

UNIVERSITY OF MINNESOTA

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

Salil G. Bapat

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.

Name of Faculty Adviser

Signature of Faculty Adviser

Date

GRADUATE SCHOOL

Improving Results for Focused and Relevance-in- context tasks

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

Salil G. Bapat

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

August, 2008

Department of Computer Science
University of Minnesota, Duluth
Duluth, MN 55812
USA

Acknowledgements

I would like to take this opportunity to thank several people who have contributed towards the successful completion of this thesis.

I would like to thank Dr. Carolyn Crouch, for providing me an opportunity to work with her, her timely advice, and guidance throughout this thesis.

I would like to thank Dr. Donald Crouch, for his suggestions and feedback during the important stages of my thesis work.

I would also like to thank Darshan Paranjape and Sarika Mehta for being supportive co-workers as well as invaluable critics of my work.

I would like to thank Aditya Mone, Nachiket Kamat and Vikram Malik for sharing their experience and helping me on various aspects of this thesis.

I would like to thank the faculty and staff of the Department of Computer Science at UMD, especially Jim Luttinen, Dr. Pete Willemsen, Lori Lucia and Linda Meek.

Last but not the least, thanks to friends at UMD for the support and encouragement they provided during the two years of this degree. Thank you for all your blessing and wishes.

Abstract

Information retrieval focuses on organizing document-related data and retrieving useful information from it. Electronic documents on web can be structured, unstructured or semi-structured. Traditionally, information retrieval was concentrating on retrieving information from unstructured documents. But with the advent of the web and XML as a preferred standard for storing documents, the center of attention is now changing to retrieve information from structured and semi-structured documents. In such documents, the elements of the document at various levels of granularity can be retrieved instead of the document as a whole. This type of retrieval, for which elements of the document at various level of granularity are retrieved, is known as the flexible retrieval [1].

The main aim of this thesis is improved performance for the INEX 2007 Ad hoc Focused and Relevant-in-Context tasks. The Focused task requires systems to find the most specific (or focused) results that satisfy the information need. The Relevant-in-Context task requires systems to find the focused results that correspond to the relevant element in each relevant article. In this thesis, we discuss and evaluate various strategies to improve the performance of the Focused and Relevant-in-Context tasks while maintaining flexible retrieval as the backbone of the system.

Table of Contents

LIST OF FIGURES	IV
LIST OF TABLES	V
1. INTRODUCTION.....	1
2. OVERVIEW.....	3
2.1 INEX.....	3
2.2 DOCUMENT COLLECTION.....	3
2.3 TOPIC COLLECTION.....	4
2.4 RELEVANCE ASSESSMENTS.....	6
2.5 2007-2008 RETRIEVAL TASKS.....	8
2.6 EVALUATION MEASURES.....	9
2.7 VECTOR SPACE MODEL.....	11
2.8 SMART RETRIEVAL ENGINE.....	11
3. THE FLEXIBLE RETRIEVAL SYSTEM.....	12
3.1 OVERVIEW OF FLEXIBLE RETRIEVAL.....	12
3.2 UNTAGGED TEXT.....	15
3.3 PRE-FLEX: NORMAL SMART RETRIEVAL.....	16
3.4 FLEX: FLEXIBLE RETRIEVAL.....	18
3.5 POST-FLEX.....	19
4. EXPERIMENTS, ANALYSIS AND OBSERVATIONS.....	24
4.1 TAGS USED.....	24
4.2 RELEVANT-IN-CONTEXT EXPERIMENTS.....	27
4.3 EXPERIMENTS FOR FOCUSED TASK.....	34
4.4 BEST-IN-CONTEXT EXPERIMENTS.....	37
5. FUTURE WORK.....	38
REFERENCES.....	39
APPENDIX A.....	41
A.1 IMPLEMENTATION DETAILS OF THE NORMAL FLEXIBLE RETRIEVAL PROCESS.....	41
A.2 STEPS TO INCORPORATE ARTICLE RETRIEVAL INTO ELEMENT RETRIEVAL.....	51

List of Figures

Fig 2.1: Sample Wikipedia document	4
Fig 2.2: A sample 2007 query (Topic id = 493)	6
Fig 2.3: Sample relevance assessment result file [Topic id = 543]	8
Fig 3.1: All- element retrieval	13
Fig 3.2 (a): Normal Smart retrieval on terminal node set (Pre – Flex)	14
Fig 3.2 (b): Flex retrieval	14
Fig 3.2 (c): Post-Flex	15
Fig 3.3: Lnu element term weighting formula	17
Fig 3.4: Ltu query term weighting formula	18
Fig 3.5: Sample Wikipedia document in its original format (article id 2614489) ..	21
Fig 3.6: Representation of the XML document after parsing (article id 2614489) ..	22
Fig A.1: Configuration file used for generating the doc-trees	40
Fig A.2: Parameters in configuration file used by Flex Driver.....	45
Fig A.3: Configuration file used for XPath expansion.....	46

List of Tables

Table 2.1: INEX 2007 Topic Components	5
Table 4.1: Tags identified as terminal nodes for INEX 2006 tasks	24
Table 4.2: Tags recognized during the flexible retrieval process	25
Table 4.3: Slope and Pivot values for INEX 2006 tasks	25
Table 4.4: Tags identified as terminal nodes for INEX 2007 tasks	26
Table 4.5: Tags recognized during the flexible retrieval process	26
Table 4.6: Slope and Pivot values for INEX 2007 tasks	27
Table 4.7: Flex + Article retrieval results for RIC task (old tag set)	28
Table 4.8 (a): Flex + Article retrieval results for RIC task (new tag set and incomplete trees)	29
Table 4.8 (b): Flex + Article retrieval results for RIC task (new tag set and complete trees)	30
Table 4.9: Results for INEX 2007 Relevant-in-Context task by incorporating article retrieval with flexible retrieval	31
Table 4.10: Results for INEX 2007 RIC task by using only the flexible retrieval ...	32
Table 4.11: Results for INEX 2007 RIC task using only article retrieval strategy ..	33
Table 4.12: Results for Focused task using old tag set	35
Table 4.13: Results for Focused task using new tag set	36
Table 4.14: Results for INEX 2007 Focused task by incorporating article retrieval with Flex (new terminal node set)	37

1. Introduction

Information retrieval focuses on organizing document-related data and retrieving useful information from it. The user fires a query in connection to his need for useful information and gets back a set of documents satisfying his query. Web search engines such as Google and Lycos are among the most visible applications of information retrieval and their success emphasizes the importance of this research area.

Electronic documents on the web can be structured, unstructured or semi-structured. In a structured document, each and every element in the document conforms to a DTD (Document Type Definition). Examples of structured documents are the IEEE documents used in INEX 2005. Unstructured documents, which have no structural constraints imposed on the data, are exemplified by documents with no XML tags. Semi-structured documents have data whose structure is incomplete or varies unpredictably. Examples of this type are the Wikipedia documents used in INEX 2006.

Traditional information retrieval concentrates on retrieving information from unstructured documents. But with the advent of the web and XML as a preferred standard for storing documents, the focus now extends to retrieving information from structured and semi-structured documents. On the web, we deal with structured or semi-structured data which is represented in XML format using various XML tags. Because of this, the elements of the document at various levels of granularity can be retrieved. We refer to this type of retrieval (where elements at various levels of granularity are retrieved rather than the document as a whole) as flexible retrieval [1]. Thus in flexible retrieval, the system can return results more specific to or tuned to the user's information need. In this thesis, we consider only retrieval from semi-structured XML documents.

INEX [9] runs a competition which promotes the development of XML-based retrieval systems. INEX provides a document collection, a query set (topics) and measures for evaluating the XML-based retrieval systems of the participants. The

University Of Minnesota Duluth (UMD) has been a participant in this competition since its beginning in 2002.

The focus of this thesis is to improve the performance of various Ad-hoc tasks as specified in INEX 2007 and INEX 2008. These tasks are explained in detail in Chapter 2 along with other INEX-related topics. Chapter 3 describes the adaptation of flexible retrieval to semi-structured XML documents, all-element retrieval (which provides a baseline for our system performance) and the flexible retrieval system which has been developed at UMD. Chapter 4 describes the experiments conducted along with results, observations and analysis. Suggestions for future research are provided in Chapter 5.

2. Overview

This chapter provides an overview of INEX in terms of the document and topic collection, relevance assessments, the retrieval tasks and the evaluation measures. It also briefly describes the Vector Space Model, the basic model of this research, and Smart, our retrieval system.

2.1 INEX

INEX [9] is the INitiative for the Evaluation of XML retrieval. It facilitates the development of effective XML-based information retrieval strategies. It was launched in 2002 and since then has provided the infrastructure required for supporting the development of XML retrieval. UMD participates in the Ad hoc task of INEX. In information retrieval literature, ad hoc retrieval is described as a simulation of how a library might be used, and it involves the searching of a static set of documents using a new set of topics [13]. The INEX query may contain both content and structural constraints. The system returns results (in the form of arbitrary XML elements) from the library. The test collection provided by INEX contains a set of XML documents, topics (queries) and relevance assessments.

