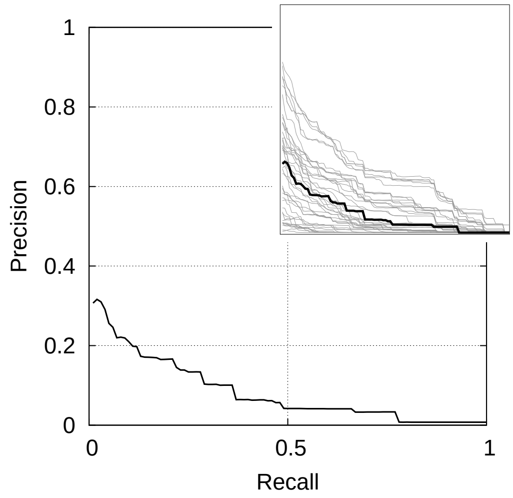
01	0.0631	11	0.0411	21	0.0215
02	0.0774	12	0.0013	22	0.3409
03	0.0038	13	1.0000	23	0.0780
04	0.0251	14	0.0004	24	0.0005
05	0.2248	15	0.0004	25	0.4221
06	0.0013	16	0.0664	26	0.0212
07	0.0015	17	0.0002	27	0.0001
08	0.8420	18	0.0809	28	0.0037
09	0.0105	19	0.0045	29	0.0057
10	0.0018	20	0.0002	30	0.1632

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

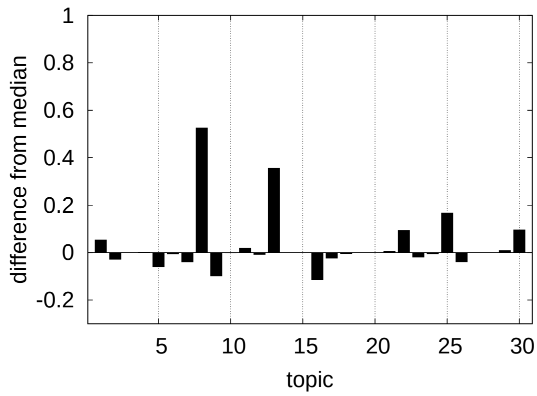


Overall average precision: 0.0831

Average precision per topic:

01	0.0631	11	0.0462	21	0.0161
02	0.0773	12	0.0025	22	0.2672
03	0.0201	13	0.3601	23	0.0686
04	0.0076	14	0.0024	24	0.0010
05	0.2025	15	0.0078	25	0.2742
06	0.0097	16	0.0506	26	0.0308
07	0.0049	17	0.0008	27	0.0001
08	0.7444	18	0.0357	28	0.0037
09	0.0091	19	0.0096	29	0.0352
10	0.0089	20	0.0006	30	0.1336

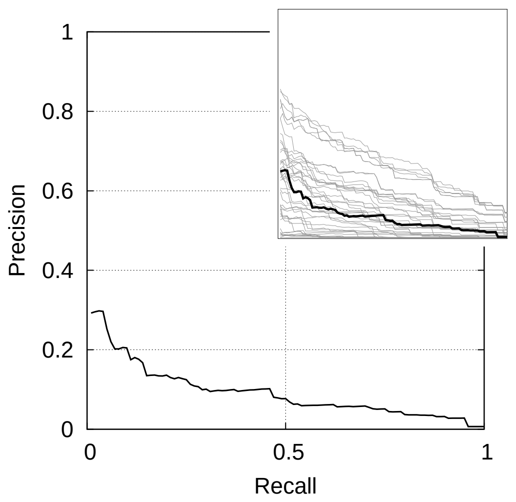
Difference from median in average precision per topic:



University of Twente
utwente1h (CAS)

Quantisation: strict

Recall / precision graph:

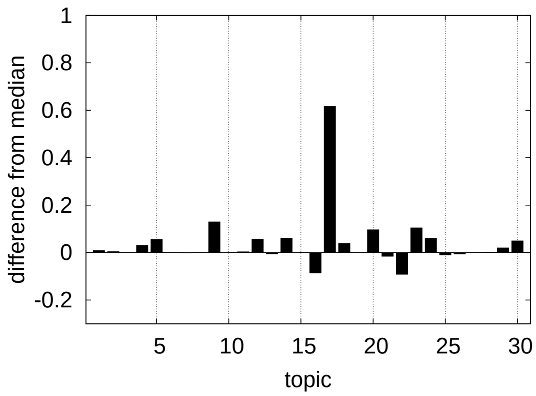


Overall average precision: 0.0923

Average precision per topic:

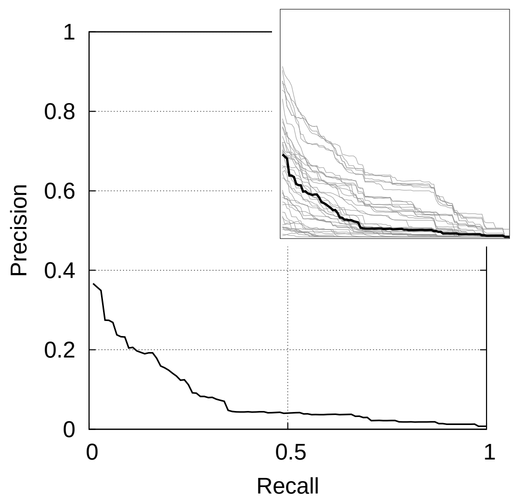
01	0.0183	11	0.0114	21	0.0010
02	0.1091	12	0.0758	22	0.0427
03	0.0029	13	0.0001	23	0.2040
04	0.0349	14	0.0629	24	0.0655
05	0.3618	15	0.0004	25	0.1193
06	0.0018	16	0.0851	26	0.0392
07	0.0195	17	0.6192	27	0.0001
08	0.2443	18	0.2067	28	0.0057
09	0.2469	19	0.0045	29	0.0267
10	0.0018	20	0.0976	30	0.0599

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

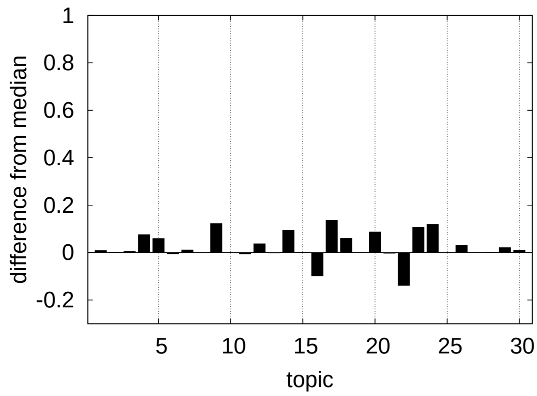


Overall average precision: 0.0789

Average precision per topic:

01	0.0183	11	0.0186	21	0.0049
02	0.1097	12	0.0496	22	0.0335
03	0.0252	13	0.0001	23	0.1977
04	0.0809	14	0.0988	24	0.1269
05	0.3229	15	0.0108	25	0.1059
06	0.0103	16	0.0663	26	0.1034
07	0.0580	17	0.1635	27	0.0001
08	0.2162	18	0.1024	28	0.0057
09	0.2321	19	0.0098	29	0.0477
10	0.0107	20	0.0889	30	0.0477

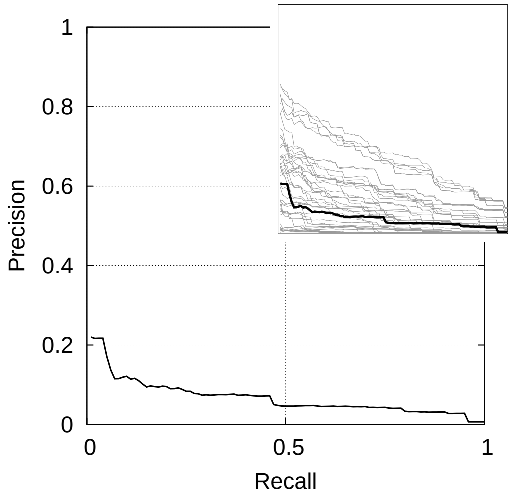
Difference from median in average precision per topic:



University of Twente
utwente1n (CAS)

Quantisation: strict

Recall / precision graph:

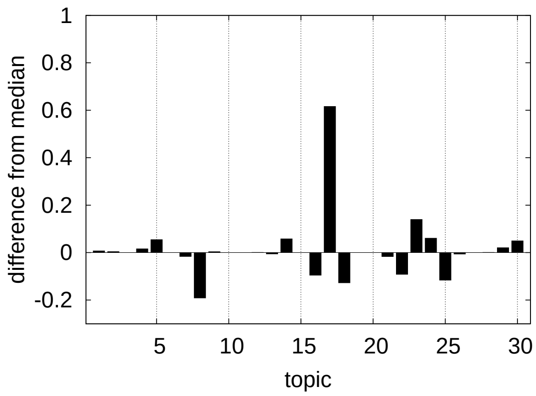


Overall average precision: 0.0670

Average precision per topic:

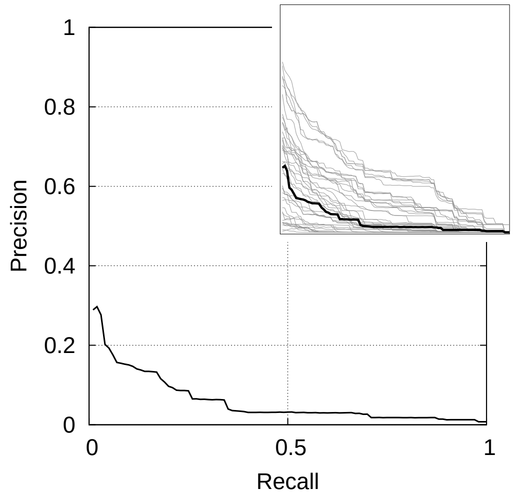
01	0.0168	11	0.0072	21	0.0002
02	0.1091	12	0.0199	22	0.0427
03	0.0025	13	0.0001	23	0.2393
04	0.0205	14	0.0600	24	0.0655
05	0.3608	15	0.0004	25	0.0133
06	0.0019	16	0.0758	26	0.0392
07	0.0040	17	0.6192	27	0.0001
08	0.0510	18	0.0387	28	0.0056
09	0.1213	19	0.0045	29	0.0273
10	0.0018	20	0.0002	30	0.0599

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

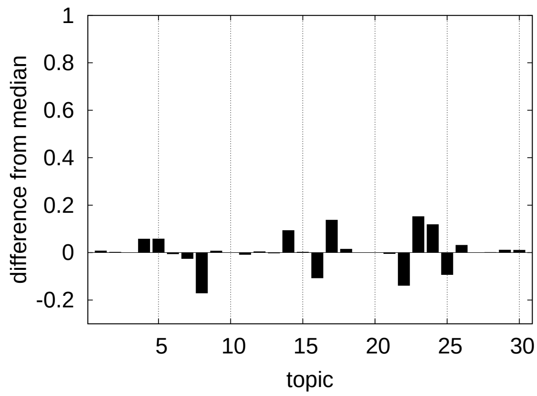


Overall average precision: 0.0592

Average precision per topic:

01	0.0168	11	0.0170	21	0.0036
02	0.1097	12	0.0162	22	0.0335
03	0.0200	13	0.0001	23	0.2417
04	0.0626	14	0.0972	24	0.1266
05	0.3218	15	0.0107	25	0.0120
06	0.0102	16	0.0574	26	0.1032
07	0.0197	17	0.1635	27	0.0001
08	0.0460	18	0.0563	28	0.0056
09	0.1166	19	0.0098	29	0.0377
10	0.0108	20	0.0008	30	0.0477

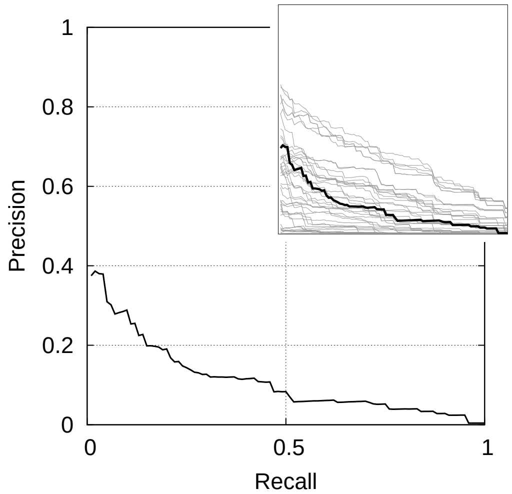
Difference from median in average precision per topic:



University of Twente
utwente1pr (CAS)

Quantisation: strict

Recall / precision graph:

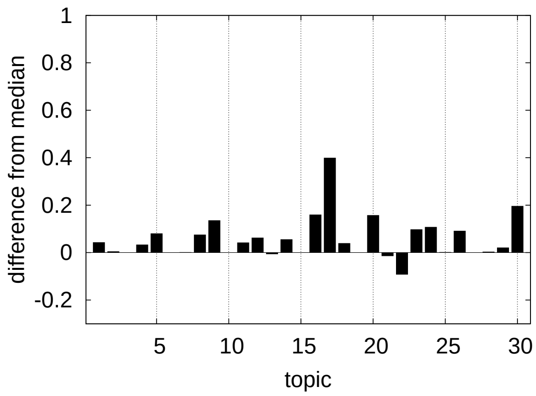


Overall average precision: 0.1115

Average precision per topic:

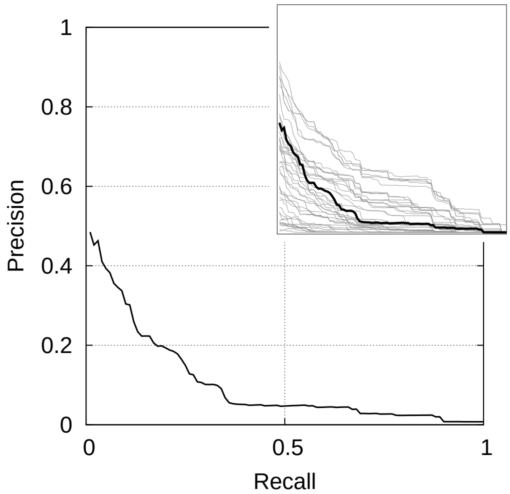
01	0.0523	11	0.0493	21	0.0029
02	0.1091	12	0.0809	22	0.0427
03	0.0030	13	0.0001	23	0.1965
04	0.0373	14	0.0567	24	0.1120
05	0.3861	15	0.0004	25	0.1325
06	0.0017	16	0.3324	26	0.1386
07	0.0234	17	0.4021	27	0.0001
08	0.3191	18	0.2071	28	0.0076
09	0.2525	19	0.0045	29	0.0269
10	0.0018	20	0.1580	30	0.2061

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

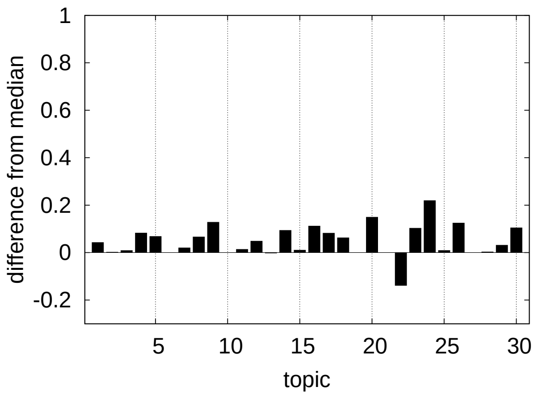


Overall average precision: 0.1026

Average precision per topic:

01	0.0523	11	0.0404	21	0.0096
02	0.1097	12	0.0609	22	0.0335
03	0.0287	13	0.0001	23	0.1926
04	0.0877	14	0.0974	24	0.2277
05	0.3320	15	0.0190	25	0.1157
06	0.0159	16	0.2782	26	0.1966
07	0.0668	17	0.1083	27	0.0001
08	0.2845	18	0.1041	28	0.0076
09	0.2376	19	0.0098	29	0.0580
10	0.0106	20	0.1508	30	0.1419

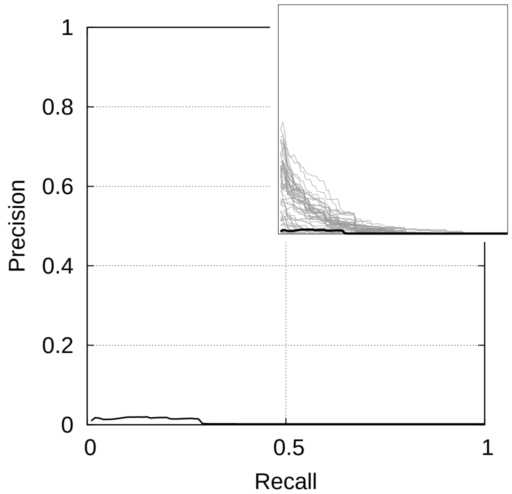
Difference from median in average precision per topic:



Centrum voor Wiskunde en Informatica (CWI)
R_all (CO)

Quantisation: strict

Recall / precision graph:

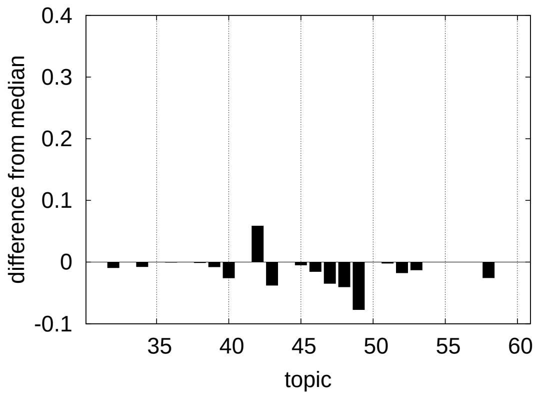


Overall average precision: 0.0061

Average precision per topic:

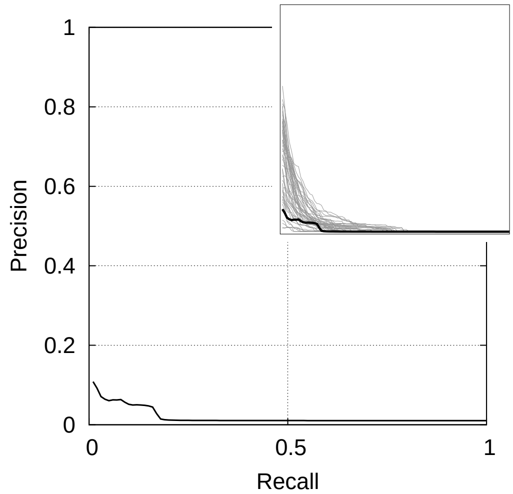
31	0.0002	41	0.0024	51	0.0016
32	0.0021	42	0.0782	52	0.0124
33	0.0001	43	0.0003	53	0.0004
34	0.0024	44	0.0005	54	–
35	–	45	0.0017	55	–
36	0.0017	46	0.0021	56	–
37	0.0032	47	0.0003	57	–
38	0.0023	48	0.0036	58	0.0075
39	0.0004	49	0.0022	59	–
40	0.0088	50	–	60	0.0065

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

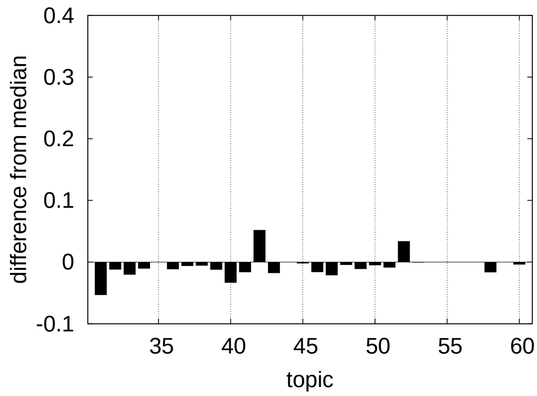


Overall average precision: 0.0189

Average precision per topic:

31	0.0035	41	0.0068	51	0.0123
32	0.0097	42	0.1076	52	0.0561
33	0.0025	43	0.0017	53	0.0124
34	0.0166	44	0.0024	54	–
35	–	45	0.0290	55	–
36	0.0079	46	0.0144	56	–
37	0.0203	47	0.0029	57	–
38	0.0273	48	0.0339	58	0.0274
39	0.0054	49	0.0072	59	–
40	0.0166	50	0.0064	60	0.0239

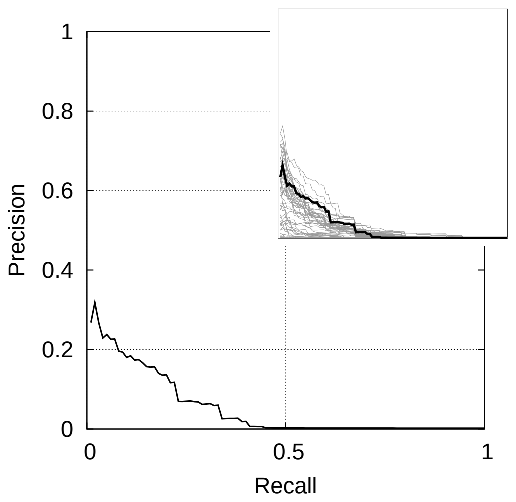
Difference from median in average precision per topic:



Centrum voor Wiskunde en Informatica (CWI)
R_article (CO)

Quantisation: strict

Recall / precision graph:

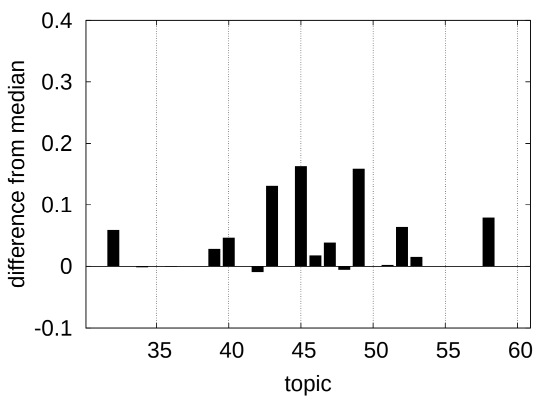


Overall average precision: 0.0520

Average precision per topic:

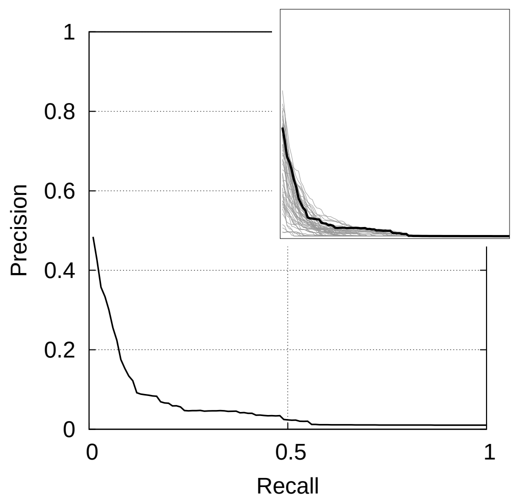
31	0.0002	41	0.0024	51	0.0061
32	0.0712	42	0.0099	52	0.0947
33	0.0001	43	0.1695	53	0.0292
34	0.0088	44	0.0005	54	–
35	–	45	0.1696	55	–
36	0.0017	46	0.0359	56	–
37	0.0032	47	0.0742	57	–
38	0.0034	48	0.0390	58	0.1128
39	0.0374	49	0.2386	59	–
40	0.0818	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

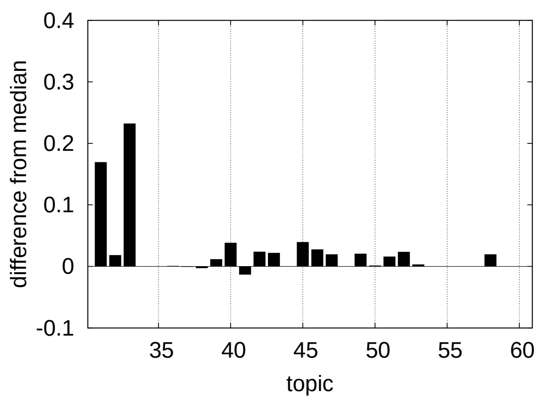


Overall average precision: 0.0555

Average precision per topic:

31	0.2265	41	0.0099	51	0.0373
32	0.0404	42	0.0795	52	0.0459
33	0.2552	43	0.0415	53	0.0165
34	0.0271	44	0.0027	54	–
35	–	45	0.0708	55	–
36	0.0204	46	0.0584	56	–
37	0.0259	47	0.0441	57	–
38	0.0302	48	0.0386	58	0.0637
39	0.0296	49	0.0391	59	–
40	0.0885	50	0.0130	60	0.0280

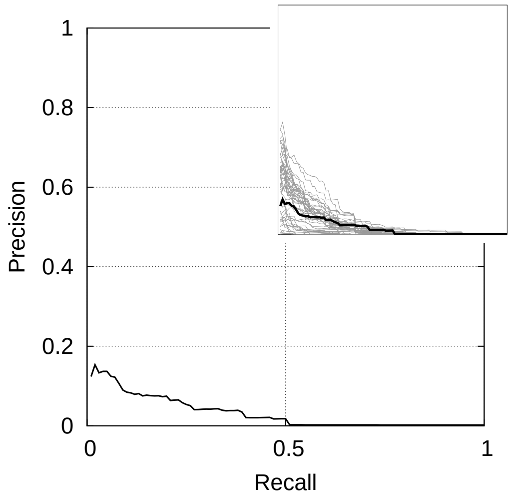
Difference from median in average precision per topic:



Centrum voor Wiskunde en Informatica (CWI)
R_prel_length (CO)

Quantisation: strict

Recall / precision graph:

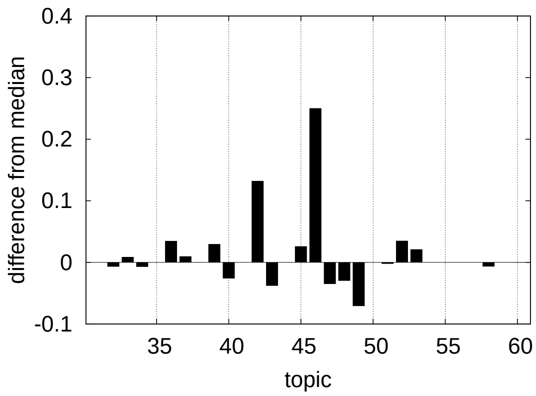


Overall average precision: 0.0319

Average precision per topic:

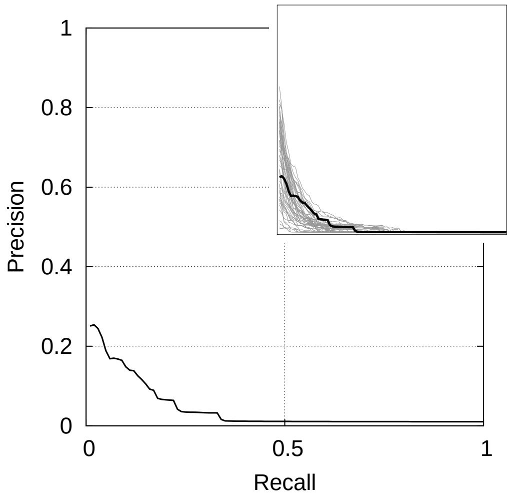
31	0.0002	41	0.0024	51	0.0016
32	0.0047	42	0.1516	52	0.0652
33	0.0090	43	0.0003	53	0.0349
34	0.0031	44	0.0009	54	–
35	–	45	0.0330	55	–
36	0.0373	46	0.2682	56	–
37	0.0130	47	0.0003	57	–
38	0.0039	48	0.0144	58	0.0266
39	0.0384	49	0.0087	59	–
40	0.0088	50	–	60	0.0065

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

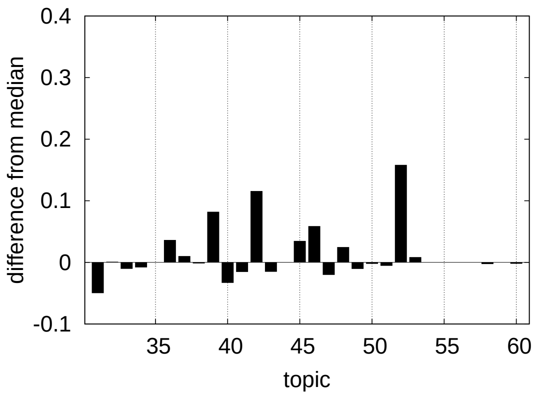


Overall average precision: 0.0423

Average precision per topic:

