

UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of a master's thesis by

Archana Bellamkonda

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Carolyn J. Crouch

Name of Faculty Advisor

Signature of Faculty Advisor

Date

GRADUATE SCHOOL

Automation of Content and Structure (or CAS) Query Processing

A thesis
submitted to the faculty of the graduate school
of the University of Minnesota
by

Archana Bellamkonda

in partial fulfillment of the requirements
for the degree of
Master of Science

September 2004

Department of Computer Science
University of Minnesota Duluth
Duluth, Minnesota 55812
U.S.A.

© Archana Bellamkonda 2004

Abstract

Information retrieval in general deals with the retrieval of relevant documents. The evolution of large, web-based collections focuses attention on structured data. XML (Extended Mark-up Language XML) is a highly structured language used to store data. It allows a document to be classified into logical levels such as abstract, body, section, paragraph, etc. Structure also allows us to form more specific queries, e.g., content-and-structure (or CAS) queries which contain constraints like year of publication, author name, etc. With structured retrieval, we can also retrieve the required *element*, i.e., a part of the document instead of the entire document itself. Thus, using XML structure, we can specify search terms, search fields, and elements to be returned in a query.

The focus of this thesis is the automatic translation of queries from XML to extended vector format based on the search terms, search fields and elements to be returned as identified in the XML query. We use the INEX test collection in XML format and our current, flexible retrieval system, based on Smart and the Extended Vector Space Model, for the XML retrieval task. Research focuses on the automatic processing of structured queries and the post-processing of results.

Acknowledgements

I sincerely thank my advisor Dr. Carolyn Crouch for her invaluable support throughout my graduate career, for teaching me the basics of research and writing, for her guidance in the field of Information Retrieval, and for her financial help.

I also owe thanks to Dr. Douglas Dunham and Dr. Joe Gallian for all their help. I specially acknowledge Dr. Joe Gallian for being available on a short notice when I needed his support.

I would like to specially thank Aniruddha Mahajan for being an excellent team mate to work with. I would also like to thank Nagendra Doddapaneni, Poorva Potnis, and Sudip Khanna for their help.

I would also like to thank the Department of Computer Science at University of Minnesota Duluth. I would specially acknowledge Lori Lucia, Linda Meek and Jim Luttinen for their assistance.

I would like to thank my family and all my friends for their support throughout my graduate career.

Table of Contents

1	Introduction	1
1.1	The Basic Retrieval Engine	1
1.2	The Extended Vector Space Model	2
1.3	Structured Retrieval under INEX	4
1.3.1	Document Collection	5
1.3.2	Topics	6
1.3.3	Assessments	7
1.3.4	Evaluation	8
2	CAS Query Processing	9
2.1	Description of XML Query	9
2.1.1	Topic	9
2.1.2	Title	10
2.1.3	Description	10
2.1.4	Narrative	10
2.1.5	Keywords	10
2.2	Processing XML Queries	11
2.2.1	Identifying the Element to be Returned	15
2.2.2	Splitting a Query into Parts	17
2.2.3	Running the Queries	20
2.2.4	Running the Flexible Program	21
2.2.5	Combining the Results for INEX Submission	23
3	Experiments and Results	30

4	Conclusions and Future Work	33
5	Bibliography	34
6	Appendix 1	36
6.1	Initial INEX 2003 Query	36
6.2	Result of a query fed to the Interpreter.....	37
6.3	Smart Results of Queries	37
6.4	Results of Flexible Program	39
6.5	Post-processing on Flexible Results - Returning the Required Element.....	40
7	Appendix 2	41
7.1	Comparison of Results between Flexible and Non-Flexible Systems.....	41
8	Appendix 3	55
8.1	Comparison of our Results with other INEX participants (2003).....	55

List of Tables

Table 1: c-types and their description	12
Table 2: Comparison of Flexible and Non-Flexible Systems for INEX 2003 Queries..	55
Table 3: Top 10 Results for INEX 2003 CAS Queries with old system.....	56
Table 4: Top 10 Results for INEX 2003 CAS Queries with hybrid system.....	57

List of Figures

Figure 1: An example of Extended Vector Space model [7].....	3
Figure 2: Structure of INEX article [7]	6
Figure 3: DTD of a XML query in INEX collection.....	9
Figure 4: An example of a simple CAS query	13
Figure 5: Query 133 from the INEX 2004 collection.....	14
Figure 6: Query 127 from the INEX 2004 collections.....	16
Figure 7: Query 128 from the INEX 2004 collection.....	17
Figure 8: Query number 64 from INEX 2003 collection	18
Figure 9: Query number 67 from INEX 2003 collection	20
Figure 10: A sample result file of Smart	21
Figure 11: Propagation of relevance values to parent nodes	22
Figure 12: Ranked Intersection of two lists [7]	24
Figure 13: Illustration of Parent Intersection.....	26
Figure 14: Illustration of Difference.....	29
Figure 15: Results of INEX 2003 CAS queries using non-flexible retrieval system.....	31
Figure 16: Results of INEX 2003 CAS queries using flexible retrieval system.....	32
Figure 17: Results of INEX 2003 CAS queries using a hybrid results.....	32
Figure 18(a): Query 61 from the INEX 2003 collection	36
Figure 18(b): First part of the result of interpreter on query 61	37
Figure 18(c): Second part of the result of interpreter on query 61	37
Figure 19(a): Smart results of Query 61 – Part 1.....	38

Figure 19(b): Smart results of Query 61 – Part 2.....	38
Figure 20: Results that can be input to Flexible Program	39
Figure 21: Sample result of Flexible Program	39
Figure 22(a) Result file before processing for element to be returned.....	40
Figure 22(b): Result file with sections only	41
Figure 23: Query 62 Non-FlexibleSystem	42
Figure 24: Query 62 FlexibleSystem.....	42
Figure 25: Query 63 Non-FlexibleSystem	42
Figure 26: Query 63 FlexibleSystem.....	42
Figure 27: Query 64 Non-FlexibleSystem	43
Figure 28: Query 64 FlexibleSystem.....	43
Figure 29: Query 65 Non-FlexibleSystem	43
Figure 30: Query 65 FlexibleSystem.....	43
Figure 31: Query 66 Non-FlexibleSystem	44
Figure 32: Query 66 FlexibleSystem.....	44
Figure 33: Query 68 Non-FlexibleSystem	44
Figure 34: Query 66 FlexibleSystem.....	44
Figure 35: Query 70 Non-FlexibleSystem	45
Figure 36: Query 70 FlexibleSystem.....	45
Figure 37: Query 71 Non-FlexibleSystem	45
Figure 38: Query 70 FlexibleSystem.....	45
Figure 39: Query 72 Non-FlexibleSystem	46
Figure 40: Query 72 FlexibleSystem.....	46

Figure 41: Query 74 Non-FlexibleSystem	46
Figure 42: Query 74 FlexibleSystem.....	46
Figure 43: Query 75 Non-FlexibleSystem	47
Figure 44: Query 75 FlexibleSystem.....	47
Figure 45: Query 77 Non-FlexibleSystem	47
Figure 46: Query 77 FlexibleSystem.....	47
Figure 47: Query 78 Non-FlexibleSystem	48
Figure 48: Query 78 FlexibleSystem.....	48
Figure 49: Query 79 Non-FlexibleSystem	48
Figure 50: Query 79 FlexibleSystem.....	48
Figure 51: Query 80 Non-FlexibleSystem	49
Figure 52: Query 80 FlexibleSystem.....	49
Figure 53: Query 81 Non-FlexibleSystem	49
Figure 54: Query 81 FlexibleSystem.....	49
Figure 55: Query 82 Non-Flexible System	50
Figure 56: Query 82 Flexible System.....	50
Figure 57: Query 83 Non-Flexible System	50
Figure 58: Query 83 Flexible System.....	50
Figure 59: Query 84 Non-Flexible System	51
Figure 60: Query 84 Flexible System.....	51
Figure 61: Query 85 Non-Flexible System	51
Figure 62: Query 85 Flexible System.....	51
Figure 63: Query 86 Non-Flexible System	52

Figure 64: Query 86 Flexible System.....	52
Figure 65: Query 87 Non-Flexible System	52
Figure 66: Query 87 Flexible System.....	52
Figure 67: Query 88 Non-Flexible System	53
Figure 68: Query 88 Flexible System.....	53
Figure 69: Query 89 Non-Flexible System	53
Figure 70: Query 89 Flexible System.....	53
Figure 71: Query 90 Non-Flexible System	54
Figure 72: Query 90 Flexible System.....	54

1 Introduction

Information retrieval deals with extracting relevant information from documents. Structural or XML retrieval allows us to handle more complicated queries than does traditional Information Retrieval (IR). The focus of this thesis is the processing of Content and Structure (or CAS) queries in a standard XML environment.

A document stored in XML format [1] is highly structured, as information is divided into logical components. XML (Extended Mark up Language) is a highly structured language that helps to represent the logical structure of a document. XML uses opening and closing tags to mark the start and the end of a logical part in the document. For example, an abstract lies within `<abstract>` and `</abstract>` tags. Hence we have a structured collection of information. With a structured system, we can search for both content and structure by specifying them in a query. For example, we can search for documents about *image retrieval* written by *Kim*, etc. Traditional retrieval can search for documents about *image retrieval* but cannot necessarily guarantee that the author is *Kim*.

1.1 The Basic Retrieval Engine

Smart [2] is used as our basic retrieval engine. Smart takes as input a single query or set of queries and a document collection. It searches the document collection and provides as output a ranked list of documents considered by the system to be relevant to the

query. An advantage of Smart is that it uses the Vector Space Model [3] for retrieval. Smart retrieval is performed in two basic steps: (1) Indexing, and (2) Retrieval.

In the Vector Space Model, each document is represented as a weighted term vector. A term vector consists of word stems from the document. Each term has a weight associated with it that represents the importance of the term in the document. Commonly used weighting schemes include *tf-idf* and *atc* [4], *lnu.Ltu* [5], etc. Smart permits an intermediate step between indexing and retrieval, where we can specify the basic weighting to be used (e.g., *nnn* or *tf* weighting, *atc*, *lnu.Ltu*, etc).

After the indexing is done, Smart retrieves documents in rank order to the query. Smart produces as output a ranked list of documents and allows us to specify the number of documents to be retrieved.

1.2 The Extended Vector Space Model

Fox [6] extended the Vector Space Model to include objective as well as subjective identifiers. Various classes of information are represented in a single vector, called the extended vector. A document vector now consists of different subvectors, and each subvector represents a different concept-type, which is called the c-type. Similarity between a pair of extended vectors can be calculated linearly by combining the similarities of the corresponding c-types. Figure 1 demonstrates an example of the extended vector.

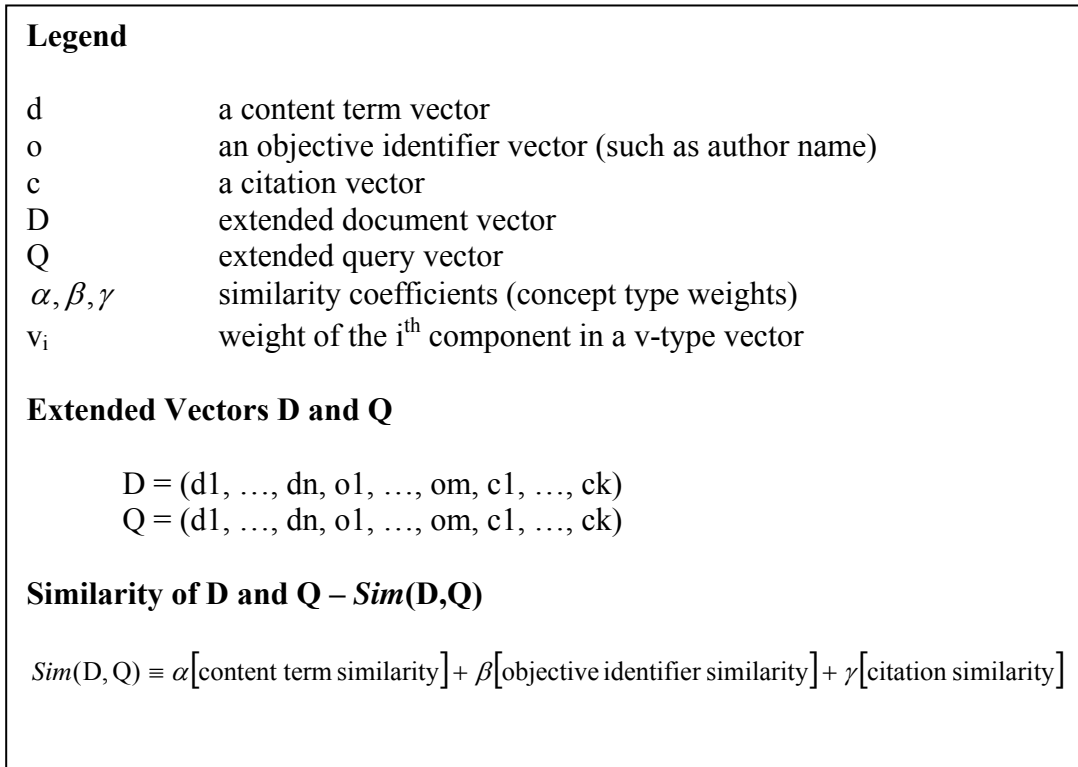


Fig. 1: An example of Extended Vector Space model [7]

Using this structure, we can divide any query into two parts: (1) the subjective part (2) the objective part. The subjective part of the extended vector query is the actual text to be searched for (i.e., the search terms). The subjective part forms the basis of the query and retrieval results depend primarily on the subjective search. For example, if a query asks for documents about *image retrieval* published in 1999, then *image retrieval* forms the subjective part of the query.