2.2 Document Collection

Prior to 2006, INEX provided a document collection consisting of IEEE documents. For 2006, the INEX collection is made up of Wikipedia documents written in English. These documents are semi-structured XML documents which contain untagged text along with the normal, tagged text. This collection is approximately 10 times the size of the earlier IEEE collection. It contains 659,388 documents (5.8 GB) and was originally available on www.wikipedia.org. The collection contains 5,000 unique tags [12]. The average number of XML nodes per article is 161.35, and the average depth of an element is 6.72 [9].

Since this document collection is semi-structured, it does not have a DTD. There is no standard hierarchy defined for the data elements, thus making the retrieval task more challenging. A typical Wikipedia document is presented in Figure 2.1.

```
<?xml version="1.0" encoding="utf-8"?>
<article>

  <name id="1442576">
    The Alchemist (producer)
  </name>

  <conversionwarning>
    0
  </conversionwarning>

  <body>
    <emph3>
      The Alchemist
    </emph3>
    , born
    <emph3>
      Alan Maman
    </emph3>
    , is one of East Coast hip hop's leading producers. , he went on to produce for many of hip hop's
    leading artists.|
    <section>
      <title>
        Biography
      </title>

      <p>
      </p>
    </section>

    <p>
    </p>
  </body>
</article>
```

Fig 2.1: Sample Wikipedia document.

A document in the Wikipedia collection typically consists of an article identifier, title and body. Each article consists of identifiable elements.

2.3 Topic Collection

Prior to 2006, INEX topic specification consisted of two types of topics, namely, Content-Only (CO) and Content and Structure (CAS). In CO topics, the focus was only

on the information need and the document structure was ignored, whereas CAS topics had explicit references to the XML structure of the document. In 2006, these topic types were merged into a single type, named Content-Only + Structure (CO+S) topics. At the beginning of the year, each participating group is asked to create a set of candidate topics which are representative of a range of real user needs over the XML collection. These topics are then assessed by the participants. The 2007 CO+S topic set has a total of 127 CO+S topics. Table 2.2 gives an overview of various components of the 2007 topics [10].

Component	Description
<i><title></i>	Defines the CO queries
<i><castitle></i>	Defines the CAS queries
<i><description></i>	One or two sentence natural language definition of the information need
<i><narrative></i>	Gives the definition of relevance and irrelevance with respect to the query

Table 2.1: INEX 2007 Topic Components

We use the *<title>* part for CO tasks and the *<castitle>* part for CAS tasks. Figure 2.2 shows a typical topic from the INEX 2007 topic collection.

```

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

<!DOCTYPE inex_topic SYSTEM "topic.dtd">

<inex_topic topic_id="493" ct_no="143">

<title>amphibians black frog forest</title>

<castitle>
/**[about(., amphibians frog) and about(., black forest)]
</castitle>

<description>
Find species of black frogs that live in forests
</description>

<narrative>
The relevant pieces of document must quote one or several species of frog with
black skin that live in forests. The document must mention all this information.
</narrative>

</inex_topic>

```

Fig 2.2: A sample 2007 query (Topic id = 493)

2.4 Relevance Assessments

In INEX, relevance is described in terms of the extent to which any component of a document focuses on the requested topic. Relevance assessments are produced by the manual assessment of a subset of the document collection against the topic. Each participating group is given about three topics to assess. In most of the cases, the topics are those which that group originally submitted. After all the participating groups finish their assessments, the assessments are pooled to form the set of all documents found relevant to a query by the human assessors. This set is considered the ideal set and the results produced by the retrieval system are assessed against this set.

For performing the assessment task, INEX provides an online assessment system called XRAI (XML Retrieval Assessment Interface) [8]. Assessors highlight only text

fragments that contain relevant information. The assessor may highlight a complete element or just a part of an element as relevant. Any document containing a relevant element is considered to be relevant. Besides highlighting relevant elements, the assessor also provides a best entry point (BEP) for each relevant document. This is the point in the document where the reader should begin reading.

Prior to INEX 2005, relevance was defined relative to two dimensions: specificity (the extent to which the document component focuses on the topic) and exhaustivity (the extent to which the document component discusses the topic). In 2006, exhaustivity was dropped. The main advantage of dropping exhaustivity is that the specificity of any partially highlighted elements can be calculated automatically as a function of the contained relevant and irrelevant content (e.g., in the simplest case, as the ratio of relevant content to all content, measured in number of words or characters) [8]. Figure 2.3 shows a sample relevance assessment file with two relevant articles for the given query. XPath paths of elements within these articles are given in the <element path> tag. The XPath of the best entry point for the document is given in the <best-entry-point path> tag.

```

<?xml version="1.0"?>
<!DOCTYPE assessments SYSTEM "../assessments.dtd">
<assessments pool="314" topic="543" version="2" whitespace-text-node="keep">

<!-- Topic definition -->
<inex_topic topic_id="543" ct_no="157">
<title>Tux</title>
<castitle>//figure[about(../image, tux)]</castitle>
<description>Find images of Tux.</description>
<narrative>Tux is penguin, being a Linux mascot since 1996. It was created by Larry Ewing.</narrative>
</inex_topic>

<!-- Topic assessments (only completed files) -->

<file collection="wikipedia" name="1044465">
  <best-entry-point path="/article[1]/body[1]/figure[1]/image[1]" />
  <passage start="/article[1]/body[1]/figure[1]/image[1]" end="/article[1]/body[1]/figure[1]/image[1]" size="12"/>
  <element path="/article[1]/body[1]/figure[1]/image[1]" exhaustivity="2" size="12" rsize="12"/>
  <element path="/article[1]/body[1]/figure[1]" exhaustivity="2" size="95" rsize="12"/>
  <element path="/article[1]/body[1]" exhaustivity="2" size="1696" rsize="12"/>
  <element path="/article[1]" exhaustivity="2" size="1741" rsize="12"/>
</file>

<file collection="wikipedia" name="18510">
  <best-entry-point path="/article[1]/body[1]/figure[1]/image[1]" />
  <passage start="/article[1]/body[1]/figure[1]/image[1]" end="/article[1]/body[1]/figure[1]/image[1]" size="9"/>
  <element path="/article[1]/body[1]/figure[1]/image[1]" exhaustivity="2" size="9" rsize="9"/>
  <element path="/article[1]/body[1]/figure[1]" exhaustivity="2" size="57" rsize="9"/>
  <element path="/article[1]/body[1]" exhaustivity="2" size="20413" rsize="9"/>
  <element path="/article[1]" exhaustivity="2" size="20429" rsize="9"/>
</file>
</assessments>

```

Fig 2.3: Sample relevance assessment result file [Topic id = 543]

2.5 2007-2008 Retrieval Tasks

The objective of the Ad hoc retrieval track of INEX is to retrieve relevant elements. The 2007 Ad hoc retrieval track is comprised of following tasks:

Focused Task: This task requires the user to return a ranked list of elements or passages for each topic. The Focused Task requires systems to find the most focused results that

satisfy an information need [2]. There should be no overlapping elements in the result returned. This means that for any element present in the result list, no other element along that path in the XML tree of the parent document can be returned.

Relevant in Context Task: This task requires the user to return a ranked list of articles and within those articles the relevant information (captured by a set of non-overlapping elements or passages). A relevant article may contain relevant information spread across different elements. The task requires systems to find a set of results that corresponds well to all relevant information in each relevant article [2].

Best in Context Task: This task requires the user to return a ranked list of articles and to identify the best-entry-point associated with each such article. Each article will have only one best entry point (even if that is the beginning of the article) [2].

Prior to 2007, the Ad-hoc task set also included the *Thorough* task. This task required the user to return a ranked list of XML elements in descending order of relevance. All the non-zero correlating elements along a path were included in the output set. The goal was to test a retrieval system's effectiveness in producing a correct ranking. Results for the Thorough task can contain overlapping elements. This task was dropped in 2007. The 2008 Ad hoc tasks are identical to those of 2007.

2.6 Evaluation Measures

The INEX 2007 evaluation measures are completely based on the retrieval of highlighted text. All the INEX tasks are simplified to highlighted text retrieval, and the assumption is made that the system will return all, and only, highlighted text. The evaluation measure compares the characters of text retrieved by a search engine to the number and location of characters of text identified as relevant by the assessor. The evaluation measure for each task follows.

Focused Task: For this task, the fraction of all highlighted text that has been retrieved forms the Recall. The fraction of retrieved text that was highlighted forms the precision. A measure is used to calculate interpolated precision at selected recall levels. INEX is most interested in what happens in the highest ranked retrieved results, and thus the INEX 2007 official measure is interpolated precision at 1% recall (iP[0.01]). The evaluation measure also provides interpolated precision at other recall points, and (mean average) interpolated precision (MAiP) over 101 standard recall points (0.00, 0.01, 0.02, ..., 1.00) as an overall measure.

Relevant in Context Task: The evaluation of the Relevant in Context Task is based on the measures of generalized precision and recall, where the per document score reflects how well the retrieved text matches the relevant text in the document. Specifically, the per document score is the harmonic mean of precision and recall in terms of the fractions of retrieved and highlighted text in the document [2]. INEX is most interested in overall performance so the official measure is mean average generalized precision (MAgP). The evaluation measure also provides the generalized precision scores at early ranks (5, 10, 25, 50).