31	0.0070	41	0.0076	51	0.0157
32	0.0229	42	0.1713	52	0.1805
33	0.0124	43	0.0043	53	0.0218
34	0.0190	44	0.0029	54	–
35	–	45	0.0660	55	–
36	0.0560	46	0.0896	56	–
37	0.0369	47	0.0042	57	–
38	0.0315	48	0.0634	58	0.0414
39	0.1000	49	0.0079	59	–
40	0.0169	50	0.0093	60	0.0257

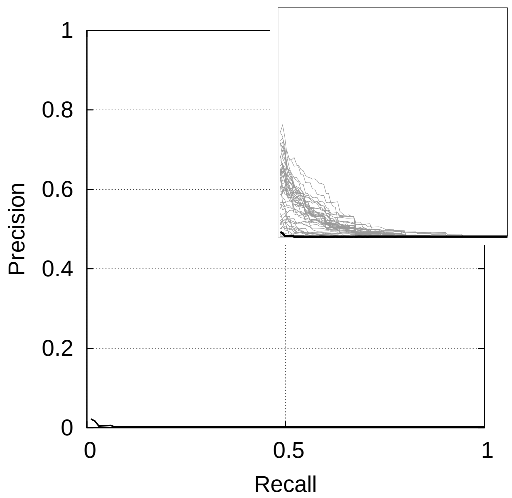
Difference from median in average precision per topic:



CSIRO Mathematical and Information Sciences
full (CO)

Quantisation: strict

Recall / precision graph:

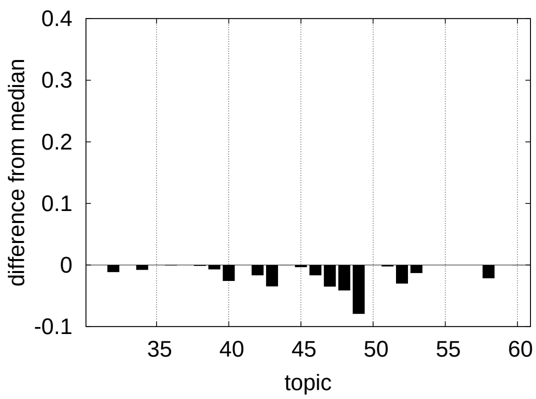


Overall average precision: 0.0026

Average precision per topic:

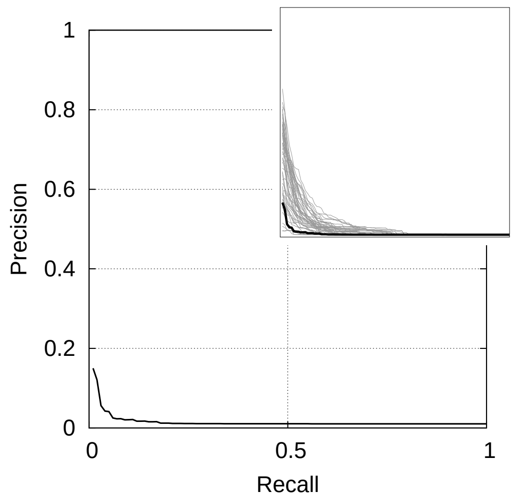
31	0.0002	41	0.0024	51	0.0016
32	0.0002	42	0.0026	52	0.0001
33	0.0001	43	0.0036	53	0.0006
34	0.0024	44	0.0005	54	–
35	–	45	0.0034	55	–
36	0.0017	46	0.0012	56	–
37	0.0032	47	0.0003	57	–
38	0.0023	48	0.0030	58	0.0119
39	0.0014	49	0.0004	59	–
40	0.0091	50	–	60	0.0065

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

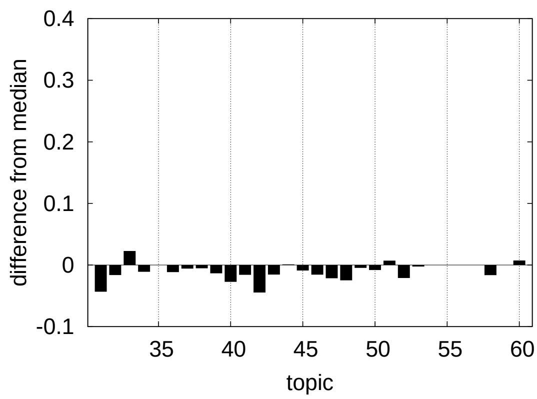


Overall average precision: 0.0152

Average precision per topic:

31	0.0136	41	0.0073	51	0.0283
32	0.0057	42	0.0109	52	0.0011
33	0.0457	43	0.0041	53	0.0108
34	0.0162	44	0.0040	54	–
35	–	45	0.0223	55	–
36	0.0081	46	0.0151	56	–
37	0.0208	47	0.0029	57	–
38	0.0278	48	0.0137	58	0.0276
39	0.0043	49	0.0138	59	–
40	0.0228	50	0.0034	60	0.0353

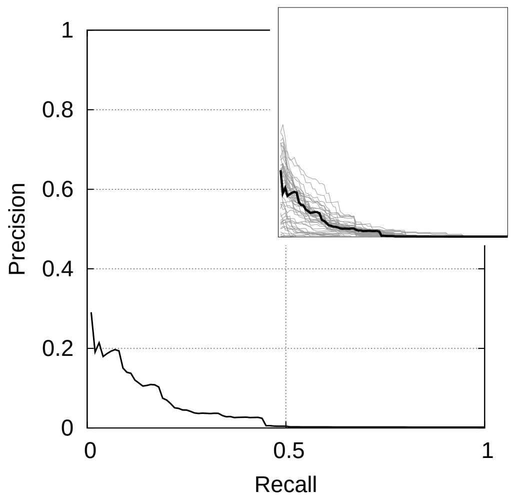
Difference from median in average precision per topic:



CSIRO Mathematical and Information Sciences
manual (CO)

Quantisation: strict

Recall / precision graph:

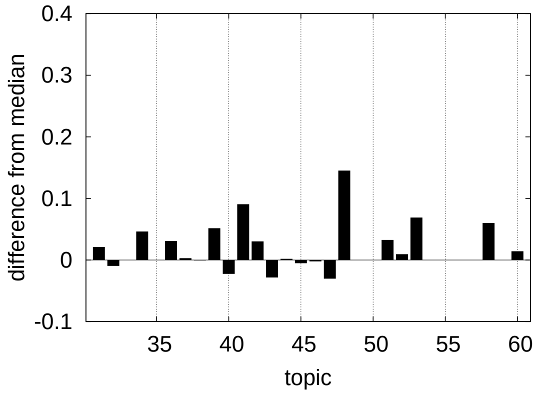


Overall average precision: 0.0398

Average precision per topic:

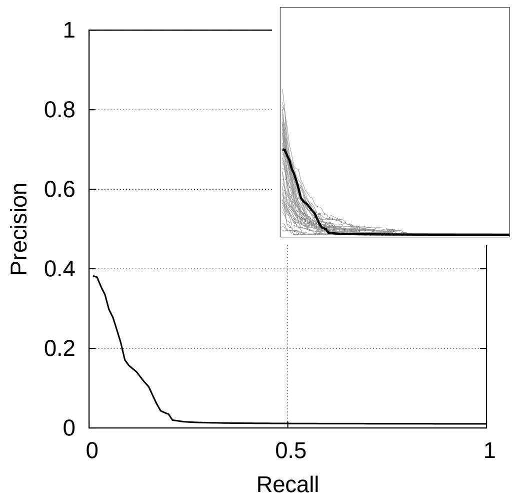
31	0.0214	41	0.0930	51	0.0365
32	0.0021	42	0.0496	52	0.0396
33	0.0001	43	0.0100	53	0.0825
34	0.0566	44	0.0027	54	–
35	–	45	0.0017	55	–
36	0.0334	46	0.0157	56	–
37	0.0063	47	0.0050	57	–
38	0.0030	48	0.1894	58	0.0934
39	0.0603	49	0.0796	59	–
40	0.0125	50	–	60	0.0209

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

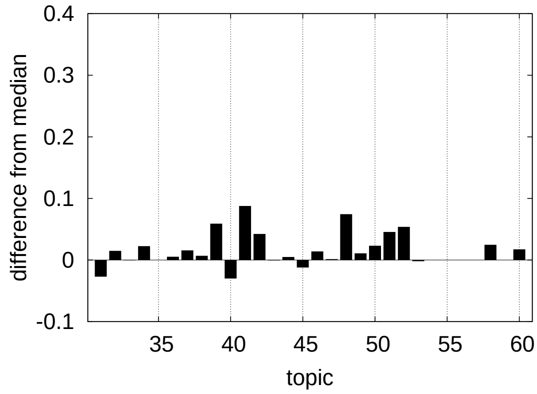


Overall average precision: 0.0464

Average precision per topic:

31	0.0300	41	0.1110	51	0.0667
32	0.0369	42	0.0979	52	0.0761
33	0.0224	43	0.0188	53	0.0112
34	0.0498	44	0.0078	54	–
35	–	45	0.0190	55	–
36	0.0249	46	0.0447	56	–
37	0.0423	47	0.0259	57	–
38	0.0400	48	0.1130	58	0.0687
39	0.0768	49	0.0292	59	–
40	0.0202	50	0.0348	60	0.0453

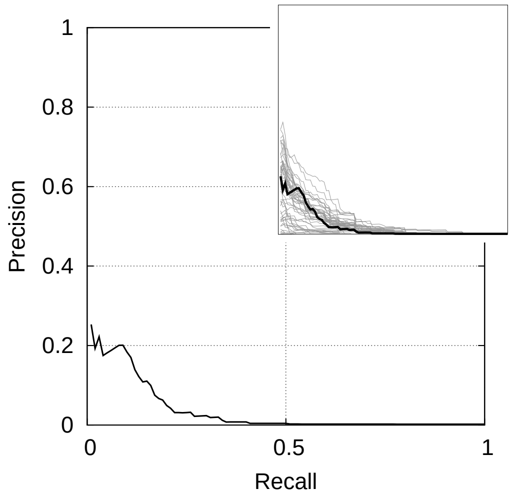
Difference from median in average precision per topic:



CSIRO Mathematical and Information Sciences
Split (CO)

Quantisation: strict	Quantisation: generalised

Recall / precision graph:

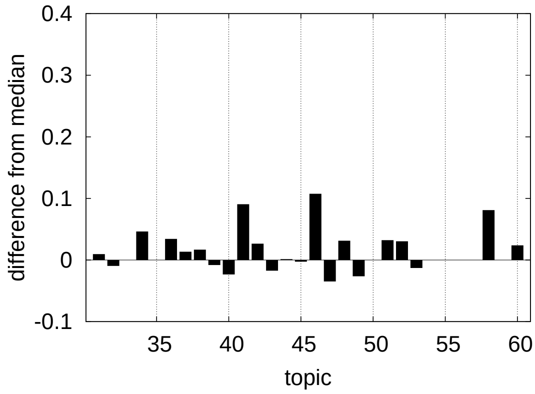


Overall average precision: 0.0356

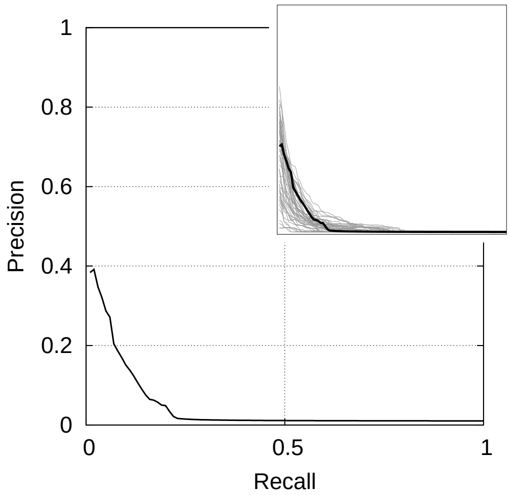
Average precision per topic:

31	0.0098	41	0.0930	51	0.0361
32	0.0021	42	0.0458	52	0.0605
33	0.0001	43	0.0210	53	0.0007
34	0.0566	44	0.0022	54	–
35	–	45	0.0041	55	–
36	0.0367	46	0.1255	56	–
37	0.0167	47	0.0005	57	–
38	0.0205	48	0.0756	58	0.1144
39	0.0004	49	0.0532	59	–
40	0.0115	50	–	60	0.0305

Difference from median in average precision per topic:



Recall / precision graph:

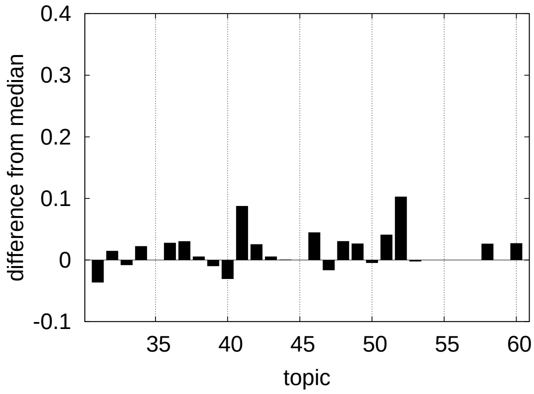


Overall average precision: 0.0447

Average precision per topic:

31	0.0205	41	0.1110	51	0.0624
32	0.0368	42	0.0812	52	0.1251
33	0.0146	43	0.0253	53	0.0109
34	0.0498	44	0.0037	54	–
35	–	45	0.0312	55	–
36	0.0475	46	0.0756	56	–
37	0.0572	47	0.0078	57	–
38	0.0388	48	0.0693	58	0.0707
39	0.0079	49	0.0452	59	–
40	0.0194	50	0.0068	60	0.0551

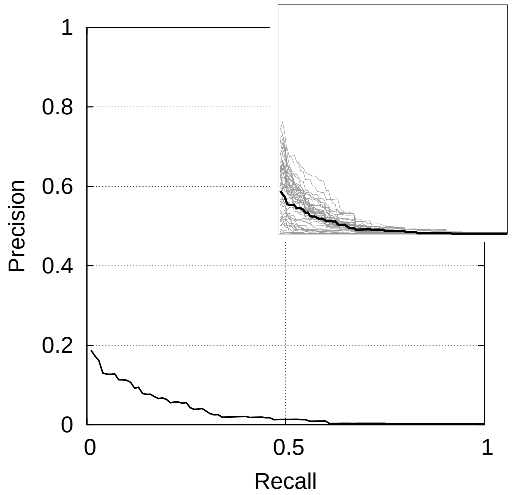
Difference from median in average precision per topic:



doctronic GmbH & Co. KG
1 (CO)

Quantisation: strict

Recall / precision graph:

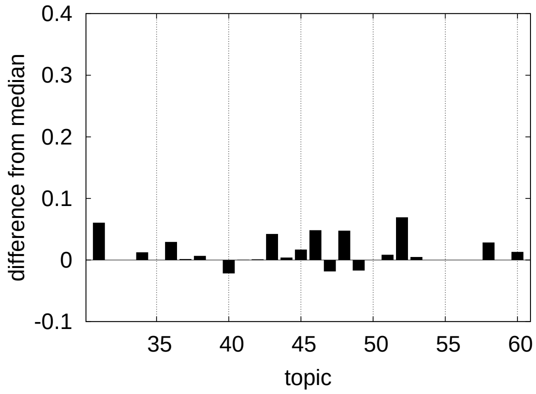


Overall average precision: 0.0325

Average precision per topic:

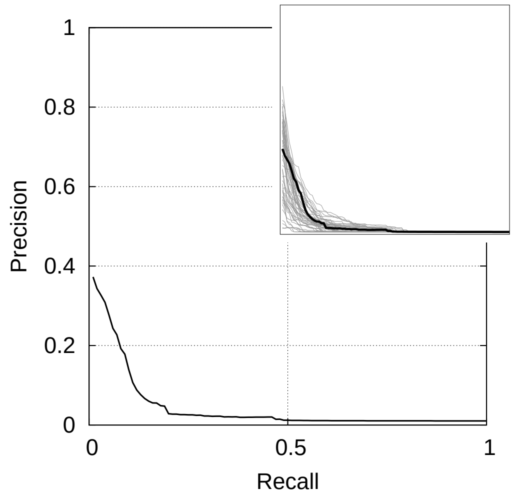
31	0.0609	41	0.0028	51	0.0125
32	0.0117	42	0.0205	52	0.0996
33	0.0001	43	0.0806	53	0.0186
34	0.0228	44	0.0047	54	–
35	–	45	0.0238	55	–
36	0.0318	46	0.0664	56	–
37	0.0048	47	0.0169	57	–
38	0.0103	48	0.0919	58	0.0618
39	0.0086	49	0.0626	59	–
40	0.0132	50	–	60	0.0199

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

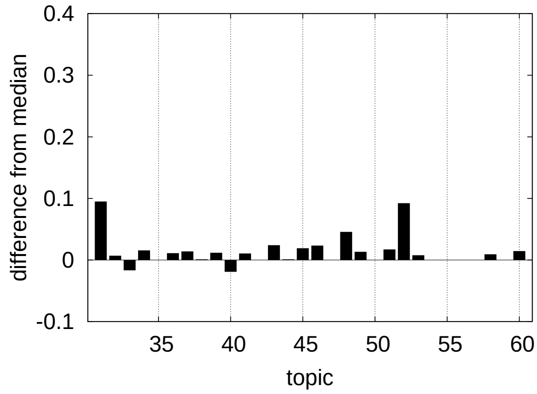


Overall average precision: 0.0441

Average precision per topic:

31	0.1520	41	0.0338	51	0.0385
32	0.0290	42	0.0555	52	0.1145
33	0.0062	43	0.0436	53	0.0210
34	0.0429	44	0.0040	54	–
35	–	45	0.0503	55	–
36	0.0307	46	0.0541	56	–
37	0.0407	47	0.0244	57	–
38	0.0343	48	0.0843	58	0.0534
39	0.0296	49	0.0318	59	–
40	0.0310	50	0.0115	60	0.0424

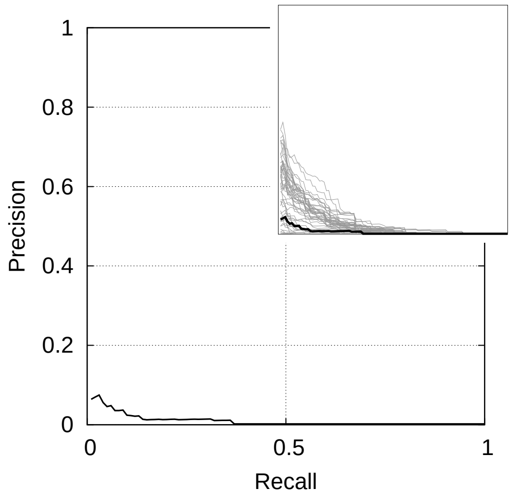
Difference from median in average precision per topic:



ETH Zurich
Augmentation0.8 (CO)

Quantisation: strict

Recall / precision graph:

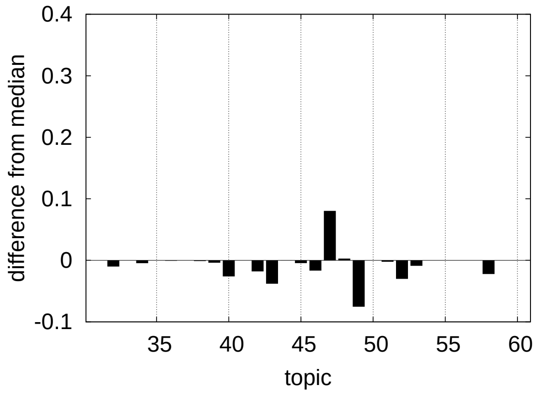


Overall average precision: 0.0099

Average precision per topic:

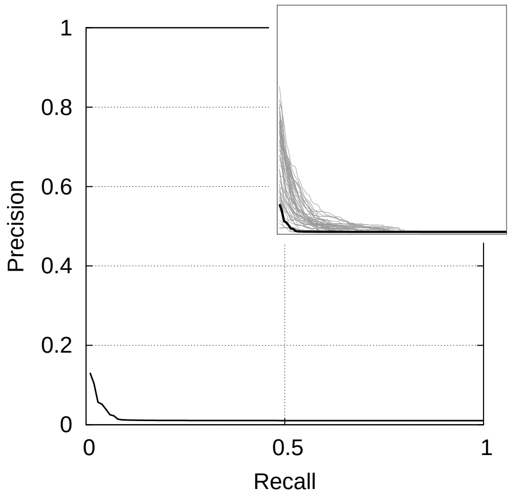
31	0.0002	41	0.0024	51	0.0016
32	0.0016	42	0.0014	52	0.0001
33	0.0001	43	0.0003	53	0.0048
34	0.0056	44	0.0005	54	–
35	–	45	0.0023	55	–
36	0.0017	46	0.0012	56	–
37	0.0032	47	0.1158	57	–
38	0.0025	48	0.0472	58	0.0111
39	0.0047	49	0.0042	59	–
40	0.0088	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

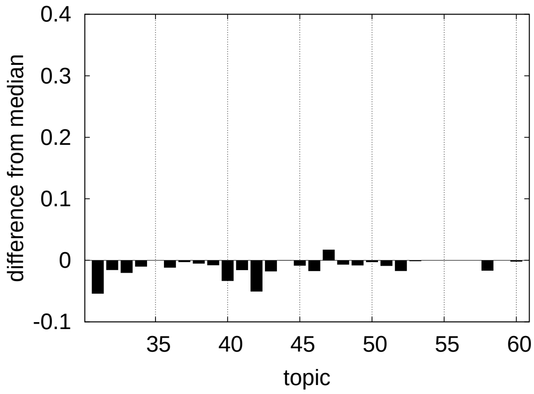


Overall average precision: 0.0142

Average precision per topic:

31	0.0028	41	0.0073	51	0.0122
32	0.0062	42	0.0048	52	0.0049
33	0.0025	43	0.0015	53	0.0118
34	0.0171	44	0.0024	54	–
35	–	45	0.0226	55	–
36	0.0077	46	0.0132	56	–
37	0.0238	47	0.0417	57	–
38	0.0278	48	0.0316	58	0.0273
39	0.0098	49	0.0102	59	–
40	0.0166	50	0.0086	60	0.0259

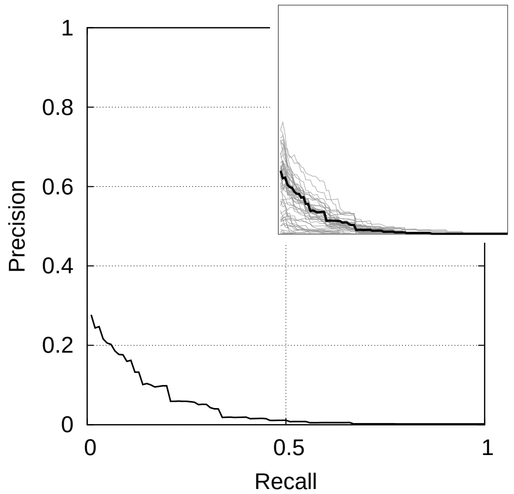
Difference from median in average precision per topic:



IBM Haifa Labs
ManualNoMerge (CO)

Quantisation: strict

Recall / precision graph:

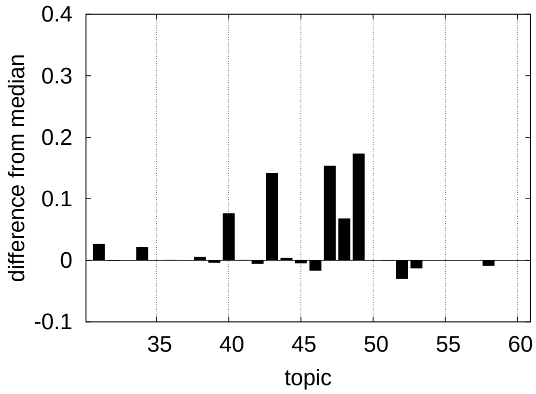


Overall average precision: 0.0434

Average precision per topic:

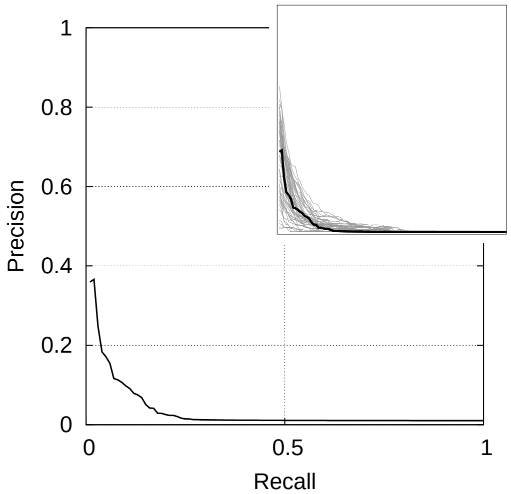
31	0.0271	41	0.0031	51	0.0033
32	0.0109	42	0.0139	52	0.0001
33	0.0001	43	0.1805	53	0.0006
34	0.0316	44	0.0047	54	–
35	–	45	0.0019	55	–
36	0.0033	46	0.0012	56	–
37	0.0032	47	0.1893	57	–
38	0.0095	48	0.1123	58	0.0246
39	0.0048	49	0.2532	59	–
40	0.1113	50	–	60	0.0067

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

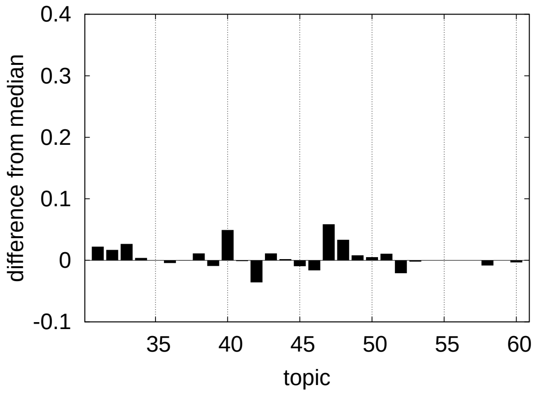


Overall average precision: 0.0337

Average precision per topic:

31	0.0792	41	0.0220	51	0.0320
32	0.0389	42	0.0198	52	0.0013
33	0.0496	43	0.0308	53	0.0111
34	0.0312	44	0.0049	54	–
35	–	45	0.0217	55	–
36	0.0152	46	0.0144	56	–
37	0.0262	47	0.0830	57	–
38	0.0444	48	0.0719	58	0.0356
39	0.0087	49	0.0266	59	–
40	0.0994	50	0.0167	60	0.0246

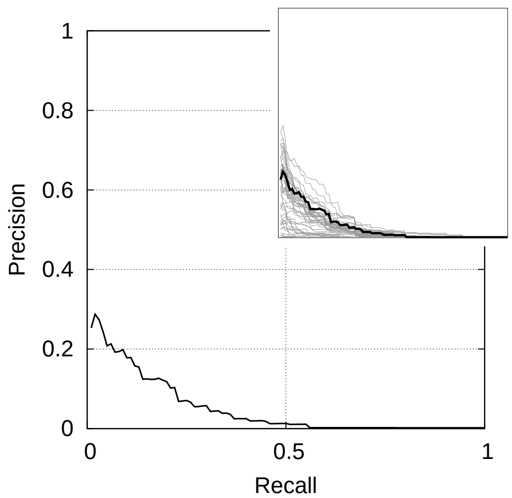
Difference from median in average precision per topic:



IBM Haifa Labs
Merge (CO)

Quantisation: strict

Recall / precision graph:

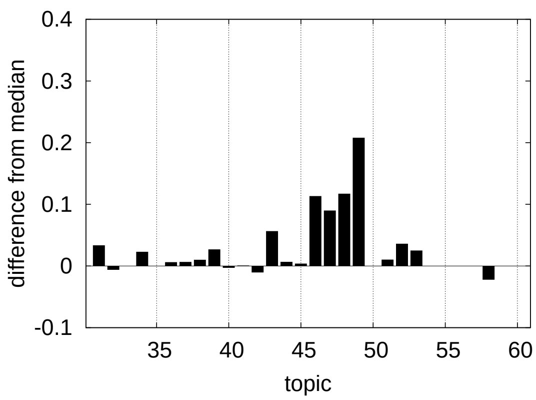


Overall average precision: 0.0496

Average precision per topic:

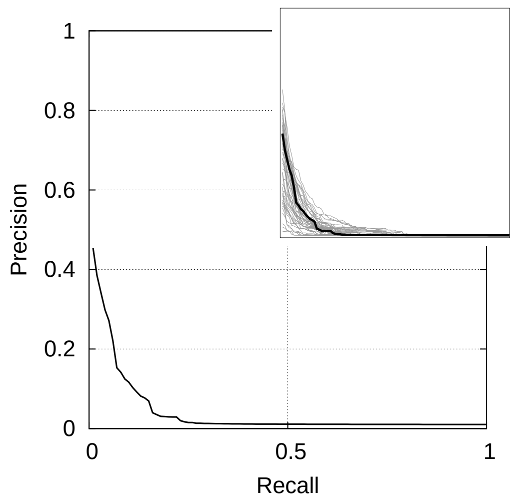
31	0.0338	41	0.0032	51	0.0143
32	0.0054	42	0.0089	52	0.0663
33	0.0001	43	0.0948	53	0.0388
34	0.0334	44	0.0074	54	–
35	–	45	0.0108	55	–
36	0.0087	46	0.1312	56	–
37	0.0098	47	0.1254	57	–
38	0.0137	48	0.1615	58	0.0111
39	0.0355	49	0.2877	59	–
40	0.0321	50	–	60	0.0067

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

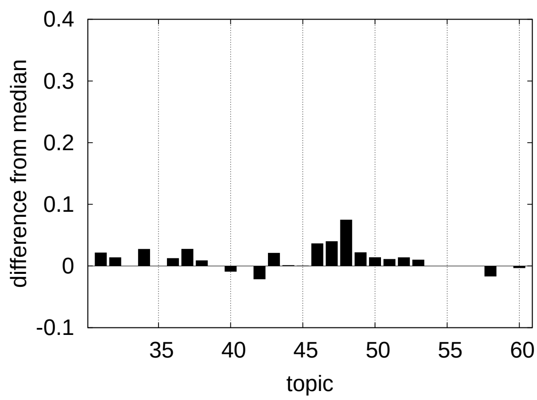


Overall average precision: 0.0404

Average precision per topic:

31	0.0788	41	0.0232	51	0.0324
32	0.0360	42	0.0341	52	0.0361
33	0.0229	43	0.0408	53	0.0234
34	0.0548	44	0.0042	54	–
35	–	45	0.0318	55	–
36	0.0323	46	0.0673	56	–
37	0.0544	47	0.0646	57	–
38	0.0422	48	0.1137	58	0.0272
39	0.0178	49	0.0405	59	–
40	0.0409	50	0.0256	60	0.0246

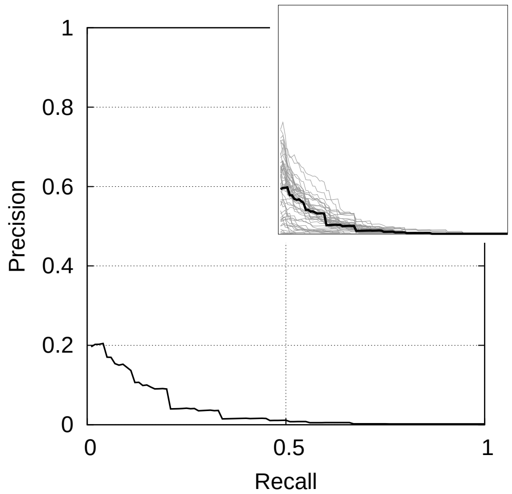
Difference from median in average precision per topic:



IBM Haifa Labs
NoMerge (CO)

Quantisation: strict

Recall / precision graph:

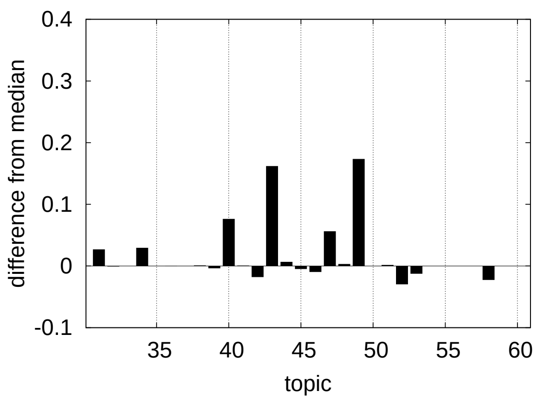


Overall average precision: 0.0367

Average precision per topic:

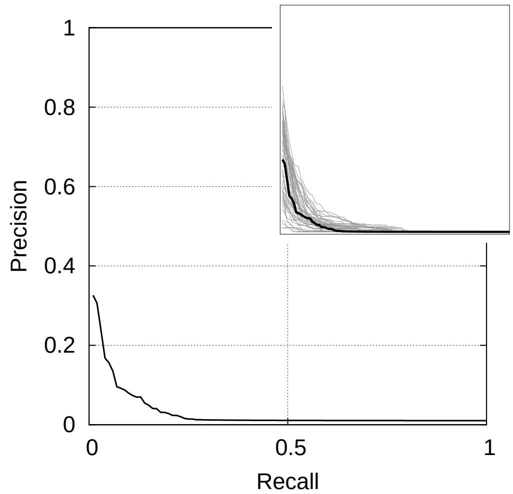
31	0.0271	41	0.0031	51	0.0055
32	0.0109	42	0.0014	52	0.0005
33	0.0001	43	0.2003	53	0.0011
34	0.0398	44	0.0074	54	–
35	–	45	0.0019	55	–
36	0.0027	46	0.0083	56	–
37	0.0032	47	0.0917	57	–
38	0.0045	48	0.0476	58	0.0107
39	0.0048	49	0.2532	59	–
40	0.1113	50	–	60	0.0067

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

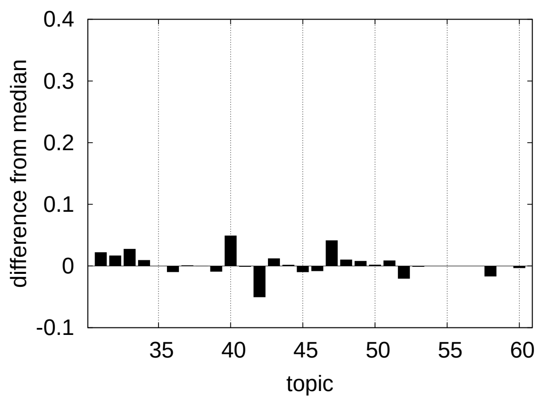


Overall average precision: 0.0309

Average precision per topic:

31	0.0792	41	0.0220	51	0.0302
32	0.0389	42	0.0049	52	0.0017
33	0.0506	43	0.0318	53	0.0119
34	0.0368	44	0.0049	54	–
35	–	45	0.0213	55	–
36	0.0098	46	0.0224	56	–
37	0.0277	47	0.0660	57	–
38	0.0328	48	0.0489	58	0.0271
39	0.0087	49	0.0266	59	–
40	0.0994	50	0.0136	60	0.0246

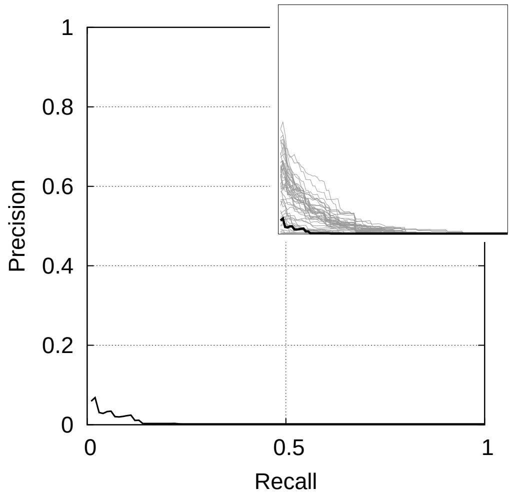
Difference from median in average precision per topic:



Institut de Recherche en Informatique de Toulouse (IRIT)
Mercure1 (CO)

Quantisation: strict

Recall / precision graph:

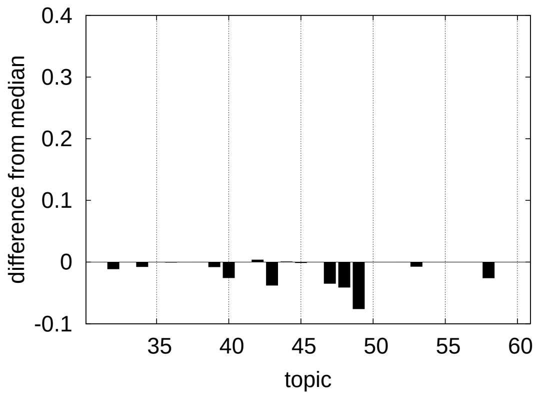


Overall average precision: 0.0058

Average precision per topic:

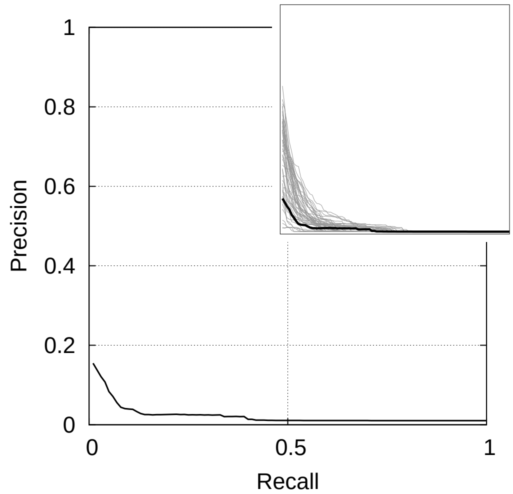
31	0.0002	41	0.0024	51	0.0038
32	0.0002	42	0.0233	52	0.0306
33	0.0001	43	0.0003	53	0.0062
34	0.0024	44	0.0016	54	–
35	–	45	0.0055	55	–
36	0.0017	46	0.0179	56	–
37	0.0032	47	0.0003	57	–
38	0.0035	48	0.0030	58	0.0072
39	0.0004	49	0.0035	59	–
40	0.0091	50	–	60	0.0065

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

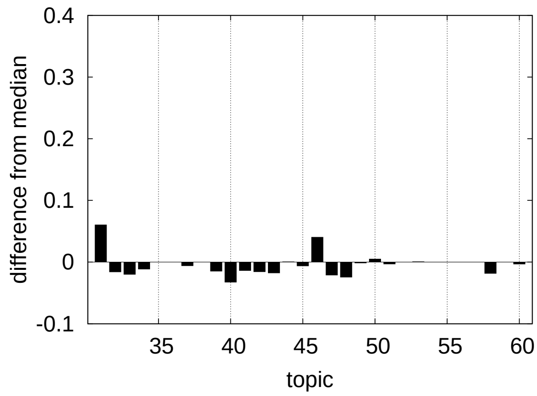


Overall average precision: 0.0224

Average precision per topic:

31	0.1177	41	0.0091	51	0.0178
32	0.0056	42	0.0395	52	0.0222
33	0.0026	43	0.0015	53	0.0141
34	0.0154	44	0.0036	54	–
35	–	45	0.0246	55	–
36	0.0194	46	0.0714	56	–
37	0.0203	47	0.0029	57	–
38	0.0331	48	0.0138	58	0.0253
39	0.0026	49	0.0166	59	–
40	0.0172	50	0.0168	60	0.0242

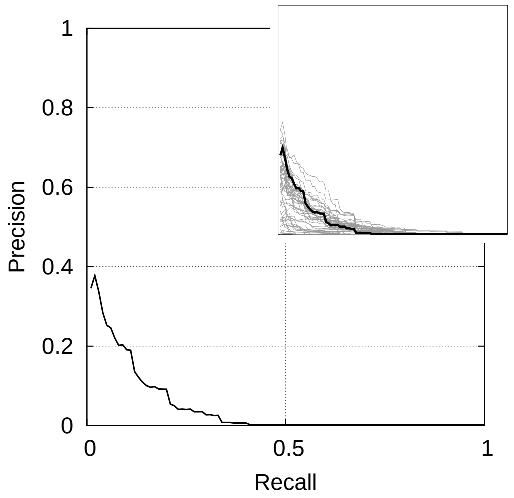
Difference from median in average precision per topic:



Nara Institute of Science and Technology
20020824-article (CO)

Quantisation: strict

Recall / precision graph:

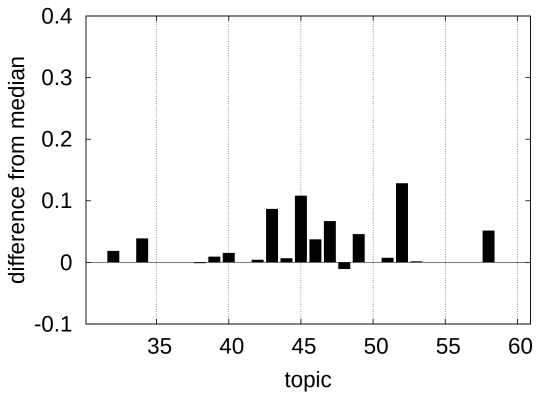


Overall average precision: 0.0445

Average precision per topic:

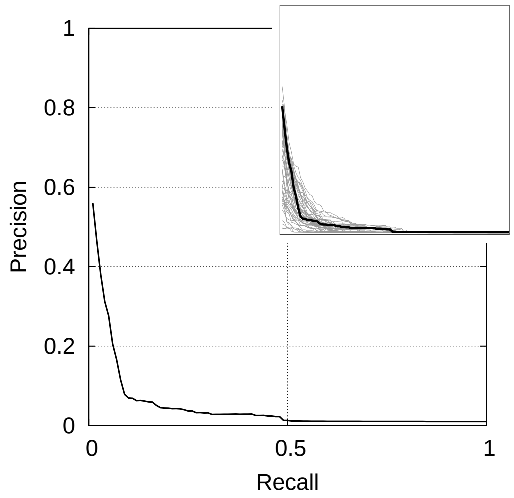
31	0.0002	41	0.0024	51	0.0114
32	0.0304	42	0.0237	52	0.1587
33	0.0001	43	0.1251	53	0.0153
34	0.0493	44	0.0074	54	–
35	–	45	0.1151	55	–
36	0.0024	46	0.0553	56	–
37	0.0032	47	0.1024	57	–
38	0.0023	48	0.0335	58	0.0850
39	0.0180	49	0.1256	59	–
40	0.0503	50	–	60	0.0070

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

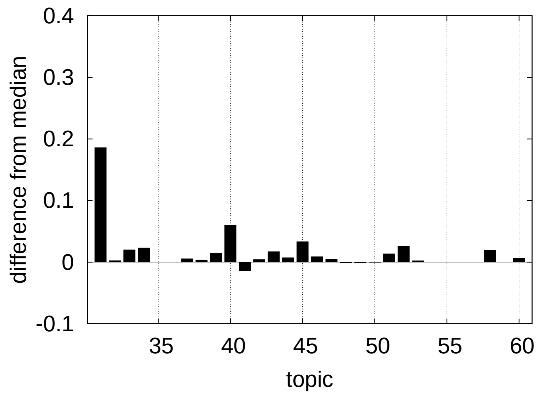


Overall average precision: 0.0461

Average precision per topic:

31	0.2432	41	0.0086	51	0.0351
32	0.0247	42	0.0601	52	0.0481
33	0.0433	43	0.0369	53	0.0158
34	0.0507	44	0.0103	54	–
35	–	45	0.0648	55	–
36	0.0199	46	0.0399	56	–
37	0.0325	47	0.0291	57	–
38	0.0370	48	0.0368	58	0.0637
39	0.0328	49	0.0174	59	–
40	0.1104	50	0.0109	60	0.0349

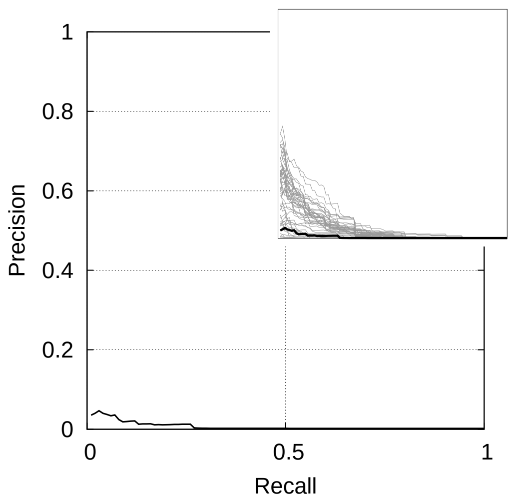
Difference from median in average precision per topic:



Queen Mary University of London
QMUL1 (CO)

Quantisation: strict

Recall / precision graph:

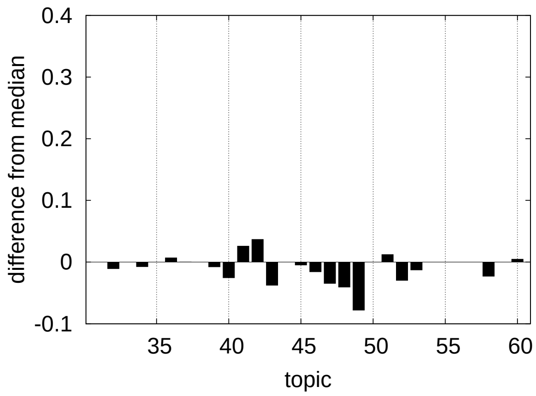


Overall average precision: 0.0071

Average precision per topic:

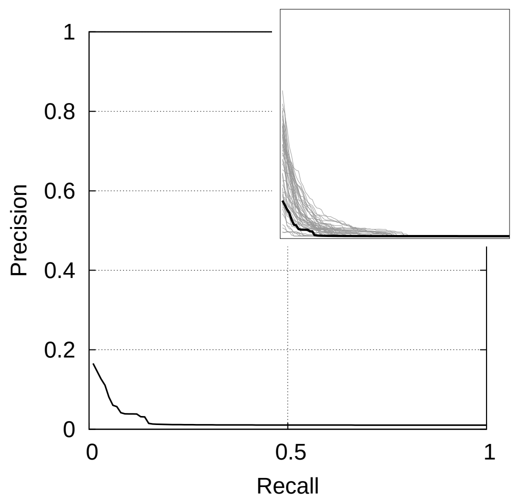
31	0.0002	41	0.0288	51	0.0164
32	0.0006	42	0.0564	52	0.0001
33	0.0001	43	0.0003	53	0.0004
34	0.0024	44	0.0005	54	–
35	–	45	0.0017	55	–
36	0.0097	46	0.0017	56	–
37	0.0037	47	0.0003	57	–
38	0.0038	48	0.0032	58	0.0100
39	0.0004	49	0.0013	59	–
40	0.0091	50	–	60	0.0117

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

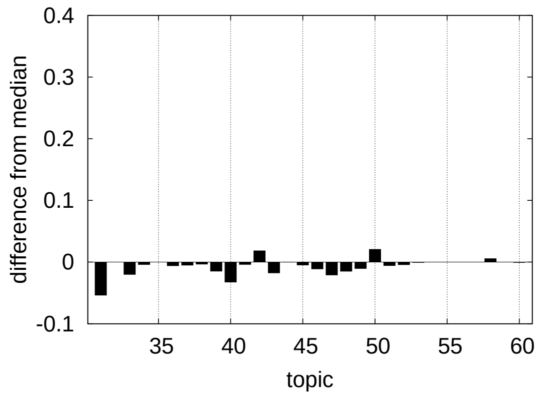


Overall average precision: 0.0194

Average precision per topic:

31	0.0029	41	0.0190	51	0.0152
32	0.0219	42	0.0741	52	0.0176
33	0.0025	43	0.0016	53	0.0121
34	0.0229	44	0.0024	54	–
35	–	45	0.0260	55	–
36	0.0133	46	0.0192	56	–
37	0.0212	47	0.0030	57	–
38	0.0294	48	0.0234	58	0.0500
39	0.0026	49	0.0076	59	–
40	0.0173	50	0.0325	60	0.0269

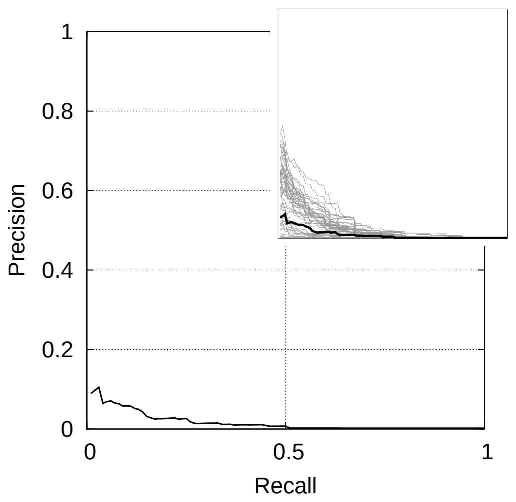
Difference from median in average precision per topic:



Queen Mary University of London
QMUL2 (CO)

Quantisation: strict

Recall / precision graph:

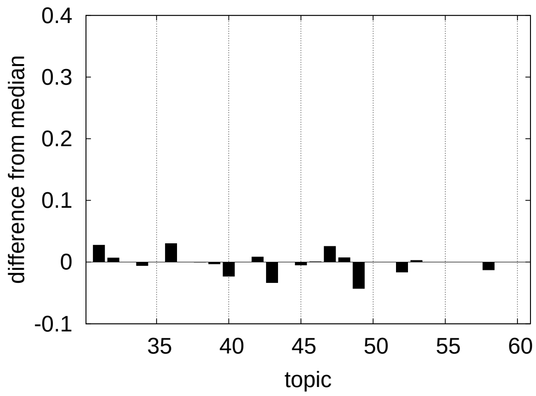


Overall average precision: 0.0163

Average precision per topic:

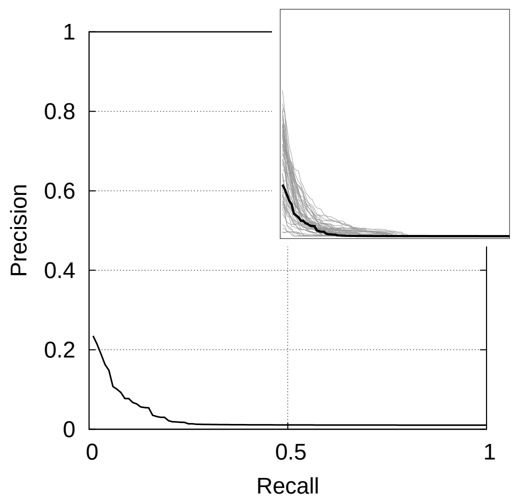
31	0.0281	41	0.0028	51	0.0040
32	0.0188	42	0.0280	52	0.0135
33	0.0001	43	0.0046	53	0.0167
34	0.0043	44	0.0008	54	–
35	–	45	0.0017	55	–
36	0.0329	46	0.0189	56	–
37	0.0033	47	0.0612	57	–
38	0.0031	48	0.0518	58	0.0202
39	0.0052	49	0.0364	59	–
40	0.0116	50	–	60	0.0072

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

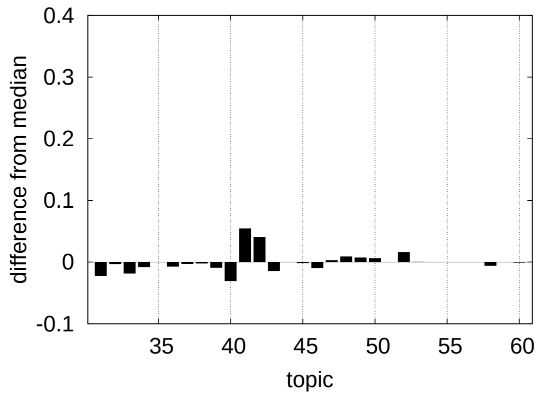


Overall average precision: 0.0275

Average precision per topic:

31	0.0346	41	0.0777	51	0.0212
32	0.0186	42	0.0962	52	0.0383
33	0.0042	43	0.0050	53	0.0137
34	0.0191	44	0.0025	54	–
35	–	45	0.0296	55	–
36	0.0123	46	0.0211	56	–
37	0.0239	47	0.0273	57	–
38	0.0308	48	0.0477	58	0.0381
39	0.0087	49	0.0258	59	–
40	0.0193	50	0.0178	60	0.0269

Difference from median in average precision per topic:

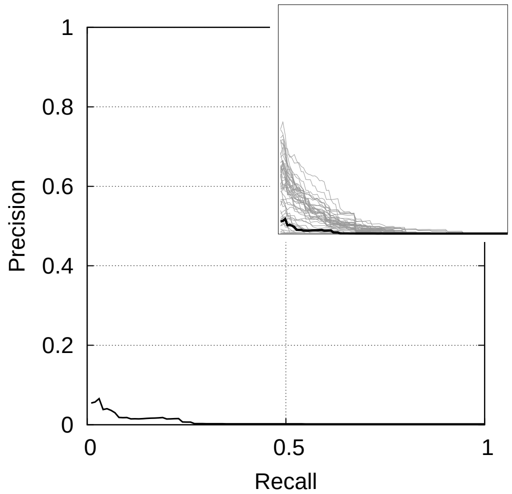


Queen Mary University of London
QMUL3 (CO)

Quantisation: strict	Quantisation: generalised

Quantisation: strict

Recall / precision graph:

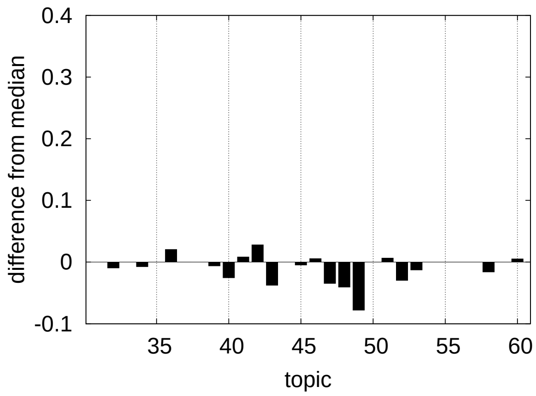


Overall average precision: 0.0077

Average precision per topic:

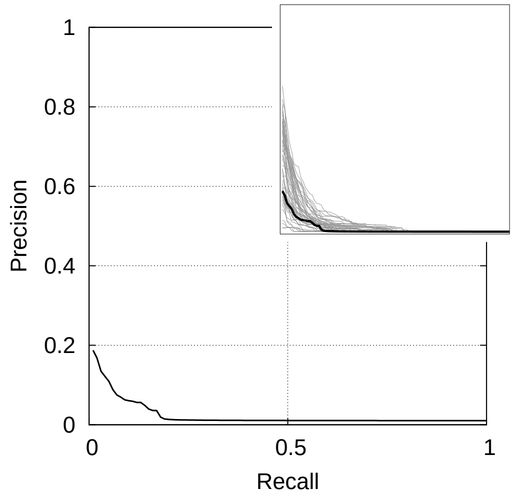
31	0.0002	41	0.0111	51	0.0107
32	0.0017	42	0.0476	52	0.0001
33	0.0001	43	0.0003	53	0.0004
34	0.0024	44	0.0005	54	–
35	–	45	0.0017	55	–
36	0.0232	46	0.0239	56	–
37	0.0037	47	0.0003	57	–
38	0.0036	48	0.0032	58	0.0169
39	0.0020	49	0.0013	59	–
40	0.0091	50	–	60	0.0123

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

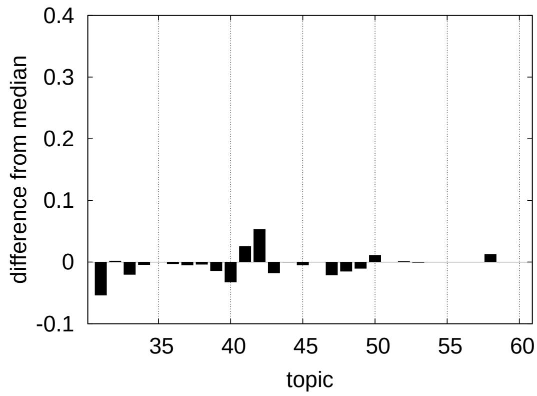


Overall average precision: 0.0232

Average precision per topic:

31	0.0029	41	0.0490	51	0.0209
32	0.0240	42	0.1088	52	0.0235
33	0.0025	43	0.0016	53	0.0122
34	0.0227	44	0.0024	54	–
35	–	45	0.0260	55	–
36	0.0165	46	0.0307	56	–
37	0.0213	47	0.0030	57	–
38	0.0291	48	0.0234	58	0.0569
39	0.0034	49	0.0078	59	–
40	0.0173	50	0.0229	60	0.0279