Consider now the objective subvectors, such as author name, publication year, author affiliation, etc. In the traditional Extended Vector Space Model as described by Fox [6], objective subvectors are used as search fields in the same way as the subjective subvectors. We might include an objective subvector for year of publication, author

name, etc. But in this model, a match in any subvector field simply increases the overall correlation between vectors (i.e., document and query); it increases the rank at which the document is retrieved. There is no concept of a condition being met. In our XML version of the Extended Vector Space Model, the objective subvectors are treated like filters; they represent conditions that must be met. Consider the same example, i.e., documents about *image retrieval* published in 1999. First, documents on *image retrieval* are retrieved based on the subjective subvectors. Then this list of documents is passed through a filter that guarantees that only documents published in 1999 are retrieved.

1.3 Structured Retrieval under INEX

Structured retrieval deals with the retrieval of documents for a query with “structure.” Structure is added by using any structured language to represent queries. Hence, we need structured queries and documents to perform retrieval tasks and conduct experiments. INEX [8] provides the test bed for these experiments. The INEX 2003 [8] and INEX 2004 collections and metrics are used in our experiments. The University of Minnesota, Duluth is a participant in INEX, which was formed to investigate retrieval in this environment.

The data that we use from the INEX database consists mainly of (1) the document collection, (2) query sets, (3) assessments, and (4) evaluation procedures.

1.3.1 Document Collection

The document collection is made up of articles marked up in XML. There are a total of 12,107 articles, published between 1995 and 2002. All these articles are publications of the IEEE Computer Society. All the articles follow a common pattern or DTD.

An article basically contains *front matter* within `<fm>` and `</fm>` tags, main *body* within `<bdy>` and `</bdy>` tags, and *back matter* between `<bm>` and `</bm>` tags. Information about year of publication, author name, author affiliation and abstract are present in the *front matter*.

The text of the article is present in the *body*. A body is further divided into sections. Text within `<sec>` and `</sec>` tags represents a *section*. A *section* can have a *section title* and it is written within `<st>` and `</st>` tags. The sections can contain *subsections*, enclosed in `<ss>` and `</ss>` tags. If there is more than one *subsection*, then these tags are numbered (e.g., `<ss1>` and `<ss2>`). Subsections are again composed of paragraphs, and each paragraph is enclosed within `<p>` and `</p>` tags. There are tags for figures, lists, list elements, tables, etc.

The *back matter* consists of bibliographical entries within `<bib>` and `</bib>` tags. Inside these tags, there are tags for bibliographical article title (`<bibl_atl>`), author first name and last name (`<bibl_au_fnm>` and `<bibl_au_snm>`), etc. The basic structure of an XML article is shown in Figure 2.

```

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

```

Fig. 2: Structure of INEX article [7]

1.3.2 Topics

The queries are developed by INEX participants. These topics are represented in XML format. The test collection contains two types of queries, Content Only (CO) and Content and Structure (CAS) queries. The Content Only queries are very similar to

traditional IR queries. In Content and Structure queries, the user can restrict the search by specifying more conditions to be met. There are 34 CAS and 40 CO queries in the INEX 2004 collection and 30 CAS and 36 CO queries in the INEX 2003 collection. Each query has attributes like topic identification number, topic type (CO or CAS), etc. Our experiments are based on the 2003 CAS query set.

1.3.3 Assessments

All documents are assessed with respect to every query in the collection. Assessments [9] are done manually by INEX participants (each group is given a set of queries to access). The final relevance of an element is assessed based not just on relevance but also on coverage. The terms used to describe relevance and coverage are exhaustivity and specificity, respectively. Exhaustivity measures how relevant an element is, and specificity measures how specific an element is to the topic. There are different levels of exhaustivity and specificity as represented by the numbers 0 (not exhaustive or not specific), 1 (marginally exhaustive or marginally specific), 2 (fairly exhaustive or fairly specific), or 3 (highly exhaustive or highly specific).

Assessments are made at the element level. Every element is assessed and given one value for exhaustivity and another for relevance. These values are propagated to the parent of that element. For example, parent should be at least as exhaustive as its children and should not be more specific than any child.

1.3.4 Evaluation

When relevance assessments are complete, the topics are distributed to INEX participants, who run the topics on their systems and submit the results to INEX for evaluation. Results are evaluated using the INEX 2003 metrics [14], which convert exhaustivity and specificity values into recall-precision curves. We used the `inex_eval` package [10] to evaluate the results of retrieval.

2 CAS Query Processing

A typical CAS query downloaded from the INEX collection is a XML-marked text file of the form shown in Figure 3.

```
<inex_topic topic_id="127" query_type="CAS" ct_no="13">
<title>.....</title>
<description>.....</description>
<narrative>.....</narrative>
<keywords>..... </keywords>
</inex_topic>
```

Fig. 3: DTD of a XML query in INEX collection

2.1 Description of XML Query

As shown in Figure 3, a query consists of (1) topic, (2) title, (3) description, (4) narrative, and (5) keywords.

2.1.1 Topic

The whole query is contained within the `<inex_topic>` and `</inex_topic>` tags. The text in the `inex_topic` tag identifies the query by identification number (`topic_id`) and query type (CO or CAS).

2.1.2 Title

Only this part of the XML query is used to construct the corresponding Smart query, by processing the text within <title> and </title> tags. The title represents what the user, who formed the query, actually wants. It contains both the subjective and objective features of the query (see Section 2.2 for more detailed information).

2.1.3 Description

This part of the query represents the description of the text in the title tag. It describes, in English, the element to be returned and what is to be searched for (i.e., the search terms).

2.1.4 Narrative

Text found within <narrative> and </narrative> tags is the actual description of the query by the person who formed the query. S/he describes in detail the documents s/he is interested in and the criteria to be used when judging a document for relevance. This tag is of little significance in processing the query, but it plays an important role in assessing documents. Assessors depend on the narrative when assessing documents and assigning relevance values to the elements.

2.1.5 Keywords

These are words that would likely be present in documents considered relevant. Keywords are provided by the person who forms the query. Keywords play a significant role in assessment. While assessing a document, one may look for keywords in the

document. Keywords are of less significance in query processing. They are used only during assessment.

2.2 Processing XML Queries

The XML files which represent the topics must be parsed to form the corresponding Smart queries. The current Smart representation of a given document consists of 18 content types or ctypes as shown in Table 1. Of these 18 tags, 6 are subjective tags and the rest are objective tags. We identify subjective and objective parts of the query from the title field of the XML query file. We also identify the element to be returned (e.g., abstract, section, paragraph, etc.).

Let us consider the syntax of an XML topic or a query. The title field of the topic contains all the information required to construct the corresponding Smart extended vector query. It tells us (1) what terms to search for, (2) where (what fields) to search, and (3) the element to return when the search is complete. The general syntax of the title field of a simple INEX topic can be represented as

A [about (.X, "...")]

Here, the information within brackets (after *about*) represents (1) where to search (search fields), and (2) the terms to be searched for (search terms). "X" represents the search field and the text within quotes represents the search terms. If no X is specified, it means that the search terms can be searched for anywhere in the document. "A" represents the element to be returned for that search. For example, consider Figure 4. The query specifies the search topic, *Godel Lukasiewicz and other fuzzy*

Table 1: c-types and their description [7]

Content type or c-type	Description	Subjective or Objective
Abs	Abstract	Subjective
Ack	Acknowledgements	Objective
Article_au_fnm	First name of article's author	Objective
Article_au_snm	Surname of article's author	Objective
Atl	Article Title	Objective
Au_aff	Author affiliation	Objective
bdy	Body	Subjective
bibl_atl	Article title in bibliography	Objective
bibl_au_fnm	Author's first name in bibliography	Objective
bibl_au_snm	Author's surname in bibliography	Objective
bibl_ti	Publication (journal) title in bibliography	Objective
ed_aff	Editor affiliation	Objective
fgc	Figure Caption	Subjective
p	Paragraph	Subjective
Pub_yr	Publication year	Objective
Sec	Section	Subjective
St	Section Title	Subjective
Vt	Vitae	Subjective

implication definitions, to be searched for in abstracts (marked as *X* below). *Section* (marked as *A* below) is the element to be returned. E.g.,

$$\begin{array}{c} \mathbf{A} \text{ [about (} \mathbf{X}, \text{ "....."} \text{)]} \\ \downarrow \qquad \downarrow \\ \text{<title> //sec[about (./abs, "Godel Lukasiewicz ... definitions")] </title>} \end{array}$$

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="127" query_type="CAS" ct_no="13">
  <title> //sec[about( ./abs, "Godel Lukasiewicz and other fuzzy implication
  definitions")] </title>
  <description>Find sections containing the definition of Godel, Lukasiewicz or
  other fuzzy-logic implications in abstracts</description>
  <narrative>Any relevant element of a section must contain the
  definition of a fuzzy-logic implication operator or a pointer to the
  element of the same article where the definition can be
  found. Elements containing criteria for identifying or comparing fuzzy
  implications are also of interest. Elements which discuss or introduce
  non-implication fuzzy operators are not relevant. </narrative>
  <keywords>Godel implication, Lukasiewicz implication, fuzzy
  implications, fuzzy-logic implication </keywords>
</inex_topic>

```

Fig. 4: An example of a simple CAS query

A query can specify more than one search field. This is done by adding more *about* expressions to a simple query. Hence, a query that specifies more than one search field can be represented as

A [about (.X, “text1”) AND/OR about (.Y, “text2”) ...]

Here, the first search term, *text1*, is to be searched for in the field *X* and the second search term, *text2*, is to be searched for in the field *Y*, and so on. For example, consider Figure 5.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="133" query_type="CAS" ct_no="67">
<title>//article[about(//atl, “Query”) and about(//st,
“optimization”)]</title>
<description>We are looking for articles that have worked on
developing optimization techniques for query
processing.</description>
<narrative>Our primary area of interest is in the field of XML
databases and developing optimization equivalences for an XML query
language algebra. The lack of research on XML Algebra and hence
optimization equivalencies for the same led us to focus on algebras
that have been developed for other query languages and the mechanism
followed in the development of such algebras. We are also interested
in the equivalencies that have been drawn for such algebras and on
their suitability to XML. Realizing that query is an important term
and must be in the title of the paper, and optimization should be in
the title of a section. This seems to be a typical property of such
papers, the keyword "query" is important to be in article title, but
optimization is usually one or more sections.</narrative>
<keywords>Optimization equivalencies, System R rules, Query
processing, Query algebra</keywords>
</inex_topic>
```

Fig. 5: Query 133 from the INEX 2004 collection


```

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE inex_topic SYSTEM "topic.dtd">

<inex_topic topic_id="128" query_type="CAS" ct_no="22">

<title>//article [about (., 'intelligent transport systems')]///sec[about(., 'on-board route planning navigation system for automobiles')]/</title>

<description>Find discussions about on-board route planning or navigation systems which are in publications about intelligent transport systems for automobiles.</description>

<narrative>I'm interested in information about on board route planning or navigation systems for automobiles. Relevant elements discuss either a requirement analysis or a concrete implementation of such a system. Elements about navigation or route planning systems that cannot be accessed within the automobile will not be considered relevant. Systems of other phenomena than automobiles will also not be judged relevant.</narrative>

<keywords>in-vehicle systems, vehicle intelligence, vehicle information systems, traffic information services, vehicle-mounted equipment</keywords>

</inex_topic>

```

Fig. 7: Query 128 from the INEX 2004 collection

2.2.2 Splitting a Query into Parts

If the query has n parts, i.e., $A_1 [\dots] A_2 [\dots] \dots A_n [\dots]$, then the query is split into n parts, $A_1 [\dots]$, $A_2 [\dots] \dots A_n [\dots]$. Each part of the query is processed, and the results are combined to get the final list of elements. A single part of the query can be further

divided into subparts if it specifies more than one search field, i.e., if the query is of the form **A [about (.X, "...") AND/OR about (.Y, "...") ...]**. The document collection is indexed to recognize paragraphs, abstracts, section titles, figure captions, lists and vitae. Hence, paragraphs, abstracts, section titles, figure captions, lists and vitae are treated as distinct search fields. If the query says that text must be searched in two or more of these fields, then the query is split into two or more queries and the results are combined to get the final result. For example, consider the query shown in Figure 8.

```

<inex_topic topic_id="64" query_type="CAS" ct_no="16">
<title>//article [about(., 'hollerith') OR about(./sec, 'Herman')]/sec[about(.,
'DEHOMAG')]</title>

<description>In articles discussing Herman Hollerith find sections that are about
DEHOMAG. </description>

<narrative>
Relevant sections deal with DEHOMAG (Deutsche Hollerith Maschinen
Gesellschaft) in documents that discuss work or life of Herman Hollerith or
sections talking about Herman.</narrative>

<keywords>Hollerith, DEHOMAG, Deutsche Hollerith Maschinen Gesellschaft
</keywords>

</inex_topic>

```

Fig. 8: Query number 64 from INEX 2003 collection

In this case, the query has two parts. The first part is *//article [about(., 'hollerith') OR about(./sec, 'Herman')]* and the second part is *//sec[about(., 'DEHOMAG')]*. Thus, the query is divided into two parts. The first part of the query can be further divided into

sub parts as the query specifies that *hollerith* can be searched anywhere in the document (i.e., the entire article), and *Herman* is to be searched in the sections.