Best in Context Task: The evaluation of the Best in Context Task is based on the measures of generalized precision and recall where the per document score reflects how well the retrieved entry point matches the best entry point in the document. Specifically, the per document score is a linear discounting function of the distance d (measured in characters, namely, $(n - d(x,b)) / n$ for $d < n$ and 0 (otherwise INEX uses $n = 1,000$ which is roughly the number of characters corresponding to the visible part of the document on a screen) [2]. INEX is most interested in overall performance, and the official measure is mean average generalized precision (MAgP). The evaluation measure also provides the generalized precision scores at early ranks (5, 10, 25, 50).

2.7 Vector Space Model

The Vector Space Model [15] is the basic model for this research. In this model, each document and query is represented as an n-dimensional vector of unique terms. These terms are weighted based on their frequency within the document. The distance between two vectors in vector space gives the correlation of a document to a query. Cosine similarity is one of the measures used to compute distance.

2.8 Smart Retrieval Engine

Smart 13.0 [14] is our basic retrieval engine. Smart is used to index the collection and to retrieve. It also provides a utility called *smart-eval* which can be use to evaluate retrieval runs in terms of precision and recall. Smart produces the following files:

textloc file (information about the physical location of the documents being indexed), *dictionary* (information about the terms in the documents and the document frequency of each term), *document vectors*, and *an inverted file* (for each term, a list of the documents vectors that contain that term).

3. The Flexible Retrieval System

This chapter presents the procedures of flexible retrieval and all-element retrieval. It also includes a detailed explanation of all the pre-processing (parsing and indexing) and post-processing (output-filtering) steps of flexible retrieval.

3.1 Overview of Flexible Retrieval

Flexible retrieval refers to the dynamic retrieval of elements in a structured environment [1]. Flexible retrieval was designed to handle structured documents but, current research extends these methods to semi-structured documents. Earlier experiments with flexible retrieval are documented in [3, 6, 7, 10, 12]. Flexible retrieval requires a base case for evaluation. An all-element index is used to generate this base case. All-element retrieval is represented in Figure 3.1 and the procedure of flexible retrieval is represented in Figure 3.2a, 3.2b, 3.2c.

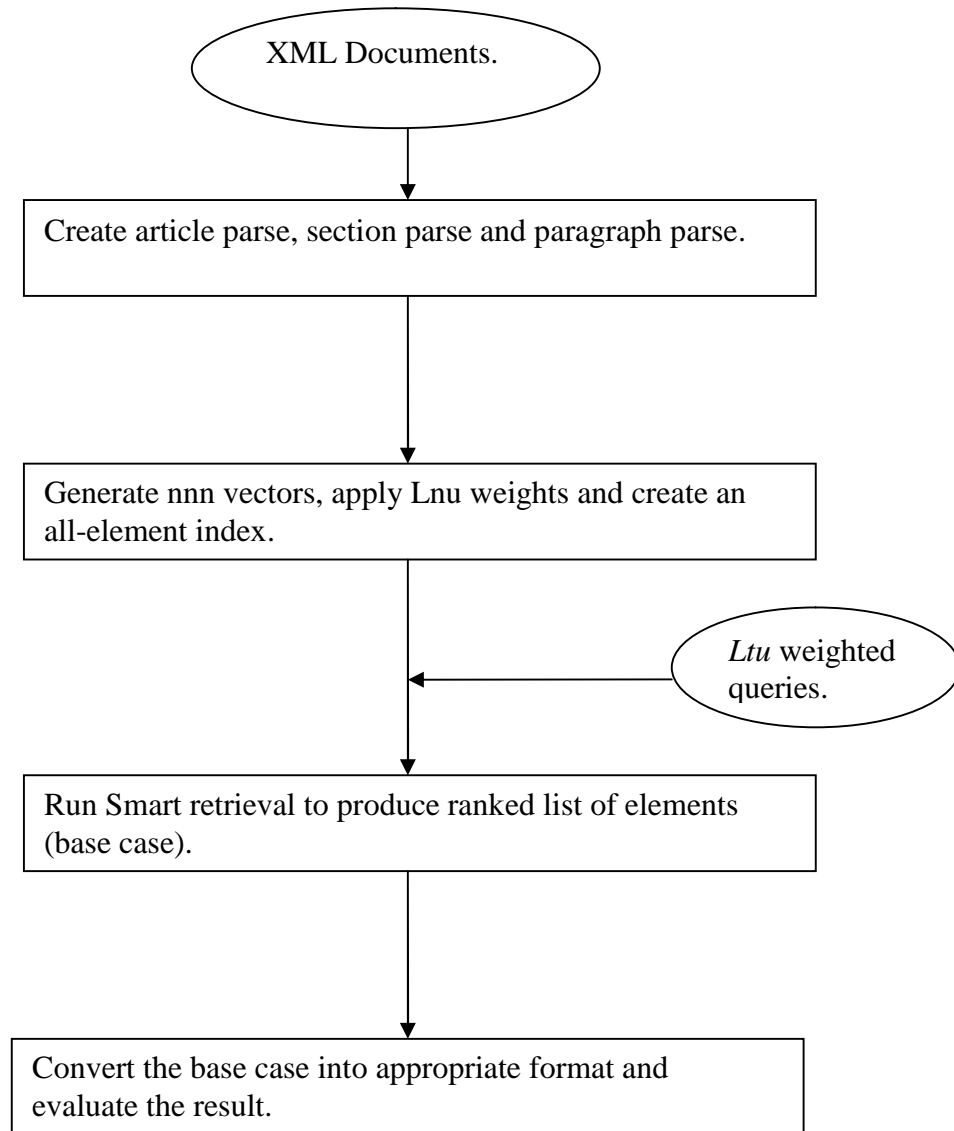


Fig 3.1: All- element retrieval

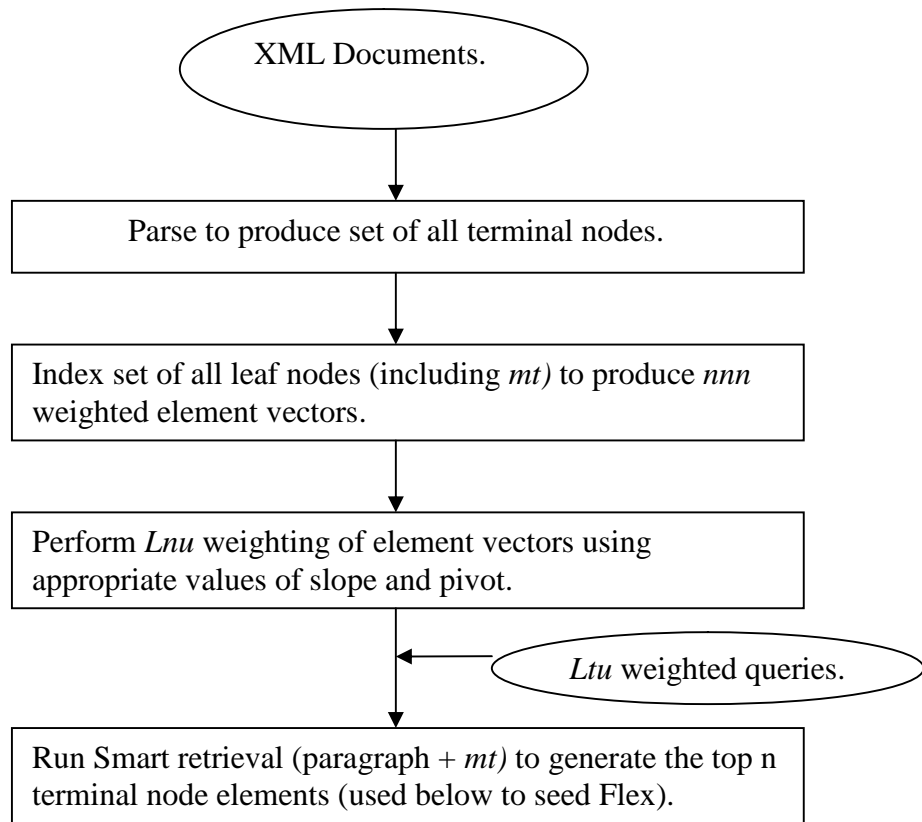


Fig 3.2 (a): Normal Smart retrieval on terminal node set (Pre – Flex)