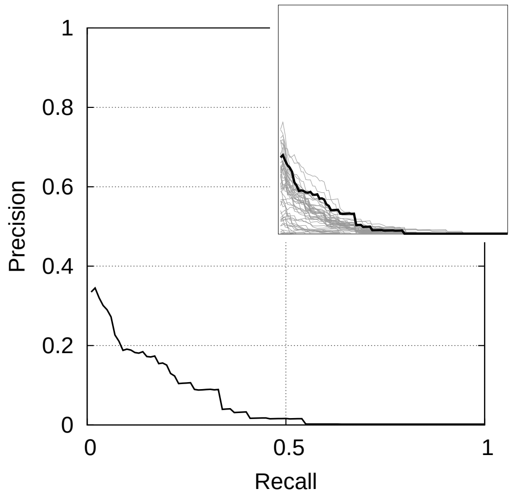
Difference from median in average precision per topic:



Queensland University of Technology
inexresult2.xml (CO)

Quantisation: strict

Recall / precision graph:

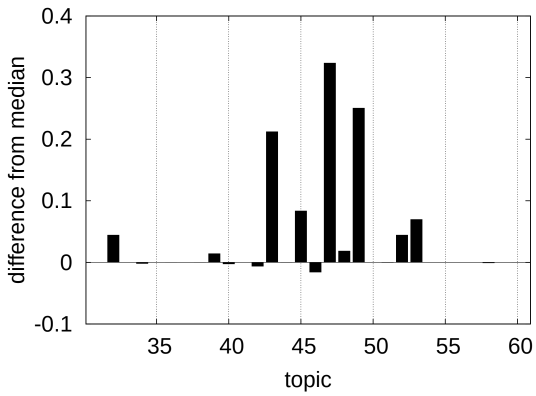


Overall average precision: 0.0627

Average precision per topic:

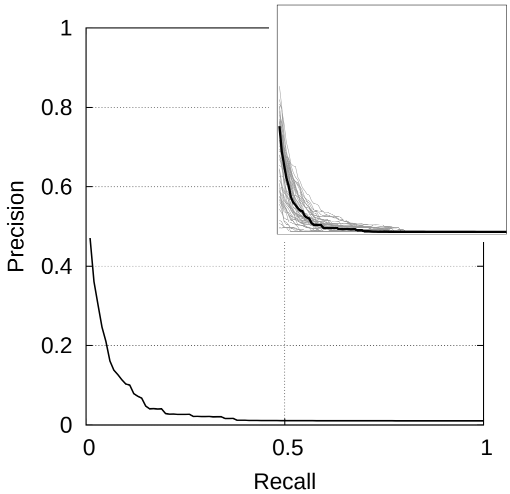
31	0.0002	41	0.0024	51	0.0035
32	0.0564	42	0.0128	52	0.0748
33	0.0001	43	0.2508	53	0.0836
34	0.0081	44	0.0010	54	–
35	–	45	0.0907	55	–
36	0.0023	46	0.0018	56	–
37	0.0032	47	0.3592	57	–
38	0.0036	48	0.0631	58	0.0322
39	0.0230	49	0.3304	59	–
40	0.0321	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

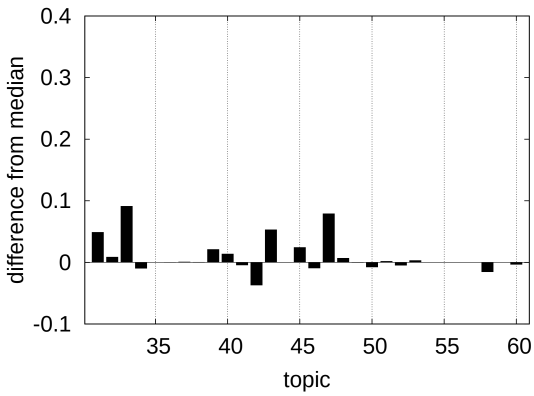


Overall average precision: 0.0385

Average precision per topic:

31	0.1062	41	0.0186	51	0.0233
32	0.0308	42	0.0183	52	0.0171
33	0.1144	43	0.0729	53	0.0166
34	0.0173	44	0.0030	54	–
35	–	45	0.0558	55	–
36	0.0194	46	0.0211	56	–
37	0.0277	47	0.1037	57	–
38	0.0337	48	0.0457	58	0.0284
39	0.0391	49	0.0180	59	–
40	0.0641	50	0.0037	60	0.0243

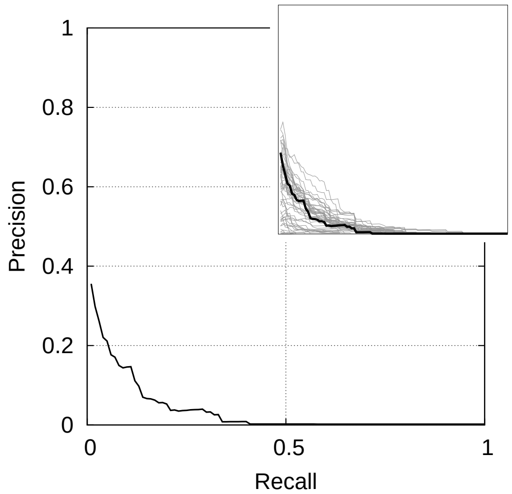
Difference from median in average precision per topic:



Queensland University of Technology
inexresults1.xml (CO)

Quantisation: strict

Recall / precision graph:

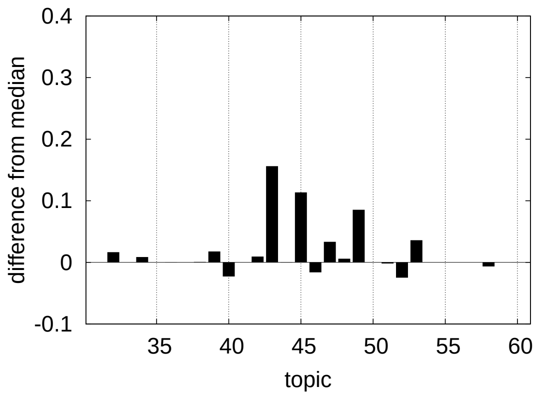


Overall average precision: 0.0356

Average precision per topic:

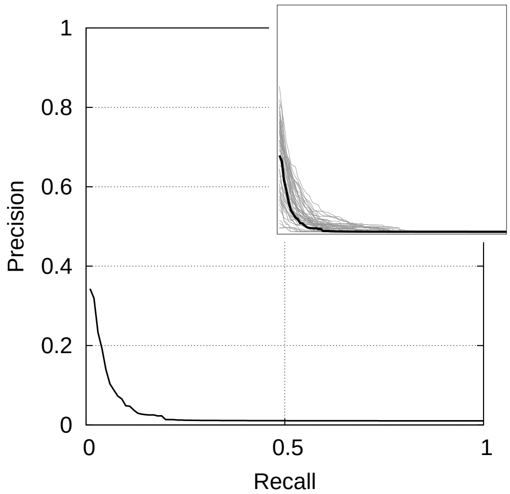
31	0.0002	41	0.0024	51	0.0021
32	0.0283	42	0.0289	52	0.0054
33	0.0001	43	0.1945	53	0.0496
34	0.0189	44	0.0005	54	–
35	–	45	0.1205	55	–
36	0.0028	46	0.0018	56	–
37	0.0032	47	0.0687	57	–
38	0.0045	48	0.0503	58	0.0268
39	0.0263	49	0.1651	59	–
40	0.0121	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

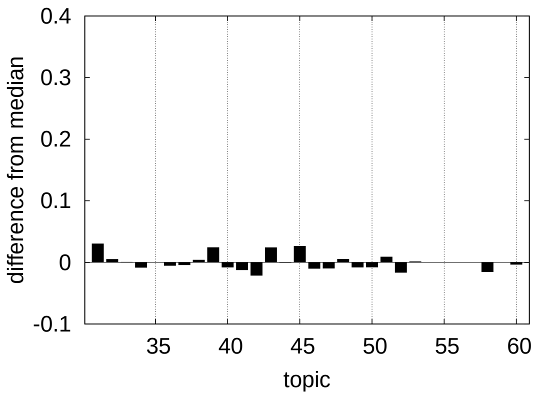


Overall average precision: 0.0275

Average precision per topic:

31	0.0875	41	0.0107	51	0.0305
32	0.0272	42	0.0341	52	0.0055
33	0.0236	43	0.0439	53	0.0148
34	0.0187	44	0.0024	54	–
35	–	45	0.0577	55	–
36	0.0142	46	0.0205	56	–
37	0.0223	47	0.0146	57	–
38	0.0373	48	0.0440	58	0.0284
39	0.0423	49	0.0103	59	–
40	0.0419	50	0.0035	60	0.0243

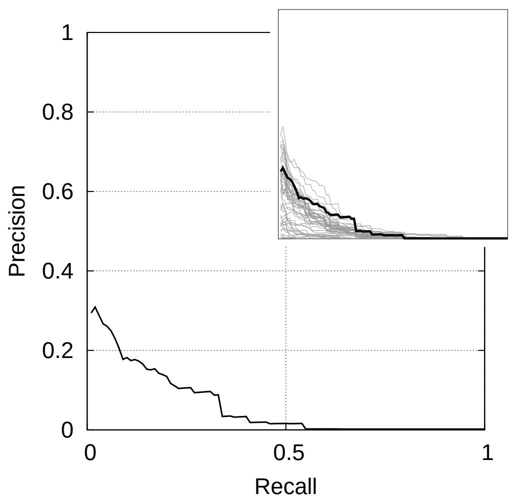
Difference from median in average precision per topic:



Queensland University of Technology
inexresults3.xml (CO)

Quantisation: strict

Recall / precision graph:

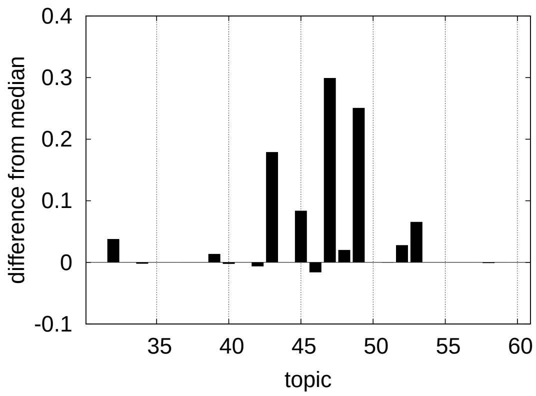


Overall average precision: 0.0590

Average precision per topic:

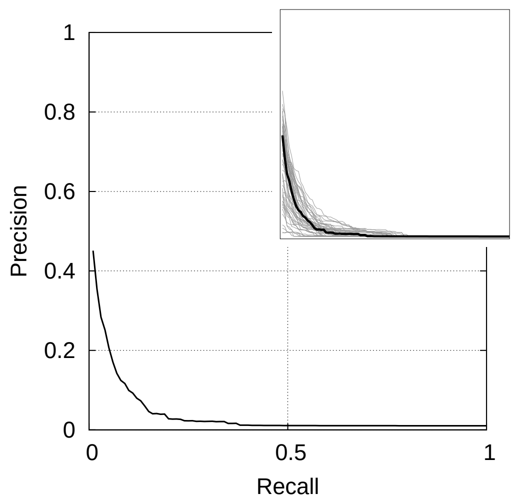
31	0.0002	41	0.0024	51	0.0035
32	0.0496	42	0.0129	52	0.0582
33	0.0001	43	0.2174	53	0.0794
34	0.0081	44	0.0007	54	–
35	–	45	0.0907	55	–
36	0.0023	46	0.0018	56	–
37	0.0032	47	0.3347	57	–
38	0.0036	48	0.0645	58	0.0322
39	0.0223	49	0.3304	59	–
40	0.0325	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

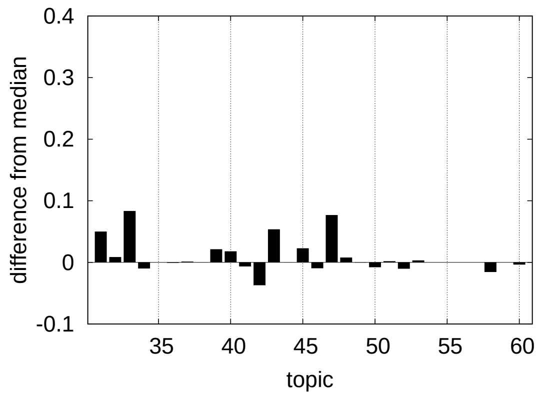


Overall average precision: 0.0379

Average precision per topic:

31	0.1071	41	0.0166	51	0.0232
32	0.0307	42	0.0184	52	0.0119
33	0.1064	43	0.0733	53	0.0164
34	0.0173	44	0.0027	54	–
35	–	45	0.0541	55	–
36	0.0188	46	0.0211	56	–
37	0.0280	47	0.1013	57	–
38	0.0333	48	0.0465	58	0.0284
39	0.0392	49	0.0180	59	–
40	0.0681	50	0.0036	60	0.0243

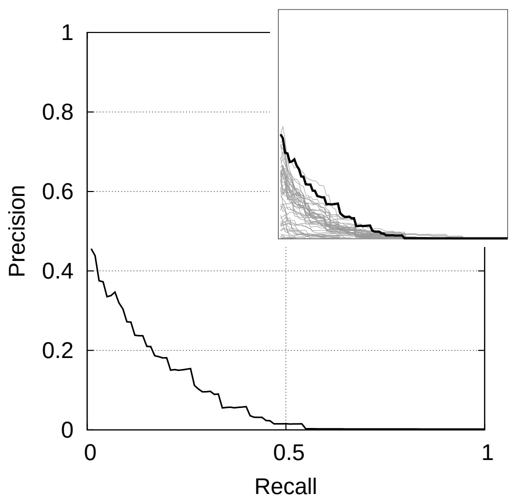
Difference from median in average precision per topic:



Royal School of Library and Information Science
bag-of-words (CO)

Quantisation: strict

Recall / precision graph:

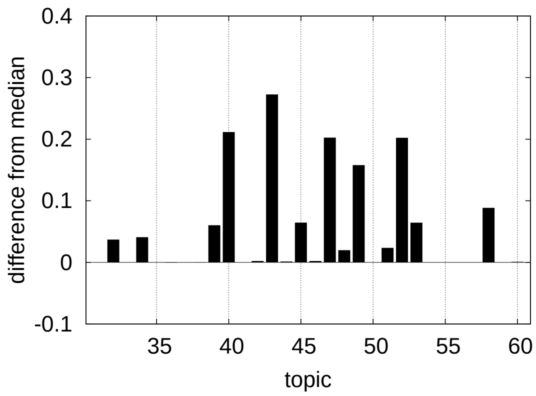


Overall average precision: 0.0809

Average precision per topic:

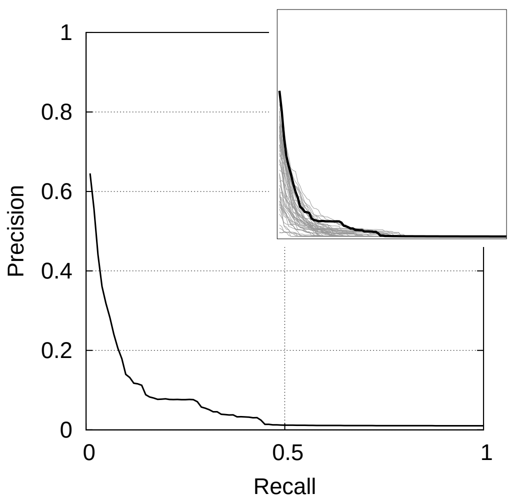
31	0.0002	41	0.0024	51	0.0275
32	0.0487	42	0.0215	52	0.2325
33	0.0001	43	0.3109	53	0.0780
34	0.0511	44	0.0022	54	–
35	–	45	0.0715	55	–
36	0.0021	46	0.0201	56	–
37	0.0032	47	0.2379	57	–
38	0.0039	48	0.0641	58	0.1219
39	0.0689	49	0.2376	59	–
40	0.2465	50	–	60	0.0077

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

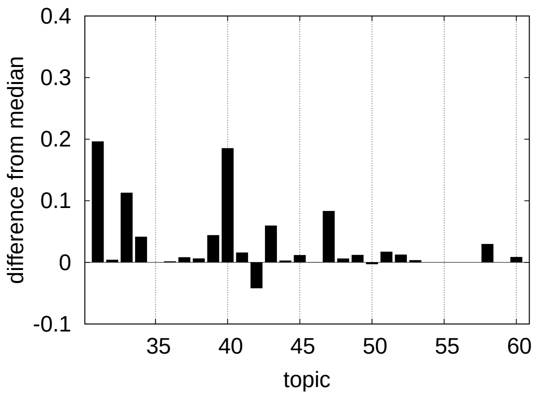


Overall average precision: 0.0618

Average precision per topic:

31	0.2535	41	0.0392	51	0.0386
32	0.0262	42	0.0134	52	0.0349
33	0.1360	43	0.0793	53	0.0168
34	0.0688	44	0.0059	54	–
35	–	45	0.0432	55	–
36	0.0213	46	0.0312	56	–
37	0.0351	47	0.1079	57	–
38	0.0395	48	0.0449	58	0.0739
39	0.0620	49	0.0305	59	–
40	0.2356	50	0.0085	60	0.0367

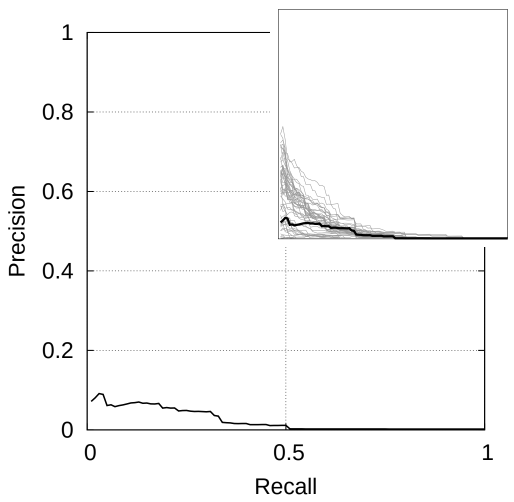
Difference from median in average precision per topic:



Royal School of Library and Information Science
boomerang (CO)

Quantisation: strict

Recall / precision graph:

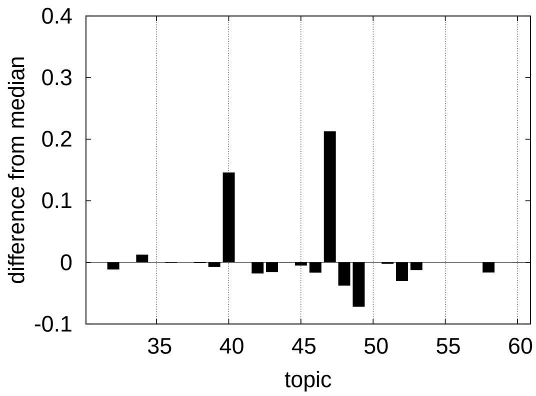


Overall average precision: 0.0231

Average precision per topic:

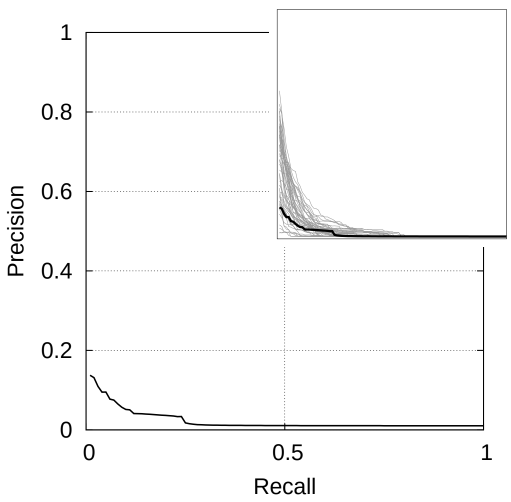
31	0.0002	41	0.0024	51	0.0016
32	0.0002	42	0.0014	52	0.0001
33	0.0001	43	0.0225	53	0.0011
34	0.0228	44	0.0005	54	–
35	–	45	0.0017	55	–
36	0.0017	46	0.0012	56	–
37	0.0032	47	0.2482	57	–
38	0.0026	48	0.0065	58	0.0170
39	0.0012	49	0.0076	59	–
40	0.1810	50	–	60	0.0065

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

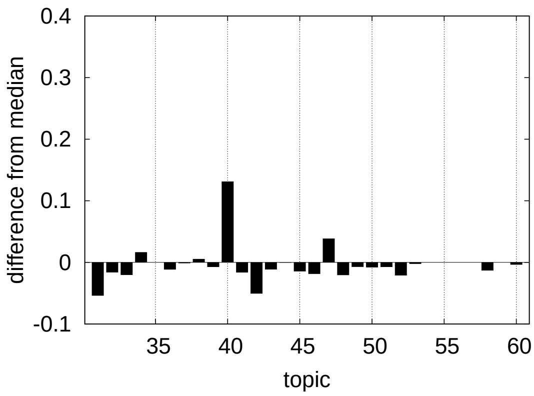


Overall average precision: 0.0227

Average precision per topic:

31	0.0029	41	0.0068	51	0.0137
32	0.0056	42	0.0048	52	0.0010
33	0.0025	43	0.0080	53	0.0107
34	0.0437	44	0.0024	54	–
35	–	45	0.0166	55	–
36	0.0080	46	0.0120	56	–
37	0.0253	47	0.0631	57	–
38	0.0387	48	0.0179	58	0.0309
39	0.0103	49	0.0111	59	–
40	0.1815	50	0.0034	60	0.0241

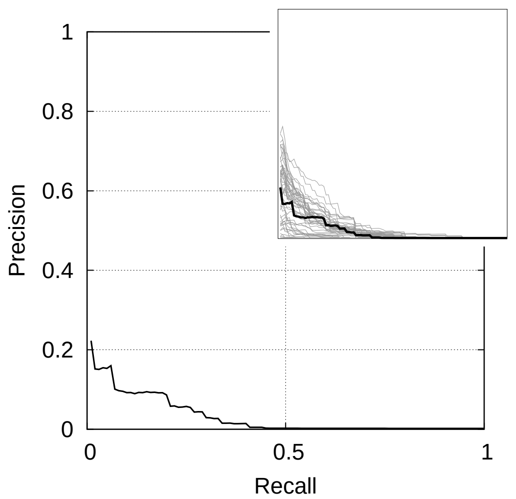
Difference from median in average precision per topic:



Royal School of Library and Information Science
polyrepresentation (CO)

Quantisation: strict

Recall / precision graph:

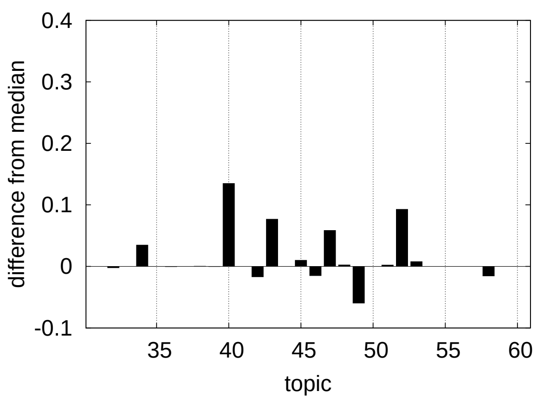


Overall average precision: 0.0313

Average precision per topic:

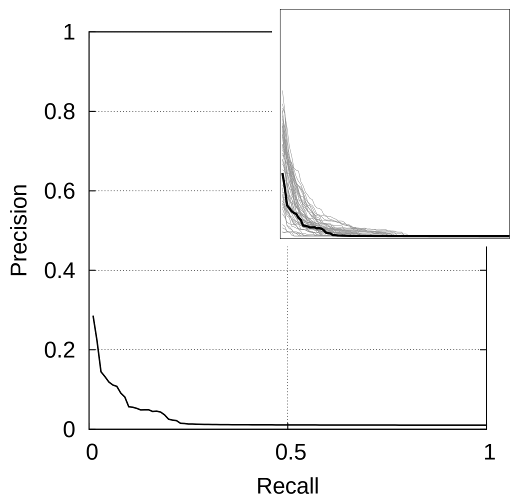
31	0.0002	41	0.0024	51	0.0065
32	0.0091	42	0.0021	52	0.1234
33	0.0001	43	0.1154	53	0.0218
34	0.0453	44	0.0005	54	–
35	–	45	0.0173	55	–
36	0.0017	46	0.0026	56	–
37	0.0032	47	0.0943	57	–
38	0.0044	48	0.0472	58	0.0174
39	0.0080	49	0.0197	59	–
40	0.1702	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

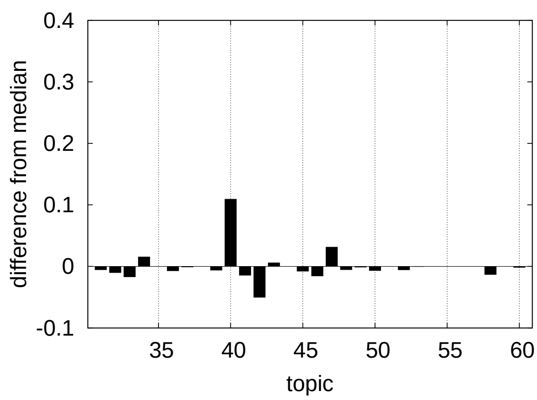


Overall average precision: 0.0271

Average precision per topic:

31	0.0512	41	0.0085	51	0.0210
32	0.0115	42	0.0050	52	0.0163
33	0.0055	43	0.0257	53	0.0126
34	0.0431	44	0.0031	54	–
35	–	45	0.0231	55	–
36	0.0121	46	0.0148	56	–
37	0.0253	47	0.0561	57	–
38	0.0327	48	0.0329	58	0.0305
39	0.0113	49	0.0168	59	–
40	0.1597	50	0.0044	60	0.0260

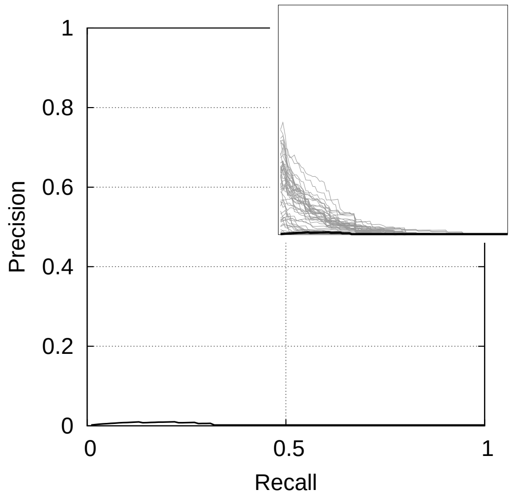
Difference from median in average precision per topic:



Salzburg Research Forschungsgesellschaft
1-corrected (CO)

Quantisation: strict

Recall / precision graph:

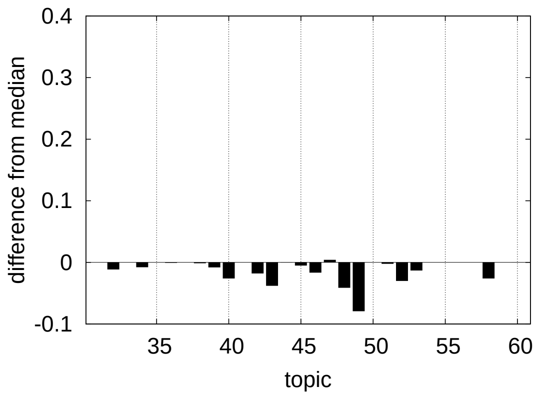


Overall average precision: 0.0037

Average precision per topic:

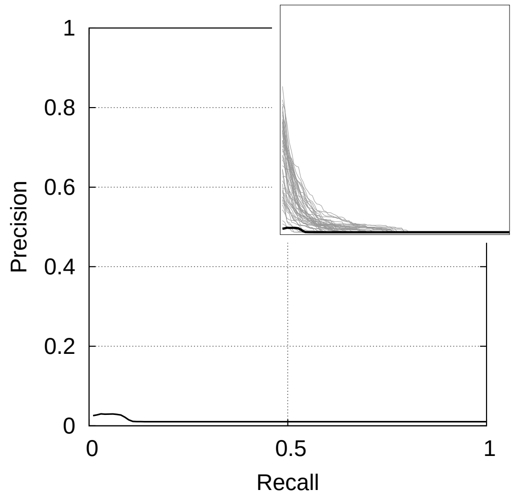
31	0.0002	41	0.0024	51	0.0016
32	0.0002	42	0.0014	52	0.0001
33	0.0001	43	0.0003	53	0.0004
34	0.0025	44	0.0005	54	–
35	–	45	0.0017	55	–
36	0.0017	46	0.0012	56	–
37	0.0032	47	0.0395	57	–
38	0.0023	48	0.0030	58	0.0072
39	0.0004	49	0.0004	59	–
40	0.0088	50	–	60	0.0065