A query can also specify the text that should *not* be present a relevant document. Such text is preceded by a minus (-) sign. For each piece of text preceded by a minus sign, a separate query is formed. Also, there can be some words that *should* be present in the document for the document to be relevant. Such term(s) are preceded by a plus (+) sign. Separate queries are formed for these terms as well.

For example, consider the query shown in Figure 9. In this query, the terms *software* and *architecture* are preceded by a plus sign and the words *distributed* and *Web* are preceded by a minus sign. This means that a relevant document **should** cover *software* and *architecture* but **not** *distributed* and *Web*. We divide the query into four parts in this case, one for each term. Thus, we get four lists of results (one for each part of the query), and these results are combined to get a single result file consisting only of article front matters (see Section 2.2.5).

Thus, a query can be divided into n parts depending on its syntax. Each part can be divided into subparts depending on the number of search fields, which can be further divided into parts depending on the presence of plus or minus signs preceding the search terms.

```
<inex_topic topic_id="67" query_type="CAS" ct_no="38">
<title>//article//fm[(about(//tig, '+software +architecture') or
about(//abs, '+software +architecture')) and about(., '-distributed
-Web')]</title>
<description> An overview of software architecture as it applies to
traditional software engineering, not to distributed nor web software
development. </description>
<narrative> To be relevant an abstract must discuss software
architecture as it applies to stand-alone application development. It
should not discuss the software architecture as it applies to web
software or distributed software engineering. The ideal abstract would
be to a document giving an overview of the topic. </narrative>
<keywords>software, architecture, distributed, web</keywords>
</inex_topic>
```

Fig. 9: Query number 67 from INEX 2003 collection

2.2.3 Running the Queries

Once the XML files are split into the required number of parts for each query, they can be processed by Smart. Smart takes as input these queries and returns as output a ranked list of documents for each query. Smart creates its own internal query numbers based on the order of queries given as input. For example, if we give query number 127 as the first query in the input to Smart, Smart produces a ranked list of documents with query number, document identification, rank and similarity value; the id of the Smart query corresponding to query 127 is 1. An example of a Smart result file is shown in Figure 10. It illustrates that the Smart results are sorted by document-id (second column).

<u>query no.</u>	<u>doc-id</u>	<u>rank</u>				<u>similarity value</u>
1	19545	987	0	0	0	15.1398
1	21747	183	0	0	0	20.0962
1	21749	394	0	0	0	18.1190
1	26506	239	0	0	0	19.4241
1	30959	952	0	0	0	15.2412
1	38642	139	0	0	0	20.7487
1	38643	677	0	0	0	16.3190
1	41045	978	0	0	0	15.1485
1	42745	922	0	0	0	15.3596
1	43464	914	0	0	0	15.3741
1	45267	362	0	0	0	18.3601
1	45282	660	0	0	0	16.3985
1	45287	605	0	0	0	16.6754
1	45290	733	0	0	0	16.0555
1	50661	108	0	0	0	21.4473
1	50681	155	0	0	0	20.5898
1	56244	211	0	0	0	19.8168
1	58277	591	0	0	0	16.7798
1	58278	270	0	0	0	19.1308
1	58299	686	0	0	0	16.2717
1	58300	668	0	0	0	16.3700

Fig. 10: A sample result file of Smart

2.2.4 Running the Flexible Program

The next step is to input the Smart results to the flexible retrieval program. [11] The whole document collection is indexed on paragraphs, abstracts, figure captions, and section titles, i.e., certain elements are treated as documents and these elements are retrieved, but not the entire documents. The program for flexible retrieval reads an element (paragraph) from the list retrieved by Smart for the query and builds a tree for the *document* it is contained in, by using the relevance values associated with the element and propagating calculated values (for relevance and coverage) to the elements higher in the DTD. For example, if Smart returns element *paragraph*, p, then the sub

section and/or section to which the paragraph belongs is assigned calculated (propagated) relevance and coverage values. Relevance and coverage are propagated similarly up the tree until the article level is reached. A sample tree is shown in Figure 11. When a tree has been built for each document with an element in the retrieved list of paragraphs, the elements are sorted based on relevance and coverage. The flexible program produces as output a list of the highest ranking elements (in terms of relevance and coverage) associated with the original query. See [11] for details.

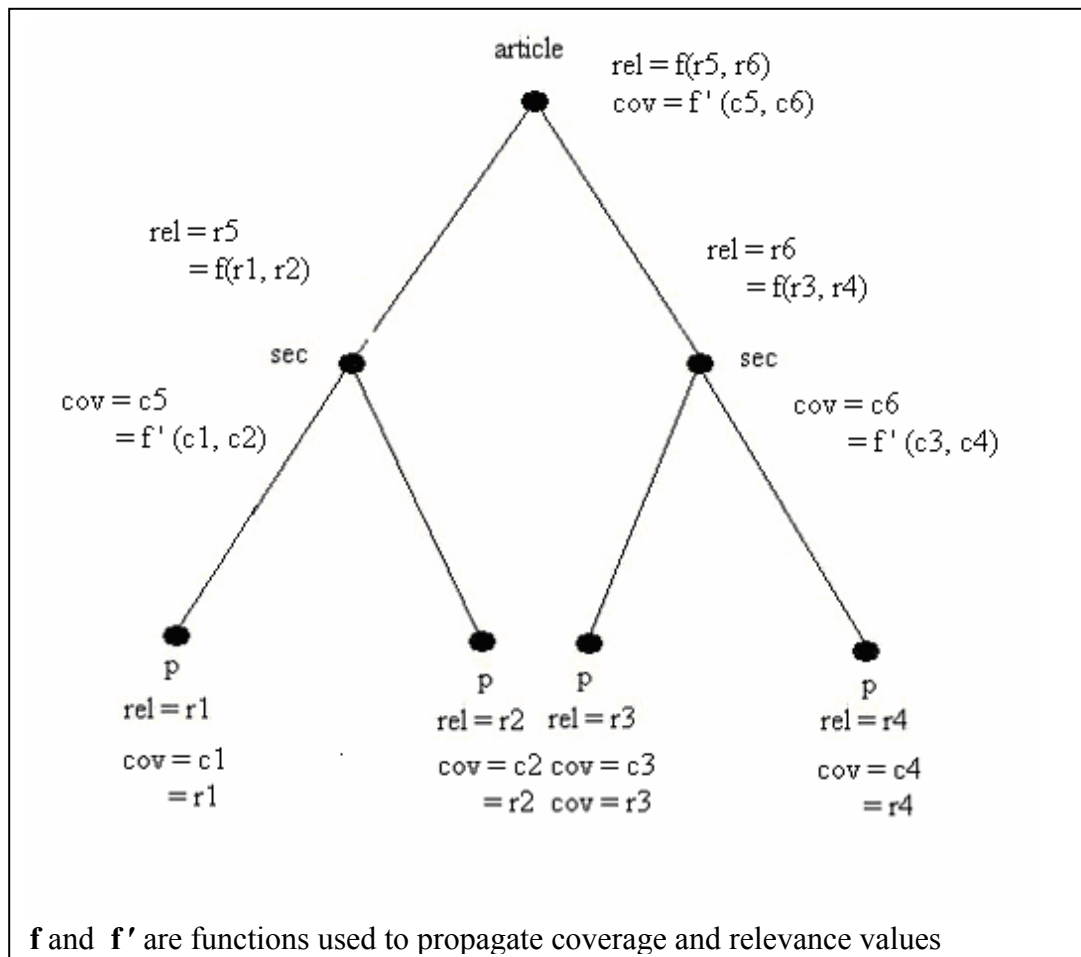


Fig. 11: Propagation of relevance values to parent nodes

2.2.5 Combining the Results for INEX Submission

We submit one set of results (elements) per query. As discussed earlier, a query may be split into n parts depending on its original structure. Once we have the results of these n parts, we must combine them to form a single list of elements. Depending on the format of the query, there are three different combinations possible: (1) ranked intersection, (2) parent intersection, and (3) difference.

2.2.5.1 Ranked Intersection

When a single part of a query is split into sub parts (i.e., when text is to be searched for in more than one subjective field in a single part of the query), we combine the results by taking the intersection of all the parts. If we want results for "Strict CAS" queries, in which all conditions have to be met for an element to be relevant, then we consider only those elements that are obtained as a result of the intersection. But if we want results for "Vague CAS" queries, then we also consider those elements that don't necessarily meet all the conditions. The elements that meet all conditions are ranked higher than the elements that meet all but one condition, elements that meet all but one condition are ranked higher than the elements that meet all but two, etc. This process is depicted in Figure 12.

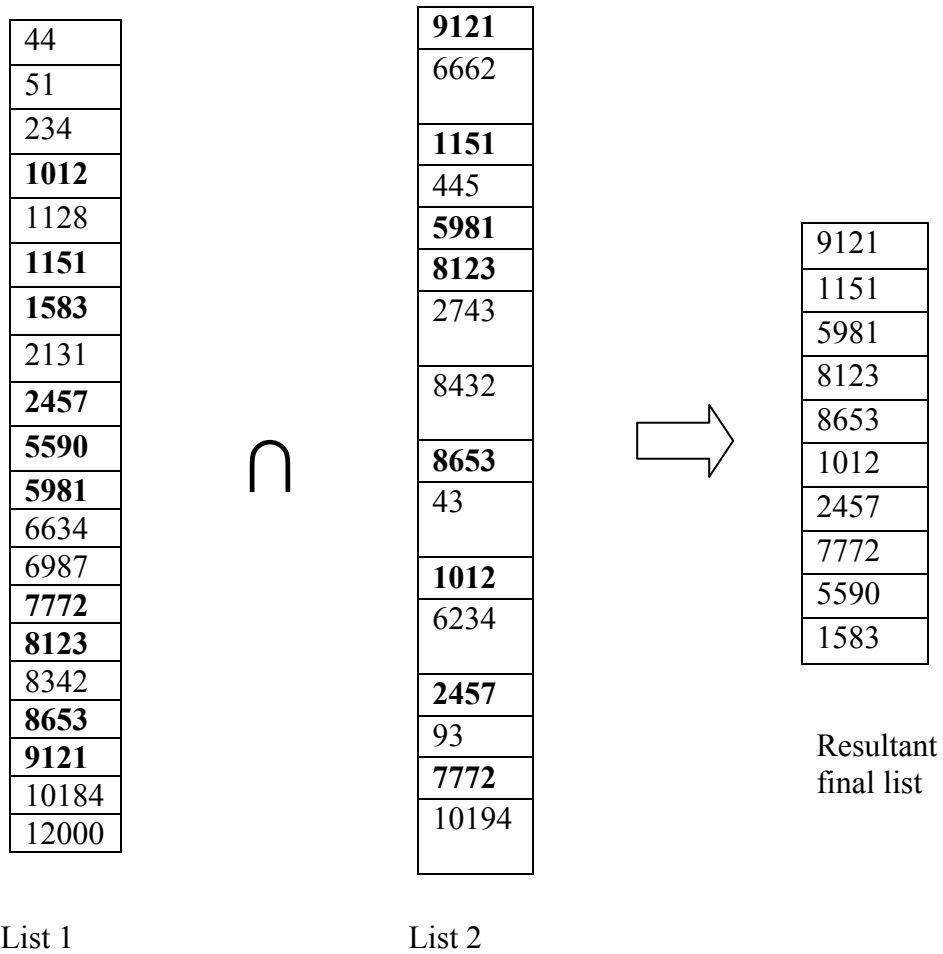


Fig. 12: Ranked Intersection of two lists [7]

2.2.5.2 Parent Intersection

In this kind of intersection, we deal with the intersection of two lists, where both the lists contain elements of a single kind (i.e., all elements are articles, or all elements are sections, etc.). Say the entire first list contains elements of type *element-one* and the entire second list contains elements of the type *element-two*. The structure of the original XML query guarantees that *element-one* is always the parent of *element-two*. For example, if the entire first list contains *articles*, then the second list will contain *sections*, *paragraphs*, or *abstracts*, etc. Any of these elements would be a child of *article*. In parent intersection, we take each element from the second list and compare its xpath [12] with that of the elements in the second list. If the xpath of an element in the first list is contained in the xpath of an element in second list, then we write that element into the output file, else we ignore that element. Figure 13 is an example of parent intersection.