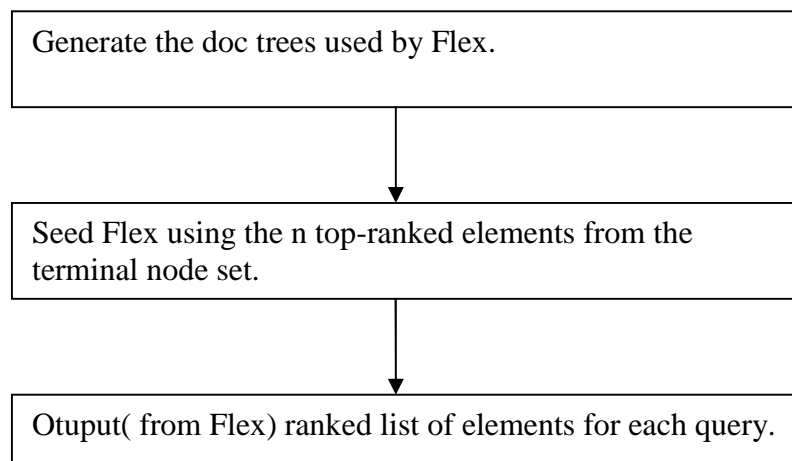


Fig 3.2 (b): Flex retrieval

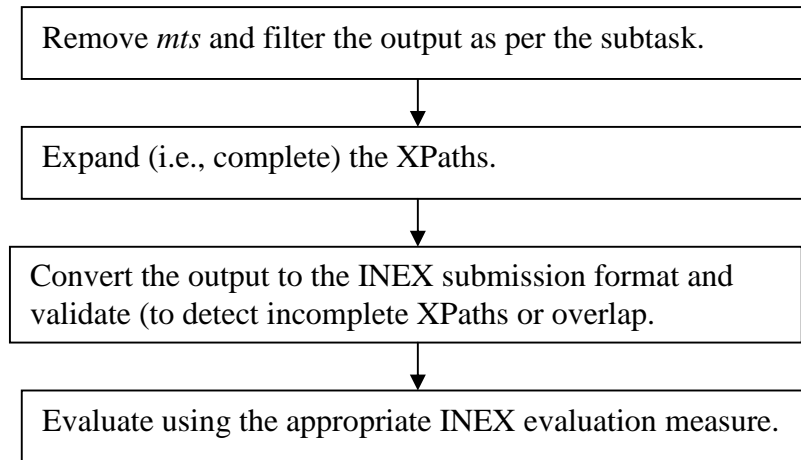


Fig 3.2 (c): Post-Flex

3.2 Untagged Text

Wikipedia documents are semi-structured and contain pieces of text not enclosed within any tags. We call this untagged text. Flex performs a vector addition of leaf nodes to build a complete document tree in a bottom-up manner. All leaf nodes must be identified before the parent can be built. Untagged text causes problems in building a parent node since it does not exist as an identifiable terminal node and hence cannot be used in building the tree. We handle this problem by pre-processing a Wikipedia document as follows:

- (1) Identify all pieces of untagged text.
- (2) Group them as a single element within the parent.
- (3) Tag this text with a special marker (we refer to this tag as *mt* or *magic text*).

All the untagged text now belongs to this new *mt* tag and the document trees can be correctly built. See [6] for a discussion of the importance of untagged text.

3.3 *Pre-Flex: Normal Smart retrieval*

This section describes each step for normal Smart retrieval (Figures 3.1 and 3.2(a)).

Parsing

The Wikipedia document collection provided by INEX contains one parent folder. Within this parent folder are 22 sub-folders. Each folder contains, on average, around 30,000 documents. In the parsing step, all these documents are parsed and four parses are generated, namely, the article, section, paragraph, and paragraph + *mt* parse.

The *article parse* creates a single element per article. The text in the document is retained and all tags except *article* and *body* are ignored. This parse is a component of all-element retrieval.

The *section parse* generates all section-level elements. Every element delimited by section corresponds to a single entity. The elements generated by this parse are also used in all-element retrieval.

The *paragraph parse* generates all the leaf nodes. For the 2007 Ad hoc tasks, we have identified a new set of specific leaf nodes, i.e., *p*, *figure*, *emph3*, *name*. (This set differs from the set previously used. See Chapter 4 for details.) Once a leaf node tag is recognized in the XML hierarchy, all tags contained within it are removed and the text is aggregated to form the leaf node. This parse is used in all-element retrieval.

In the *paragraph + mt* parse, the *mt* tag is also considered as a leaf node (along with the aforementioned leaf nodes). This parse is used to generate the *paragraph + mt* element set; (the elements are indexed, retrieval is performed and top n elements retrieved are then used to seed Flex).

Indexing

After parsing, the following indices are generated using Smart 13.0:

- 1) All-element index: This uses the article, section and paragraph parse as its input set. There is a high level of redundancy present since overlapping elements are treated as independent documents.

- 2) Article index: This uses the article parse as input. There is no redundancy present in this set.
- 3) Paragraph + mt index: This index uses the *paragraph* + *mt* parse elements as input. There is no redundancy present in this index. It is used for flexible retrieval.
- 4) Index query: The title part of each query is also indexed to produce the indexed queries.

Weighting Scheme

Traditional weighting schemes have a bias towards longer elements with more terms and terms with higher frequency. In our research, we address this issue by using the *Lnu-ltu* weighting scheme. [7] The formula for *Lnu* element term weighting is given in Figure 3.3, and the formula for *ltu* query term weighting is given in Figure 3.4. The *Lnu-ltu* weighting scheme depends on two parameters, namely *slope* and *pivot*. These values are determined empirically.

$$\frac{\frac{1 + \log(\text{tf})}{1 + \log(\text{average tf})}}{(1 - \text{slope}) + \text{slope} * (\# \text{ unique terms})/\text{pivot}}$$

where, tf is the term frequency (as in the *nnn* vector)
average tf is the average term frequency of all terms in this vector
unique terms is the number of distinct terms in this vector
slope & pivot are empirically determined constants

Fig 3.3: *Lnu* element term weighting formula [10]

$$\frac{1 + \log(\text{tf}) * \log(N/n_k)}{(1 - \text{slope}) + \text{slope} * (\# \text{ unique terms})/\text{pivot}}$$

where, tf is the term frequency
N is the collection size
n_k is the number of documents that contain this term
slope & pivot are empirically determined constants
unique terms is the number of distinct terms in this vector

Fig 3.4: *Ltu* query term weighting formula [10]

Lnu is used to convert an nnn-weighted element vector to a corresponding *Lnu* weighted vector. Details of the *Lnu-ltu* weighting scheme can be found in [10] and [16].

Smart Retrieval

The input to Flex is produced by a Smart retrieval against the *paragraph + mt* index to obtain a list of highly correlating leaf nodes. *Lnu* weights are applied to the element vectors and the queries are weighted using the *ltu* weighting scheme. Finally, an inner product of these elements with the query vector is computed, producing a ranked list of elements based on similarity score.

3.4 Flex: Flexible retrieval

This section describes the steps of Flex retrieval (Figure 3.2(b)).

Seeding Flex

Retrieval against the *paragraph + mt* index is required to seed Flex. This set of top-ranked elements from this retrieval is used as the seed. A basic assumption is made for this process: if any element from a document has a strong correlation with the topic, there is a good chance that other elements from the same document will correlate highly with

the topic as well. Apart from the *paragraph* + *mt* retrieval, the seeding process requires some other inputs:

1. A parameter, *n*, which indicates the number of top-ranked elements used to seed Flex.
2. A document schema (which we call doc-trees) is also given as an input to the seeding process. This schema specifies the structure of the documents which will be generated by Flex.
3. The docid-docpath mapping (from the *paragraph* + *mt* index) associates the XPath of each terminal node element identified in the schema of the seeded document with its corresponding Smart identifier.

Flex Retrieval

Flex reads the schema information of the seeded document. It then constructs the vectors for the non-terminal elements based their child elements. It then computes the correlation score for these non-terminal vectors. So correlation of every element in the tree with the query vector is computed. The output of Flex is a ranked list of elements for every query.

3.5 Post-Flex

This section provides the details of steps performed after Flex retrieval to prepare the output required for the Ad hoc tasks (See Figure 3.2(c)).

Filtering the output

Given the output from Flex, several filtering tasks are performed to prepare the output required by the various Ad hoc tasks, namely:

(1) *Removing mts*: the first task is to remove all the elements tagged with *mt*. These elements must be removed since this tag is used for internal purposes only. After all *mt* elements are removed, the output is valid for the *Thorough* task but needs further filtering for the 2007 Ad hoc tasks.

(2a) *Focused task*: Output of the Focused task can be obtained from the output of Thorough task by removing all overlapping elements from it. To remove overlap, we use

two strategies. In the first strategy, in case of an overlap, we always include the deeper element (child element) in the output and ignore the parent element. Another strategy outputs the element with the higher correlation score, whether it is child or parent. See [13] for details.

(2b) *Best-in-Context task*: The output of the Best-in-Context task is obtained by returning a single element per “relevant” article. We use several different strategies for selecting this element. See [11] for more details.

(2c) *Relevant-in-Context task*: The output of the *Relevant-in-Context* task is a list of non-overlapping elements sorted by article as per the initial article retrieval. The overlap removal strategies are same as those used in the *Focused* task. There are several variations of this basic strategy with which we have experimented. Details of all these experiments are given in Chapter 4.

XPath expansion

There are around 5,000 unique tags in all the Wikipedia documents [12]. To reduce the amount of processing required, a number of “low importance” (i.e., mostly formatting) tags are ignored throughout the flexible retrieval process. These tags are removed during the parsing step itself. Figure 3.5 shows of an original Wikipedia document. Figure 3.6 shows the same XML file after parsing.

In the original XML document (Figure 3.5) the tags *approximate*, *engine*, *http_* were discarded during parsing to produce the document in Figure 3.6. The XPaths of the elements in this document include:

```
/article[1]/body[1]/section[1]/p[1]
```

```
/article[1]/body[1]/section[1]/p[2]
```

These are the correct XPaths for the document in Figure 3.6, but these XPaths are invalid with respect to the original document (i.e., Figure 3.5). (This is because the paragraphs [p[1] and p[2]] are actually enclosed within the *approximate*, *engine* and *http_* xml tags. Hence these XPaths must be corrected before they are submitted to INEX.)

```
<?xml version="1.0" encoding="utf-8"?>
<article>
  <name id="2614489"> U.S. health reform under FDR </name>
  <conversionwarning> 1 </conversionwarning>

  <body>
    <section>
      <title> Introduction </title>
      <approximate>
      <engine>
        U.S. health reform under FDR began with efforts
        to include national healthinsurance in the
        original SocialSecurity bill.
        <p>
        However, FDR ultimately removed itdue to
        concerns that fierce opposition from the AMA
        would jeopardizeenactment of Social Security.
        Although the record suggests FDRplanned to
        return to this issue after World War II,
        hisuntimely death in early 1945 precluded this,
        so itwas left to Harry S.Truman to fight this
        battle.
        </p>

        <http_>
          <p>
            The first proposal for federal compulsory
            health insurance wasintroduced in 1943 by
            Senators
            ..
            ..
            ..
          </p>
        </http_>
      </engine>
    </approximate>
  </section>
</body>
</article>
```

Fig 3.5: Sample Wikipedia document in its original format (article id 2614489)

```
<?xml version="1.0" encoding="utf-8"?>
<article>
  <name id="2614489"> U.S. health reform under FDR </name>
  <conversionwarning> 1 </conversionwarning>

  <body>
    <section>
      <title> Introduction </title>

      U.S. health reform under FDR began with efforts to
      include national healthinsurance in the original
      SocialSecurity bill.
      <p>
        However, FDR ultimately removed itdue to concerns that
        fierce opposition from the AMA would
        jeopardizeenactment of Social Security. Although the
        record suggests FDRplanned to return to this issue
        after World War II, hisuntimely death in early 1945
        precluded this, so itwas left to Harry S.Truman to
        fight this battle.
      </p>

      <p>
        The first proposal for federal compulsory health
        insurance was introduced in 1943 by Senators
        ..
        ..
        ..
      </p>
    </section>
  </body>
```

Fig 3.6: Representation of the XML document after parsing (article id 2614489)

To solve this problem, we wrote a Perl script which takes incorrect XPath's and from them builds the correct XPath's. In particular, we are interested in "correcting" our Flex results. This script takes the elements returned by Flex, identifies the article for each element, checks the path of this element within the article and expands (corrects) it if necessary. It then outputs a new Flex result file with corrected XPath's. So in the above case, if we run the Perl script on the incorrect XPath's of Figure 3.6f, it will output the following:

/article[1]/body[1]/section[1]/approximate[1]/engine[1]/p[1]

/article[1]/body[1]/section[1]/approximate[1]/engine[1]/http_[1]/p[1]

These corrected XPathS will be accepted by INEX. The correctness of an XPath can be checked using the validation software provided by INEX.

Conversion to INEX format

Before evaluation, the output of the various tasks is first converted into an XML file. INEX specifies the XML output format (i.e., “submission format”) which is slightly different for each task.

Validating the outputs

The XML file generated in the previous step must first be validated to guarantee correct XPathS and the absence of overlap. INEX has provided a package [4] which checks the XML file for invalid XPathS or overlap. If either occurs, it reports an error.

Evaluation

After converting the output of the three tasks to their corresponding XML format and validating them as above, we use Perl scripts provided by INEX to evaluate these outputs. INEX provides a different script to evaluate the output of each of the three tasks. These scripts (foc_dom_eval2.pl, ric_dom_eval2.pl and bic_dom_eval2.pl) evaluate the output of the Focused, Relevant-in-Context and Best-in-Context tasks, respectively. Inputs for the scripts are:

1. DB_File: A database of all elements with global offsets.
2. Relevance Assessments: These can be downloaded from INEX website [5]
3. The XML file which is output from validation.

4. Experiments, Analysis and Observations

This chapter describes experiments performed to improve the performance of the 2007 Ad hoc tasks.

4.1 Tags used

Table 4.1 gives the list of tags that we used for indexing for INEX 2006 tasks. We call these “tags to index.” These are the tags which are considered terminal node tags and retrieved during Smart retrieval. Table 4.2 gives the list of tags that we considered during flexible retrieval for the INEX 2006 Ad hoc tasks. We call these “tags to keep.” During parsing, only these “tags to keep” are kept in the document (for building the trees later); all other tags are ignored. Table 4.3 specifies the slope and pivot values used for the 2006 retrieval tasks.

Tag Name	Tag Representation
Paragraph	<p> ... </p>
Normal-list	<normallist> </normallist>
Ordered-list	
Unordered-list	
Number - list	<numberlist> </numberlist>
Definition-list	<definitionlist> ... </definitionlist>
Figure	<figure> ... </figure>
Table	<table> ... </table>

Table 4.1: Tags identified as terminal nodes for INEX 2006 tasks (old tags)

Tag Name	Tag Representation
Article	<article> </article>
Section	<section> </section>
Body	<body> </body>
Paragraph	<p> ... </p>
Normal-list	<normallist> </normallist>
Ordered-list	
Unordered-list	
Number - list	<numberlist> </numberlist>
Definition-list	<definitionlist> ... </definitionlist>
Figure	<figure> ... </figure>
Magic Text	<mt> ... </mt>

Table 4.2: Tags recognized during the flexible retrieval process (INEX 2006)

Retrieval Type	Slope	Pivot
Article	0.04	120
Paragraph	0.12	18
All-element	0.12	38

Table 4.3: Slope and Pivot values for INEX 2006 tasks

After thorough analysis of the relevance assessments provided by INEX for the 2007 Ad hoc tasks, it appears that only a few of the tags mentioned in Table 4.1 are marked relevant by the assessor. Hence we defined a new of terminal nodes for the 2007 Ad hoc tasks. This new set is shown in Table 4.4. Table 4.5 lists the “tags to keep” for the 2007 Ad hoc tasks.

Tag Name	Tag Representation
Paragraph	<p> ... </p>
Figure	<figure> ... </figure>
Name	<name> ... </name>
Emphasis	<emph3> ... </emph3>

Table 4.4: Tags identified as terminal nodes for INEX 2007 tasks (new tags)

Tag Name	Tag Representation
Article	<article> </article>
Section	<section> </section>
Body	<body> </body>
Paragraph	<p> ... </p>
Normal-list	<normallist> </normallist>
Ordered-list	
Unordered-list	
Number - list	<numberlist> </numberlist>
Definition-list	<definitionlist> ... </definitionlist>
Figure	<figure> ... </figure>
Name	<name> ... </name>
Magic Text	<mt> ... </mt>
Title	<title> ... </title>
Emphasis	<emph3> ... </emph3>

Table 4.5: Tags recognized during the flexible retrieval process (INEX 2007)

Since tags which identify a terminal node were changed, the average length of an element vector also changed. This warranted a change in the *slope and pivot* values used

for paragraph retrieval. Table 4.6 lists the slope and pivot values used for the INEX 2007 Ad hoc tasks.

Retrieval Type	Slope	Pivot
Article	0.04	120
Paragraph	0.12	15
All-element	0.12	38

Table 4.6: Slope and Pivot values for INEX 2007 tasks

4.2 Relevant-in-Context Experiments

This section describes the experiments which were performed to improve the performance for the RIC task.

Experiment 1: Using Flex (for production of elements) and Article retrieval (for filtering and ordering) using the old tag set

Our first experiment involves both article and flexible retrieval. The results of both retrievals are used to produce the final output. We perform the following steps in this experiment. For each query:

- (1) Perform a normal flexible retrieval (which gives us a ranked list of elements).
- (2) Perform an article retrieval (which gives us a ranked list of articles).
- (3) Filter the output of the flexible retrieval on the basis of the article retrieval. That is, the elements produced by Flex are collected by article and reported in the order determined by the article retrieval. All elements associated with other articles (i.e., non-ranked articles) are deleted. This experiment results in an undefined number of elements per article being reported. Some article(s) in the top-ranked set may not in fact be reported (if none of its elements were generated by Flex).

This experiment was performed by using the old tag set. For this experiment, we utilize 2500 elements from Smart (para+mt) retrieval. The number of elements used to seed Flex is kept constant at 1500 ($n=1500$) in each case. While populating the upto 1500 trees generated by Flex, only terminal nodes found in this set of 2500 are populated. That is, only for those nodes are correlation values available. As a result complete trees may not be generated. The results of this experiment for INEX 2007 RIC task are listed in Table 4.7.

Number of Articles	MAgP
1000	0.0471
500	0.0577
250	0.0818
200	0.0830
100	0.0885
50	0.0897
25	0.0935
10	0.0587
5	0.0448

Table 4.7: Flex + Article retrieval results for RIC task (old tag set)

INEX evaluates the results of the RIC task on basis of *MAgP* (See Chapter 2 for details). The best result was obtained at 25 articles ($MAgP = 0.0935$). This value ranks our result at 38.

Experiment 2: Using Flex (for production of elements) and Article retrieval (for filtering and ordering) using the new tag set

After recognizing the importance of the new tag set (see Section 4.1), we performed a new experiment, identical to Experiment 1 but using the new tag set. The number of articles in this experiment varied from 5 to 1,000. The number of elements used to seed

Flex was kept constant at 1500 ($n = 1500$). The results of this task are listed in Table 4.8 (a).

For the new tag set, the best value of $MAgP$ was obtained at $m=1,000$ ($MAgP = 0.1142$). This value ranks our result 21 in the INEX result set. Clearly, significant improvement is realized when the new tag set is used.

Number of Articles	MAgP
1000	0.1142
500	0.1132
250	0.1087
200	0.1066
100	0.0982
50	0.0801
25	0.0721
10	0.0610
5	0.0521

Table 4.8 (a): Flex + Article retrieval results for RIC task (new tag set & incomplete trees)

We then repeat this experiment by retrieving 125,000 elements from the Smart *para+mt* index (instead of 2,500). This ensures that the trees generated are complete. The results of this task are listed in Table 4.8 (b). For the new tag set, the best value of $MAgP$ was obtained at $m=1,000$ ($MAgP = 0.1148$). This value ranks our result at 20 in the INEX result set. So though there is a difference produced by using complete trees, the difference is not remarkable.

Number of Articles	MAgP
1000	0.1148
500	0.1139
250	0.1078
200	0.1053
100	0.1006
50	0.0831
25	0.0763
10	0.0610
5	0.0521

Table 4.8 (b): Flex + Article retrieval results for RIC task (new tag set and complete trees)

Experiment 3: Incorporating Article retrieval in flexible retrieval

Experiment 2 ranked our results at 20. In order to further improve our performance, we analyzed the 2007 relevance assessments provided by INEX. Analysis shows that the trees that Flex builds in Experiment 1 and 2 for the RIC task are not in agreement with the articles in the relevance assessments. That is, in Experiments 1 and 2, for each query, Flex determines the trees that are built (i.e., the articles that are represented in the output). This list of trees often overlaps with the list of articles in the relevance assessments but the number of articles in common varies. We need to increase the size of this set (i.e., the overlap). With this in mind, a new experiment is performed as follows:

- (1) Perform article retrieval to produce a ranked list of articles per query.
- (2) Seed Flex with elements only from these articles.
- (3) Report the output in terms of elements from the article at rank 1, followed by the elements from the article at rank 2, etc. Thus the output of this experiment is, for every article in the ranked ordered list, its correlating elements.

This experiment was performed using the new set of tags. Results of this strategy are listed in Table 4.9.

No. of Articles	#Paras = 100	#Paras = 250	#Paras = 500	#Paras = 1000	#Paras = 1500	#Paras = 4000	#Paras = 8000
25	0.0591	0.0688	0.0706	0.0706	0.0706	0.0706	0.0706
50	0.0643	0.0748	0.0817	0.0828	0.0829	0.0829	0.0829
100	0.0692	0.0799	0.0860	0.0930	0.0939	0.0940	0.0940
150	0.0706	0.0811	0.0893	0.0970	0.0993	0.1003	0.1003
200	0.0712	0.0823	0.0903	0.0965	0.1009	0.1021	0.1021
250	0.0711	0.0831	0.0911	0.0975	0.1025	0.1047	0.1047
400	0.0697	0.0941	0.0962	0.0969	0.1049	0.1071	0.1071
500	0.0704	0.0927	0.0960	0.0967	0.1007	0.1068	0.1068

Table 4.9: MAgP results for INEX 2007 Relevant-in-Context task by incorporating article retrieval with flexible retrieval.

In this experiment, 125,000 elements were retrieved from the Smart para+mt index. This ensures that the trees generated are complete. As it can be seen from Table 4.9, performance in terms of MAgP improves as we increase the number of articles and number of elements retrieved from Flex. However after 400 articles and 4,000 elements, the *MAgP* value decreases. The best value we get is for 400 articles and 4,000 elements (*MAgP* = 0.1071). This value ranks our result at 29.

Experiment 4: Using Only Flexible Retrieval.

This experiment is similar to Experiment 2 but the article retrieval is not performed. In this experiment we use only the flexible retrieval process to get output for the RIC task. Article retrieval is not involved at any stage. We perform the following steps in this experiment.

- 1) Perform flexible retrieval using various numbers of elements to seed Flex.
- 2) From the Flex result, remove *mt* tagged elements and remove overlap.
- 3) Then reorder the flex file as follows:
 - Take the top ranked element.
 - Identify the article it belongs to.
 - Scan the Flex result file and get all the elements from this article.
 - All these elements go to the top of output file.
 - Take the next ranked element and follow the same procedure.

The results of this method are given in Table 4.10.

n	<i>MAgP</i> Value
2000	0.1034
1500	0.1025
1000	0.0978
500	0.0931
250	0.0863
100	0.0797

Table 4.10: Results for INEX 2007 RIC task by using only the flexible retrieval.

In this experiment, we retrieve 125,000 elements from Smart to ensure generation of complete trees. The best result we get is for 2,000 elements (used to seed Flex) ($MAgP = 0.1034$). This ranks our result at 32. The main issue with this method is the way in which articles are ordered. We assume that the article of the top ranked element is the most relevant article. This is unlikely to be the case.

Experiment 5: Using Article Retrieval and XPath

In this experiment, only the results from article retrieval are used. The process of flexible retrieval is not involved. In this experiment we perform following steps:

- 1) Perform article retrieval (normal Smart retrieval). 500 articles are retrieved in this step.
- 2) We have created, for each article, a file containing all the XPath in those articles. For each article, select from its XPath file, the first e XPath in that file.
- 3) These XPath are then reported based on article rank. This forms the Relevant-in-Context task output.

Results of this task are given in Table 4.11.

Elements per article (e)	MAgP value
60	0.1097
40	0.1093
20	0.1077
10	0.0719
5	0.0511

Table 4.11: Results for INEX 2007 RIC task using only article retrieval strategy.

As seen in Table 4.11, as we increase the number of elements selected per article, the results improve. The best value was obtained at $n = 60$ ($MAgP = 0.1097$). This value ranks our result at 25. However one problem with this strategy is that as we increase the number of elements per article, the file size increases rapidly. For $n = 60$ the file size for RIC output is around 400 Mbs. (These large files mitigated against increasing the number of elements in this experiment).

Conclusion:

From the experiments we performed for RIC task, conclusion can be drawn that the best results for this task are obtained by using new tag set and performing flexible and article retrieval independently.

4.3 Experiments for Focused task

This section describes the experiments which were performed to improve the performance for the focused task.

Experiment 1: Overlap removal

The first experiment involved changing our overlap removal strategy. Given the results from Flex, the overlap must be removed to produce Focused task results. Two different strategies for overlap removal were tried. The correlation score strategy gives priority to the highest correlating element along a path. The terminal node strategy gives precedence to the terminal node on the XPath. Given two overlapping elements, the terminal node which is deeper in the tree is always selected.

Various experiments were performed to compare the results of the two strategies. Results show that the terminal node strategy is more effective. This strategy is used for the experiments which follow. See [13] for details.

Experiment 2: Normal Flexible retrieval (old set tags)

In this experiment, the retrieval steps which were followed were those of normal flexible retrieval (Figure 3.2) using the old set of terminal node tags. The overlap from the Flex result was removed using the terminal node strategy. For this experiment, we retrieve 2500 elements from Smart retrieval. While populating the trees used by Flex, only terminal nodes found in this set of 2500 are used. That is, only for these nodes are correlation values inserted. As a result complete trees may not be generated. Results of this experiment are listed in Table 4.12. The results are evaluated at $iP[0.01]$. This method ranked our results at 32.

n	iP[0.01]
1000	0.4590
500	0.4583
250	0.4585
100	0.4586
50	0.4582
25	0.4642
10	0.4613
5	0.4336
1	0.3121

Table 4.12: Results for Focused task using old tag set.

Experiment 3: Normal Flexible retrieval (new set tags)

This experiment is identical to Experiment 2 but using the new tag set. Also, in this experiment, 125,000 elements are generated from Smart retrieval to ensure that the trees being built are complete. So here, we perform a normal flexible retrieval using the new set of terminal node tags. We then remove the overlap using the terminal node strategy, expand the XPath, convert to INEX format, validate and evaluate. The results using this strategy are listed in Table 4.13.

We ran this experiment for $n=1$ to 1,000 in defined steps. The best result occurs at $n = 1,000$ ($iP[0.01] = 0.5027$). Here n is the number of paragraphs used to seed Flex. This value ranks us at 11 as per the INEX result set.

n	iP[0.01]
1000	0.5027
500	0.5006
250	0.4983
100	0.4969
50	0.4963
25	0.4956
10	0.4701
5	0.4400
1	0.2574

Table 4.13: Results for Focused task using new tag set

Experiment 4: Incorporating Article Retrieval with Flexible Retrieval

In this experiment, we incorporate article retrieval with flexible retrieval (as we did earlier for the RIC task). This experiment is identical to the Experiment 3 for the RIC task. Overlap is removed using the terminal node strategy. For this experiment, we generate 125,000 elements from Smart retrieval. This ensures generation of complete trees. Best results produced by this method are given in Table 4.14.

The 1st column gives the number of top-ranked articles returned. Elements from these articles are used to seed Flex. The number of paragraphs indicates the number of elements produced by Flex as output. The difference between the values specified in this table and values specified in [13] for same experiment is attributed to the fact that here 125,000 elements were retrieved from Smart as opposed to 25,000 in [13]. So trees generated here are complete.

From Table 4.14 we can see that the best value ($iP[0.01] = 0.5266$) is obtained with 25 articles and 500 elements. This value ranks our result at 2. This in fact is a very good result since the top-ranked result is 0.5271, which is very close to our value.

Number of Articles	#Paras = 100	#Paras = 250	#Paras = 500	#Paras = 750	#Paras = 1000	#Paras = 1250	#Paras = 1500
25	0.4901	0.5263	0.5266	0.5262	0.5262	0.5262	0.5262
50	0.4965	0.5260	0.5282	0.5293	0.5260	0.5260	0.5260
100	0.4639	0.4983	0.5158	0.5217	0.5173	0.5178	0.5140
150	0.4635	0.4739	0.5136	0.5106	0.5189	0.5162	0.5161
200	0.4648	0.4659	0.4954	0.5021	0.5043	0.5134	0.5120
250	0.4575	0.4587	0.4842	0.4979	0.4954	0.5045	0.5084

Table 4.14: iP[0.01] results for INEX 2007 Focused task by incorporating article retrieval with Flex (new terminal node set).

Conclusion:

From the experiments we performed for focused task, conclusion can be drawn that using the new set of terminal node tags and incorporating article retrieval with flexible retrieval gives us our best results for the INEX 2007 Focused task.

4.4 Best-in-Context Experiments

Other members of our team carried out various experiments to improve the Best-in-Context results. Details of these experiments and results can be found in [11].

5. Future Work

In this section, we make suggestions for improvements and future work.

To improve our INEX performance rankings, we can fine tune the values of slope and pivot and refine the terminal node tag set. The tag set can be based on a thorough analysis of the relevance assessments (once they are released by INEX for the 2008 tasks).

To improve operational efficiency, the entire process of flexible retrieval can be automated by writing a single script. All the parameters must be specified beforehand and then used in the scripts for flexible retrieval.

There is more work to be done with respect to the Relevant-in-Context task. At this stage it is unclear how the RIC performance can be improved. As new relevance assessments are released by INEX, these can be analyzed and the experiments performed for RIC task can be fine tuned or modified to get better results.

A major aspect of the future work involves extension of flexible retrieval into passage retrieval. Analyzing the relevance assessments can give us insight into what forms a good passage. We may then be able to combine the elements retrieved in such a way as to produce a good passage. We also need a ranking scheme for the passages that are formed by combining elements. Another possibility is to modify the flexible retrieval process itself so that it works to produce passages instead of elements.

References

- [1] Crouch, C. Dynamic element retrieval in structured environment. *ACM TOIS*, 24(4): 437-454, 2006

- [2] Fuhr, N., Lalmas M., Trotman, A. Pre-Proceedings of INEX 2007, <http://inex.is.informatik.uni-duisburg.de/2007/inex07/pdf/2007-preproceedings.pdf>

- [3] Ganpathibotla, M. Query Processing in a Flexible Retrieval Environment, MS Thesis, Department of Computer Science, University of Minnesota Duluth, 2005. http://www.d.umn.edu/cs/thesis/satyanarayana_ganapathibhotla_ms.pdf

- [4] <http://inex.is.informatik.uni-duisburg.de/2007/adhoc-protected/submissions.html>

- [5] <http://inex.is.informatik.uni-duisburg.de/2007/adhoc-protected/assessments.html>

- [6] Kamat, N. Impact of Untagged Text in Dynamic Element Retrieval, MS Thesis, Department of Computer Science, University of Minnesota Duluth, 2007. http://www.d.umn.edu/cs/thesis/nachiket_kamat_ms.pdf

- [7] Khanna, S. Design and Implementation of Flexible Retrieval System, MS Thesis, Department of Computer Science, University of Minnesota Duluth, 2005. http://www.d.umn.edu/cs/thesis/sudip_khanna_ms.pdf

- [8] Lalmas, M., Piwowarski, B. INEX 2007 Relevance Assessment Guide, http://inex.is.informatik.uni-duisburg.de/2007/inex07/adhoc-protected/downloads/Relevance_Assessment2007.pdf

- [9] Malik, S., Trotman, A., Lalmas, M., Fuhr, N. Overview of INEX 2007 Workshop Proceedings, <http://inex.is.informatik.uni-duisburg.de/2007>
- [10] Malik, V. Impact of Terminal Node Processing on Element Retrieval. MS Thesis, Department of Computer Science, University of Minnesota Duluth, 2007. http://www.d.umn.edu/cs/thesis/vikram_malik_ms.pdf
- [11] Mehta, S. Finding the Best Entry Point. MS Thesis, Department of Computer Science, University of Minnesota Duluth, 2008. http://www.d.umn.edu/cs/thesis/sarika_mehta_ms.pdf
- [12] Mone, A. Dynamic Element Retrieval for Semi-Structured Documents, MS Thesis, Department of Computer Science, University of Minnesota Duluth, 2007. http://www.d.umn.edu/cs/thesis/aditya_mone_ms.pdf
- [13] Paranjape, D. Improving Focused Results. MS Thesis, Department of Computer Science, University of Minnesota Duluth, 2008. http://www.d.umn.edu/cs/thesis/darshan_paranjape_ms.pdf
- [14] Salton, G., editor, *The SMART Retrieval System – Experiments in Automatic Document Retrieval*, Prentice-Hall, Eaglewood Cliffs, NJ, 1971.
- [15] Salton, G., Wong, A., Yang, C. *A vector space model for information retrieval*. JASIS, 18(11):613-620, 1975.
- [16] Singhal, A., Buckley, C., Mitra M. Pivoted Document Length Normalization, *Proceedings of 19th Annual International Conference on Research and Development in Information Retrieval*, Zurich. 19-21, 1996.

Appendix A

A.1 Implementation details of the Normal Flexible retrieval process.

1. *Convert vectors to text format.* In this step we convert various vectors generated during indexing process into their text format. This is done so that the vectors can be easily read in the scripts. A Smart utility named `smprint` is used to perform this conversion. Following is its usage.
 - 1) `/smart/smart.13.0/src/bin/smprint vec query.nnn > query.nnn.txt`
 - 2) `/smart/smart.13.0/src/bin/smprint inv inv.words > inv.words.txt`
 - 3) `/smart/smart.13.0/src/bin/smprint vec doc.nnn > doc.nnn.txt`
 - 4) `/smart/smart.13.0/src/bin/smprint textloc textloc > textloc.txt`
2. *Generate the doc-trees.* Doc-trees represent the schema of each document. This schema is represented in form of their XPath. Following is the usage of script that generates doc trees:

```
perl generate_doc_trees_one_mt.pl config.txt 2008_collection_parsed
doctrees_dir
```

Parameters:

- 1) Configuration file that includes tags to keep and tags to index. Example of this config file is given in Figure A.1.

```
tags_to_keep
article,section,body,p,normallist,ol,ul,numberlist,definitionlist
,figure,name,mt,title,name
tags_to_index
p,figure,emph3,name,mt
```

Figure A.1 Configuration file used for generating the doc-trees.

2) Path to Document collection

3) Output path

3. *Generate doc id doc path mapping.* The doc id doc path mapping file gives the mapping of each indexed element to its corresponding Smart identifier. Following is usage of the same:

```
perl generate_docid_docpath_mapping.pl textloc.txt
                                docid_docpath_mappings.txt
```

Parameters:

- 1) textloc.txt file from which is created during generation of para + mt index.
- 2) output file which contains the docid doc path mappings.

4. *Creating query vectors.*

- a) Create the inverted query file. This step will create a file which will list each word in the query and corresponding document in which that word has occurred.

```
perl make_query_inv_list.pl query.nnn.txt inv.words.txt query_inv.words.txt
```

Parameters:

- 1) query.nnn.txt file generated while indexing the queries.
- 2) inv.words.txt file generated while creating the para+mt index. This file contains a listing of each word and the documents in which this word has occurred.
- 3) Output file.

- b) List the “relevant” elements from each document for each query. This step will create a list of document and the paragraph or section within that document in

which the query word has occurred.

```
perl get_n_subk_all_subvecs.pl docid_docpath_mappings.txt  
paras_mt_index/indexed
```

We get output file as `inv_vector_ctype0`

Parameters:

- 1) `docid_docpath_mappings.txt` from `para+mt` index
- 2) Output path (Should be indexed folder inside `para+mt` index)

- c) Convert `inv_vector_ctype0` to a vector form. We use a Smart utility for doing this operation. Following is the usage:

```
/smart/smart.13.0/src/bin/smart convert  
paras_mt_index/indexed/spec proc  
convert.obj.text_vec in inv_vector_ctype0 out n_stats.ctype0
```

Parameters:

- 1) Path of `inv_vector_ctype0` file we created earlier
- 2) Output file (which must be `n_stats.ctype0`)

5. *Move files in all parts of doctrees to a single folder.* INEX provides the Wikipedia collection divided into 22 different directories. In earlier step, the doctrees are generated for each Wikipedia document with the directory structure maintained. This step brings all the doctrees (which are present in 22 different directories) into a single directory. This is done for simplicity sake so that all files will be accessible under one directory.

```
perl moving_large_number_of_files.pl doctrees_dir/part-0
```

Parameters:

- 1) Path of each part in doctrees folder. This script has to be run once for each of the 22 folders.
- 2) Path to a folder in which all the doctrees will get stored.

6. *Conversion and Retrieval*

- a) Conversion: This step converts all the nnn element vectors to their corresponding Lnu format and also converts all the nnn query vectors in their corresponding Ltu format. Script to perform this operation is:

```
./nnn_to_Lnu
```

Before running this script, open the script and change the following to their desired values:

1) Slope and pivot values

2) Following paths:

current: Path to the para+mt index

database: Path to the output directory which will have output of the conversion step.

temp_dir: Path to a directory which will be used as a temporary directory in this process.

- b) Retrieval: This step performs a Smart retrieval and retrieves a ranked list of elements (by performing inner product of Lnu weighted element vectors and Ltu weighted query vectors). Following script is used to perform the Smart retrieval:

```
./retrieve_Lnu
```

Before running this script, open the script and change the following to their

desired values:

1) Change the number of elements to retrieve

2) Change the paths as follows:

current: Path to the para+mt index

database: Path to the conversion output which is obtained in above step.

Lnu_retrieval : Path to the output directory.

7. *Seed the doc trees.* This step takes the retrieval results from previous step. It then opens doctree of each relevant article and assigns correlation score determined in previous step to the corresponding XPath in the doctree. Following is the script for performing this operation.

```
perl convert_smart_tr_to_trees.pl tr.Lnu.txt docid_docpath_mappings.txt  
all_in_one_doctrees 1500 SeedResults
```

Parameters:

1) tr.Lnu.txt file from retrieval done in previous step

2) Path to docid_docpath_mappings.txt

3) Path to doctrees

4) No. of elements to be seeded.

5) Output path

8. *Run Flex.* In this step we run the Flex driver which reads the schema information of the seeded document. Lnu weights are then applied to the element vectors and the queries are weighted using the Itu weighting scheme.

```
./FlexDriver config_flex.txt
```

Parameters:

1) Config file for flex. This config file contains a number of parameters. The list of

parameters is given in Figure A.2

1. **DOC_INDEX_PATH:** paras+mt_index/indexed/doc.nnn
2. **QUERY_INDEX_PATH:** paras+mt_index/indexed/query.nnn
3. **OUTPUT_PATH:** flex_results/flex_output.out
4. **TREES_PATH:** /smart/para0101/flex_results/SeedResults
5. **NUM_OUTPUT_ELEMS:** 2500
6. **SLOPE_PARA:** 0.12
7. **SLOPE_SEC:** 0.12
8. **SLOPE_BDY:** 0.12
9. **SLOPE_ALLELEMS:** 0.12
10. **PIVOT_PARA:** 18
11. **PIVOT_SEC:** 38
12. **PIVOT_BDY:** 38
13. **PIVOT_ALLELEMS:** 38
14. **N_PARA_VALUES:** Number of paras. Get this no from textloc.txt in para index
15. **N_SEC_VALUES:** Number of sections. Get this no from textloc.txt in section index
16. **N_BDY_VALUES:** Number of articles. Get this no from textloc.txt in article

Figure A.2 Parameters in configuration file used by Flex Driver.

9. *mt Removal.* Remove all the mt elements from the Flex output.

```
perl remove_mt_from_flex_output.pl flex_output.out flex_output_nomt.out
```

Parameters:

- 1) Flex result file
- 2) Output file path: this file contains xpath's with mt's removed

10. *Expand the XPath's.* This step corrects all the XPath's by inserting all the intermediate tags which were omitted during the Flexible retrieval process.

```
perl blowUpFlexOutput.pl config_blowUpFlex.txt flex_output_nomt.out  
2008_collection_parsed_in_one_base_dir
```

flex_output_nomt_expanded.out

Parameters:

1) Configuration file. Sample config file for this script is given in Figure A.3. Here `tags_to_keep` contain a list of all the tags which are used in the Wikipedia documents. `Tags_to_index` are the tags which get indexed. `Tags_kept` are the tags which were used during the flexible retrieval process (these are the same tags used in config files of the doctrees).

```
tags_to_keep
article,section,body,p,normallist,ol,ul,numberlist,definitionlist,figure,name,collectionlink,item,unknownlink,cell,emph2,emph3,template,row,outsidelink,language,link,name,conversionwarning,br,td,caption,image,figure,wikipedialink,cadre,indentation1,tr,table,numberlist,emph5,sup,math,small,sub,weblink,value,term,definitionlist,center,indentation2,div,li,term,i,th,font,b,tt,code,blockquote,u,big,ul,span,gallery,cite,indentation3,s,hr,emph4,em,strong,h4,hiero,h3,redirectlink,http:,st3,var,ol,dd,h2,dl,title,mt,call,fun,http_,approximate,engine,ng,r,number,ww.w.cms.hhs.gov,page_name,sum,stdlib.h,stdin,iostream,valuegivingprotocol,objc,int,style
tags_to_index
p,figure,emph3,name,mt
tags_kept
article,section,body,p,normallist,ol,ul,numberlist,definitionlist,figure,name,mt,title,name
```

Figure A.3 Configuration file used for XPath expansion.

- 2) Path of flex result without mt tag.
 - 3) Path to the directory containing all the Wikipedia documents in one directory.
 - 4) Output path.
11. After expanding the XPaths, convert these flex results to the outputs of various Ad hoc tasks. Following are the scripts used for each task:

Focused:

```
perl gen_focused_output.pl flex_output_nomt_expanded.out
flex_focused.out
```

Best-in-Context:

```
perl generate_best_in_context.pl flex_100_nomt_expanded.out flex_BIC.out
```

Relevant-in-Context:

For this task, an article retrieval is first performed using article index. (Same procedure as in step 6 but done using article index). Then the following script is run

```
perl generate_fetch_browse_retrieval.pl flex_focused.out
                                lnu_article_retrieval/tr.Lnu.txt
                                num_eles num_articles
                                flex_RIC.out
```

Parameters:

- 1) Path of the focused result file.
- 2) Path of tr.Lnu.txt from article retrieval.
- 3) Number of elements to be considered from flex file
- 4) Number of articles to be considered from article retrieval
- 5) Output file path

12. Then convert the results in INEX format. In this step each result is converted to INEX submission format. Following is the script to perform this conversion.

```
perl convert_to_inex_v2007.pl part_id run_id task type num_elements
                                start_query_id flex_res_file inex_out_file
```

Parameters:

part_id: The unique identifier for each participant. This can be found on INEX website.

run_id: An identifier used to define each run. Should be unique for each run that is

submitted.

task: Should be either “Focussed” or “RelevantInContext” or “BestInContext” depending on the task for which this conversion is performed.

Type: Should be either “element” or “passage” depending on the type of retrieval being performed.

num_elements: The top n number of elements to be considered for each query.

start_query_id: The id of the first query.

flex_res_file: Path to the result file after converting the Flex output to one of the Ad hoc tasks.

inex_out_file: Path to the inex format output file

13. *Validation.* This step performs the validation the output file from step 12 for presence of overlap or presence of incorrect XPath. The script to perform validation is:

```
./check_submission inex_out_file
```

Parameters:

- 1) Complete path to the XML file which is generated in previous step.

Before running this script, open it and change the following:

```
dtd => 'Submission dtd format provided by INEX'
```

```
inex => 'Path to the Wikipedia collection'
```

```
pathdb => 'Path to the database generated for validation purpose.'
```

Then change the parameter

```
my @topics = (544 .. 828);
```

Here in place of 544 and 828, put the starting number and ending number of the

query set.

pathdb is the database which has to be built before checkrun is run. This database has to be built only if the document collection changes. The size of pathdb is around 22 gb.

Command to build the pathdb is:

```
./build_pathdb_wiki -inex collection_dir -pathdb actual_path_db(this is the output directory)
```

14. *Evaluation.* After performing the validation step, perform the evaluation. (Redirect output to record results)

Focused:

```
perl foc_dom_eval2.pl DB_File relevance_assessments inex_out_file
```

Best-in-context:

```
perl bic_dom_eval2.pl DB_File relevance_assessments inex_out_file
```

Rel-in-Context:

```
perl ric_dom_eval2.pl DB_File relevance_assessments inex_out_file
```

Parameters:

- 1) The database for evaluation. This database must be created each time the document collection is changed. It contains all elements with global offsets.
- 2) Relevance Assessments: These can be downloaded from INEX website [14]
- 3) XML file which is output from the previous step (i.e., the output of validation)

A.2 Steps to incorporate article retrieval into element retrieval.

1. Perform Steps from 1 to 7 which are mentioned in section A.1.
2. Do the article retrieval for various article values (e.g. 25, 50,100,150,200,250).
3. Convert tr.Lnu.txt for each retrieval to flex format.

```
perl smart_to_flex.pl article_docid_docpath_mappings.txt  
lnu_article_retrieval/tr.Lnu.txt tr.Lnu.out
```

Parameters:

- 1) The docid docpath mapping file created for articles.
 - 2) Path to the article retrieval file.
 - 3) Output file, which is file in Flex format.
4. *Generate list of articles.* For each query, a list of top m articles is generated. We create different lists for various values of m for this experiment.

```
perl generate_article_list_new.pl tr.Lnu.out article_list.txt
```

Parameters:

- 1) The article retrieval file in Flex format (generated as output of step 3)
 - 2) Output file.
5. *Create a subset of trees from element retrieval.* This subset will contain only trees of articles present in the list generated at step 3.

```
perl copying_seeding_files.pl article_list.txt SeedResults seed_subset
```

Parameters:

- 1) The list of articles generated in step 4.
 - 2) The result of seeding generated in step 7 of section A.1.
 - 3) The subset of seeding elements which will be used to seed Flex.
-
6. In Flex configuration file, use the `seed_subset` directory instead of the whole `seeding_directory` (parameter 4 in Figure A.2).
 7. Perform steps 8 to 14 from section A.1.