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

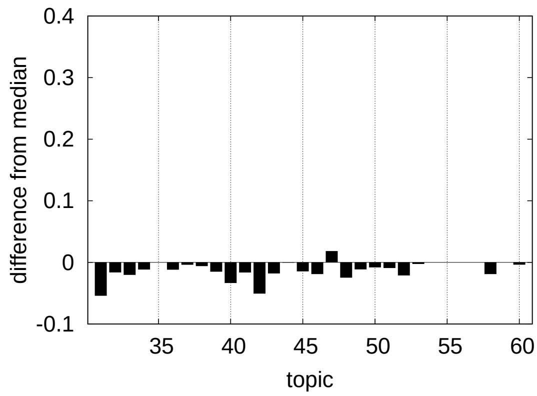


Overall average precision: 0.0120

Average precision per topic:

31	0.0028	41	0.0069	51	0.0120
32	0.0056	42	0.0049	52	0.0010
33	0.0025	43	0.0015	53	0.0106
34	0.0156	44	0.0024	54	–
35	–	45	0.0166	55	–
36	0.0078	46	0.0117	56	–
37	0.0227	47	0.0427	57	–
38	0.0270	48	0.0138	58	0.0250
39	0.0026	49	0.0070	59	–
40	0.0166	50	0.0034	60	0.0241

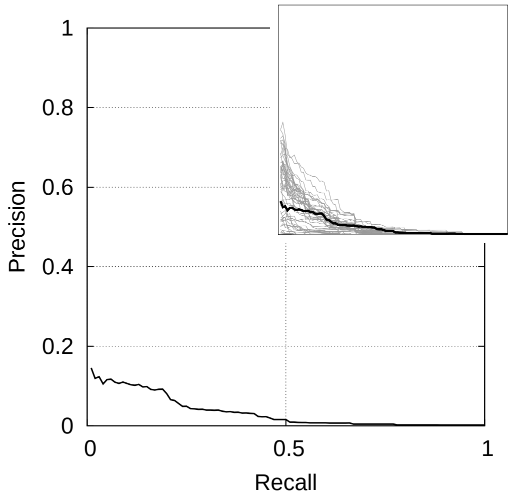
Difference from median in average precision per topic:



Sejong Cyber University
TitleKeywordsWLErr (CO)

Quantisation: strict

Recall / precision graph:

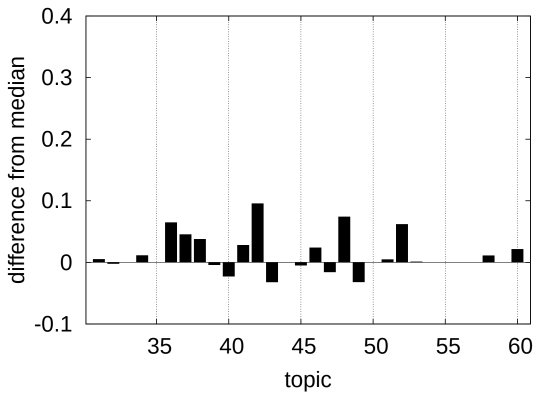


Overall average precision: 0.0340

Average precision per topic:

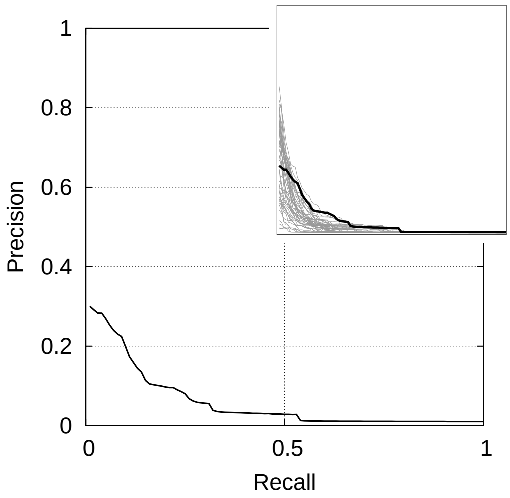
31	0.0055	41	0.0307	51	0.0087
32	0.0094	42	0.1151	52	0.0924
33	0.0001	43	0.0060	53	0.0149
34	0.0218	44	0.0009	54	–
35	–	45	0.0019	55	–
36	0.0674	46	0.0420	56	–
37	0.0488	47	0.0194	57	–
38	0.0416	48	0.1186	58	0.0446
39	0.0043	49	0.0476	59	–
40	0.0120	50	–	60	0.0284

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

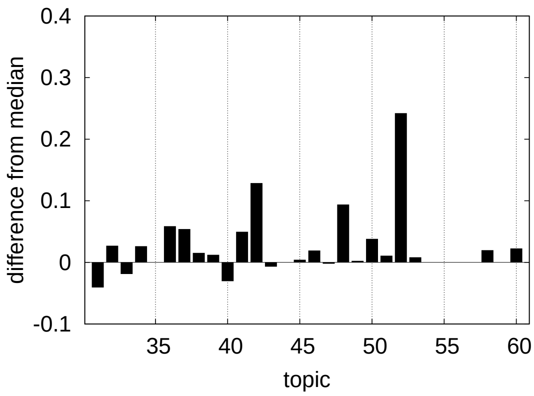


Overall average precision: 0.0582

Average precision per topic:

31	0.0163	41	0.0729	51	0.0321
32	0.0490	42	0.1843	52	0.2645
33	0.0040	43	0.0126	53	0.0214
34	0.0535	44	0.0028	54	–
35	–	45	0.0355	55	–
36	0.0784	46	0.0500	56	–
37	0.0808	47	0.0222	57	–
38	0.0485	48	0.1326	58	0.0638
39	0.0301	49	0.0209	59	–
40	0.0195	50	0.0497	60	0.0506

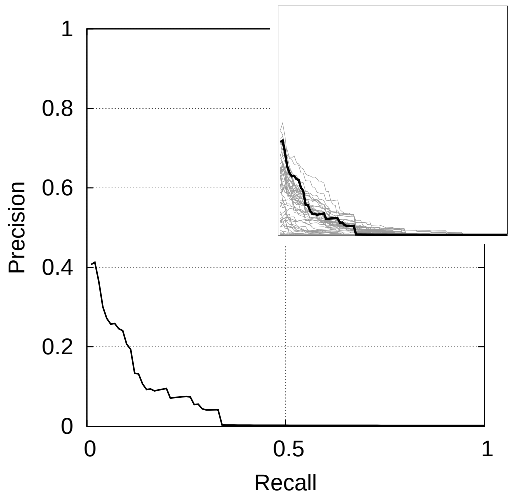
Difference from median in average precision per topic:



Tarragon Consulting Corporation
tgnCO_base (CO)

Quantisation: strict

Recall / precision graph:

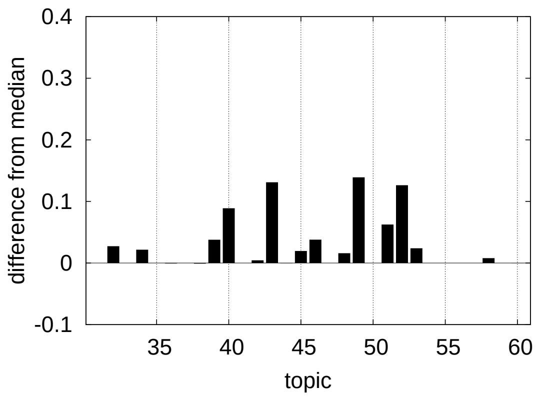


Overall average precision: 0.0500

Average precision per topic:

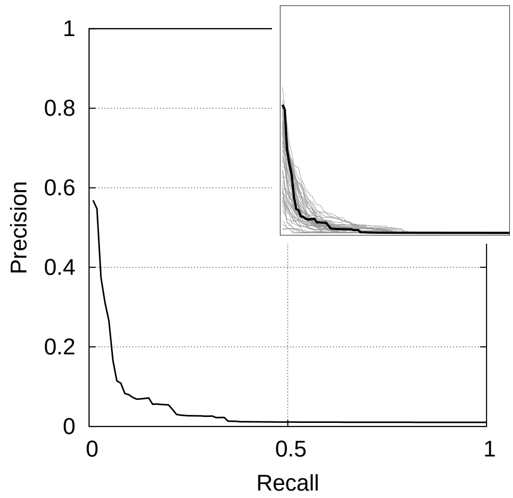
31	0.0002	41	0.0024	51	0.0664
32	0.0390	42	0.0237	52	0.1565
33	0.0001	43	0.1694	53	0.0376
34	0.0319	44	0.0005	54	–
35	–	45	0.0264	55	–
36	0.0017	46	0.0559	56	–
37	0.0032	47	0.0354	57	–
38	0.0027	48	0.0601	58	0.0413
39	0.0464	49	0.2188	59	–
40	0.1239	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

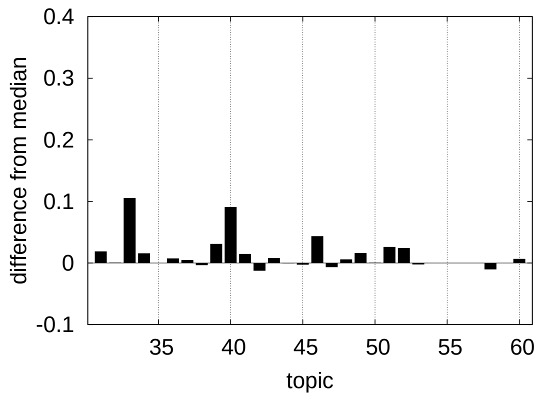


Overall average precision: 0.0435

Average precision per topic:

31	0.0759	41	0.0380	51	0.0474
32	0.0227	42	0.0429	52	0.0465
33	0.1284	43	0.0276	53	0.0109
34	0.0430	44	0.0025	54	–
35	–	45	0.0284	55	–
36	0.0271	46	0.0743	56	–
37	0.0315	47	0.0177	57	–
38	0.0296	48	0.0446	58	0.0337
39	0.0489	49	0.0346	59	–
40	0.1410	50	0.0121	60	0.0347

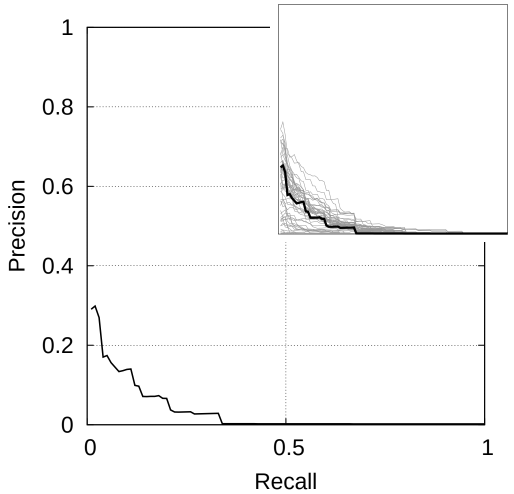
Difference from median in average precision per topic:



Universität Bayreuth
IRStream (CO)

Quantisation: strict

Recall / precision graph:

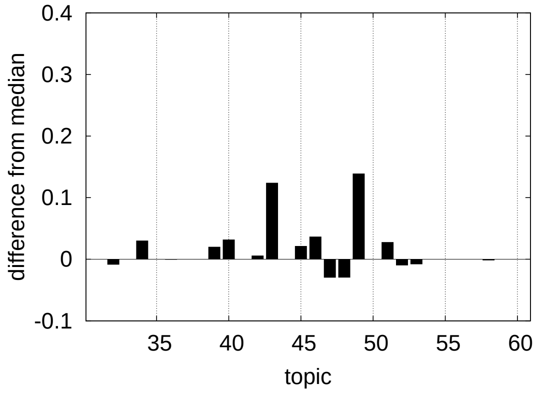


Overall average precision: 0.0329

Average precision per topic:

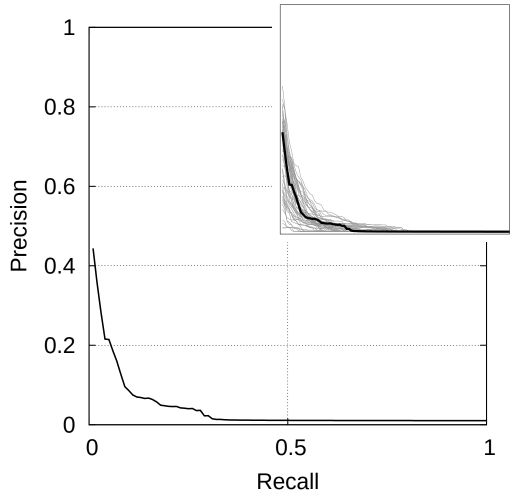
31	0.0002	41	0.0024	51	0.0317
32	0.0029	42	0.0253	52	0.0203
33	0.0001	43	0.1624	53	0.0055
34	0.0406	44	0.0005	54	–
35	–	45	0.0284	55	–
36	0.0017	46	0.0547	56	–
37	0.0032	47	0.0055	57	–
38	0.0033	48	0.0146	58	0.0315
39	0.0288	49	0.2188	59	–
40	0.0669	50	–	60	0.0065

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

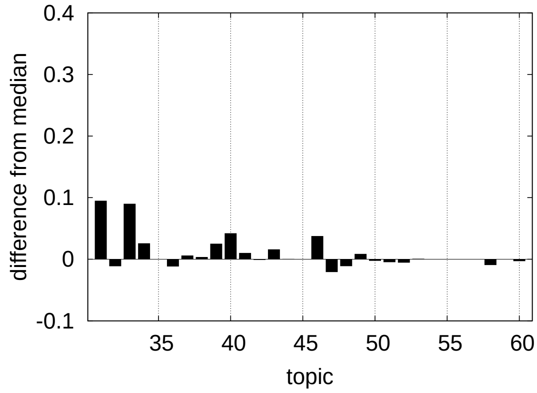


Overall average precision: 0.0392

Average precision per topic:

31	0.1520	41	0.0335	51	0.0165
32	0.0106	42	0.0544	52	0.0167
33	0.1130	43	0.0357	53	0.0140
34	0.0531	44	0.0034	54	–
35	–	45	0.0314	55	–
36	0.0079	46	0.0683	56	–
37	0.0327	47	0.0037	57	–
38	0.0367	48	0.0273	58	0.0346
39	0.0431	49	0.0271	59	–
40	0.0924	50	0.0090	60	0.0248

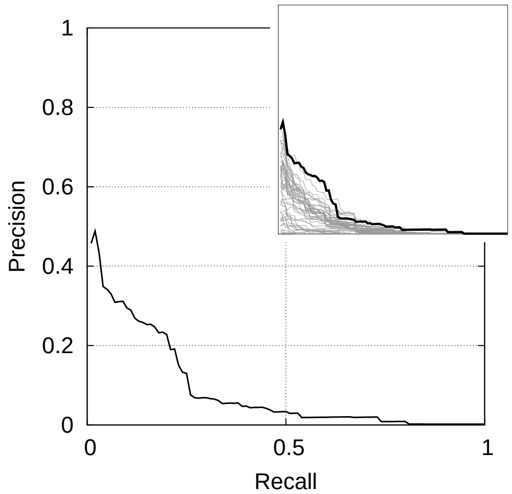
Difference from median in average precision per topic:



Universität Dortmund / Universität Duisburg-Essen
Epros03 (CO)

Quantisation: strict

Recall / precision graph:

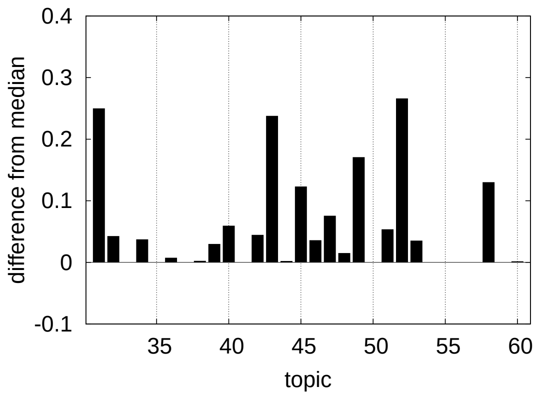


Overall average precision: 0.0883

Average precision per topic:

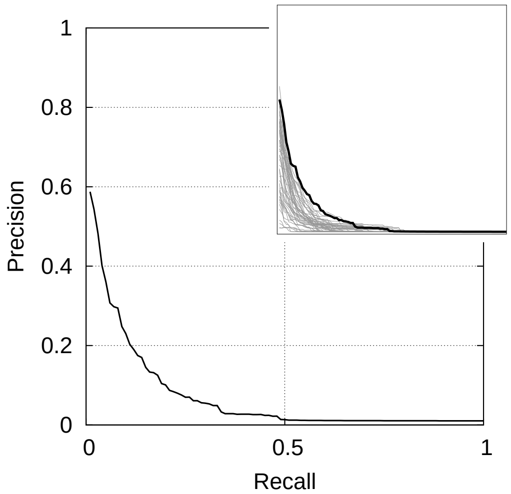
31	0.2503	41	0.0024	51	0.0575
32	0.0544	42	0.0641	52	0.2964
33	0.0001	43	0.2762	53	0.0489
34	0.0477	44	0.0029	54	–
35	–	45	0.1300	55	–
36	0.0100	46	0.0539	56	–
37	0.0032	47	0.1111	57	–
38	0.0061	48	0.0594	58	0.1635
39	0.0385	49	0.2504	59	–
40	0.0945	50	–	60	0.0085

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

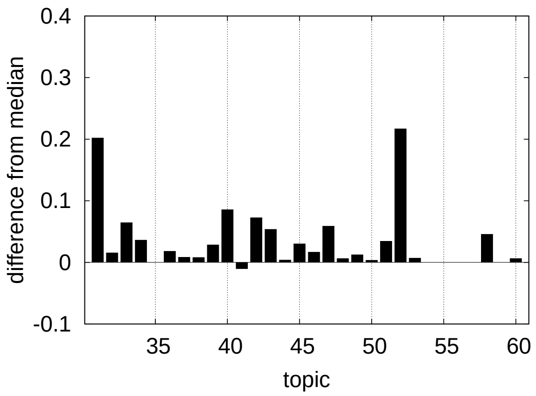


Overall average precision: 0.0705

Average precision per topic:

31	0.2592	41	0.0126	51	0.0560
32	0.0377	42	0.1284	52	0.2394
33	0.0878	43	0.0735	53	0.0204
34	0.0637	44	0.0070	54	–
35	–	45	0.0616	55	–
36	0.0380	46	0.0476	56	–
37	0.0353	47	0.0836	57	–
38	0.0415	48	0.0452	58	0.0900
39	0.0465	49	0.0310	59	–
40	0.1361	50	0.0153	60	0.0346

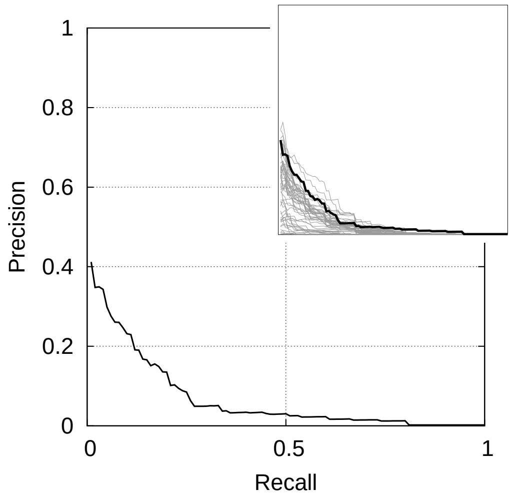
Difference from median in average precision per topic:



Universität Dortmund / Universität Duisburg-Essen
Epros06 (CO)

Quantisation: strict

Recall / precision graph:

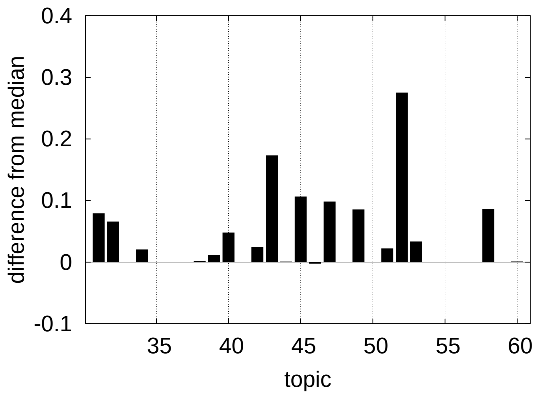


Overall average precision: 0.0670

Average precision per topic:

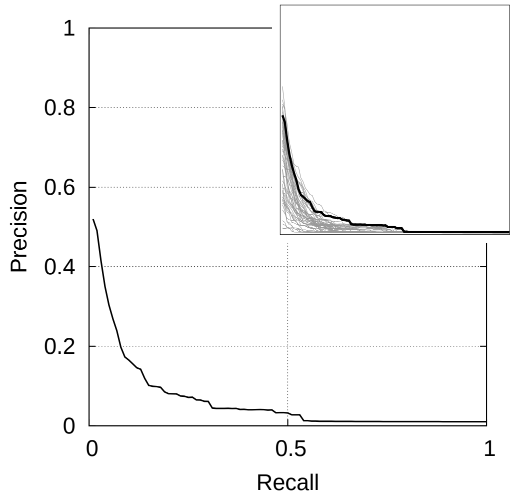
31	0.0794	41	0.0024	51	0.0262
32	0.0776	42	0.0442	52	0.3055
33	0.0001	43	0.2115	53	0.0471
34	0.0308	44	0.0016	54	–
35	–	45	0.1133	55	–
36	0.0030	46	0.0155	56	–
37	0.0032	47	0.1338	57	–
38	0.0057	48	0.0443	58	0.1195
39	0.0205	49	0.1652	59	–
40	0.0829	50	–	60	0.0078

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

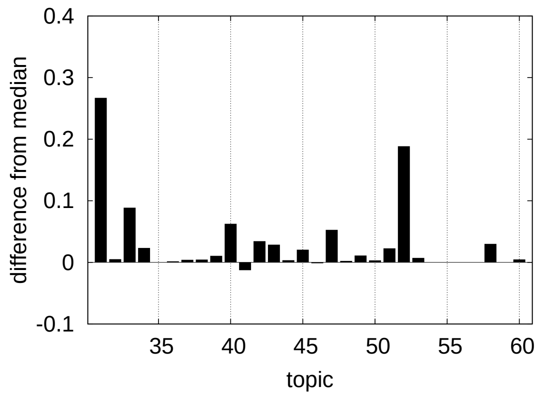


Overall average precision: 0.0635

Average precision per topic:

31	0.3241	41	0.0105	51	0.0440
32	0.0271	42	0.0900	52	0.2107
33	0.1116	43	0.0483	53	0.0204
34	0.0507	44	0.0064	54	–
35	–	45	0.0518	55	–
36	0.0213	46	0.0293	56	–
37	0.0308	47	0.0773	57	–
38	0.0376	48	0.0409	58	0.0741
39	0.0284	49	0.0296	59	–
40	0.1127	50	0.0148	60	0.0327

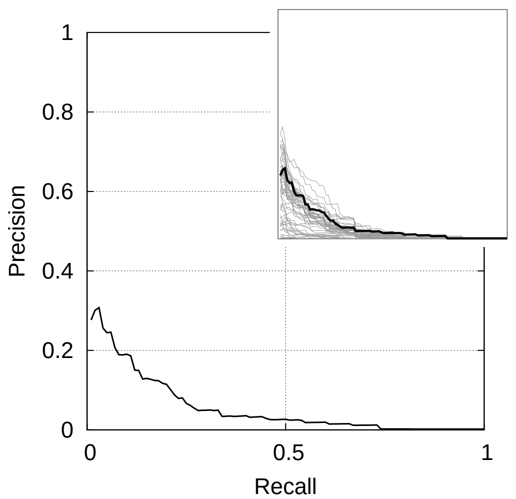
Difference from median in average precision per topic:



Universität Dortmund / Universität Duisburg-Essen
plain hyrex (CO)

Quantisation: strict

Recall / precision graph:

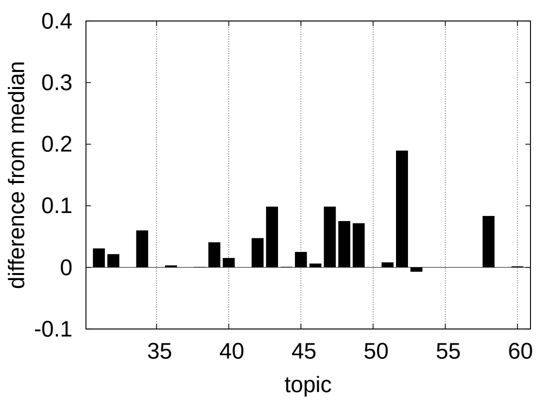


Overall average precision: 0.0556

Average precision per topic:

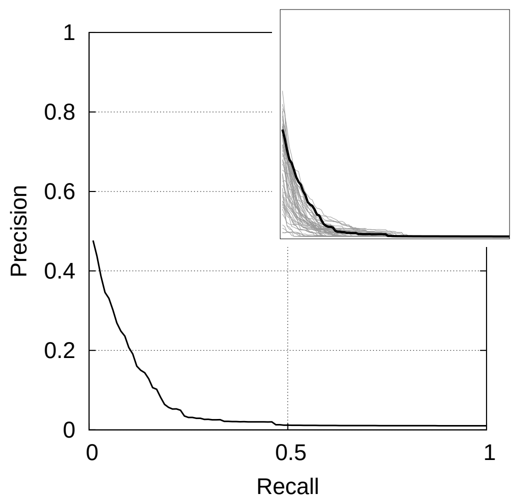
31	0.0311	41	0.0024	51	0.0120
32	0.0332	42	0.0669	52	0.2197
33	0.0001	43	0.1368	53	0.0065
34	0.0703	44	0.0017	54	–
35	–	45	0.0319	55	–
36	0.0057	46	0.0241	56	–
37	0.0032	47	0.1340	57	–
38	0.0044	48	0.1194	58	0.1168
39	0.0493	49	0.1514	59	–
40	0.0502	50	–	60	0.0085

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

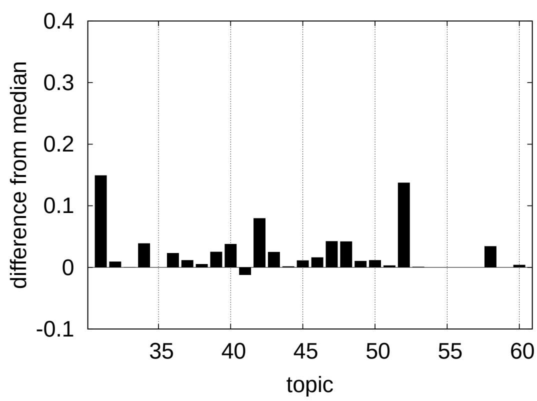


Overall average precision: 0.0572

Average precision per topic:

31	0.2065	41	0.0109	51	0.0245
32	0.0314	42	0.1354	52	0.1597
33	0.0236	43	0.0447	53	0.0142
34	0.0663	44	0.0045	54	–
35	–	45	0.0426	55	–
36	0.0429	46	0.0470	56	–
37	0.0385	47	0.0671	57	–
38	0.0386	48	0.0808	58	0.0785
39	0.0432	49	0.0289	59	–
40	0.0881	50	0.0234	60	0.0320

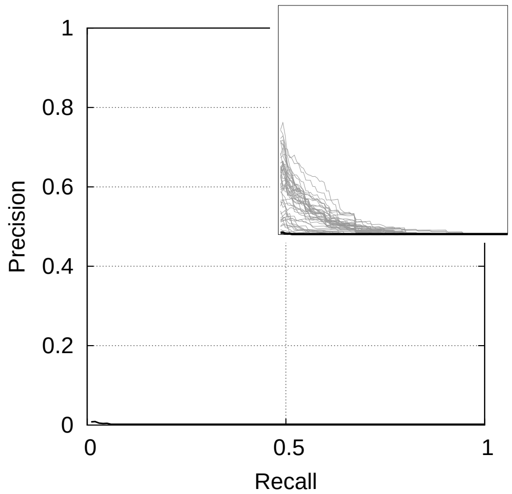
Difference from median in average precision per topic:



Université Pierre et Marie Curie
bayes-2 (CO)

Quantisation: strict

Recall / precision graph:

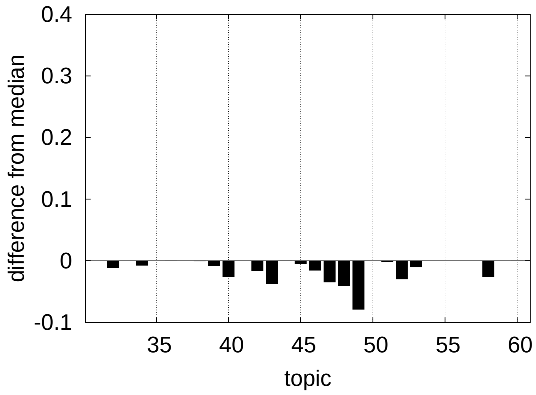


Overall average precision: 0.0023

Average precision per topic:

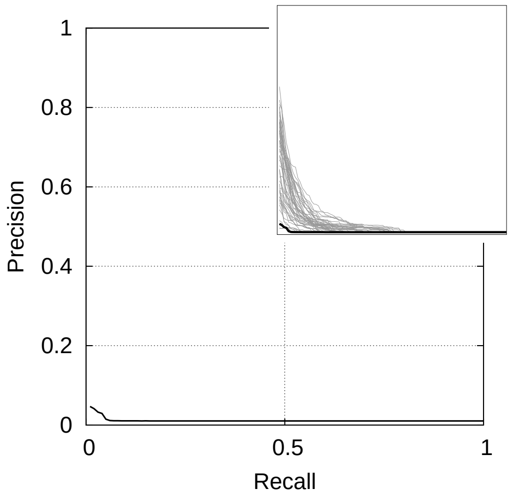
31	0.0002	41	0.0024	51	0.0016
32	0.0002	42	0.0029	52	0.0001
33	0.0001	43	0.0003	53	0.0031
34	0.0024	44	0.0005	54	–
35	–	45	0.0019	55	–
36	0.0017	46	0.0020	56	–
37	0.0032	47	0.0003	57	–
38	0.0028	48	0.0030	58	0.0072
39	0.0004	49	0.0004	59	–
40	0.0088	50	–	60	0.0065

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

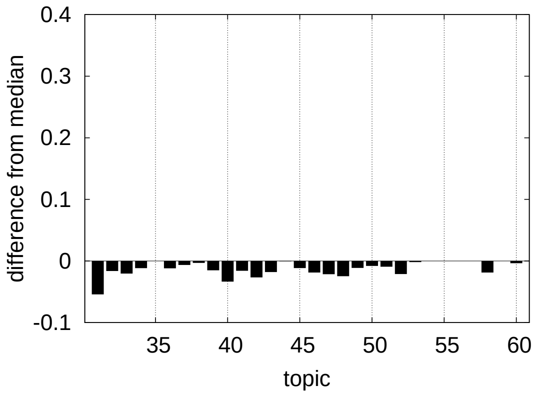


Overall average precision: 0.0115

Average precision per topic:

31	0.0028	41	0.0073	51	0.0120
32	0.0056	42	0.0289	52	0.0010
33	0.0025	43	0.0015	53	0.0113
34	0.0156	44	0.0024	54	–
35	–	45	0.0196	55	–
36	0.0077	46	0.0119	56	–
37	0.0202	47	0.0029	57	–
38	0.0300	48	0.0138	58	0.0253
39	0.0026	49	0.0072	59	–
40	0.0166	50	0.0035	60	0.0242

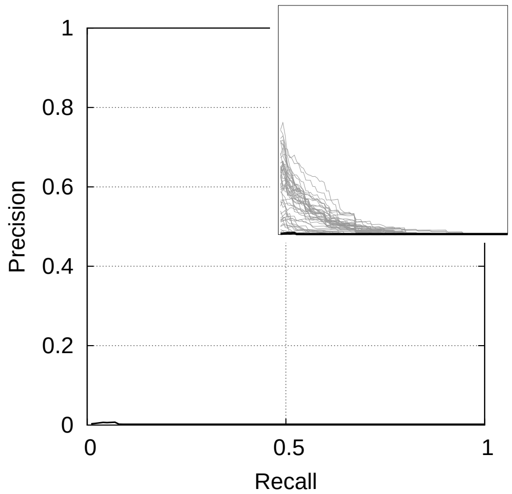
Difference from median in average precision per topic:



Université Pierre et Marie Curie
bayes-3 (CO)

Quantisation: strict

Recall / precision graph:

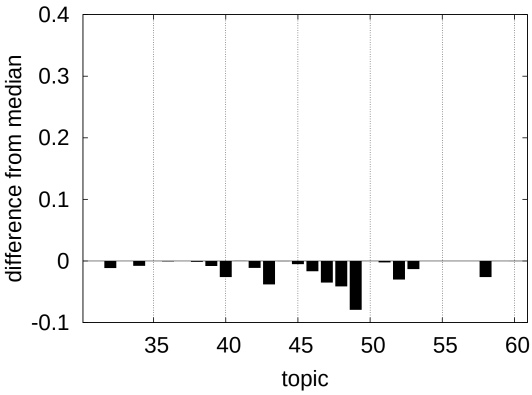


Overall average precision: 0.0023

Average precision per topic:

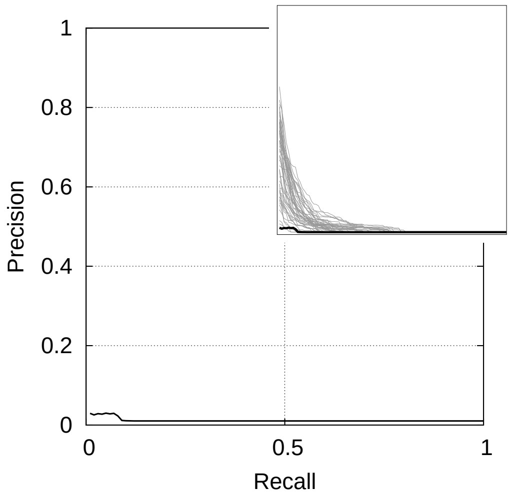
31	0.0002	41	0.0027	51	0.0016
32	0.0002	42	0.0081	52	0.0001
33	0.0001	43	0.0003	53	0.0004
34	0.0024	44	0.0005	54	–
35	–	45	0.0017	55	–
36	0.0017	46	0.0012	56	–
37	0.0032	47	0.0003	57	–
38	0.0023	48	0.0030	58	0.0072
39	0.0004	49	0.0004	59	–
40	0.0088	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

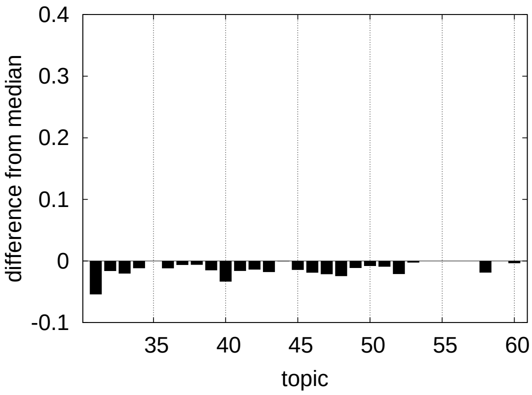


Overall average precision: 0.0117

Average precision per topic:

31	0.0028	41	0.0070	51	0.0120
32	0.0056	42	0.0416	52	0.0010
33	0.0025	43	0.0015	53	0.0106
34	0.0155	44	0.0024	54	–
35	–	45	0.0167	55	–
36	0.0077	46	0.0117	56	–
37	0.0202	47	0.0029	57	–
38	0.0269	48	0.0139	58	0.0252
39	0.0026	49	0.0071	59	–
40	0.0166	50	0.0034	60	0.0242

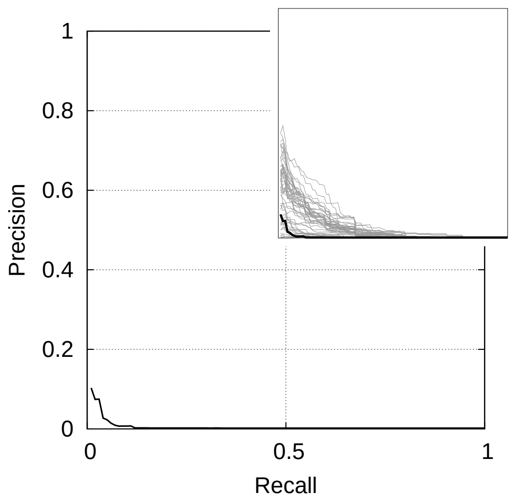
Difference from median in average precision per topic:



Université Pierre et Marie Curie
simple (CO)

Quantisation: strict

Recall / precision graph:

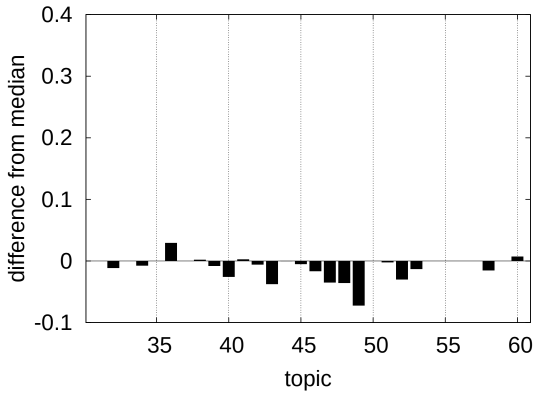


Overall average precision: 0.0055

Average precision per topic:

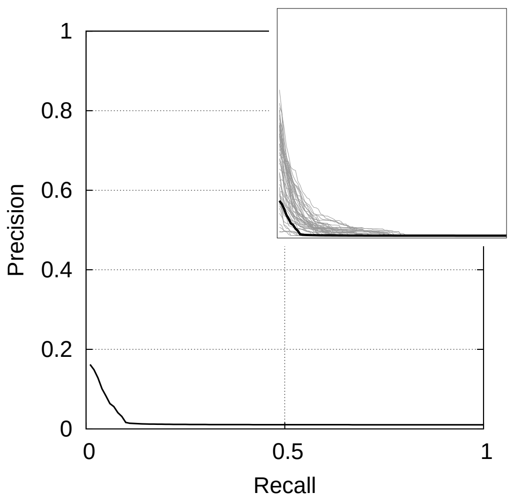
31	0.0002	41	0.0051	51	0.0016
32	0.0002	42	0.0134	52	0.0001
33	0.0001	43	0.0007	53	0.0004
34	0.0027	44	0.0005	54	–
35	–	45	0.0017	55	–
36	0.0318	46	0.0012	56	–
37	0.0032	47	0.0003	57	–
38	0.0058	48	0.0084	58	0.0179
39	0.0004	49	0.0073	59	–
40	0.0091	50	–	60	0.0139

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

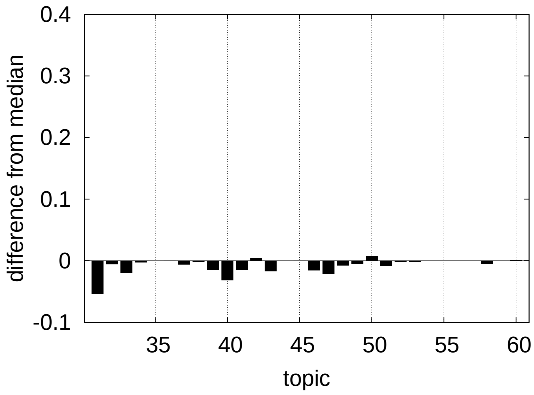


Overall average precision: 0.0181

Average precision per topic:

31	0.0030	41	0.0081	51	0.0126
32	0.0161	42	0.0601	52	0.0198
33	0.0025	43	0.0024	53	0.0108
34	0.0243	44	0.0028	54	–
35	–	45	0.0310	55	–
36	0.0190	46	0.0149	56	–
37	0.0203	47	0.0030	57	–
38	0.0311	48	0.0307	58	0.0387
39	0.0027	49	0.0133	59	–
40	0.0183	50	0.0195	60	0.0288

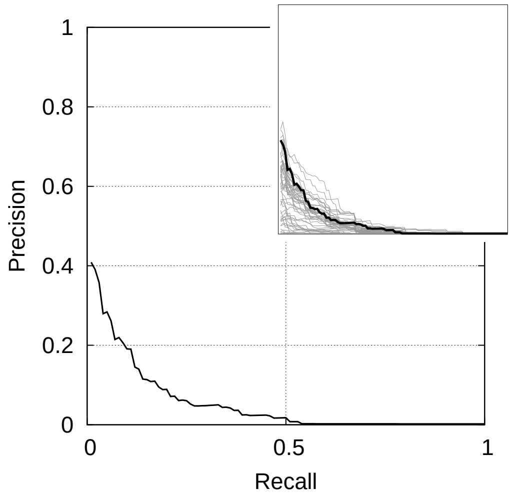
Difference from median in average precision per topic:



University of Amsterdam
UAmsI02NGiSt (CO)

Quantisation: strict

Recall / precision graph:

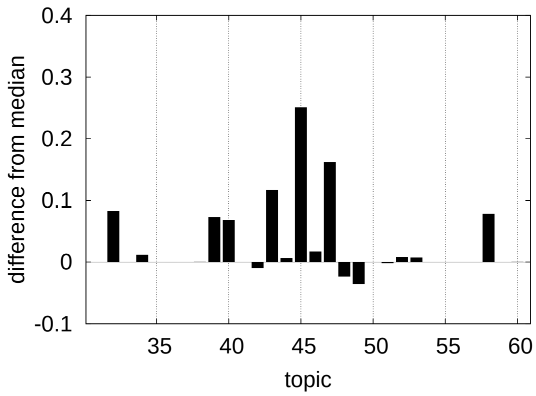


Overall average precision: 0.0532

Average precision per topic:

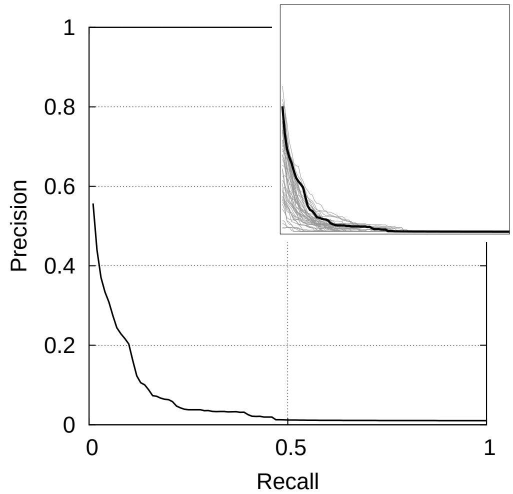
31	0.0002	41	0.0024	51	0.0019
32	0.0948	42	0.0097	52	0.0386
33	0.0001	43	0.1555	53	0.0211
34	0.0222	44	0.0074	54	–
35	–	45	0.2578	55	–
36	0.0027	46	0.0350	56	–
37	0.0032	47	0.1973	57	–
38	0.0042	48	0.0207	58	0.1117
39	0.0813	49	0.0443	59	–
40	0.1035	50	–	60	0.0072

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

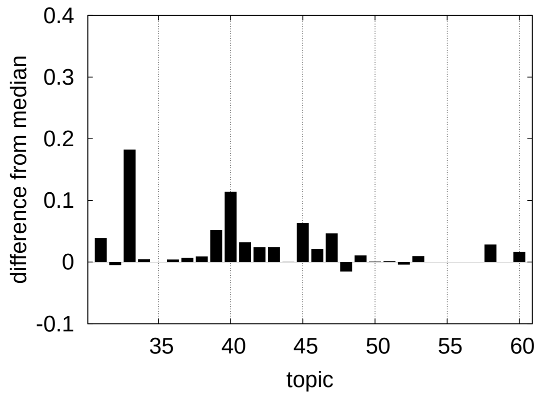


Overall average precision: 0.0554

Average precision per topic:

31	0.0960	41	0.0552	51	0.0225
32	0.0169	42	0.0795	52	0.0181
33	0.2054	43	0.0437	53	0.0227
34	0.0316	44	0.0034	54	–
35	–	45	0.0949	55	–
36	0.0237	46	0.0520	56	–
37	0.0336	47	0.0709	57	–
38	0.0422	48	0.0232	58	0.0725
39	0.0701	49	0.0291	59	–
40	0.1642	50	0.0124	60	0.0446

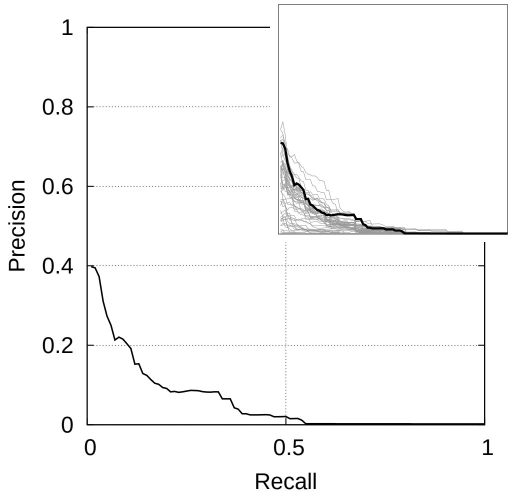
Difference from median in average precision per topic:



University of Amsterdam
UAmsI02NGram (CO)

Quantisation: strict

Recall / precision graph:

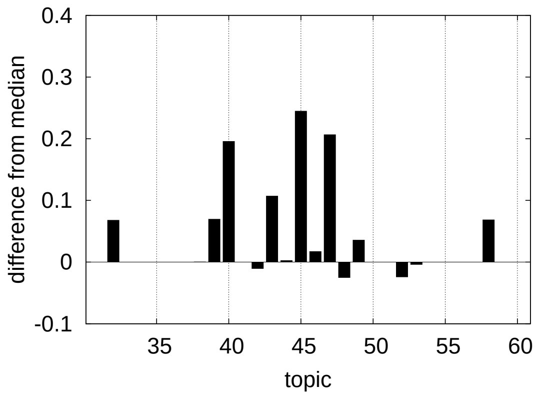


Overall average precision: 0.0592

Average precision per topic:

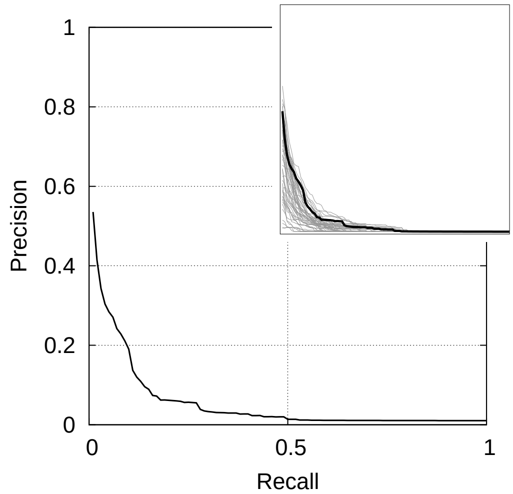
31	0.0002	41	0.0024	51	0.0039
32	0.0799	42	0.0085	52	0.0057
33	0.0001	43	0.1457	53	0.0094
34	0.0103	44	0.0036	54	–
35	–	45	0.2519	55	–
36	0.0024	46	0.0354	56	–
37	0.0032	47	0.2423	57	–
38	0.0043	48	0.0187	58	0.1022
39	0.0785	49	0.1156	59	–
40	0.2311	50	–	60	0.0071

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

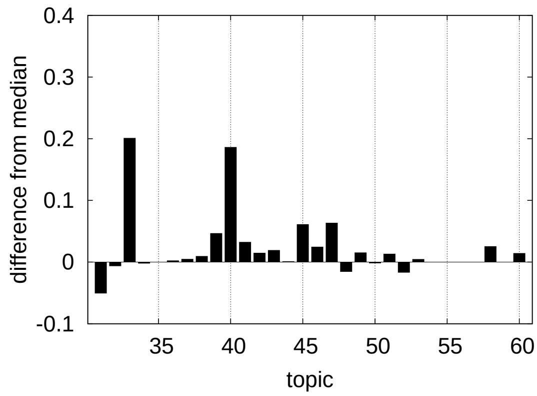


Overall average precision: 0.0546

Average precision per topic:

31	0.0062	41	0.0559	51	0.0347
32	0.0154	42	0.0705	52	0.0052
33	0.2241	43	0.0390	53	0.0180
34	0.0249	44	0.0041	54	–
35	–	45	0.0925	55	–
36	0.0222	46	0.0556	56	–
37	0.0317	47	0.0881	57	–
38	0.0428	48	0.0228	58	0.0696
39	0.0647	49	0.0340	59	–
40	0.2366	50	0.0096	60	0.0423

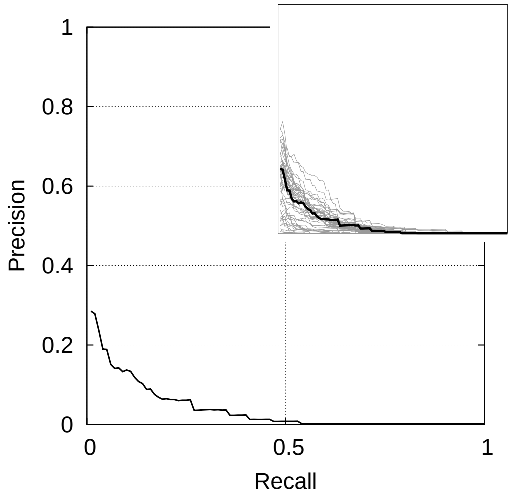
Difference from median in average precision per topic:



University of Amsterdam
UAmsI02Stem (CO)

Quantisation: strict

Recall / precision graph:

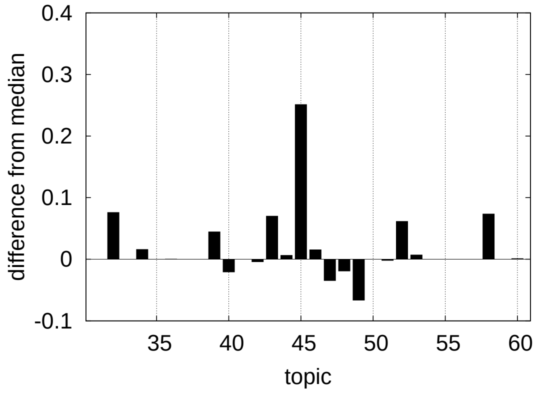


Overall average precision: 0.0385

Average precision per topic:

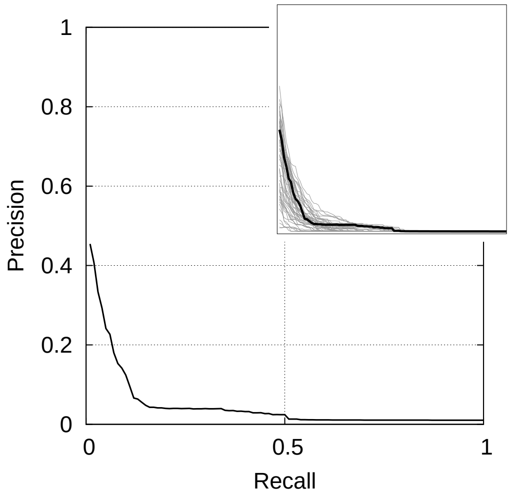
31	0.0002	41	0.0024	51	0.0016
32	0.0880	42	0.0148	52	0.0920
33	0.0001	43	0.1087	53	0.0211
34	0.0266	44	0.0074	54	–
35	–	45	0.2585	55	–
36	0.0032	46	0.0337	56	–
37	0.0032	47	0.0003	57	–
38	0.0035	48	0.0248	58	0.1073
39	0.0535	49	0.0128	59	–
40	0.0138	50	–	60	0.0082

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

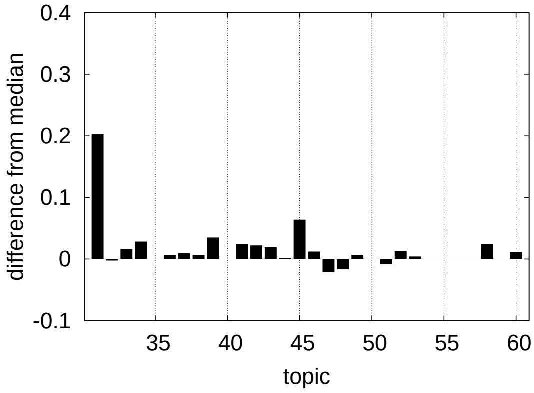


Overall average precision: 0.0466

Average precision per topic:

31	0.2597	41	0.0473	51	0.0131
32	0.0195	42	0.0777	52	0.0348
33	0.0389	43	0.0388	53	0.0174
34	0.0556	44	0.0046	54	–
35	–	45	0.0951	55	–
36	0.0258	46	0.0429	56	–
37	0.0360	47	0.0035	57	–
38	0.0398	48	0.0219	58	0.0688
39	0.0527	49	0.0250	59	–
40	0.0501	50	0.0112	60	0.0390

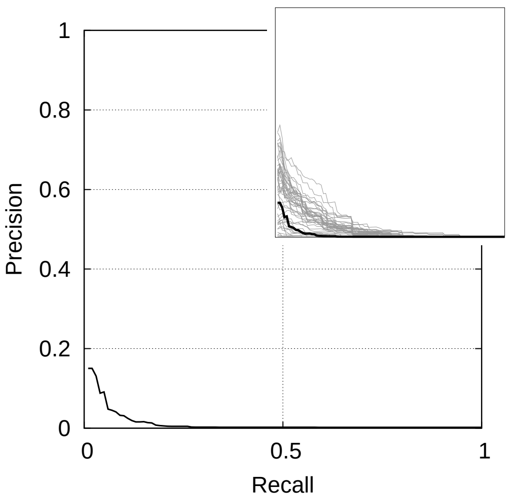
Difference from median in average precision per topic:



University of California, Berkeley
Berkeley01 (CO)

Quantisation: strict

Recall / precision graph:

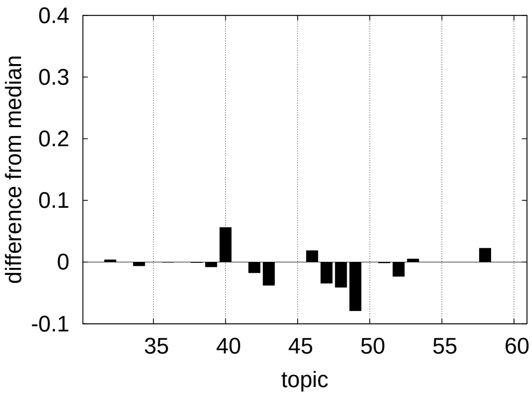


Overall average precision: 0.0114

Average precision per topic:

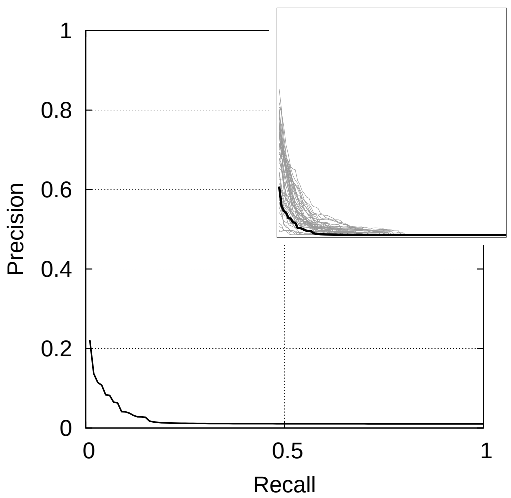
31	0.0002	41	0.0024	51	0.0022
32	0.0158	42	0.0016	52	0.0066
33	0.0001	43	0.0003	53	0.0191
34	0.0038	44	0.0005	54	–
35	–	45	0.0068	55	–
36	0.0017	46	0.0369	56	–
37	0.0032	47	0.0006	57	–
38	0.0025	48	0.0030	58	0.0561
39	0.0004	49	0.0004	59	–
40	0.0914	50	–	60	0.0069