In Figure 13, the first list contains all articles and the second list contains all sections. The result list is the result of parent intersection of both the lists. Consider the xpath, “mi/1999/m5084, books[1]/journal[1]/article[1]/bdy[1]/sec[1].” Here the parent article, “mi/1999/m5084, books[1]/journal[1]/article[1]” is present in the first list. Hence, this xpath is present in the result list.

it/1999/f1057, books[1]/journal[1]/article[1]
o/1999/r7094, books[1]/journal[1]/article[1]
so/1999/s1009, books[1]/journal[1]/article[1]
mi/1999/m5084, books[1]/journal[1]/article[1]
mi/1999/m5084, books[1]/journal[1]/article[1]
co/1999/r1028, books[1]/journal[1]/article[1]
so/1999/s2066, books[1]/journal[1]/article[1]
cs/1999/c3005, books[1]/journal[1]/article[1]
it/1999/f5058, books[1]/journal[1]/article[1]
co/1999/r3024, books[1]/journal[1]/article[1]

List 1

mi/1999/m5084, books[1]/journal[1]/article[1]/bdy[1]/sec[1]
co/1998/s1094, books[1]/journal[1]/article[1]/bdy[1]/sec[1]
ex/1999/e1009, books[1]/journal[1]/article[1]/bdy[1]/sec[2]
it/1999/f1057, books[1]/journal[1]/article[1]/bdy[1]/sec[3]
co/1999/c5084, books[1]/journal[1]/article[1]/bdy[1]/sec[6]
it/1999/g1028, books[1]/journal[1]/article[1]/bdy[1]/sec[4]
so/2000/s2066, books[1]/journal[1]/article[1]/bdy[1]/sec[1]
ex/1999/c3005, books[1]/journal[1]/article[1]/bdy[1]/sec[1]
cg/1999/f5058, books[1]/journal[1]/article[1]/bdy[1]/sec[1]
cs/1999/c3005, books[1]/journal[1]/article[1]/bdy[1]/sec[1]

List 2

PI



mi/1999/m5084, books[1]/journal[1]/article[1]/bdy[1]/sec[1]
it/1999/f1057, books[1]/journal[1]/article[1]/bdy[1]/sec[3]
cs/1999/c3005, books[1]/journal[1]/article[1]/bdy[1]/sec[1]

Result List

Fig. 13: Illustration of Parent Intersection

2.2.5.3 Difference

Difference is similar to mathematical subtraction. Given two lists, say A and B, performing (A difference B) would result in all the elements of the list A that are not present in the list B. Difference is used in the cases when it is specified in the query that some text should *not* be present in the document for it to be relevant. This is done by preceding a specific search term or terms by a minus sign. Figure 14 illustrates the difference between two lists.

it/1999/f1057, books[1]/journal[1]/article[1]
co/1999/r7094, books[1]/journal[1]/article[1]
so/1999/s1009, books[1]/journal[1]/article[1]
mi/1999/m5084, books[1]/journal[1]/article[1]
mi/1999/m5084, books[1]/journal[1]/article[1]
co/1999/r1028, books[1]/journal[1]/article[1]
so/1999/s2066, books[1]/journal[1]/article[1]
cs/1999/c3005, books[1]/journal[1]/article[1]
it/1999/f5058, books[1]/journal[1]/article[1]
co/1999/r3024, books[1]/journal[1]/article[1]

—

mi/1999/m5084, books[1]/journal[1]/article[1]/
co/1998/s1094, books[1]/journal[1]/article[1]/
ex/1999/e1009, books[1]/journal[1]/article[1]/
it/1999/f1057, books[1]/journal[1]/article[1]/
co/1999/c5084, books[1]/journal[1]/article[1]/
it/1999/g1028, books[1]/journal[1]/article[1]/
so/2000/s2066, books[1]/journal[1]/article[1]/
ex/1999/c3005, books[1]/journal[1]/article[1]/
cg/1999/f5058, books[1]/journal[1]/article[1]/
cs/1999/c3005, books[1]/journal[1]/article[1]/

List 1

List 2



co/1999/r7094, books[1]/journal[1]/article[1]
so/1999/s1009, books[1]/journal[1]/article[1]
mi/1999/m5084, books[1]/journal[1]/article[1]
co/1999/r1028, books[1]/journal[1]/article[1]
so/1999/s2066, books[1]/journal[1]/article[1]
it/1999/f5058, books[1]/journal[1]/article[1]
co/1999/r3024, books[1]/journal[1]/article[1]

Result List

Fig. 14: Illustration of Difference

3 Experiments and Results

This chapter illustrates the various experiments performed on the INEX 2003 CAS queries. Smart is the retrieval engine used for all the experiments. The weighting scheme used for our experiments is *Lnu-ltu* as the best results are obtained in our system with this weighting scheme. [13] A Perl interpreter is used to convert the INEX topics in XML format to the extended vector queries that can be used by Smart as input. The results of the Smart retrieval are post-processed (as described in the Section 2.2.5) to ensure that all the conditions of the query are met. An illustration of the automatic query processing at different stages of processing is shown in Appendix 1.

Experiments are performed using either *article* level indexing or *paragraph* level indexing. In *article* level indexing, (where the basic unit of indexing is the article [document]) the element to be returned is the *article* for all the queries. In *paragraph* level indexing, we perform flexible retrieval that allows us to return the highest ranked elements (article, section, paragraph, etc.) as required to match the element specified in the query.

Once we have the final results (list of elements) for each topic, we evaluate them using the *inex_eval* package. [10] Evaluation can be done on a query-by-query basis (yielding one precision-recall curve for each query) or it can be done for the entire set of queries as a whole (resulting in one graph). See Appendix 2 for the graphs.

We observe that better results are obtained with the non-flexible retrieval system when *article* is the element to be returned for a query. This is because the non-flexible system runs on an *article* level indexing and hence the retrieved articles are more accurate representations of the document content than the articles retrieved by the flexible system, where the value of the article is constructed from its parts. On the other hand, when the element to be returned is not an article, we observe better results with the flexible system (See Appendix2).

We also include a comparison of our results produced by our system (flexible program + automatic query processing) to those of other 2003 INEX participants. (See Appendix3) Results using our old system, new system and a hybrid system are shown below in Figures 14, 15 and 16, respectively. The hybrid system uses the non-flexible system for queries which return articles and the flexible system for all other queries. It demonstrates a substantial increase in precision.

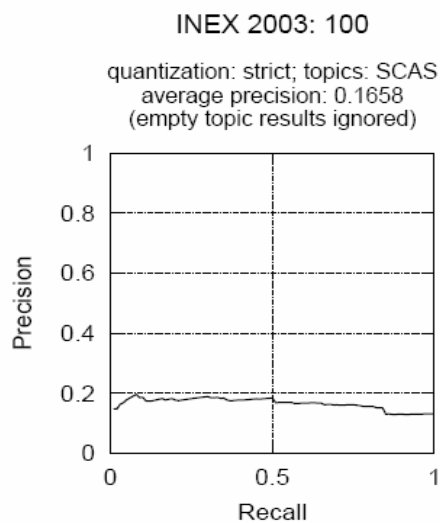


Fig. 15: Results of INEX 2003 CAS queries using non-flexible retrieval system

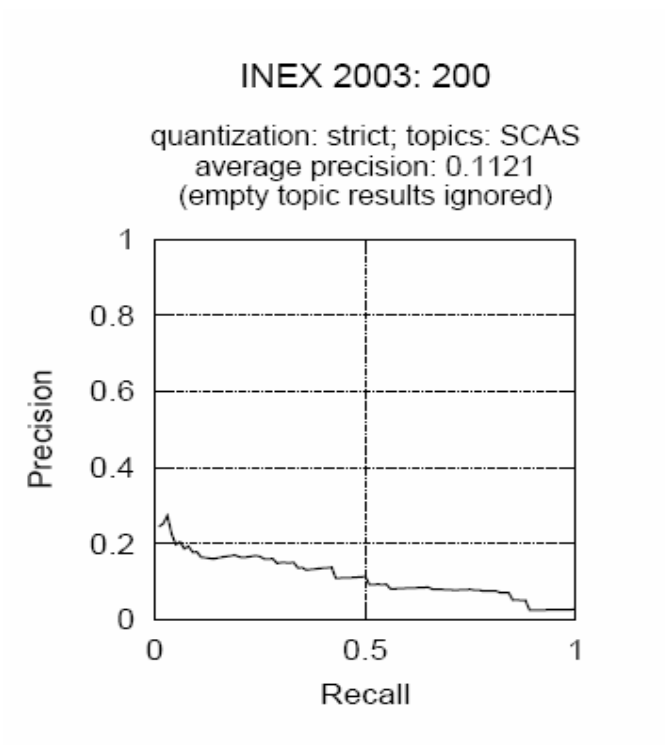


Fig. 16: Results of INEX 2003 CAS queries using flexible retrieval system

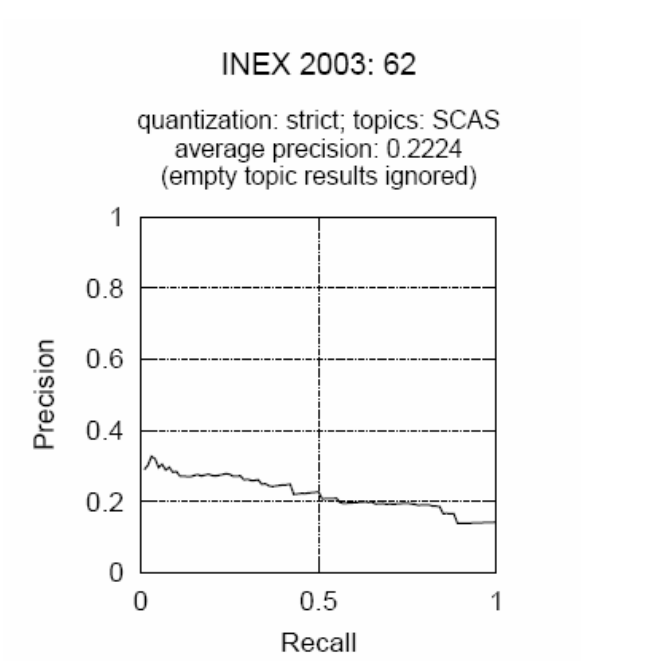


Fig. 17: Results of INEX 2003 CAS queries using a hybrid retrieval system

4 Conclusions and Future Work

Automation of Content and Structure (CAS) queries goes hand-in-hand with the design of the flexible retrieval system. A Perl translator has been built for the processing of CAS queries that takes as input XML queries and produces as output converted extended vector queries for Smart to process. The system post processes the results of the flexible retrieval program to produce the output required by the original XML query. An illustration of this process is contained in Appendix 1. We observe improvement in our results in comparison to those of other INEX participants (2003) using our system (flexible program + automatic query processing). Lastly, we illustrate comparison on a query-by-query and collection basis of the results generated by our new system (flexible program + automatic query processing) versus our old system (non-flexible retrieval and no automatic query processing) and observe that a hybrid system gives better results. (See Appendix 2)

Smart allows us to specify the number of documents to be retrieved. Our current system specifies 1000 elements to be returned. We can experiment with more elements, as better results are possible. Consider for example, a query that asks to retrieve documents published in 1999 or 2000 or 2001. We observed that relevant elements are greater than 1000 in number, and hence we could not retrieve all the relevant elements. Due to time and cost constraints we were not able to experiment with greater numbers. Also, a better approach might be possible to add elements to a file when dealing with VCAS queries. (see Chapter 2) These options are left as a future work.

5 Bibliography

- [1] Flynn P., The XML FAQ. <http://www.ucc.ie/xml/>
- [2] Salton, G., editor. *The SMART Retrieval System – Experiments in Automatic Document Retrieval*, Prentice-Hall, Englewood Cliffs, NJ, 1971.
- [3] Salton, G., Wong, A., and Yang C. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, v.18 (11), 613-620, Nov.1975.
- [4] Salton G., and Buckley, C. Term Weighting approaches in Automatic Text Retrieval. *IP&M* 24(5), 513-523, 1998.
- [5] Singhal, A., Salton, G., Mitra, M., and Buckley C. Document Length Normalization. *Information Processing and Management*, 32(5):619-633, 1996.
- [6] Fox, E. 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.
- [7] Bapat, Harsh. Adapting the Extended Vector Space Model for Structured XML Retrieval. M.S. Thesis, Department of Computer Science, University of Minnesota Duluth, 2003.
- [8] Fuhr N., Malik S., Lalmas M. Overview of the INitiative for the Evaluation of XML Retrieval (INEX) 2003. In: *Proceedings of the INitiative for the Evaluation of XML Retrieval (INEX) 2003 Workshop*, Schloss Dagstuhl, Germany, 2003.
- [9] Kazai G., Lalmas M., Piwowarski B. INEX 2003 Relevance Assessment Guide.
- [10] INEX Retrieval Result Submission Format and Procedure. In *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, (pp. 182-183), Dagstuhl, Germany, 2002.
- [11] Mahajan, Aniruddha. Flexible Retrieval in a Structured Environment. M.S. Thesis, Department of Computer Science, University of Minnesota Duluth, 2004.
- [12] XML Path Language (Xpath). <http://www.w3.org/TR/xpath>

- [13] Apte, S. Using the Extended Vector Space Model for Content Oriented XML Retrieval. M.S. Thesis, Department of Computer Science, University of Minnesota Duluth, 2003.
- [14] Gövert, N. and Kazai, G. Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2002. In *Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval (INEX)*, Dagstuhl, Germany, 2002.

6 Appendix 1

6.1 Initial INEX 2003 Query

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="61" query_type="CAS" ct_no="2">
<title>//article[about(.,'clustering +distributed') and about(./sec,'java')]</t
itle>
<description>Retrieve articles about clustering with a section
describing usage of the java programming language and the java virtual
machine.</description>
<narrative> The article should contain a description of usage of java
based applications or techniques for running applications on several
computers (clustering). Please note that I am not interested in the
"clustering" concept from statistics and pattern
recognition. </narrative>
<keywords>clustering,distributed, java</keywords>
</inex_topic>
```

Fig. 18(a): Query 61 from the INEX 2003 collection

Figure 18(a) shows an XML topic from the INEX 2003 collection that must be parsed by the interpreter to produce one or more queries that can be used as Smart input.

6.2 Result of a query fed to the Interpreter

```
<article>
cas-61-1--1
<bdy>
clustering +distributed
```

Fig. 18(b): First part of the result of interpreter on query 61 from the INEX 2003 collection

```
<article>
cas-61-2--2
<bdy>
java
```

Fig. 18(c): Second part of the result of interpreter on query 61 from the INEX 2003 collection

After the queries are formed that can be fed to Smart, a Smart retrieval is done on these queries.

6.3 Smart Results of Queries

Figure 18 and Figure 19 show a small part of Smart results of the first part of query 61 (shown in Figure 16) and the second part of query 61 (shown in Figure 17). Smart results contain columns for query number, document-id, rank, and relevance values (as shown in Figures 18 and 19). These results are converted automatically to the format shown in Figure 20 and are input to the flexible program. The input file to the flexible

program has columns for query number, document-id, relevance, rank and xpath (see Figure 20).

Q.No	Doc-id	Rank				Relevance Value
1	19545	987	0	0	0	15.1398
1	21747	183	0	0	0	20.0962
1	21749	394	0	0	0	18.1190
1	26506	239	0	0	0	19.4241
1	30959	952	0	0	0	15.2412
1	38642	139	0	0	0	20.7487
1	38643	677	0	0	0	16.3190
1	41045	978	0	0	0	15.1485
1	42745	922	0	0	0	15.3596
1	43464	914	0	0	0	15.3741
1	45267	362	0	0	0	18.3601
1	45282	660	0	0	0	16.3985

Fig. 19(a): Smart results of Query 61 (from INEX 2003 collection) – Part 1

Q.No	Doc-id	Rank				Relevance Value
2	127585	718	0	0	0	23.1850
2	127588	429	0	0	0	25.4427
2	127589	529	0	0	0	24.5024
2	127590	486	0	0	0	24.8328
2	127624	572	0	0	0	24.1263
2	127628	139	0	0	0	29.0886
2	127629	792	0	0	0	22.7945
2	127630	933	0	0	0	21.9437
2	127654	778	0	0	0	22.8836
2	128884	298	0	0	0	26.8335
2	128895	306	0	0	0	26.7484

Fig. 19(b): Smart results of Query 61 (from INEX 2003 collection) – Part2

Q.no	Docid	Rel	Rank	xpath
1	987	15.1398	19545	an/1999/a1004/article[1]/bdy[1]/sec[6]/ss1[4]/ip1[1]
1	183	20.0962	21747	an/2000/a3088/article[1]/bdy[1]/sec[11]/p[1]
1	394	18.1190	21749	an/2000/a3088/article[1]/bdy[1]/sec[11]/p[3]
1	239	19.4241	26506	cg/1995/g1090/article[1]/bdy[1]/sec[1]/ss1[4]/p[1]
1	952	15.2412	30959	cg/1995/g5101/article[1]/bdy[1]/sec[1]/ss1[8]/p[1]
1	139	20.7487	38642	cg/1996/g6033/article[1]/bm[1]/app[1]/p[5]
1	677	16.3190	38643	cg/1996/g6033/article[1]/bm[1]/app[1]/p[6]
1	978	15.1485	41045	cg/1997/g2006/article[1]/bdy[1]/sec[8]/p[2]

Fig. 20: Results that can be input to Flexible Program

6.4 Results of Flexible Program

The flexible program produces a list of all the relevant elements for every query. Figure 21 illustrates the output of the flexible program.

Q.No	Rank	Doc-id	Rel	Cov	xpath
1	1	1	299.3498	4.2844	cg/1999/g5032, books[1]/journal[1]/article[1]
1	2	2	299.3498	4.2844	cg/1999/g5032, books[1]/journal[1]/article[1]/bdy[1]
1	3	3	287.347	7.3403	tp/1998/i1011, books[1]/journal[1]/article[1]
1	4	4	287.347	7.3403	tp/1998/i1011, books[1]/journal[1]/article[1]/bdy[1]
1	5	5	280.863	1.6095	tk/1999/k0595, books[1]/journal[1]/article[1]/bdy[1]
1	6	6	280.863	1.6095	tk/1999/k0595, books[1]/journal[1]/article[1]

Fig. 21: Sample Result of Flexible Program

The result file has columns for query number, rank, document-id, relevance value, coverage value and the xpath.

6.5 Post-processing on Flexible Results - Returning the Required Element

If there are two or more parts of a query, then the results of the parts of the query are combined to form a single result file. Once, a single result file is formed, a new list is formed from this list that contains only the required elements that are to be returned. Consider for example, a query where we want to return only sections. Figure 22 illustrates the results before and after processing for the element to be returned. There are columns for query number, rank, relevance, coverage and xpath.

Q.No.	Rank	Relevance	Coverage	Xpath
8	1	432.6139	3.8204	an/1995/a2033, books[1]/journal[1]/article[1]/bdy[1]
8	2	432.6139	3.8204	an/1995/a2033, books[1]/journal[1]/article[1]
8	3	166.2825	13.8569	an/1995/a2033, books[1]/journal[1]/article[1]/bdy[1]/sec[12]
8	4	138.1071	7.2688	an/1995/a2033, books[1]/journal[1]/article[1]/bdy[1]/sec[9]
8	5	127.8221	1.4525	an/1997/a4064, books[1]/journal[1]/article[1]
8	6	127.8221	1.4525	an/1997/a4064, books[1]/journal[1]/article[1]/bdy[1]
8	7	127.8221	5.8101	an/1997/a4064, books[1]/journal[1]/article[1]/bdy[1]/sec[4]
8	8	81.1473	0.6269	an/1997/a2031, books[1]/journal[1]/article[1]

Fig. 22 (a): Result file before processing for element to be returned

Q.No.	Rank	Relevance	Coverage	Xpath
8	3	166.2825	13.8569	an/1995/a2033, books[1]/journal[1]/article[1]/bdy[1]/sec[12]
8	4	138.1071	7.2688	an/1995/a2033, books[1]/journal[1]/article[1]/bdy[1]/sec[9]
8	7	127.8221	5.8101	an/1997/a4064, books[1]/journal[1]/article[1]/bdy[1]/sec[4]
8	11	76.3265	3.8163	an/1997/a4072, books[1]/journal[1]/article[1]/bdy[1]/sec[2]
8	13	70.8916	7.8768	an/1995/a2007, books[1]/journal[1]/article[1]/bdy[1]/sec[1]
8	16	58.4514	0.0504	an/1995/a4086, books[1]/journal[1]/article[1]/bdy[1]/sec[1]
8	18	58.3367	8.3338	an/1995/a2033, books[1]/journal[1]/article[1]/bdy[1]/sec[10]

Fig. 22(b): Result file with only sections

7 Appendix 2

7.1 Comparison of Results between Flexible and Non-Flexible Systems

The INEX 2003 CAS queries are numbered 61 to 90. [8] The recall-precision graphs for some of the queries using flexible retrieval and without using flexible retrieval are shown below. If there are no relevant documents for a topic in the assessments, then it means that none of the elements in the document collection is considered relevant by the assessors while assessing this query. Hence, there would be no assessments, i.e., no graphs for such topics.

Query 61

There are no relevant elements for topic 61

Query 62

Elements to return are articles for this query.

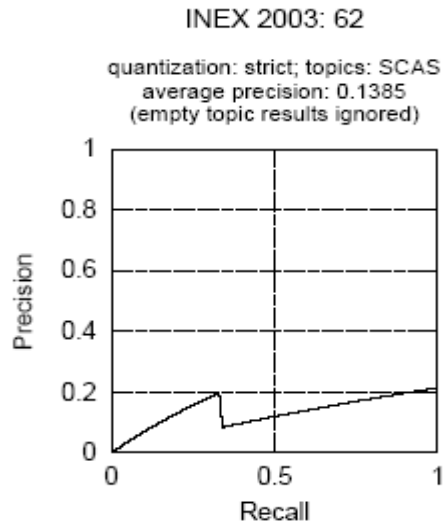


Fig. 23: Non flexible retrieval system

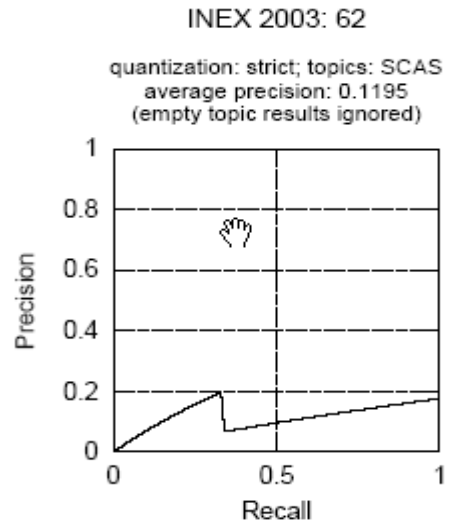


Fig. 24: Flexible retrieval system

Query 63

Elements to return are articles for this query.

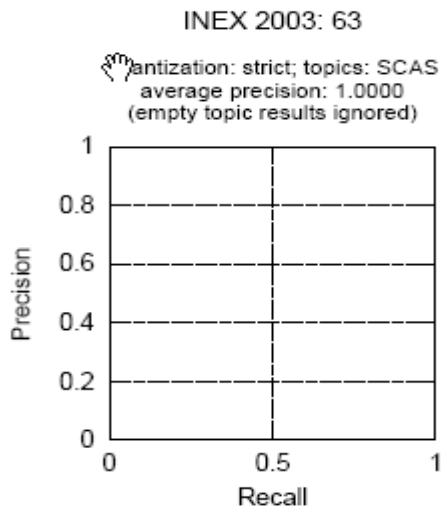


Fig. 25: Non flexible retrieval system

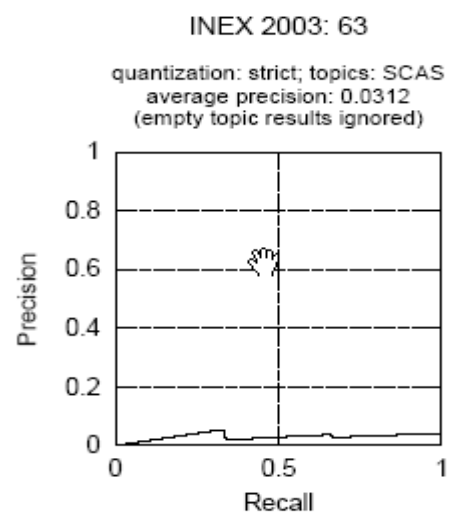


Fig. 26: Flexible retrieval system

Query 64

Elements to return are sections for this topic.

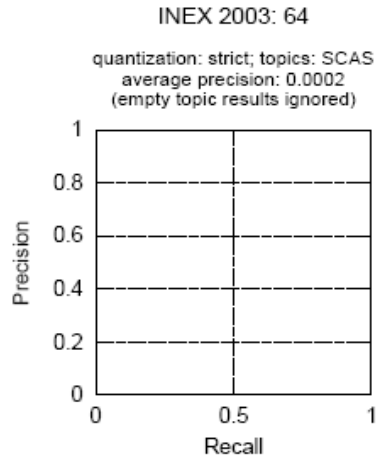


Fig. 27: Non-Flexible Retrieval System

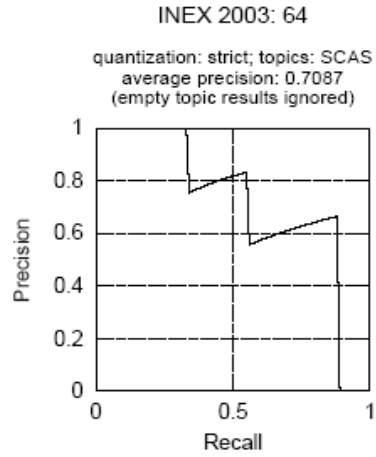


Fig. 28: Flexible Retrieval System

Query 65

Elements to return are articles for this query.

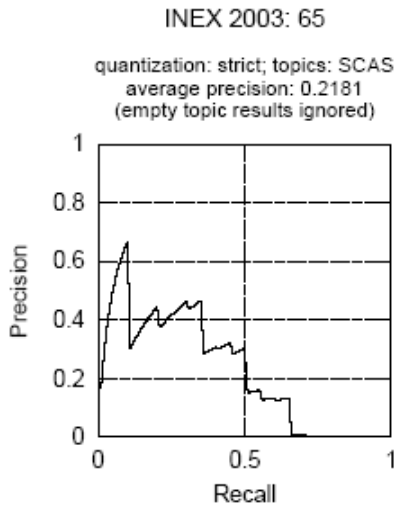


Fig. 29: Non-Flexible Retrieval System

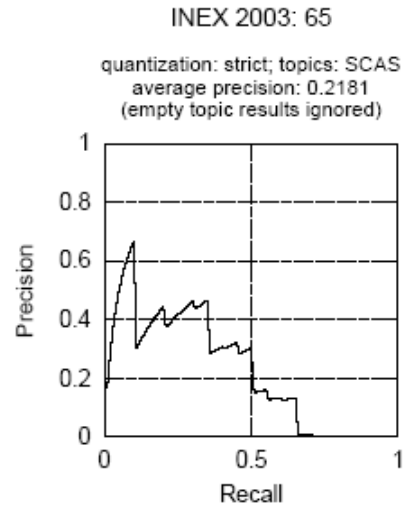


Fig. 30: Flexible Retrieval System

Query 66

Elements to return are sections for this query.

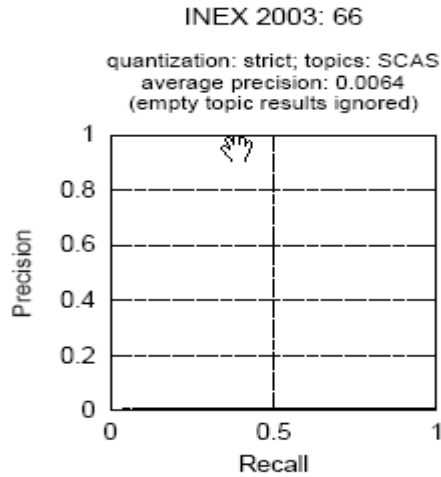


Fig. 31: Non-Flexible Retrieval System

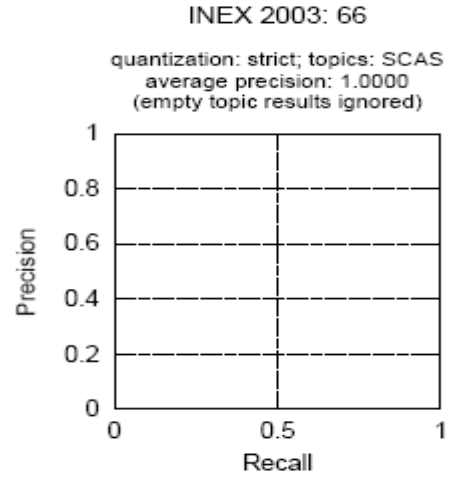


Fig. 32: Flexible Retrieval System

Query 68

Elements to return are sections for this query.

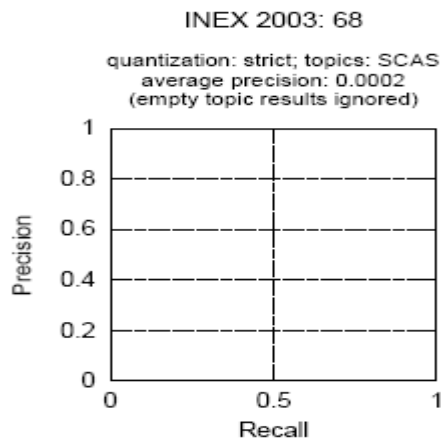


Fig. 33: Non-Flexible Retrieval System

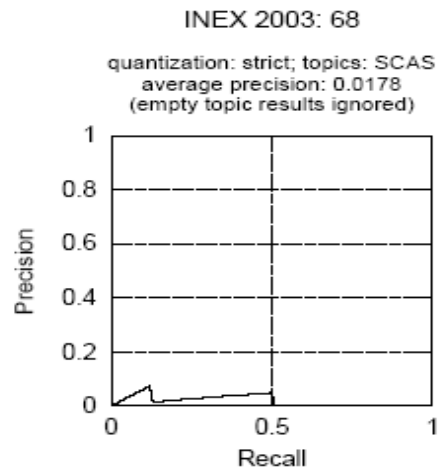


Fig. 34: Flexible Retrieval System

Query 67, 69, 73

There are no relevant elements for topic 67, topic 69 and topic 73.

Query 70

Elements to return are articles for this query.

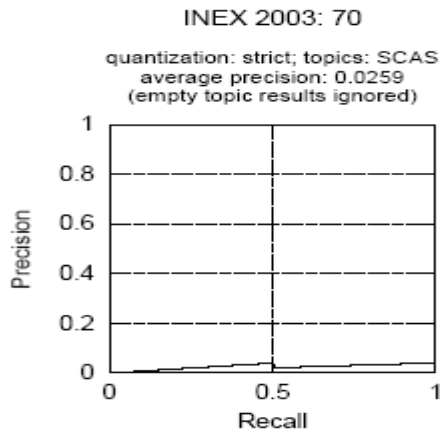


Fig. 35: Non-Flexible Retrieval System

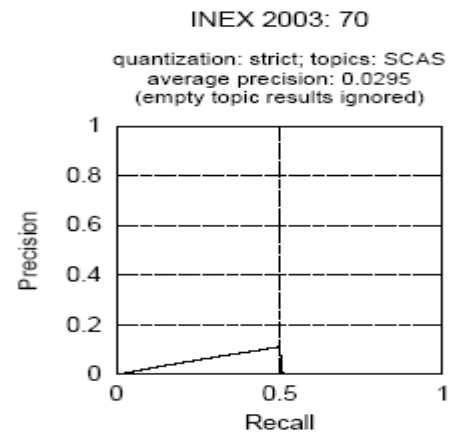


Fig. 36: Flexible Retrieval System

Query 71

Elements to return are parts of body for this query.

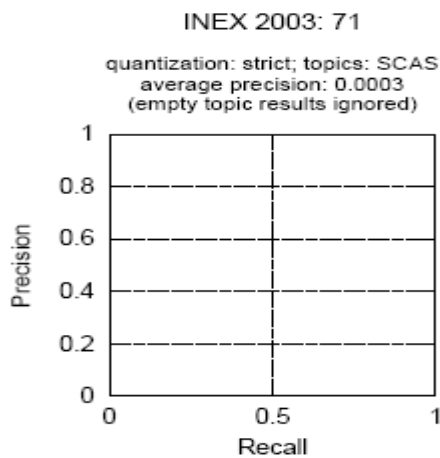


Fig. 37: Non-Flexible Retrieval System

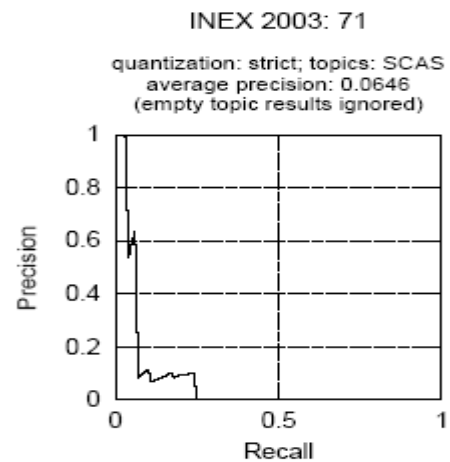


Fig. 38: Flexible Retrieval System

Query 72

Elements to return are parts of body for this query.

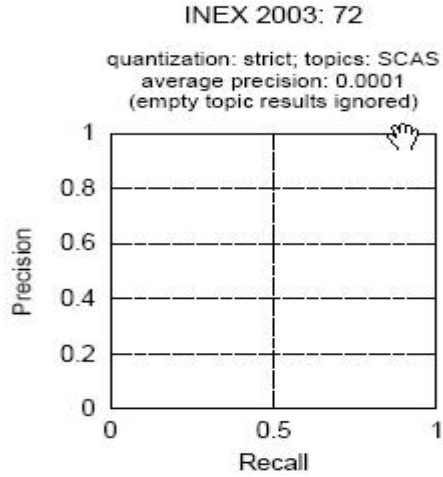


Fig. 39: Non-Flexible Retrieval System

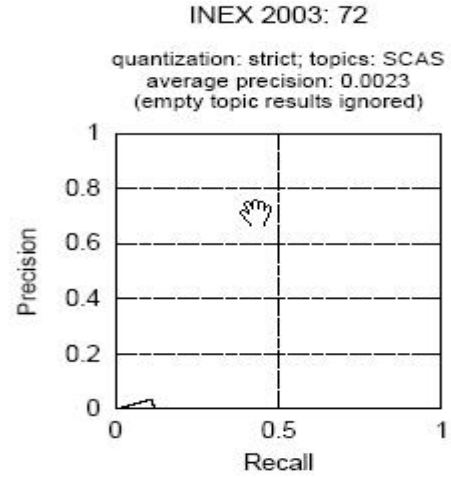


Fig. 40: Flexible Retrieval System

Query 74

Elements to return are sections for this query.

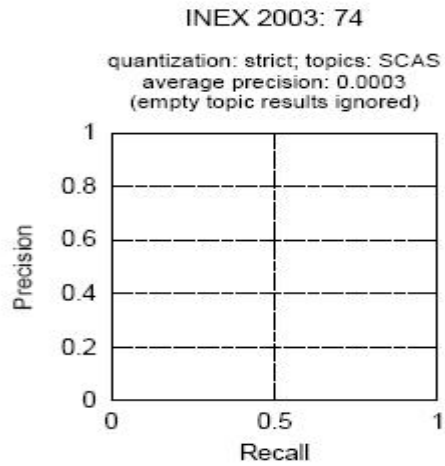


Fig. 41: Non-Flexible Retrieval System

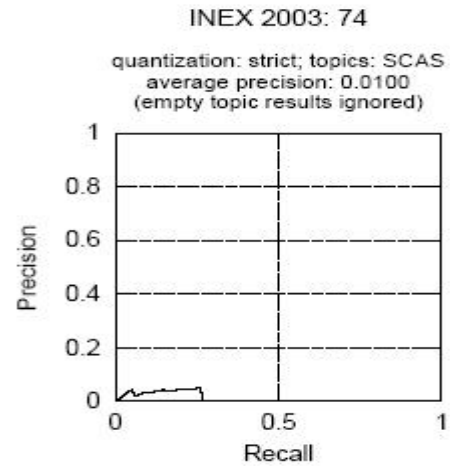


Fig. 42: Flexible Retrieval System

Query 75

Elements to return are articles for this query.

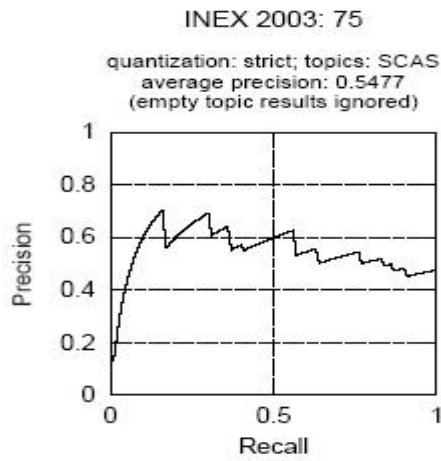


Fig. 43: Non-Flexible Retrieval System

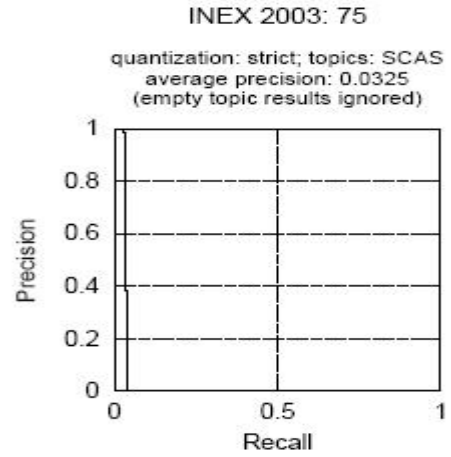


Fig. 44: Flexible Retrieval System

Query 76

There are no relevant elements for topic 76.

Query 77

Elements to return are sections for this query.

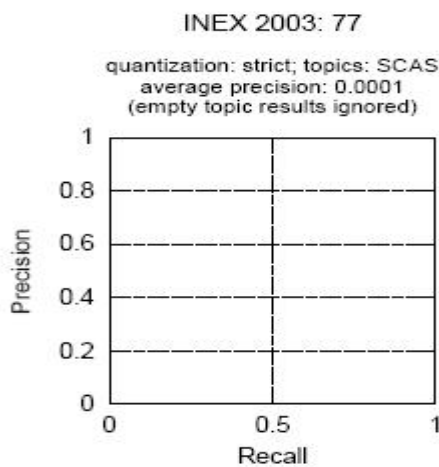


Fig. 45: Non-Flexible Retrieval System

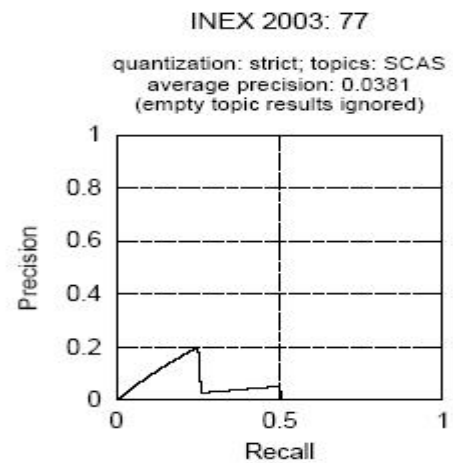


Fig. 46: Flexible Retrieval System

Query 78

Elements to return are vitae for this query.

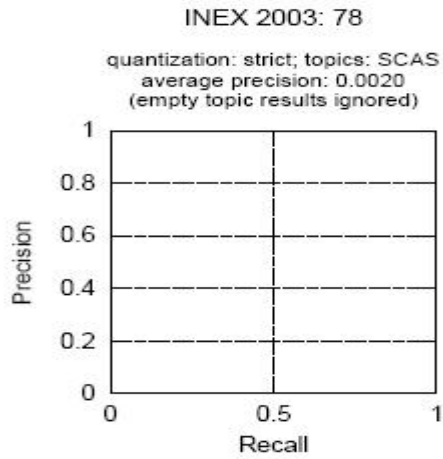


Fig. 47: Non-Flexible Retrieval System

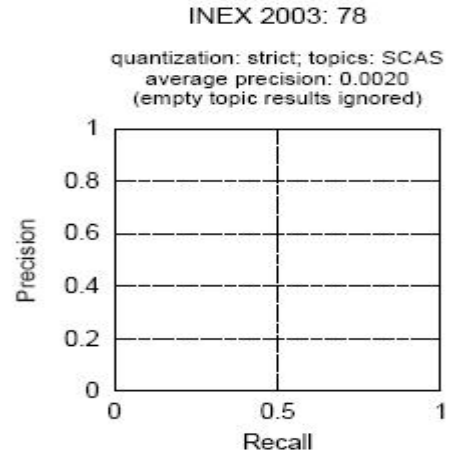


Fig. 48: Flexible Retrieval System

Query 79

Elements to return are articles for this query.

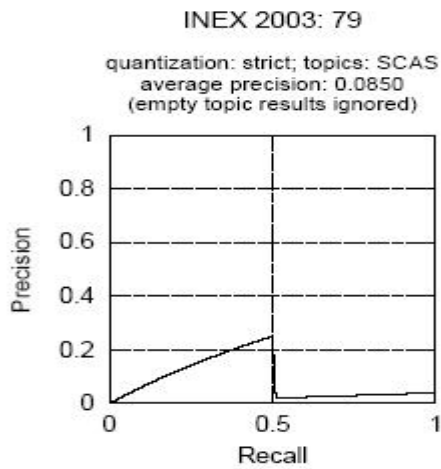


Fig. 49: Non-Flexible Retrieval System

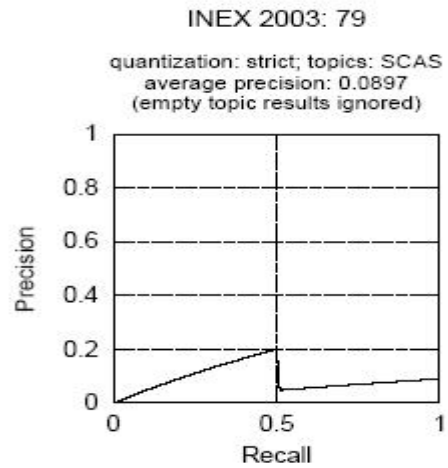


Fig. 50: Flexible Retrieval System

Query 80

Elements to return are sections for this query.

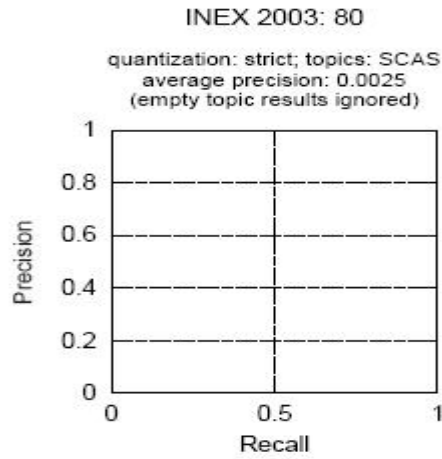


Fig. 51: Non-Flexible Retrieval System

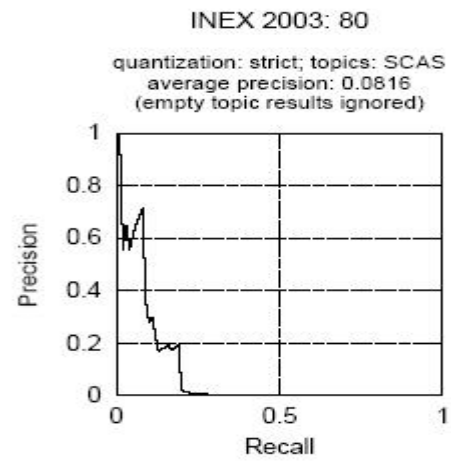


Fig. 52: Flexible Retrieval System

Query 81

Elements to return are articles for this query.

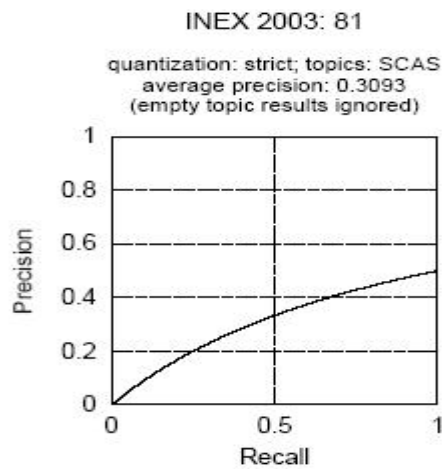


Fig. 53: Non-Flexible Retrieval System

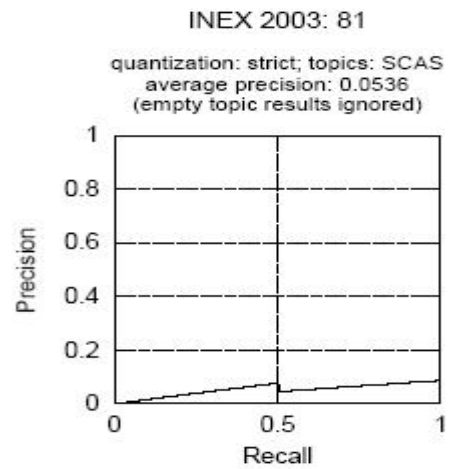


Fig. 54: Flexible Retrieval System

Query 82

Elements to return are articles for this query.

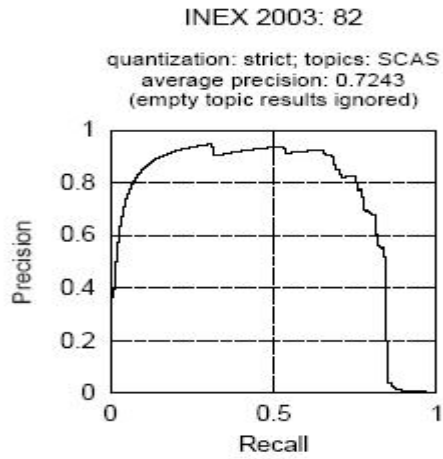


Fig. 55: Non-Flexible Retrieval System

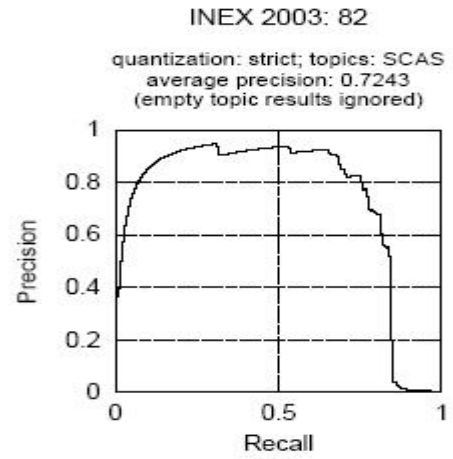


Fig. 56: Flexible Retrieval System

Query 83

Elements to return are abstracts for this query.

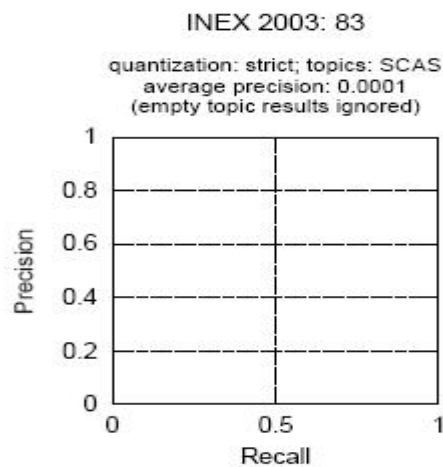


Fig. 57: Non-Flexible Retrieval System

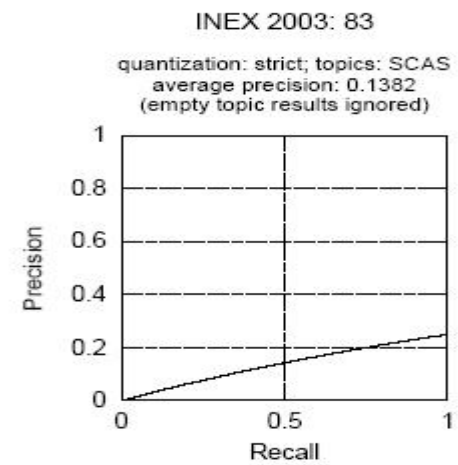


Fig. 58: Flexible Retrieval System

Query 84

Elements to return are paragraphs for this query.

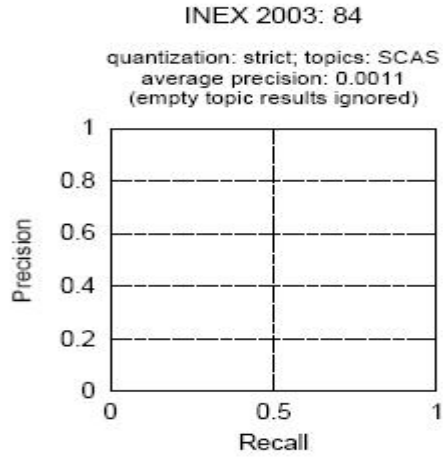


Fig. 59: Non-Flexible Retrieval System

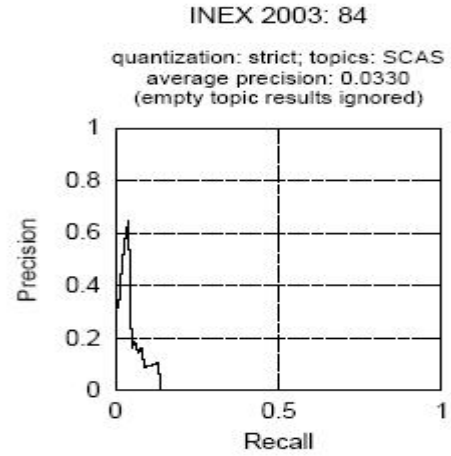


Fig. 60: Flexible Retrieval System

Query 85

Elements to return are sections for this query.

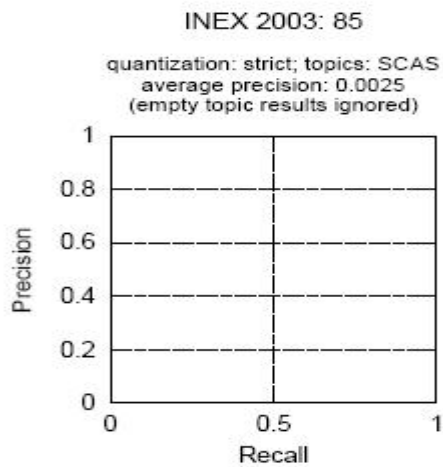


Fig. 61: Non-Flexible Retrieval System

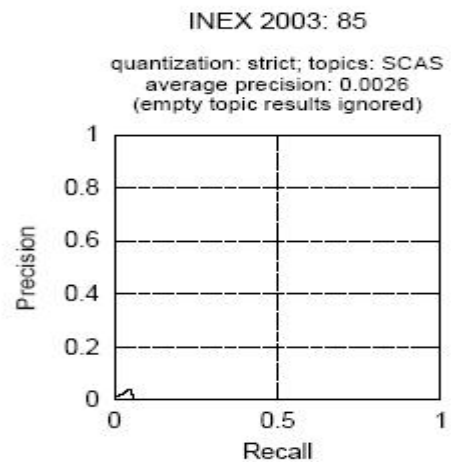


Fig. 62: Flexible Retrieval System

Query 86

Elements to return are sections for this query.

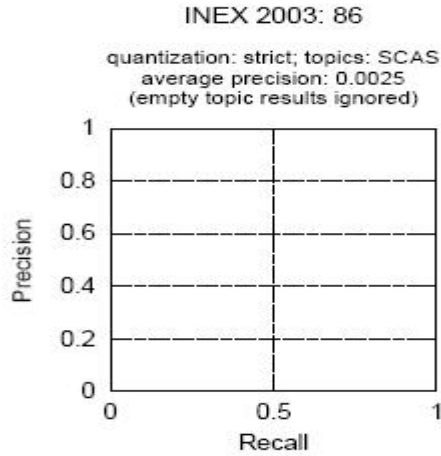


Fig. 63: Non-Flexible Retrieval System

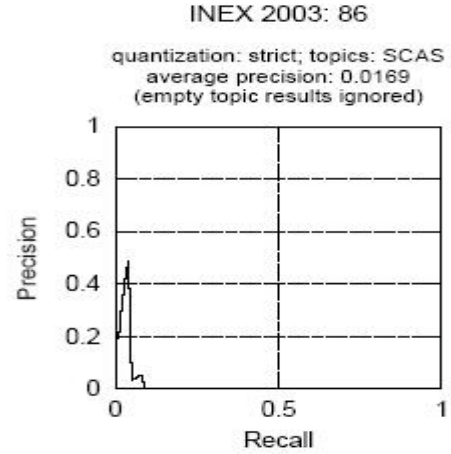


Fig. 64: Flexible Retrieval System

Query 87

Elements to return are articles for this query.

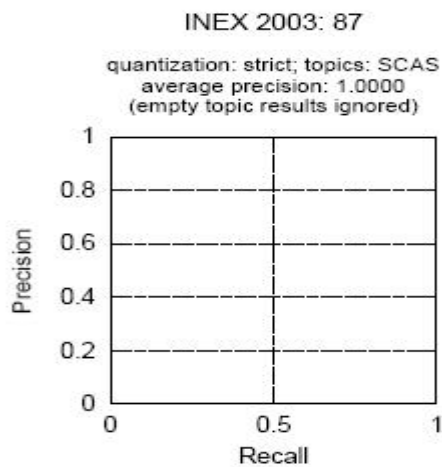


Fig. 65: Non-Flexible Retrieval System

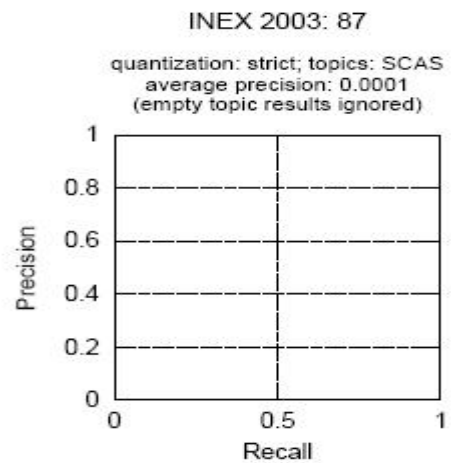


Fig. 66: Flexible Retrieval System

Query 88

Elements to return are articles for this query.

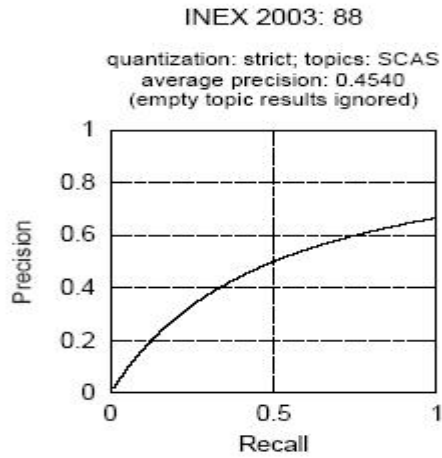


Fig. 67: Non-Flexible Retrieval System

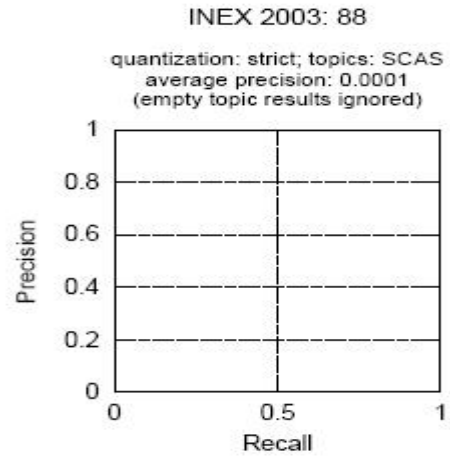


Fig. 68: Flexible Retrieval System

Query 89

Elements to return are bibliographical elements for this query.

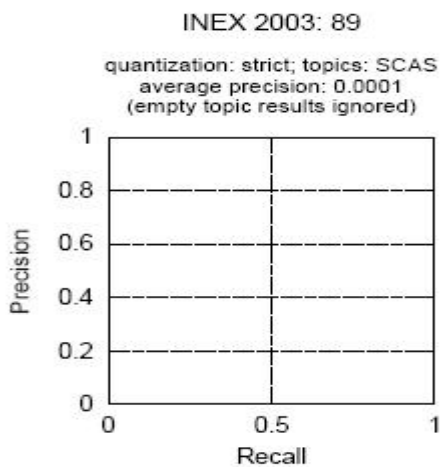


Fig. 69: Non-Flexible Retrieval System

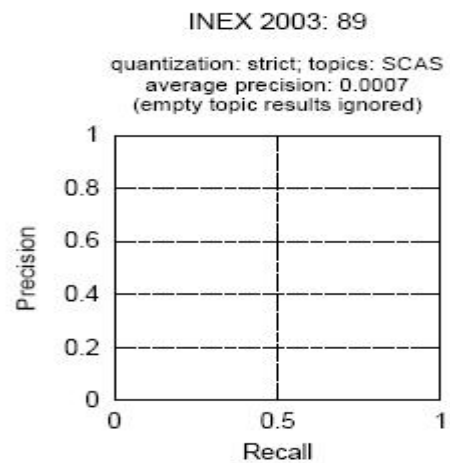


Fig. 70: Flexible Retrieval System

Query 90

Elements to return are abstracts for this query.

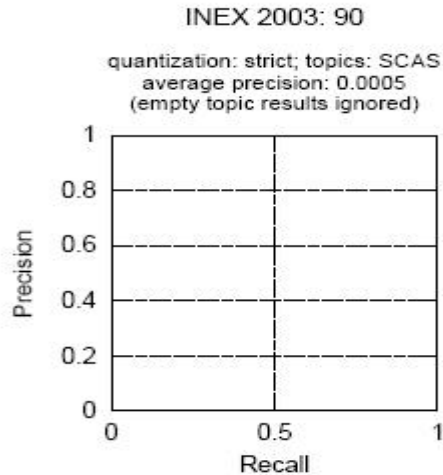


Fig. 71: Non-Flexible Retrieval System

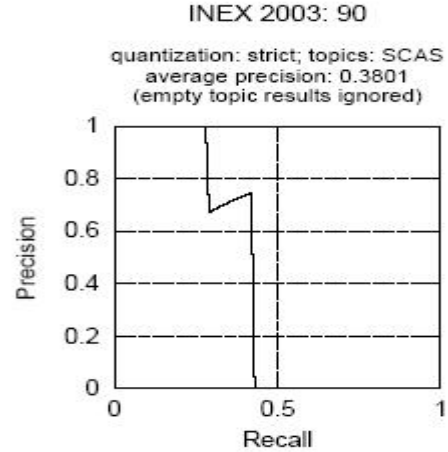


Fig. 72: Flexible Retrieval System

When we compare the two graphs for all queries, i.e., results using non-flexible system and results using flexible system, we observe the difference in results between both systems. The precision values for all the queries in both the systems, along with the element to be returned for each query are tabulated in Table 2. Table 2 does not show entries for topics with no relevant elements. We obtain a single graph for all the CAS queries to observe the overall results. Figure 15 (See Chapter 3) shows the graph obtained by results of all the 30 CAS queries together using non-flexible retrieval system, Figure 16 (See chapter 3) shows the graph for the flexible retrieval system and Figure 17 (See chapter 3) shows the graph with hybrid results. Hybrid results are the results obtained by using non-flexible retrieval system when *article* is the element to be returned and flexible retrieval system when other elements are to be returned for a query. We observe the improvement in results using a hybrid system.

Table 2: Comparison of Flexible and Non-Flexible Systems for INEX 2003 Queries

Query Number	Precision of Non-Flexible System	Precision of Flexible System	Element to be Returned
62	0.1385	0.1195	Article
63	1.0000	0.0312	Article
64	0.0002	0.7087	Section
65	0.2181	0.2181	Article
66	0.0064	1.0000	Section
68	0.0002	0.0178	Section
70	0.0259	0.0295	Article
71	0.0003	0.0646	Part of Body
72	0.0001	0.0023	Part of Body
74	0.0003	0.0100	Section
75	0.5477	0.0325	Article
77	0.0001	0.0381	Section
78	0.0020	0.0020	Vitae
79	0.0850	0.0897	Article
80	0.0025	0.0816	Section
81	0.3093	0.0536	Article
82	0.7243	0.7243	Article
83	0.0001	0.1382	Abstract
84	0.0011	0.0330	Paragraph
85	0.0025	0.0026	Section
86	0.0025	0.0169	Section
87	1.0000	0.0001	Article
88	0.4540	0.0001	Article
89	0.0001	0.0007	Bibliography
90	0.0005	0.3801	Abstract

8 Appendix 3

8.1 Comparison of our Results with other INEX participants (2003)

We observe that we do not stand within top 10 results with our old system (precision is 0.1658) for CAS query evaluation. However, we figure at rank 6 with our hybrid system that uses both our old and new systems (precision is 0.2224) as illustrated in Table 4.

Table 3: Top 10 Results for INEX 2003 CAS Queries with old system

Rank	Participant Name	Precision Value
1	Language and Inference Technology Group, ILLC, U. of Amsterdam	0.2989
2	Language and Inference Technology Group, ILLC, U. of Amsterdam	0.2456
3	Language and Inference Technology Group, ILLC, U. of Amsterdam	0.2451
4	IBM, Haifa Research Lab	0.2399
5	IBM, Haifa Research Lab	0.2378
6	IBM, Haifa Research Lab	0.2222
7	University of Twente and CWI	0.2212
8	Queensland University of Technology	0.2050
9	Universität Duisburg-Essen	0.1934
10	Queensland University of Technology	0.1893

Table 4: Top 10 Results for INEX 2003 CAS Queries with hybrid system

Rank	Participant	Precision
1	Language and Inference Technology Group, ILLC, U. of Amsterdam	0.2989
2	Language and Inference Technology Group, ILLC, U. of Amsterdam	0.2456
3	Language and Inference Technology Group, ILLC, U. of Amsterdam	0.2451
4	IBM, Haifa Research Lab	0.2399
5	IBM, Haifa Research Lab	0.2378
6	University of Minnesota, Duluth	0.2224
7	IBM, Haifa Research Lab	0.2222
8	University of Twente and CWI	0.2212
9	Queensland University of Technology	0.2050
10	Universität Duisburg-Essen	0.1934

s