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

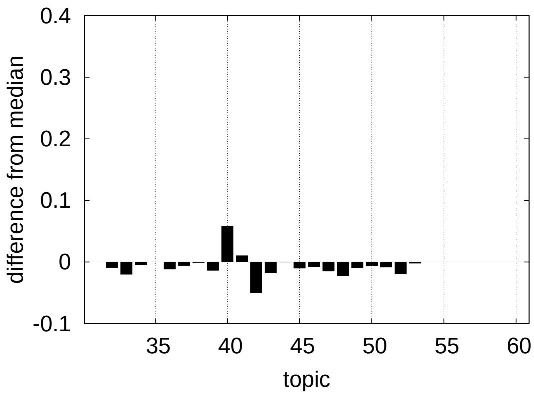


Overall average precision: 0.0204

Average precision per topic:

31	0.0570	41	0.0338	51	0.0126
32	0.0127	42	0.0049	52	0.0023
33	0.0026	43	0.0015	53	0.0109
34	0.0225	44	0.0024	54	–
35	–	45	0.0208	55	–
36	0.0077	46	0.0224	56	–
37	0.0206	47	0.0093	57	–
38	0.0320	48	0.0155	58	0.0440
39	0.0038	49	0.0083	59	–
40	0.1089	50	0.0052	60	0.0282

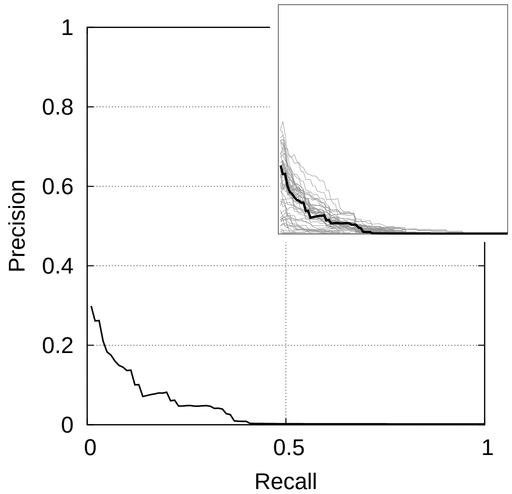
Difference from median in average precision per topic:



University of California, Berkeley
Berkeley02 (CO)

Quantisation: strict	Quantisation: generalised

Recall / precision graph:

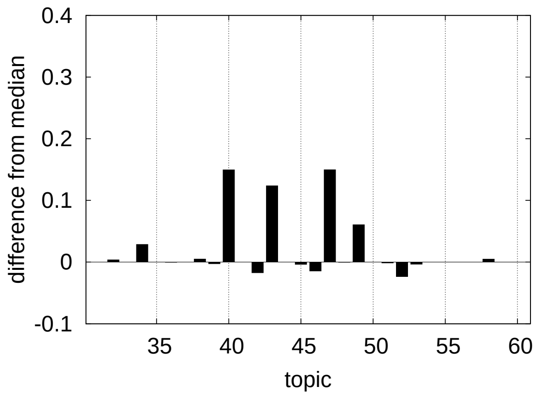


Overall average precision: 0.0376

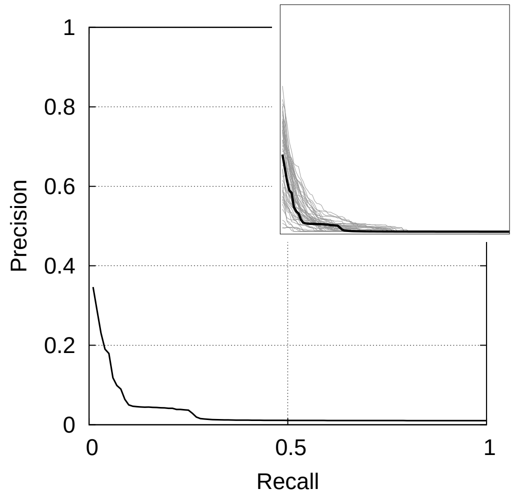
Average precision per topic:

31	0.0002	41	0.0024	51	0.0020
32	0.0158	42	0.0016	52	0.0062
33	0.0001	43	0.1624	53	0.0097
34	0.0393	44	0.0005	54	–
35	–	45	0.0027	55	–
36	0.0017	46	0.0031	56	–
37	0.0032	47	0.1855	57	–
38	0.0089	48	0.0435	58	0.0385
39	0.0055	49	0.1407	59	–
40	0.1849	50	–	60	0.0065

Difference from median in average precision per topic:



Recall / precision graph:

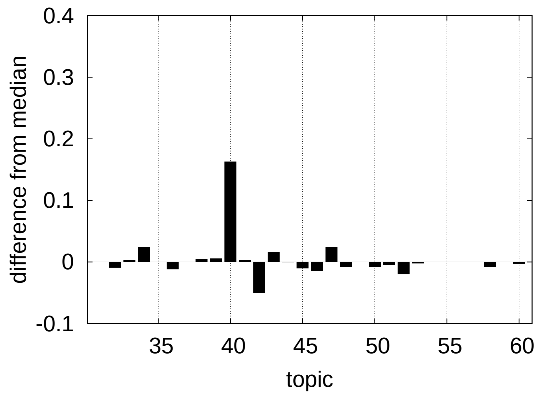


Overall average precision: 0.0314

Average precision per topic:

31	0.0570	41	0.0267	51	0.0167
32	0.0127	42	0.0049	52	0.0023
33	0.0257	43	0.0358	53	0.0109
34	0.0515	44	0.0024	54	–
35	–	45	0.0208	55	–
36	0.0077	46	0.0158	56	–
37	0.0265	47	0.0489	57	–
38	0.0377	48	0.0306	58	0.0357
39	0.0236	49	0.0184	59	–
40	0.2132	50	0.0035	60	0.0251

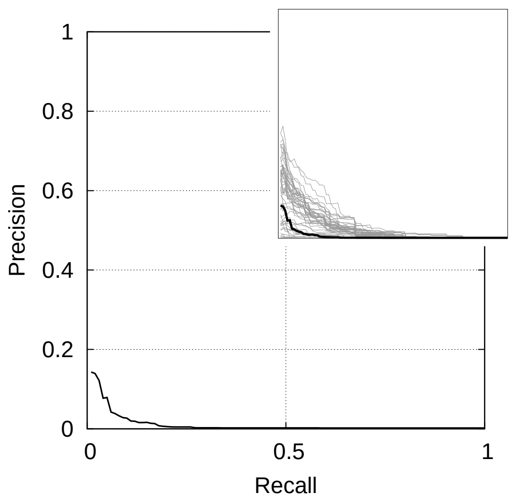
Difference from median in average precision per topic:



University of California, Berkeley
Berkeley03 (CO)

Quantisation: strict

Recall / precision graph:

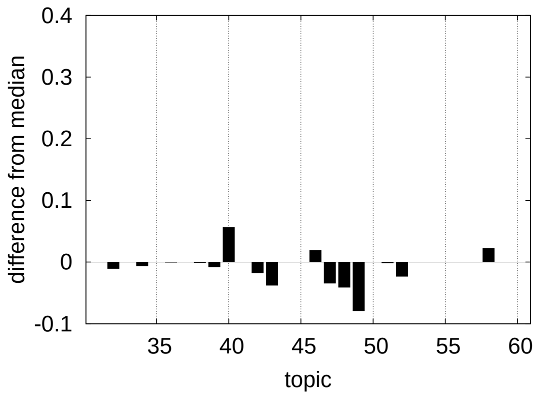


Overall average precision: 0.0106

Average precision per topic:

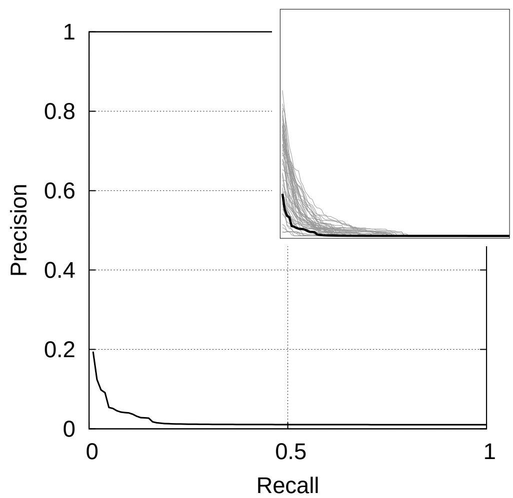
31	0.0002	41	0.0024	51	0.0022
32	0.0009	42	0.0016	52	0.0066
33	0.0001	43	0.0003	53	0.0137
34	0.0038	44	0.0005	54	–
35	–	45	0.0068	55	–
36	0.0017	46	0.0374	56	–
37	0.0032	47	0.0006	57	–
38	0.0025	48	0.0030	58	0.0561
39	0.0004	49	0.0004	59	–
40	0.0914	50	–	60	0.0069

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

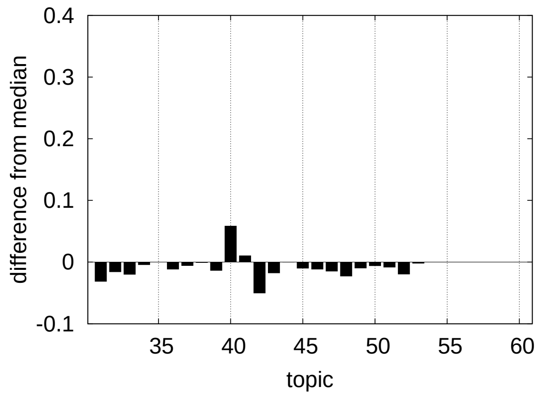


Overall average precision: 0.0187

Average precision per topic:

31	0.0253	41	0.0338	51	0.0126
32	0.0058	42	0.0049	52	0.0023
33	0.0026	43	0.0015	53	0.0109
34	0.0225	44	0.0024	54	–
35	–	45	0.0208	55	–
36	0.0077	46	0.0188	56	–
37	0.0206	47	0.0093	57	–
38	0.0320	48	0.0154	58	0.0440
39	0.0038	49	0.0083	59	–
40	0.1089	50	0.0052	60	0.0282

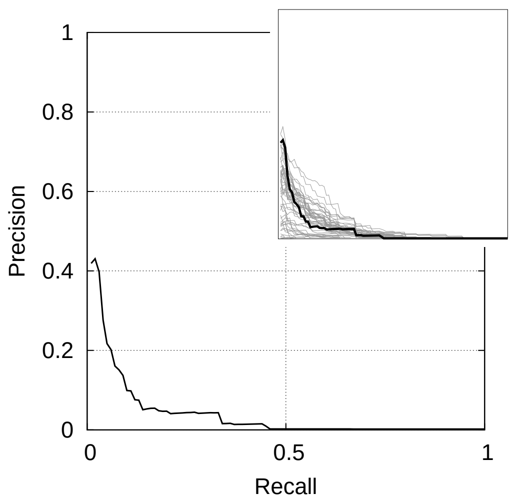
Difference from median in average precision per topic:



University of California, Los Angeles
CorrectedFormat (CO)

Quantisation: strict

Recall / precision graph:

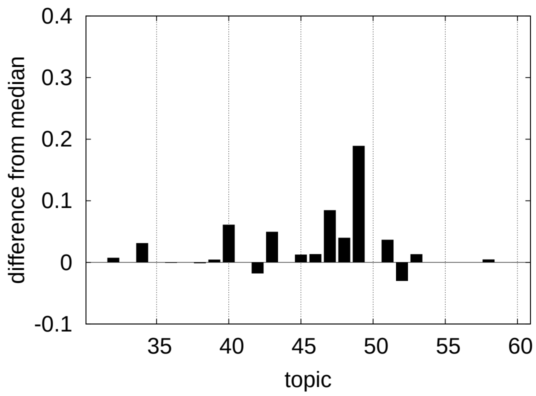


Overall average precision: 0.0394

Average precision per topic:

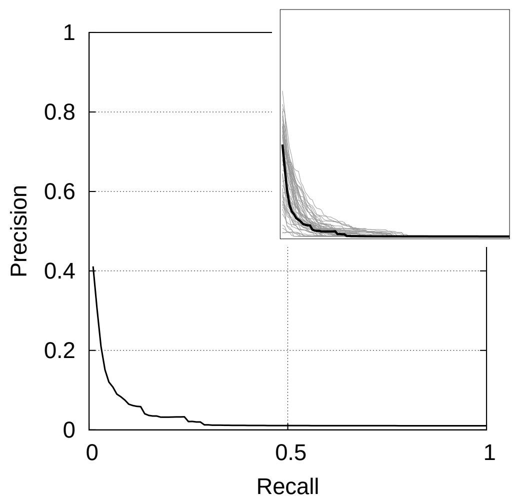
31	0.0002	41	0.0024	51	0.0406
32	0.0192	42	0.0014	52	0.0001
33	0.0001	43	0.0880	53	0.0269
34	0.0416	44	0.0007	54	–
35	–	45	0.0195	55	–
36	0.0017	46	0.0313	56	–
37	0.0032	47	0.1202	57	–
38	0.0023	48	0.0842	58	0.0382
39	0.0131	49	0.2689	59	–
40	0.0962	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

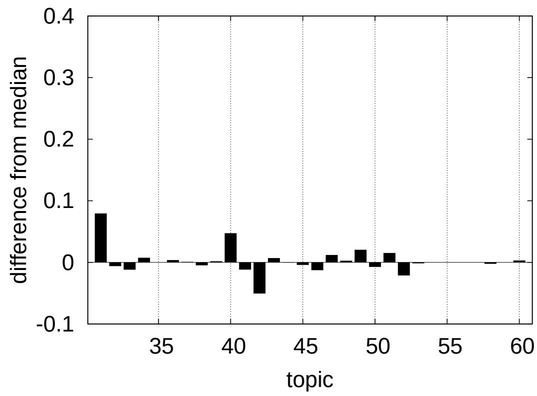


Overall average precision: 0.0303

Average precision per topic:

31	0.1364	41	0.0115	51	0.0365
32	0.0159	42	0.0050	52	0.0010
33	0.0110	43	0.0265	53	0.0117
34	0.0347	44	0.0025	54	–
35	–	45	0.0271	55	–
36	0.0234	46	0.0180	56	–
37	0.0276	47	0.0365	57	–
38	0.0283	48	0.0413	58	0.0417
39	0.0196	49	0.0389	59	–
40	0.0974	50	0.0042	60	0.0310

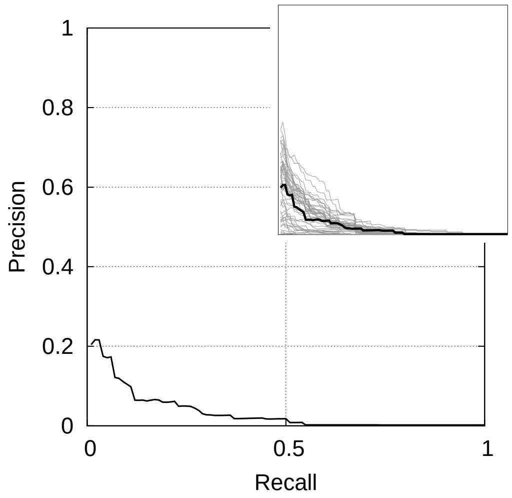
Difference from median in average precision per topic:



University of Melbourne
um_mgx21_short (CO)

Quantisation: strict

Recall / precision graph:

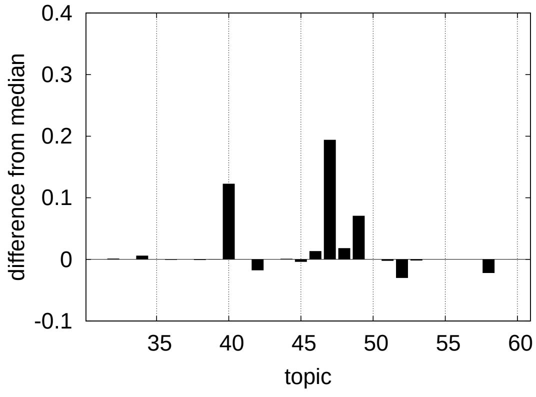


Overall average precision: 0.0329

Average precision per topic:

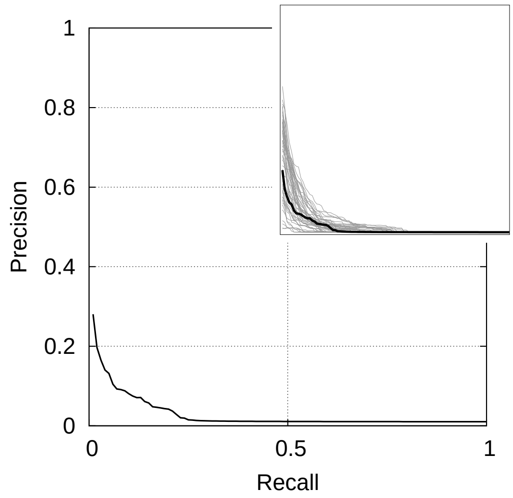
31	0.0002	41	0.0026	51	0.0016
32	0.0131	42	0.0017	52	0.0001
33	0.0001	43	0.0383	53	0.0118
34	0.0164	44	0.0018	54	–
35	–	45	0.0028	55	–
36	0.0017	46	0.0313	56	–
37	0.0032	47	0.2294	57	–
38	0.0026	48	0.0626	58	0.0112
39	0.0091	49	0.1504	59	–
40	0.1578	50	–	60	0.0069

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

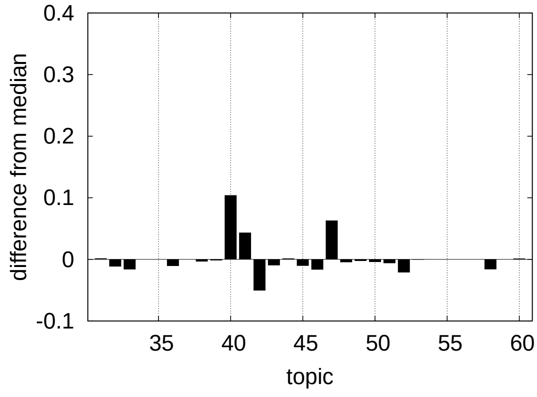


Overall average precision: 0.0287

Average precision per topic:

31	0.0587	41	0.0668	51	0.0151
32	0.0103	42	0.0049	52	0.0010
33	0.0066	43	0.0099	53	0.0127
34	0.0272	44	0.0044	54	–
35	–	45	0.0207	55	–
36	0.0089	46	0.0140	56	–
37	0.0266	47	0.0876	57	–
38	0.0296	48	0.0339	58	0.0278
39	0.0161	49	0.0158	59	–
40	0.1542	50	0.0073	60	0.0293

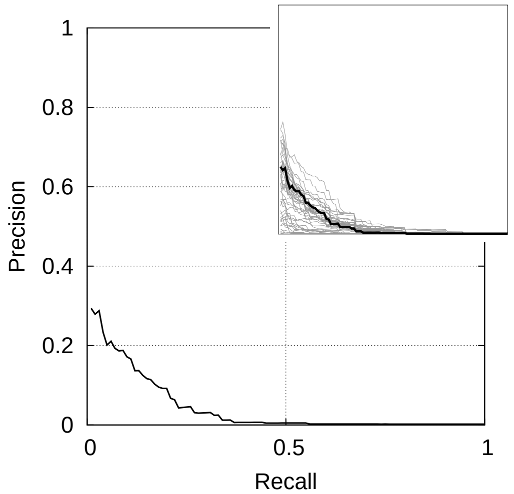
Difference from median in average precision per topic:



University of Melbourne
um_mgx26_long (CO)

Quantisation: strict

Recall / precision graph:

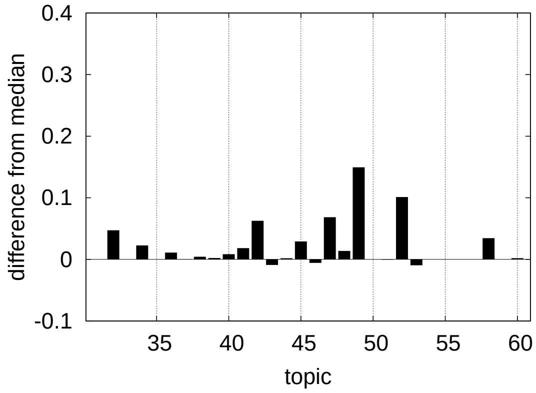


Overall average precision: 0.0418

Average precision per topic:

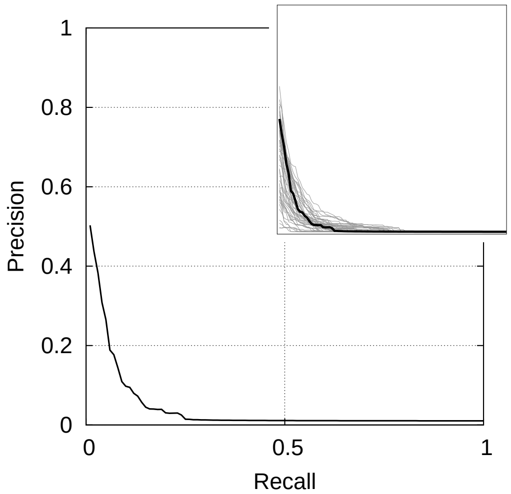
31	0.0002	41	0.0208	51	0.0033
32	0.0589	42	0.0820	52	0.1314
33	0.0001	43	0.0292	53	0.0041
34	0.0329	44	0.0025	54	–
35	–	45	0.0360	55	–
36	0.0135	46	0.0123	56	–
37	0.0036	47	0.1039	57	–
38	0.0080	48	0.0580	58	0.0678
39	0.0108	49	0.2291	59	–
40	0.0433	50	–	60	0.0086

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

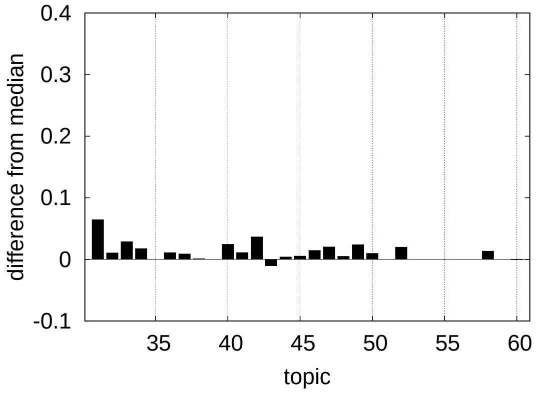


Overall average precision: 0.0411

Average precision per topic:

31	0.1218	41	0.0347	51	0.0212
32	0.0329	42	0.0924	52	0.0424
33	0.0520	43	0.0088	53	0.0132
34	0.0450	44	0.0072	54	–
35	–	45	0.0369	55	–
36	0.0309	46	0.0456	56	–
37	0.0358	47	0.0450	57	–
38	0.0345	48	0.0439	58	0.0578
39	0.0179	49	0.0426	59	–
40	0.0749	50	0.0216	60	0.0269

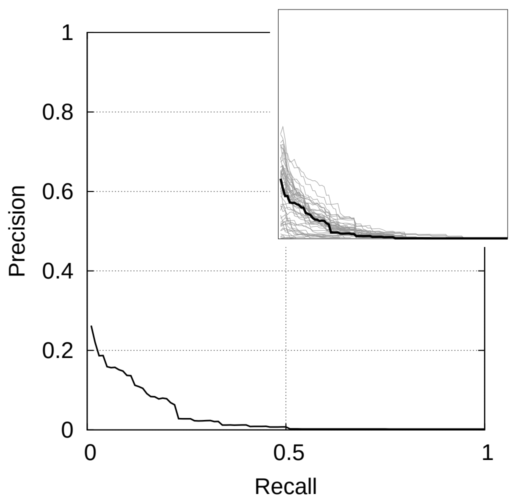
Difference from median in average precision per topic:



University of Melbourne
um_mgx2_long (CO)

Quantisation: strict

Recall / precision graph:

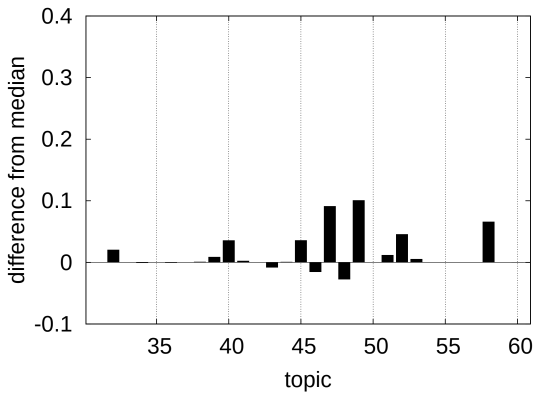


Overall average precision: 0.0340

Average precision per topic:

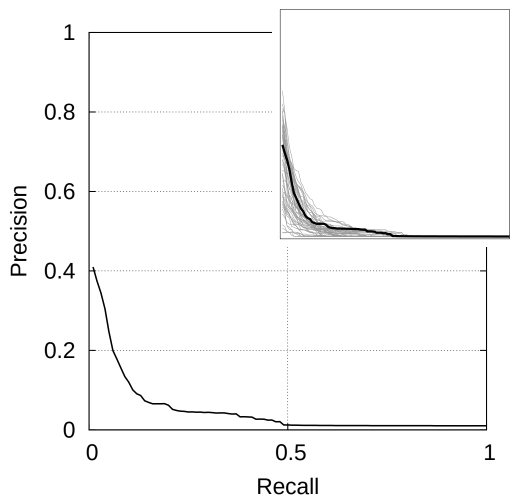
31	0.0002	41	0.0050	51	0.0159
32	0.0323	42	0.0193	52	0.0760
33	0.0001	43	0.0299	53	0.0192
34	0.0092	44	0.0017	54	–
35	–	45	0.0428	55	–
36	0.0017	46	0.0023	56	–
37	0.0032	47	0.1267	57	–
38	0.0046	48	0.0166	58	0.0995
39	0.0175	49	0.1806	59	–
40	0.0708	50	–	60	0.0069

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

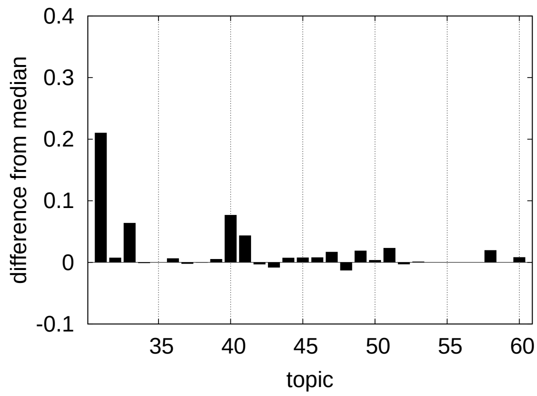


Overall average precision: 0.0483

Average precision per topic:

31	0.2675	41	0.0670	51	0.0446
32	0.0296	42	0.0523	52	0.0191
33	0.0870	43	0.0111	53	0.0145
34	0.0261	44	0.0104	54	–
35	–	45	0.0391	55	–
36	0.0262	46	0.0388	56	–
37	0.0244	47	0.0415	57	–
38	0.0327	48	0.0254	58	0.0638
39	0.0232	49	0.0374	59	–
40	0.1271	50	0.0153	60	0.0363

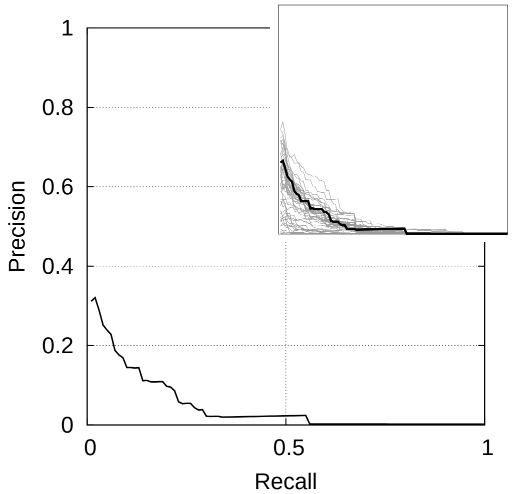
Difference from median in average precision per topic:



University of Michigan
allow-duplicate (CO)

Quantisation: strict

Recall / precision graph:

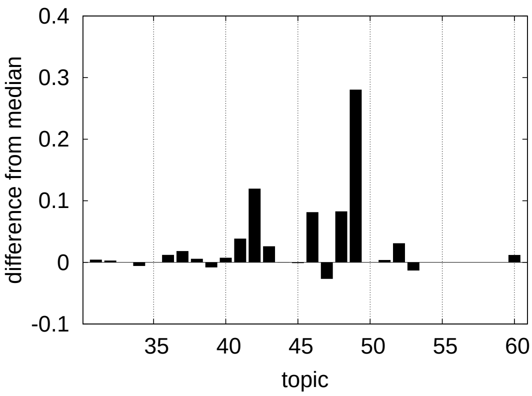


Overall average precision: 0.0470

Average precision per topic:

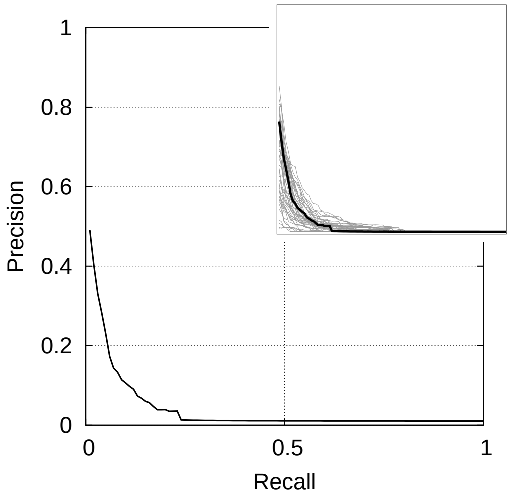
31	0.0046	41	0.0410	51	0.0077
32	0.0147	42	0.1391	52	0.0612
33	0.0001	43	0.0644	53	0.0004
34	0.0045	44	0.0005	54	–
35	–	45	0.0057	55	–
36	0.0146	46	0.0995	56	–
37	0.0217	47	0.0086	57	–
38	0.0094	48	0.1271	58	0.0333
39	0.0004	49	0.3601	59	–
40	0.0425	50	–	60	0.0188

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

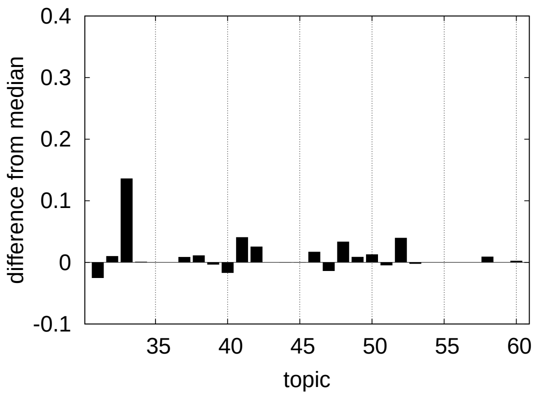


Overall average precision: 0.0397

Average precision per topic:

31	0.0317	41	0.0641	51	0.0165
32	0.0321	42	0.0812	52	0.0621
33	0.1592	43	0.0195	53	0.0108
34	0.0282	44	0.0027	54	–
35	–	45	0.0309	55	–
36	0.0196	46	0.0479	56	–
37	0.0354	47	0.0105	57	–
38	0.0446	48	0.0722	58	0.0533
39	0.0142	49	0.0274	59	–
40	0.0331	50	0.0246	60	0.0305

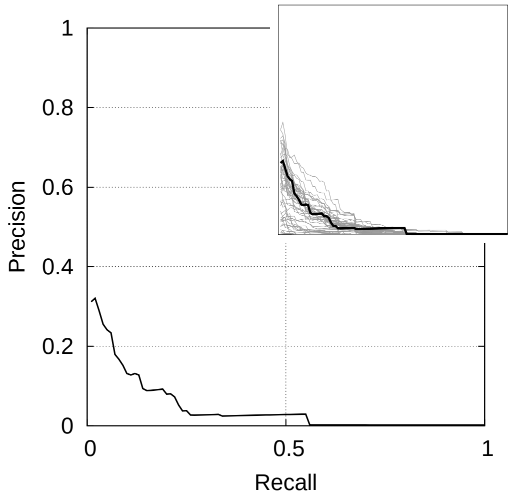
Difference from median in average precision per topic:



University of Michigan
no-duplicate (CO)

Quantisation: strict	**Quantisation: generalised**

Recall / precision graph:

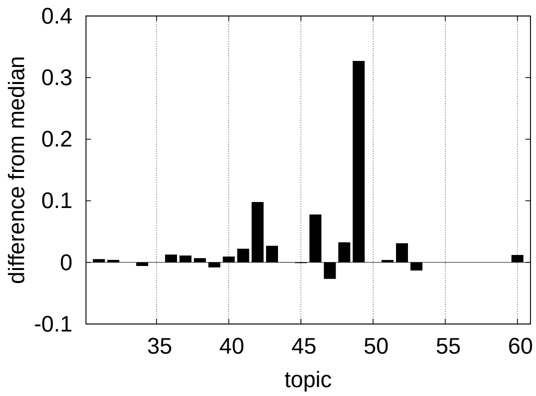


Overall average precision: 0.0449

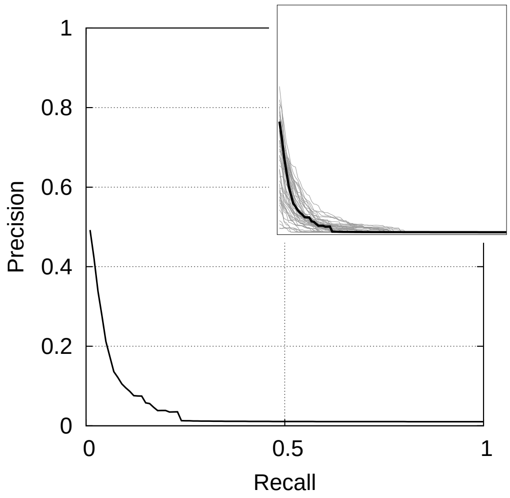
Average precision per topic:

31	0.0055	41	0.0245	51	0.0077
32	0.0157	42	0.1173	52	0.0612
33	0.0001	43	0.0652	53	0.0004
34	0.0045	44	0.0005	54	–
35	–	45	0.0057	55	–
36	0.0152	46	0.0958	56	–
37	0.0143	47	0.0086	57	–
38	0.0104	48	0.0769	58	0.0333
39	0.0004	49	0.4066	59	–
40	0.0444	50	–	60	0.0188

Difference from median in average precision per topic:



Recall / precision graph:

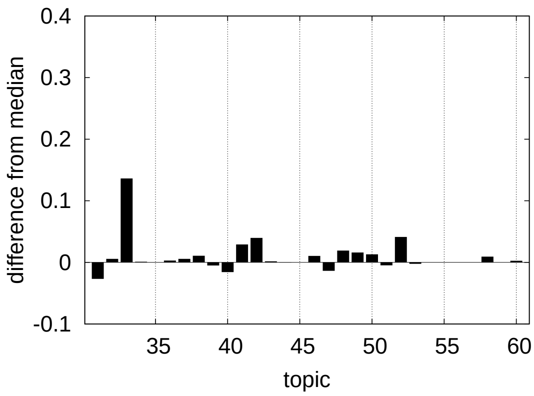


Overall average precision: 0.0390

Average precision per topic:

31	0.0302	41	0.0524	51	0.0165
32	0.0276	42	0.0952	52	0.0635
33	0.1592	43	0.0212	53	0.0109
34	0.0282	44	0.0027	54	–
35	–	45	0.0312	55	–
36	0.0225	46	0.0412	56	–
37	0.0324	47	0.0107	57	–
38	0.0438	48	0.0577	58	0.0533
39	0.0128	49	0.0345	59	–
40	0.0344	50	0.0246	60	0.0305

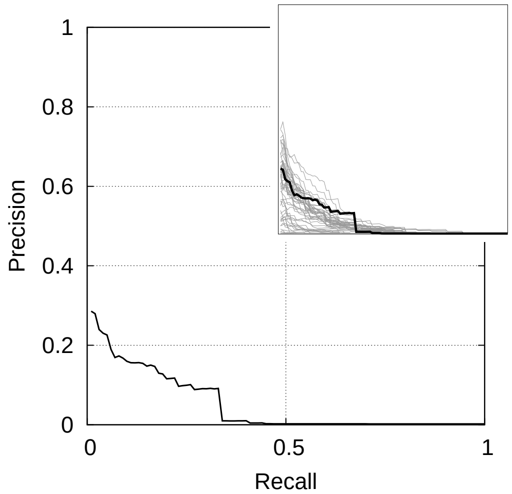
Difference from median in average precision per topic:



University of Minnesota Duluth
01 (CO)

Quantisation: strict

Recall / precision graph:

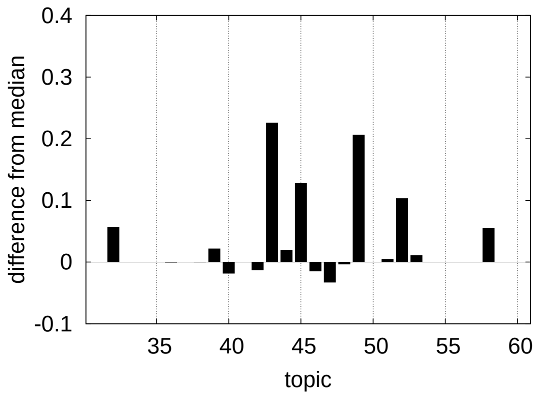


Overall average precision: 0.0503

Average precision per topic:

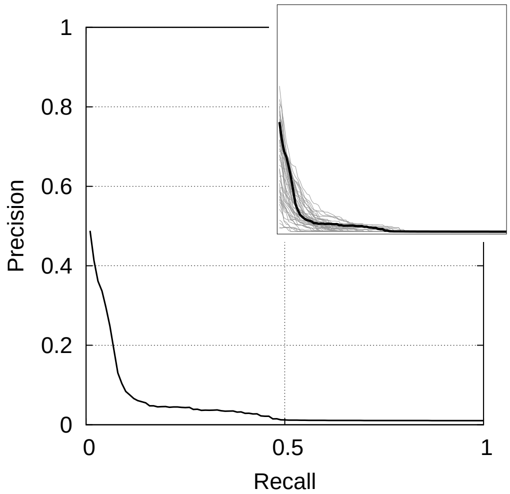
31	0.0002	41	0.0024	51	0.0089
32	0.0688	42	0.0063	52	0.1336
33	0.0001	43	0.2643	53	0.0248
34	0.0105	44	0.0206	54	–
35	–	45	0.1348	55	–
36	0.0017	46	0.0030	56	–
37	0.0032	47	0.0022	57	–
38	0.0031	48	0.0406	58	0.0889
39	0.0305	49	0.2862	59	–
40	0.0164	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

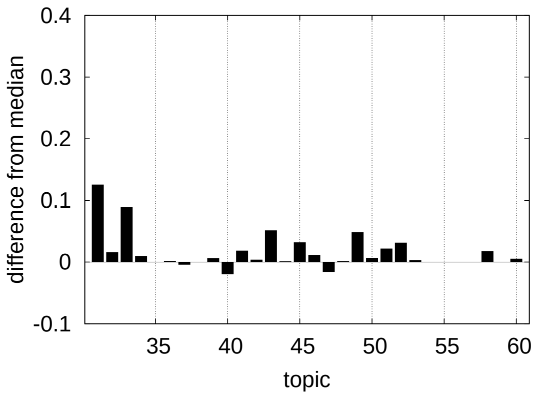


Overall average precision: 0.0469

Average precision per topic:

31	0.1827	41	0.0417	51	0.0432
32	0.0380	42	0.0595	52	0.0537
33	0.1122	43	0.0710	53	0.0164
34	0.0371	44	0.0041	54	–
35	–	45	0.0631	55	–
36	0.0216	46	0.0423	56	–
37	0.0224	47	0.0085	57	–
38	0.0333	48	0.0403	58	0.0620
39	0.0243	49	0.0670	59	–
40	0.0304	50	0.0184	60	0.0333

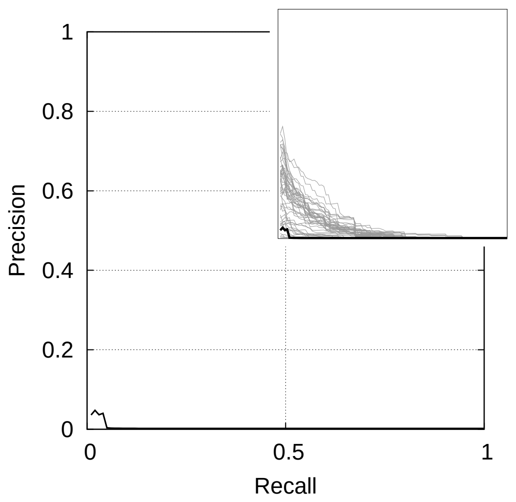
Difference from median in average precision per topic:



University of North Carolina at Chapel Hill
irt (CO)

Quantisation: strict

Recall / precision graph:

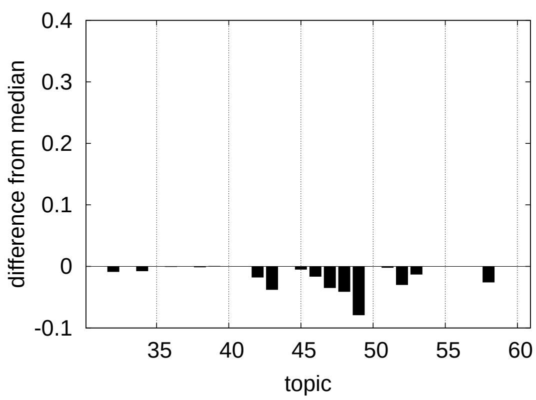


Overall average precision: 0.0037

Average precision per topic:

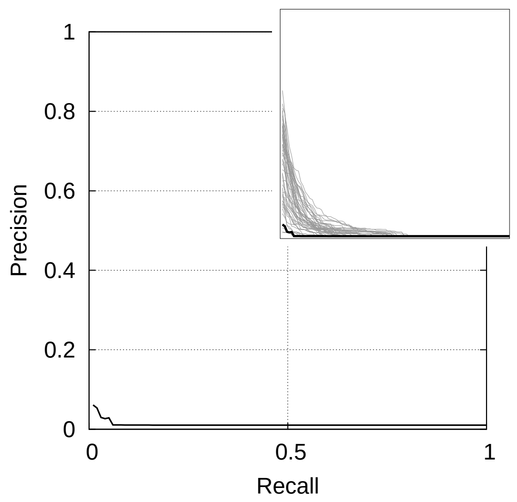
31	0.0002	41	0.0025	51	0.0016
32	0.0028	42	0.0014	52	0.0001
33	0.0001	43	0.0003	53	0.0004
34	0.0025	44	0.0005	54	–
35	–	45	0.0018	55	–
36	0.0017	46	0.0013	56	–
37	0.0032	47	0.0003	57	–
38	0.0023	48	0.0030	58	0.0073
39	0.0092	49	0.0004	59	–
40	0.0350	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

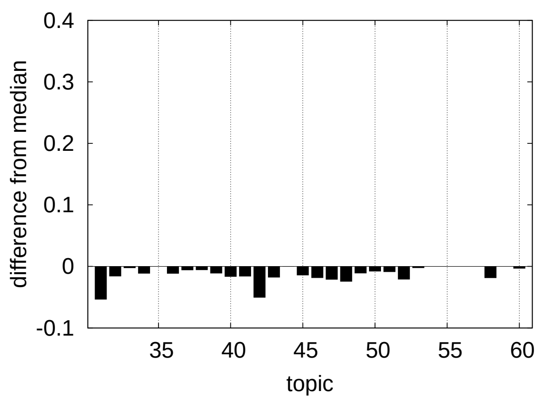


Overall average precision: 0.0119

Average precision per topic:

31	0.0031	41	0.0069	51	0.0122
32	0.0058	42	0.0049	52	0.0010
33	0.0201	43	0.0015	53	0.0106
34	0.0156	44	0.0024	54	–
35	–	45	0.0168	55	–
36	0.0078	46	0.0119	56	–
37	0.0204	47	0.0029	57	–
38	0.0271	48	0.0139	58	0.0252
39	0.0064	49	0.0071	59	–
40	0.0333	50	0.0034	60	0.0244

Difference from median in average precision per topic:

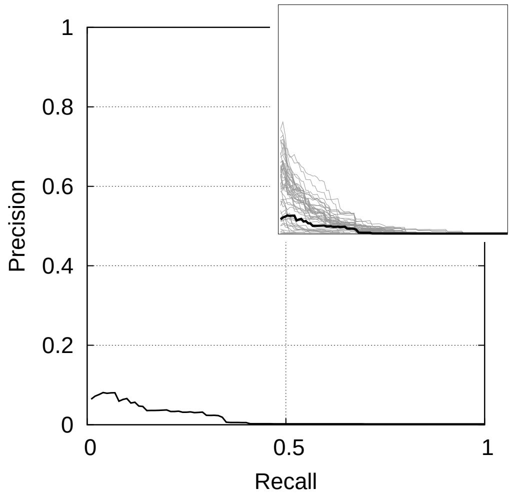


University of Twente
utwente1h (CO)

Quantisation: strict	Quantisation: generalised

Quantisation: strict

Recall / precision graph:

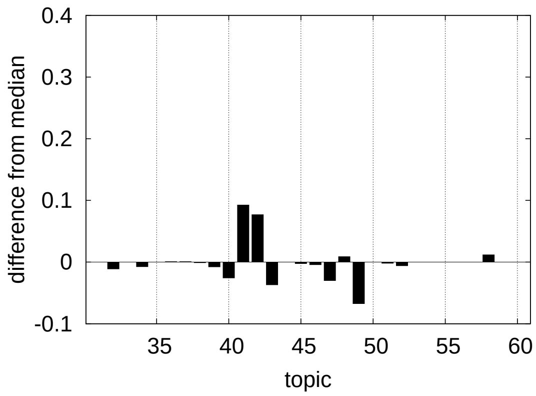


Overall average precision: 0.0172

Average precision per topic:

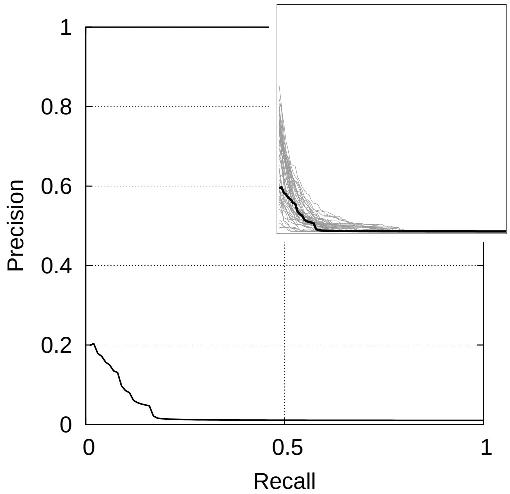
31	0.0002	41	0.0954	51	0.0016
32	0.0002	42	0.0966	52	0.0238
33	0.0001	43	0.0010	53	0.0136
34	0.0024	44	0.0005	54	–
35	–	45	0.0041	55	–
36	0.0035	46	0.0133	56	–
37	0.0043	47	0.0050	57	–
38	0.0023	48	0.0534	58	0.0455
39	0.0004	49	0.0118	59	–
40	0.0088	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

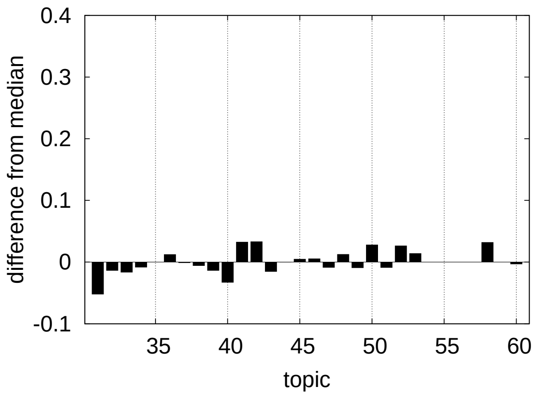


Overall average precision: 0.0279

Average precision per topic:

31	0.0046	41	0.0559	51	0.0121
32	0.0080	42	0.0889	52	0.0489
33	0.0061	43	0.0039	53	0.0274
34	0.0185	44	0.0024	54	–
35	–	45	0.0362	55	–
36	0.0321	46	0.0363	56	–
37	0.0252	47	0.0153	57	–
38	0.0271	48	0.0514	58	0.0763
39	0.0039	49	0.0089	59	–
40	0.0168	50	0.0398	60	0.0244

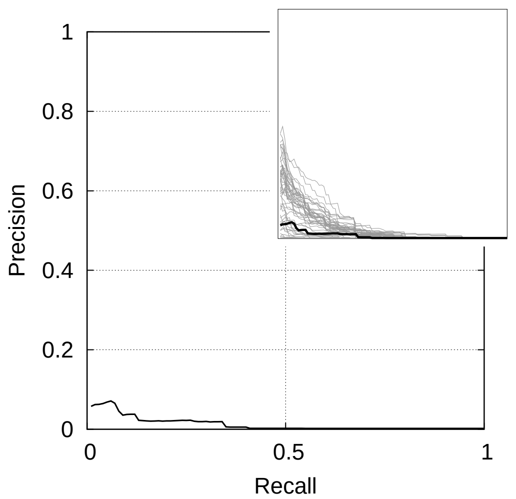
Difference from median in average precision per topic:



University of Twente
utwente1n (CO)

Quantisation: strict

Recall / precision graph:

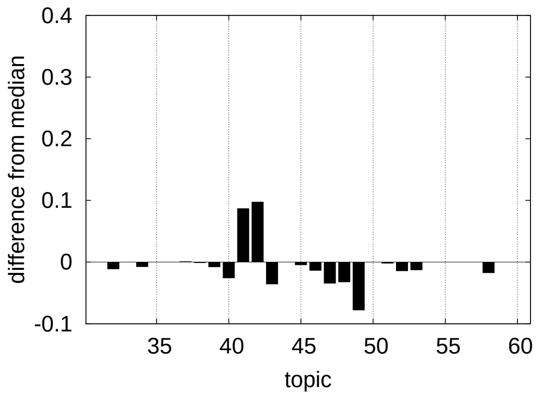


Overall average precision: 0.0126

Average precision per topic:

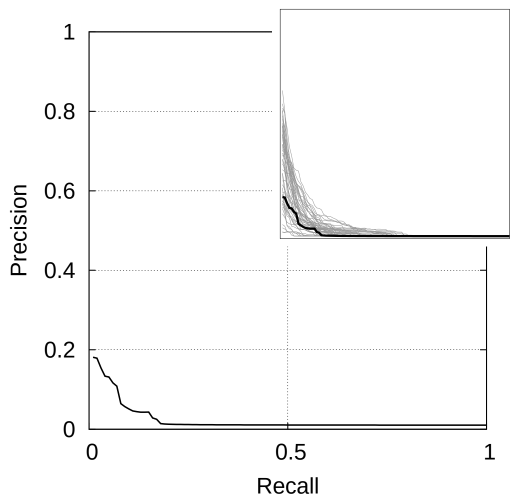
31	0.0002	41	0.0895	51	0.0016
32	0.0002	42	0.1171	52	0.0156
33	0.0001	43	0.0023	53	0.0006
34	0.0024	44	0.0005	54	–
35	–	45	0.0019	55	–
36	0.0028	46	0.0039	56	–
37	0.0043	47	0.0006	57	–
38	0.0023	48	0.0116	58	0.0156
39	0.0004	49	0.0016	59	–
40	0.0088	50	–	60	0.0066

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:

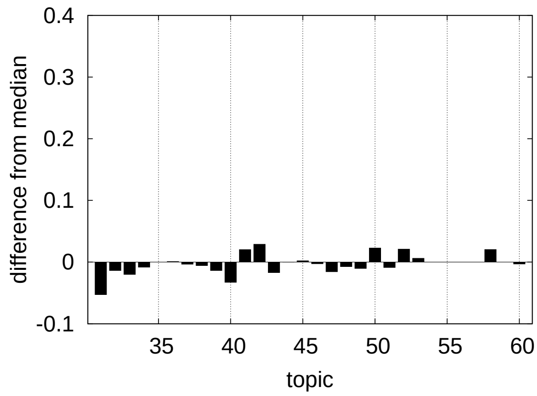


Overall average precision: 0.0235

Average precision per topic:

31	0.0038	41	0.0438	51	0.0121
32	0.0078	42	0.0848	52	0.0435
33	0.0025	43	0.0021	53	0.0197
34	0.0186	44	0.0024	54	–
35	–	45	0.0337	55	–
36	0.0207	46	0.0276	56	–
37	0.0227	47	0.0084	57	–
38	0.0271	48	0.0308	58	0.0648
39	0.0037	49	0.0077	59	–
40	0.0168	50	0.0348	60	0.0244

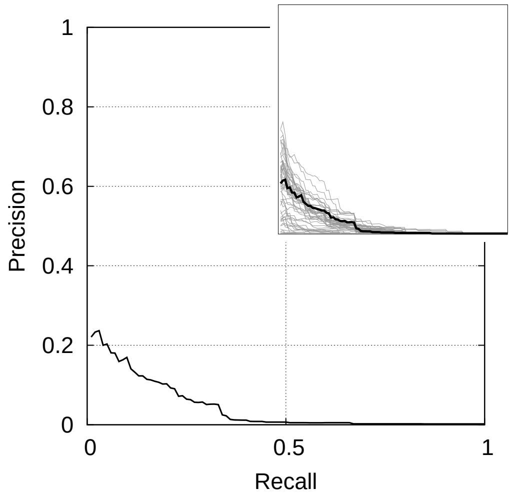
Difference from median in average precision per topic:



University of Twente
utwente1pr (CO)

Quantisation: strict

Recall / precision graph:

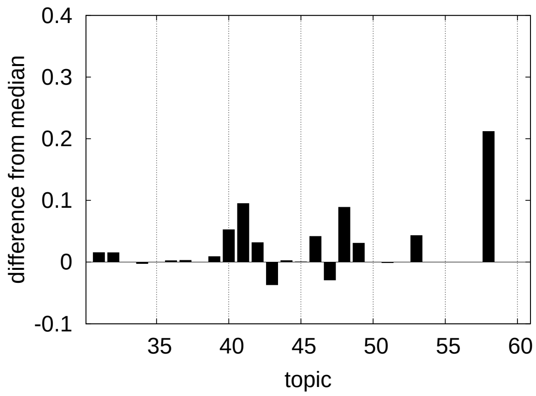


Overall average precision: 0.0429

Average precision per topic:

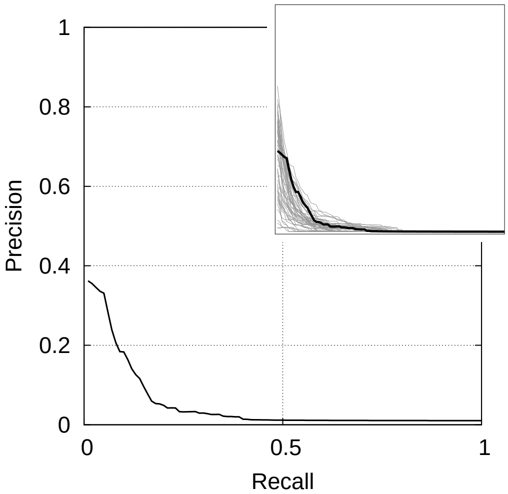
31	0.0160	41	0.0978	51	0.0025
32	0.0273	42	0.0513	52	0.0302
33	0.0001	43	0.0010	53	0.0571
34	0.0074	44	0.0036	54	–
35	–	45	0.0077	55	–
36	0.0052	46	0.0600	56	–
37	0.0065	47	0.0058	57	–
38	0.0040	48	0.1336	58	0.2456
39	0.0179	49	0.1107	59	–
40	0.0879	50	–	60	0.0065

Difference from median in average precision per topic:



Quantisation: generalised

Recall / precision graph:



Overall average precision: 0.0499

Average precision per topic:

31	0.0994	41	0.0754	51	0.0328
32	0.0163	42	0.0794	52	0.0586
33	0.1571	43	0.0040	53	0.0371
34	0.0176	44	0.0027	54	–
35	–	45	0.0400	55	–
36	0.0429	46	0.0593	56	–
37	0.0598	47	0.0160	57	–
38	0.0307	48	0.0848	58	0.1099
39	0.0181	49	0.0181	59	–
40	0.0813	50	0.0329	60	0.0242

Difference from median in average precision per topic: