

Dynamic Element Retrieval for Semi-Structured Documents

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

Aditya Shrikant Mone

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

July, 2007

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

UNIVERSITY OF MINNESOTA

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

Aditya Shrikant Mone

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

Dr. Carolyn Crouch

Name of the Faculty Adviser

Signature of the Faculty Adviser

Date

GRADUATE SCHOOL

© Aditya Shrikant Mone, 2007

Acknowledgements

Many people have contributed towards the successful completion of this thesis.

I would like to take this opportunity to thank Dr Carolyn Crouch for her continuous guidance and valuable feedback on my work. I am really grateful to her for giving me opportunity to work in this exciting field of information retrieval.

I would like to thank Dr Donald Crouch for his useful suggestions and feedback. I would like to thank Nachiket Kamat and Vikram Malik for their useful suggestions and prompt help. I would also thank Darshan Paranjape for his help with experiments.

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

Finally I would like to thank my family, friends, and ‘team-1107 E’ for their support and encouragement during my two years at Duluth.

Abstract

Flexible retrieval refers to the dynamic retrieval of elements in a structured environment. Applying Flex (our system of flexible retrieval) to semi-structured documents presents an interesting challenge. The adaptation from structured to semi-structured documents includes issues of tagging untagged text and its usage as terminal XML nodes. When successfully adapted, Flex generates results identical to those generated by all-element retrieval.

Seeding Flex with a varying number of terminal nodes shows that it is possible at certain values to obtain better results with Flex than with all-element retrieval. Flex uses an index which is free from redundancy and hence requires less disk space. The value of n (the number of terminal nodes used to seed Flex) determines the number of documents Flex generates. These experiments show that low values of n produce superior results and minimize the time required for tree generation. The time and space requirements for Flex operation are specified along with the appropriate values for seeding. The results for the INEX 2006 Ad-hoc task (Thorough, Focused, Best-in-context and All-in-context) are also presented.

Table of Contents

List of Figures.....	v
List of Tables.....	vi
Chapter 1. Introduction.....	1
Chapter 2. Flexible Retrieval System for Semi-structured Documents.....	4
2.1 The Vector Space Model.....	4
2.2 The Smart Retrieval System.....	4
2.3 Flexible Retrieval.....	5
2.4 Adaptation of Flexible Retrieval to Semi-Structured Documents.....	5
2.5 Element Vector Weighting.....	8
2.6 Query Weighting.....	9
2.7 Pivoted Length Normalization.....	10
2.8 Similarity Computation.....	11
2.9 Output Processing.....	11
Chapter 3. Procedure for Flexible and All-element Retrieval.....	12
3.1 Procedure.....	12
3.2 Parsing.....	13
3.3 Indexing.....	15
3.4. Slope and Pivot Computation.....	15
3.5 Retrieval Using Computed Slope and Pivot.....	16
3.6 Seeding Flex.....	16
3.7 Flex.....	17

3.8 Filtering the Output.....	18
3.9 Evaluation.....	18
Chapter 4. Experimental Setup.....	20
4.1 Document Collection.....	20
4.2. Query Collection.....	21
4.3 Relevance Assessments.....	22
4.4 Evaluation Measures.....	23
Chapter 5. Experiments, Observations and Analysis.....	27
5.1 Procedure to Validate Flex.....	27
5.2 Results, Observations and Conclusions.....	28
5.3 Determining a Realistic Value of n.....	31
5.4 Results, Observations and Conclusions.....	31
5.5 Time and Space Requirement of Flex.....	40
Chapter 6. Future work.....	44
References.....	45
Appendix A.....	47
A.1 Configuration File for Flex.....	47
A.2 Configuration Files for EvalJ.....	48
A.3 Sample Output from Flex and All-element Retrieval.....	49
A.4 Results for the All In Context Subtask for INEX 2006.....	50
A.5 Results for the Best In Context Subtask for INEX 2006.....	51

List of Figures

2.4 Semi-structured nature of XML documents.....	7
2.5 <i>Lnu</i> element term weighting formula.....	9
2.6 <i>ltu</i> query term weighting formula.....	10
3.1 Procedure for the all-element retrieval experiments.....	12
3.1 Procedure for the normal flexible retrieval experiments.....	13
4.1 Sample Wikipedia document.....	21
4.2 Sample query (topic) [Topic id = 294].....	22
4.4 Calculation of nxCG and effort-precision (ep).....	26
A.1 A typical configuration file for Flex.....	47
A.2 A typical configuration file for EvalJ (Thorough sub-task).....	48
A.2 A typical configuration file for EvalJ (Focused sub-task).....	48
A.3 Sample all-element output.....	49
A.3 Sample Flex output.....	49

List of Tables

2.4 Tags identified as terminal nodes for the INEX 2006 document collection.....	6
2.4 Number and type of terminal nodes in the INEX 2006 document collection.....	6
2.4 Typical semi-structured article in terms of its elements.....	7
2.6 Number of elements in INEX 2006.....	9
3.4 Slope and Pivots values for INEX 2006.....	16
3.6 Requirements for seeding Flex and their purpose.....	17
4.1 Specification of the INEX 2006 document collection.....	20
5.2 Slope = 0.12, Pivot = 38, Task: Thorough, Assessment v4.....	28
5.2 Slope = 0.12, Pivot = 38, Task: Thorough, Assessment v5.....	28
5.2 Slope = 0.12, Pivot = 38, Task: Focused (Off), Assessment v4.....	29
5.2 Slope = 0.12, Pivot = 38, Task: Focused (Off), Assessment v5.....	29
5.2 Slope = 0.12, Pivot = 38, Task: Focused (On), Assessment v4.....	29
5.2 Slope = 0.12, Pivot = 38, Task: Focused (On), Assessment v5.....	29
5.4 Slope = 0.12, Pivot = 38, Task = Thorough, Assessment v4.....	32
5.4 Slope = 0.12, Pivot = 38, Task = Thorough, Assessment v5.....	32
5.4 Slope = 0.2, Pivot = 110, Task = Thorough, Assessment v4.....	33
5.4 Slope = 0.2, Pivot = 110, Task = Thorough, Assessment v5.....	33
5.4 Slope = 0.12, Pivot = 38, Task = Focused (off), Assessment v4.....	34
5.4 Slope = 0.12, Pivot = 38, Task = Focused (off), Assessment v5.....	35
5.4 Slope = 0.12, Pivot = 38, Task = Focused (on), Assessment v4.....	35
5.4 Slope = 0.12, Pivot = 38, Task = Focused (on), Assessment v5.....	36

5.4 Slope = 0.2, Pivot = 110, Task = Focused (off), Assessment v4.....	36
5.4 Slope = 0.2, Pivot = 110, Task = Focused (off), Assessment v5.....	37
5.4 Slope = 0.2, Pivot = 110, Task = Focused (on), Assessment v4.....	37
5.4 Slope = 0.2, Pivot = 110, Task = Focused (on), Assessment v5.....	38
5.5 Tree generation statistics, Paragraph retrieval: Slope = 0.12, Pivot = 18; Flex: Slope = 0.12, Pivot = 38.....	41
5.5 Tree generation statistics, Paragraph retrieval: Slope = 0.12, Pivot = 18; Flex: Slope = 0.2, Pivot = 110.....	42
5.5 Data-space requirements.....	43
A.4 Slope = 0.12, Pivot = 38, Task = All-In-Context, Assessment v4.....	50
A.4 Slope = 0.12, Pivot = 38, Task = All-In-Context, Assessment v5.....	51
A.5 Slope = 0.12, Pivot = 38, Task = Best-In-Context, Assessment v5.....	51

1. Introduction

Information is a resource sought by people across the globe. Web-based search engines provide users with the means of locating information quickly. Systems like Google, Yahoo and others have become an integral part of our lives. Electronic data is commonly represented in text, video and/or audio forms, but most web documents primarily contain text. Searching these documents for information satisfying the user's need is an important task.

Electronic documents can be *structured*, *unstructured* or *semi-structured*. *Structured* documents are well defined; every element (i.e., identifiable piece of text) in the document is compliant with a specific hierarchy as defined in its DTD (Document Type Definition). Every element in the hierarchy contains information of a specific type. (The IEEE documents used in INEX 2005 are structured in nature.) *Unstructured* documents by definition lack structure. Data occurs freely without structural constraints. Text without structural bindings (i.e., XML tags) may be considered unstructured. Semi-structured data is data that may be irregular or incomplete and have a structure that may change rapidly or unpredictably. [1] Semi-structured XML documents have a general structure, but untagged text may be present within it. (Wikipedia documents used in INEX 2006 are semi-structured in nature.)

Most web-based data is formatted in XML (Extensible Markup Language). A primary purpose of XML is to facilitate the sharing of data across different systems. Text in an XML document is represented as

elements in a hierarchy. Every element has a defined position in the hierarchy of the document. An element is delimited by special XML labels called tags. In the INEX 2006 collection, Wikipedia, the *article* element is marked by the presence of <article> and </article> tags, respectively. An article element contains different sub-elements like *section*, *paragraph*, *figure*, etc.

Earlier information retrieval systems commonly retrieved references to documents in response to a query. The user had to browse through each document to find the relevant portion(s) of text. XML promotes the development of element retrieval, i.e., the return of a set of elements rather than the document as a whole. It also allows the user to specify structural constraints as a part of the query. He or she can ask for any element within an article or the article itself. This helps the system to return results tuned to the user's information need. In this thesis, we consider only semi-structured XML documents.

The University of Minnesota Duluth (UMD) is a participant in the INEX (Initiative for the Evaluation of XML Retrieval) competition. INEX encourages the development of XML-based retrieval systems. INEX provides all participant organizations with a set of documents (test collection), queries (topics) and metrics for evaluating their XML-based retrieval systems. [2]

The focus of this thesis is flexible retrieval in a semi-structured document collection. We compare this output with that generated by all-element retrieval (the baseline) and determine the conditions under which the two methods achieve comparable results. An all-element index uses

every element in the collection as its input set. [3] The research focuses on subtasks of the Ad-hoc task in INEX 2006 and 2007.

Chapter 1 presents introductory information. Chapter 2 presents an overview of the adaptation of flexible retrieval to semi-structured XML documents. Chapter 3 describes the all-element as well as the flexible retrieval. Chapter 4 presents the details of the INEX 2006 and 2007 test-bed. Chapter 5 presents the experiments conducted, results, observations and analysis. Chapter 6 presents future work.

2. Flexible Retrieval System for Semi-structured Documents

Our flexible retrieval system [3] is designed for structured XML documents. The extension of this system to semi-structured XML documents is a step towards its generalization. In this section we present the retrieval model (framework) used, the retrieval engine used and the steps for the extension of flexible retrieval system to semi-structured documents.

2.1 The Vector Space Model

The *Vector Space Model* [4] is the basic model of this research. The vector space model represents each document and query as an n-dimensional vector of unique terms. The terms are weighted based on their frequency within the document. The relationship of a document to a query is determined by the distance between the two vectors in vector space. The closer the two vectors, the more potentially relevant the document is (cosine similarity is one of the measures used to compute the distance). Our retrieval engine, Smart 13.0, is based on the Vector Space Model.

2.2 The Smart Retrieval System

Smart 13.0 is our retrieval engine. Smart is used to index, weight and retrieve the documents correlating with a query. It also provides a facility to evaluate retrieval runs with a utility called *smart-eval*. This utility uses precision-recall to evaluate results. Smart produces the *textloc* file (information about the physical location of the documents being indexed),

dictionary (information about the terms in the documents and number of documents each term is contained in), document vectors, and inverted file (for each term, the documents that contain that term are listed).

2.3 Flexible Retrieval

Flexible retrieval refers to the dynamic retrieval of elements in a structured environment [5]. Flexible retrieval is designed to handle structured XML documents. The current research is an attempt to extend this idea to semi-structured XML documents.

Earlier experiments with flexible retrieval are documented in [3], [6], [7]. The details of flexible retrieval can be found in [5]. Flexible retrieval requires a base case for evaluation. The all-element index is used to generate this base case. The disadvantage of the all-element index is the redundancy created due to overlapping elements and large size of the files created. [3]

2.4 Adaptation of Flexible Retrieval to Semi-structured Documents

Our system of flexible retrieval is called Flex. Flex uses only the leaf nodes of a XML document and builds all the non-leaf nodes. Leaf node(s) of an XML document are the smallest identifiable and meaningful unit(s) of the document. Identified leaf nodes (terminal elements) are listed in Table 1. The total number of leaf nodes of every type is shown in Table 2.

The documents used for INEX 2006 and 2007 are Wikipedia documents in XML format. This document set is semi-structured in nature. A typical example of a Wikipedia document is shown in Figure 1. The semi-structured nature of Wikipedia poses an interesting challenge to Flex.

Flex builds non-leaf nodes (parent nodes) using the data contained in its child node(s). All terminal nodes must be present for the parent to be generated properly. Semi-structured documents may have text (i.e., untagged text) present in the parent node which is not contained in any of its identifiable child nodes. Figure 1 represents a typical semi-structured document. The XML nodes and the text contained in each element are listed in Table 3.

Table 1: Tags identified as terminal nodes for INEX 2006 document collection

Identified terminal elements	Terminal tags
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 2: Number and type of terminal nodes in the INEX 2006 document collection

Type of terminal element	Count of the terminal element
Leaf nodes	4,071,665
' <i>mt</i> ' nodes *	2,181,351
Leaf nodes + ' <i>mt</i> ' *	6,253,016

*Refer to section 2.4 for details

```

<article>
  sun
  <section>
    hello
    <p>
      world
    </p>
    <p>
      earth
    </p>
    solar system
  </section>
</article>

```

Figure 1: Semi-structured nature of XML document

Table 3: Typical semi-structured article in terms of its elements

Node	X-path of element	Text contained in the element
1	<i>/article[1]</i>	<i>sun hello world earth solar system</i>
2	<i>/article[1]/section[1]</i>	<i>hello world earth solar system</i>
3	<i>/article[1]/section[1]/p[1]</i>	<i>World</i>
4	<i>/article[1]/section[1]/p[2]</i>	<i>Earth</i>

Consider the two leaf nodes represented by nodes 3 and 4 in Table 3. An attempt to generate the parent node, i.e. section, from its children will fail due to the presence of untagged text (i.e., *hello, solar, system*) in the parent. We refer to this untagged text as *magic text*. Magic text must be considered and incorporated to ensure the proper generation of a parent node from its children. To achieve this objective, all untagged text is marked with special tags, namely, *<mt>* and *</mt>*. [8] If magic text is present within a node (i.e., text is scattered within the parent), it is aggregated as one single entity. The number of nodes tagged as *mt* is shown in Table 2. The tagging of aggregated magic text and the generation of *mt* leaf nodes is done during the parsing of the documents. [See section 3.2 for details.] The nodes tagged as *mt* are added to the set of leaf nodes.

2.5 Element Vector Weighting

The normal term weighting/retrieval schemes tend to be unfairly biased towards larger elements and thereby increase their probability of retrieval [9]. Thus we use *Lnu-ltu* weighting scheme for this work. One advantage of *Lnu*-weighted element vectors is the independence of the weighting scheme from the global statistics. [3] The formula for *Lnu* weighting is given in Figure 2. This formula is used to convert a *nnn*-weighted element vector to a corresponding *Lnu*-weighted vector.

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

where, tf is term frequency (as in the *nmn* 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.

Figure 2: Lnu element term weighting formula

2.6 Query Weighting

The Topic vectors (or queries) are weighted using the *ltu*-weighting scheme. This formula is shown in Figure 3. The variables N and n_k represent the global statistics. N is the total number of elements of all types in the entire Wikipedia collection. Table 4 gives the elements of each type and the total of elements, N , for INEX 2006 and 2007.

Table 4: number of elements in INEX 2006

element-type	total number of elements
Article [Body]	659,166
Section	1,600,709
Paragraph	4,071,665
Paragraph + mt	6,253,016
N (total number of elements)	6,331,540 / 8,512,891

n_k represents the number of elements containing a specific query term. This global statistic can be calculated by using the method described by Ganapathibotla in [6].

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

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

Figure 3: Itu query term weighting formula

2.7 Pivoted Length Normalization

The pivoted length normalization is used in the *Lnu-ltu* weighting scheme. [9] Its aim is to ameliorate the undue advantage that longer elements have over shorter elements. [3] Slope and pivot are calculated empirically. We calculate slope and pivot values based on the type of element in question (i.e., body, paragraph and all-element).

2.8 Similarity Computation

Similarity between the topic and the element vectors is calculated by the inner product between them. Flex computes the similarity between the *Lnu*-weighted document and *ltu*-weighted query vector. All the element vectors (per topic) are sorted in descending order of similarity and are presented to the user as the output.

2.9 Output Processing

The nodes tagged by *mt* are not present in the actual XML document. The output of Flex is therefore filtered to remove all the elements tagged as *mt*. [8] The filtered output is then processed according to the requirement of specific sub-tasks (i.e., Thorough, Focused, All-in-context, Best-in-context) to generate the respective outputs. This is followed by the application of structural constraints and fluffing to generate the final output. [10]

3. Procedure for Flexible and All-element Retrieval

In this chapter, the procedures for flexible retrieval and all-element retrieval are presented. The chapter also includes a detailed explanation of parsing, indexing, output processing and evaluation of results.

3.1 Procedure

The procedure for all-element retrieval is represented in Figure 4, whereas the procedure for the flexible retrieval is represented in Figure 5.

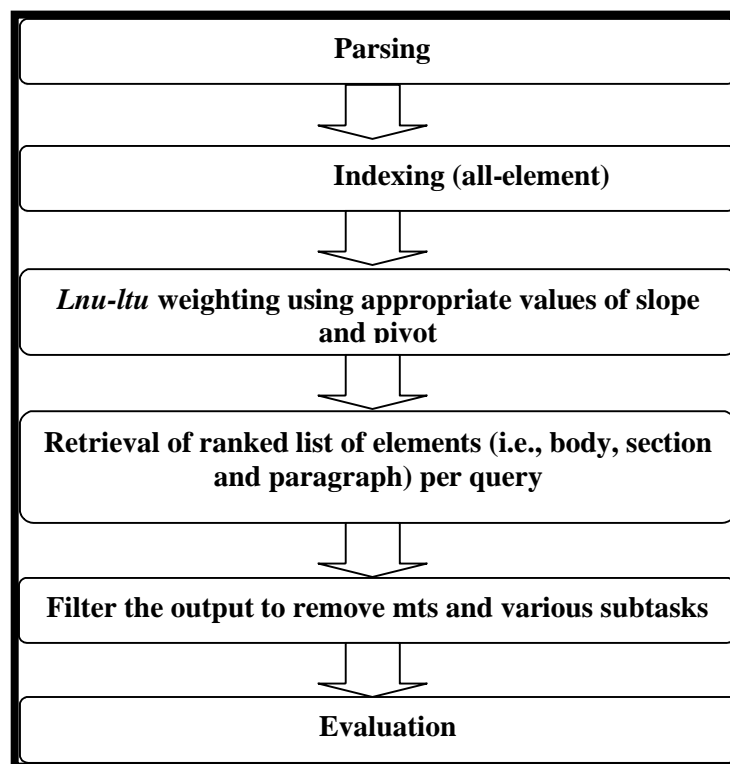


Figure 4: Procedure for the all-element retrieval experiments

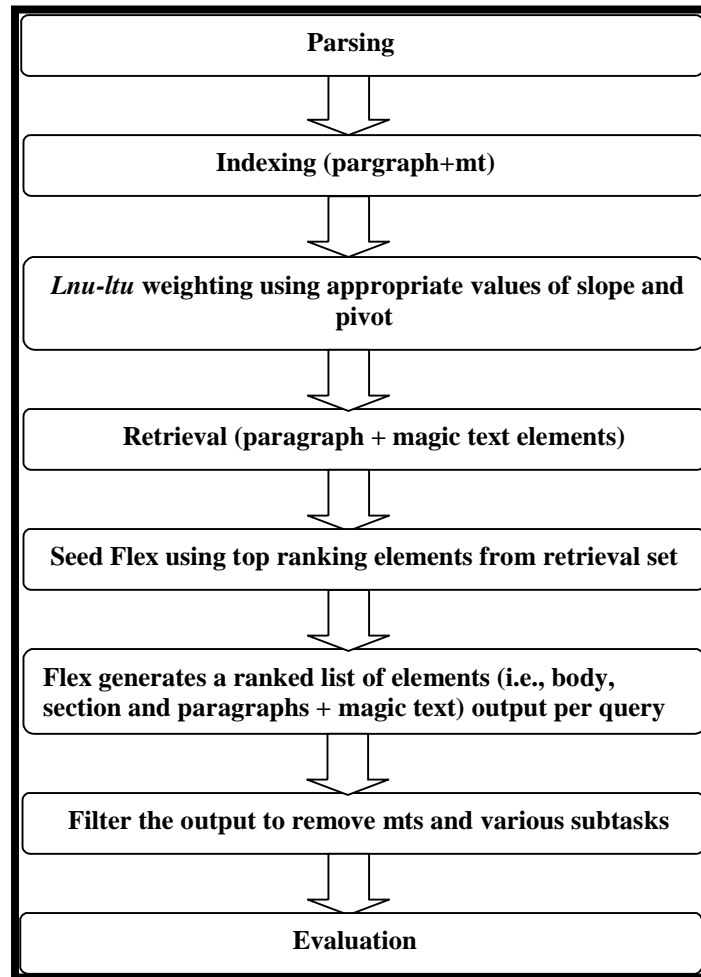


Figure 5: Procedure for the normal flexible retrieval experiments

3.2 Parsing

The INEX document collection is downloadable in form of a tar.gz file. After extracting, the document collection consists of a parent folder containing 22 sub-folders. Each sub-folder has 30,000 documents on

average. The documents are parsed into constituent elements. Four parses are generated:

1) *Article* parse: This parse creates a single element per article. We use only the part of the article delimited by *body* tags to represent the article. (The article vector is also called *body-only* vector) The text in the document is retained whereas all the tags except *article* and *body* are ignored. This parse is used to produce elements for the all-element index and for the article index.

2) *Section* parse: This parse generates all the section-level elements. Every element delimited by *section* corresponds to a single entity. For every section, all the tags contained in it are removed and the text is aggregated to form the section level element. The elements generated by this parse are used to contribute to all-element index.

3) *Paragraph* parse: This parse generates all the leaf nodes. (The tags delimiting the terminal elements are given in Table 1.) Once a leaf node tag is reached in the xml hierarchy any tags contained in it are removed and the text is aggregated to form the leaf node. The elements generated by this parse are used to contribute to all-element index.

4) *Paragraph + mt* parse: This parse generates the magic text elements in addition to the leaf nodes (i.e., paragraphs). This parse is used to generate the paragraph + mt index.

3.3 Indexing

Parsing is followed by creation of indices using Smart 13.0. The following indices are generated for experimentation:

1) *all – element* index: - The all-element index uses the *article*, *section* and *paragraph* parses as its input set. This index has a high level of redundancy due to overlapping elements being treated as independent documents. Every element is treated independent of the others and indexed accordingly.

2) *paragraph + mt* index: - The *paragraph + mt* index uses the *paragraph + mt* parse as its input set. Redundancy is not present in this index. The flexible retrieval system uses this index.

3) *article* index: - The *article* index uses the article parse as its input set. Redundancy is not present in this index.

Queries (i.e., the title part of every query) are also indexed along with the elements.

3.4. Slope and Pivot Computation

In order to perform pivoted length normalization, slope and pivot are computed for all the indices. The computation of slope and pivot is empirical. The final (best) value is the one giving best results upon evaluation. Values of slope and pivot for various indices are found in Table 5.

Table 5: Slope and Pivots values for INEX 2006

	Slope	Pivot
All-element	0.12	38
Article	0.04	120
Paragraph + mt	0.12	18

3.5 Retrieval Using Computed Slope and Pivot

The best value of slope and pivot is then used to generate various retrievals. These retrievals may be filtered according to different subtasks of the Ad-hoc tasks of INEX 2006 and 2007.

3.6 Seeding Flex

A retrieval against the paragraph + mt index is required to seed Flex. The set of top-ranked elements from this retrieval is used as the seed. (This set of top-ranked elements which identify documents containing possibly relevant elements is referred to as seeding Flex.) It is based on the assumption that if any element from a document has a strong correlation with the topic, there is a good probability that other elements, from the same document, will correlate highly with the topic as well. Various inputs and their purpose required for seeding Flex are listed in Table 6.

Table 6: Requirements for seeding Flex and their purpose

Input	Purpose
initial retrieval (paragraph + mt)	to identify document(s) with possibly relevant elements.
n , number of top-ranked elements used to seed Flex	to determine the smallest number of trees required to produce a result comparable to all-element retrieval.
document schemas (doc-trees)	to provide structure for documents generated by Flex.
docid-docpath mapping from paragraph + mt index	to relate x-path of every terminal element identified from the schema of the seeded document with its corresponding Smart identifier.

3.7 Flex

Flex reads the schema information of the seeded document. It then computes the vectors for the non-terminal elements and computes the correlation of every element with the query vector using *Lnu-ltu* weighting with inner product. Flex outputs a ranked set of correlating elements. A typical configuration file for Flex is shown in Appendix A.1. For the *All-in-context* and *Best-in-context* subtasks, retrieval on the article index is used to seed Flex. The process thereafter is same as the normal process for running Flex.

3.8 Filtering the Output

Once the output is filtered and all the nodes tagged by *mt* are removed, the output is valid for the *Thorough* subtask. In this subtask, a ranked-ordered list of all positively-correlating elements is expected. Overlapping elements are allowed in the output. The output for the *Focused* subtask can be obtained from the *Thorough* output by filtering out the overlapping elements. If two overlapping elements are found in the output, the element higher in the hierarchy is removed, thereby retaining the lower level element.

For the *Best-in-context* and *All-in-context* subtasks, Flex output is filtered to produce the required results. The output for the *Best-in-context* sub-task is obtained by returning a single element per “relevant” article (this is the element with highest correlation in a particular article). The elements are ranked in the order of their containing articles as per the initial article retrieval. The output for the *All-in-context* subtask is a list of non-overlapping elements sorted by article as per the initial article retrieval. Overlapping elements higher in the hierarchy are removed while the lower elements are retained. Outputs for all subtasks must be converted to XML format before submission to INEX.

3.9 Evaluation

The output in XML format can be evaluated using *EvalJ* (the INEX 2006 evaluation package). *EvalJ* has two versions: one is used to evaluate outputs of the *Thorough* and *Focused* subtasks and the other (INEX-eval) is used to evaluate the output of the *Best-In-Context* subtask. For evaluating the output

of the *All-In-Context* subtask, a Perl script is downloaded from the INEX 2006 website. This evaluation can also be done online (see [2]).

4. Experimental Setup

INEX provides a testbed for conducting experiments. It includes a document collection, set of queries (topics), relevance assessments and evaluation software.

4.1 Document Collection

The INEX 2006 (and 2007) document collection is a set of Wikipedia documents in XML format. The specifications of the collection are in the Table 7.

Table 7: Specification of INEX 2006 document collection [2]

Specifications of INEX 2006 document collection (Wikipedia)	
1.	Number of documents in the collection = 659,388
2.	Size of the document collection \approx 4.6 GB
3.	Number of categories covered = 113,483
4.	Number of unique tags = 5000
5.	Average number of XML nodes per document = 161.35
6.	Average depth of an element = 6.72

The document collection comes without a DTD. An article consists of the article-identifier, title and the body. Each article consists of identifiable elements. A typical Wikipedia document is presented in Figure 6.

```
<?xml version="1.0" encoding="UTF-8"?>
<article>
  <name id="1000">
    Hercule Poirot
  </name>
  <conversionwarning>
    0
  </conversionwarning>
  <body>
    <figure>
      <caption>
      </caption>
    </figure>
    <emph3>
      Hercule Poirot
    </emph3>
    <p>
    </p>
    data
    <section>
      <title>
        Major novels
      </title>
      <p>
      </p>
      <p>
      </p>
    </section>
    <section>
    </section>
  </body>
</article>
```

Figure 6: Sample Wikipedia document

4.2. Query Collection

INEX 2006 queries specify both content (information to be retrieved found in the title field) and structural constraints (found in the castitle field). These

queries are referred to as CO + S (Content Only + Structure) queries. There are 125 CO+S queries in INEX 2006, whereas there are 130 CO+S queries in INEX 2007. Figure 7 represents a typical CO + S query.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="294" ct_no="16">
<title>user interface design usability guidelines</title>
<castitle>//article[about(.,user interface)]//section[about(.,design usability guidelines)]
</castitle>
<description>Find information about user interface design and usability guidelines
</description>
<narrative>Today there is a wide usage of graphical user interface in every
product we use, ranging from TV, DVD, PDA, computer software etc.
I would like to learn the guidelines of GUI design.
And to know what is considered "good" user interface,
with emphasis on human interaction and usability.
This could help me avoid the problem of creating
a user interface for a product or a computer application
that will be unusable for the average user.
</narrative>
<ontopic_keywords>HIG heuristic evaluation</ontopic_keywords>
</inex_topic>
```

Figure 7: Sample query (topic) [Topic id = 294]

4.3 Relevance Assessments

INEX provides relevance assessments against which results are evaluated. All INEX participants contribute towards the creation of these assessments. The process is as follows. Once the queries (topics) are finalized, INEX retrieves for each query a number of top-ranked articles (using their own TopX retrieval engine). Several topics and the associated articles are then returned to each participating group. Using the web-based tool provided by INEX, participants read each article and mark the relevant portion(s) of text in it. They are also asked to mark the best entry point for each article they have judged relevant.

Relevance was defined in INEX 2005 based on two factors, namely, *specificity* and *exhaustivity*. *Exhaustivity* describes the extent to which the document element discusses topic. [11]. *Specificity* describes the extent to which the document element focuses on the topic. [11]

In INEX 2006, only specificity is used. It's measured as the ratio of the highlighted (selected) length of an element compared to its the total length. Its value lies in the closed interval [0.0, 1.0]. Exhaustivity can have two values, namely 0 and 2. A value of 2 indicate the element is exhaustive and value of 0 indicates the element is to be ignored. A document with values of [1.0, 2] for specificity and exhaustivity is considered highly relevant.

4.4 Evaluation Measures

All information in this section is based on [12] by Kazai and Lamas. For the purpose of evaluation, the eXtended Cumulative Gain (XCG) measures are used. XCG is an extension of the cumulated gain or CG-based metrics which consider the dependency of XML elements within the evaluation. The XCG measure includes both the user-oriented measure of nxCG (normalized extended cumulative gain) and the system-oriented ep/gr (effort-precision / gain recall) measure.

These metrics are evaluated against a specific recall-base. Recall-base refers to the collection of assessments within the test collection that forms the ground-truth for the evaluation experiments [12]. Depending on the usage, either a full recall-base or an ideal recall-base is used. The full recall-base is defined as the one in which overlapping elements are allowed; (e.g.,

123/article[1]/body[1] and *123/article[1]/body[1]/section[1]*). The ideal recall-base is defined as the one in which only a single element along a path is allowed (i.e., no overlap is present).

xCG can be calculated as a cumulative gain of a result vector. If a ranked list of elements is returned for a particular query, we can compute the xCG value at a specific value of rank, say i . The xCG value at that rank is denoted by $xCG[i]$.

$$xCG[i] := \sum_{j=1}^i xG[j]$$

The nxCG value at a particular rank is given as the ratio of the xCG value at that rank for a run being evaluated to the xCG value at that rank for the ideal run.

$$nxCG[i] := \frac{xCG[i]}{xCI[i]}$$

where xCI is the xCG value for the ideal run. Results are typically evaluated with this metric at $i = 5, 10, 25$ and 50 . The nxCG metric is used to evaluate the results of the *Focused* subtask.

On the other hand, the effort-precision gain-recall metric evaluates the effort required of the user to reach a given level of cumulated gain when scanning a given ranking compared to the ideal ranking, i.e.,

$$ep[r] := \frac{i_{ideal}}{i_{run}}$$

where i_{ideal} is the rank position at which the cumulative gain of r is reached by the ideal curve and i_{run} is the rank position at which the cumulative gain of r is reached by the system run. A score of 1 reflects ideal performance.

Effort precision can be calculated at arbitrary gain-recall points, where gain-recall is calculated as:

$$gr[i] := \frac{xCG[i]}{xCI[n]} = \frac{\sum_{j=1}^i xG[j]}{\sum_{j=1}^n xI[j]}$$

where n is the total number of documents in the recall-base. The output for the *Thorough* subtask is evaluated using MAep or non-interpolated mean average effort precision. It is calculated by averaging the effort-precision values measured at 1500 which is a defined recall-point.

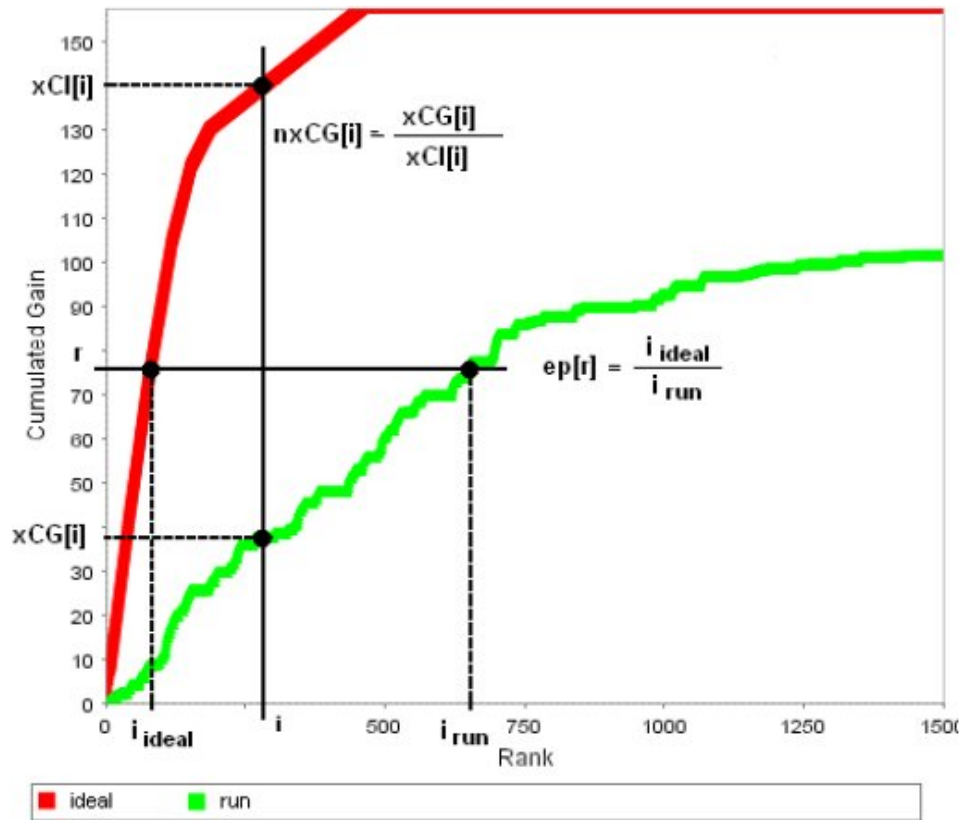


Figure 8: Calculation of nxCG and effort-precision (ep) [[12], page 7].

5. Experiments, Observations and Analysis

As a part of this thesis, the adaptation of Flex to semi-structured documents was carried out. The following section explains the details of the experiments carried out to verify that Flex was successfully adapted.

5.1 Procedure to Validate Flex

As a prerequisite to running Flex, all the terminal nodes (paragraph + mt elements) required to seed Flex are generated. After retrieval against the paragraph + mt index, Flex is seeded with n (a varying number of) terminal nodes. Once the seeded documents are identified, Flex generates the higher level elements, computes the correlation score with the query and generates a ranked list of elements (The correlation scores depend on various parameters provided in configuration file for Flex [refer to appendix A.1]).

The set of 1500 top ranked elements from the all-element retrieval using the best values of slope and pivot is used as the baseline in all the experiments. To verify the correct adaptation of Flex, Flex must be seeded with the terminal nodes identifying each of the documents whose elements constitute the baseline. It is found that a minimum of 123,103 top ranking terminal nodes are needed for seeding Flex to generate a baseline-compatible output. For comparison purposes, Flex is seeded with 125,000 terminal nodes per-query. Flex is then run to populate the documents identified by the terminal nodes mentioned above. To maintain compatibility, the values of slope and pivot used for Flex are the same as

those used for the corresponding all-element retrieval. The evaluation results are presented in Tables 8 to 13. The outputs are evaluated using EvalJ (the configuration files for the runs are found in the Appendix A.2). EvalJ uses the relevance assessments for computing the MAep and nxCg values while evaluating the Thorough and Focused outputs. Two versions of these assessments are used; version 4 and version 5. Version 4 has 111 queries in the assessment set. This version ignores the different link tags in relevant xml documents. Version 5 has 114 queries in the assessment set. This version includes the link tags in relevant xml documents.

5.2 Results, Observations and Conclusions

In this section the experimental results are tabularized. The observations and the conclusions made using those results are also presented. Here n , the number of terminal nodes seeded to Flex, is 125,000, as shown in Tables 8-13.

Table 8: Slope = 0.12, Pivot = 38, Task: Thorough, Assessment Version: v4

N	MAep@ 10	MAep@ 20	MAep@ 50	MAep@ 100	MAep@ 500	MAep@ 1500
baseline	0.0085	0.0122	0.0185	0.0238	0.0325	0.0356
Flex	0.0085	0.0122	0.0185	0.0238	0.0325	0.0356

Table 9: Slope = 0.12, Pivot = 38, Task: Thorough, Assessment Version: v5

N	MAep@ 10	MAep@ 20	MAep@ 50	MAep@ 100	MAep@ 500	MAep@ 1500
baseline	0.0045	0.0065	0.0098	0.0126	0.0171	0.0188
Flex	0.0045	0.0065	0.0098	0.0126	0.0171	0.0188

Table 10: Slope = 0.12, Pivot = 38, Task: Focused (Off),

Assessment Version: v4

n	nxCG@	nxCG@	nxCG@	nxCG@
	5	10	25	50
baseline	0.3653	0.3163	0.2575	0.2078
Flex	0.3653	0.3163	0.2575	0.2078

Table 11: Slope = 0.12, Pivot = 38, Task: Focused (Off),

Assessment Version: v5

n	nxCG@	nxCG@	nxCG@	nxCG@
	5	10	25	50
baseline	0.3625	0.3134	0.2538	0.2042
Flex	0.3625	0.3134	0.2538	0.2042

Table 12: Slope = 0.12, Pivot = 38, Task: Focused (On),

Assessment Version: v4

n	nxCG@	nxCG@	nxCG@	nxCG@
	5	10	25	50
baseline	0.3233	0.2795	0.2275	0.196
Flex	0.3233	0.2795	0.2275	0.196

Table 13: Slope = 0.12, Pivot = 38, Task: Focused (On),

Assessment Version: v5

n	nxCG@	nxCG@	nxCG@	nxCG@
	5	10	25	50
baseline	0.3241	0.2787	0.222	0.1815
Flex	0.3241	0.2787	0.222	0.1815

From these tables we observe the following:

1. From Tables 8 to 13 it is observed that the Flex output for the *Thorough* and *Focused* tasks is same as for the corresponding all-element output.
2. Flex generates exactly the same correlation scores as those of the all-element index. This may be observed by looking at the actual outputs. (A sample of both outputs is present in Appendix A.3.)
3. Large numbers of terminal elements are required to seed Flex for ensuring the generation of results compatible with the base-line.

We conclude from this data that

1. Using our method Flex has been successfully adapted to handle semi-structured documents.
2. For the semi-structured documents the terminal node vectors can be effectively used to generate all the non-terminal node vectors.
3. Since the correlations from Flex are identical to those in all-element retrieval, the only requirement for obtaining the same output from Flex and the all-element approach is to make sure that Flex is seeded with terminal elements identifying every document whose elements form the all-element output.
4. To generate results identical to the baseline, a large number of terminal nodes is required to seed Flex. This indicates that there are some documents in the baseline with terminal nodes which correlate poorly with the query.

5.3 Determining a Realistic Value of n

We now consider the issue of determining realistic values of n , (That is, the minimal number of terminal nodes required to seed Flex and produce results at least as good as that produced by the corresponding all-element retrieval.)

As a part of this experiment, Flex is seeded with varying values of n . The outputs generated in this fashion are evaluated using EvalJ and compared with the baseline. We use two sets of values of slope and pivot. The values tuned for the Wikipedia collection are given in Table 3.4. Another set of values used for experimentation are the TREC values (i.e., slope = 0.2 and pivot = 110). Results are evaluated using 2 versions of the relevance assessments (i.e., version 4 and version 5).

5.4 Results, Observations and Conclusions

An initial retrieval is done using the paragraph + mt index. (This retrieval uses the slope and pivot values for paragraph + mt index [see Table 3.4].) For each experiment, Flex uses the same slope and pivot values used for the retrieval against the all-element index (base case) [see Table 3.4].

Tables 14 to 17 show MAep evaluation at different values of n ; all pertain to the Thorough task. Here the highest value at each rank is in bold type.

Table 14: Slope = 0.12, Pivot = 38, Task = Thorough, Assessment version = 4

n	MAep@ 10	MAep@ 20	MAep@ 50	MAep@ 100	MAep@ 500	MAep@ 1500
baseline	0.0085	0.0122	0.0185	0.0238	0.0325	0.0356
1000	0.0085	0.0122	0.0186	0.0238	0.0325	0.0357
500	0.0085	0.0122	0.0186	0.0239	0.0326	0.0371
250	0.0085	0.0122	0.0185	0.0238	0.0332	0.0396
100	0.0086	0.0122	0.0185	0.0237	0.0361	0.0415
50	0.0086	0.0121	0.0184	0.0245	0.0385	0.0401
25	0.0083	0.0119	0.0190	0.0263	0.0364	0.0365
10	0.0083	0.0126	0.0208	0.0263	0.0296	0.0296
5	0.0083	0.0133	0.0210	0.0242	0.0250	0.0250
1	0.0084	0.0120	0.0149	0.0154	0.0154	0.0154

Table 15: Slope = 0.12, Pivot = 38, Task = Thorough, Assessment version = 5

n	MAep @ 10	MAep @ 20	MAep@ 50	MAep @ 100	MAep @ 500	MAep @ 1500
baseline	0.0045	0.0065	0.0098	0.0126	0.0171	0.0188
1000	0.0045	0.0065	0.0098	0.0126	0.0171	0.0188
500	0.0045	0.0065	0.0098	0.0126	0.0172	0.0196
250	0.0045	0.0065	0.0098	0.0125	0.0175	0.0209
100	0.0046	0.0065	0.0098	0.0125	0.0190	0.0219
50	0.0045	0.0064	0.0097	0.0129	0.0203	0.0212
25	0.0044	0.0063	0.0100	0.0138	0.0192	0.0192
10	0.0044	0.0067	0.0111	0.0140	0.0158	0.0158
5	0.0045	0.0072	0.0113	0.0129	0.0133	0.0133
1	0.0043	0.0061	0.0075	0.0078	0.0078	0.0078

Table 16: Slope = 0.2, Pivot = 110, Task = Thorough, Assessment version = 4

n	MAep @ 10	MAep @ 20	MAep@ 50	MAep @ 100	MAep @ 500	MAep @ 1500
baseline	0.0083	0.0119	0.0183	0.0236	0.0322	0.0353
1000	0.0083	0.0119	0.0184	0.0236	0.0322	0.0355
500	0.0083	0.0119	0.0184	0.0237	0.0324	0.0368
250	0.0082	0.0119	0.0184	0.0236	0.0330	0.0391
100	0.0082	0.0119	0.0183	0.0237	0.0357	0.0410
50	0.0082	0.0119	0.0183	0.0242	0.0380	0.0396
25	0.0080	0.0118	0.0186	0.0260	0.0359	0.0361
10	0.0081	0.0126	0.0207	0.0260	0.0293	0.0293
5	0.0084	0.0132	0.0209	0.0241	0.0249	0.0249
1	0.0075	0.0105	0.0135	0.0140	0.0140	0.0140

Table 17: Slope = 0.2, Pivot = 110, Task = Thorough, Assessment version = 5

n	@ 10	@ 20	@ 50	@ 100	@ 500	@ 1500
baseline	0.0044	0.0063	0.0097	0.0124	0.0170	0.0186
1000	0.0044	0.0063	0.0097	0.0124	0.0170	0.0187
500	0.0044	0.0064	0.0097	0.0125	0.0171	0.0194
250	0.0044	0.0064	0.0097	0.0124	0.0174	0.0206
100	0.0044	0.0063	0.0097	0.0125	0.0188	0.0216
50	0.0044	0.0063	0.0096	0.0127	0.0200	0.0209
25	0.0043	0.0063	0.0098	0.0137	0.0189	0.0190
10	0.0043	0.0067	0.0110	0.0138	0.0156	0.0156
5	0.0045	0.0072	0.0112	0.0128	0.0132	0.0132
1	0.0038	0.0053	0.0068	0.0071	0.0071	0.0071

The Focused subtask requires that all non-overlapping nodes from the XML document be returned in response to a query. To produce this output we

filter the output for the Thorough subtask by removing all overlapping elements. Tables 18 to 25 present equivalent results for the Focused task. n refers to the number of terminal elements used to seed Flex. To evaluate Focused runs, EvalJ can be configured with two options, i.e., *overlap = on* and *overlap = off*. EvalJ uses the ideal-recall base [see section 4.4] when option *overlap = on* is specified and uses the full recall base [see section 4.4] when the option *overlap = off* is specified.

Table 18: Slope = 0.12, Pivot = 38, Task = Focused, Assessment version = 4

Overlap = off

N	nxCG@ 5	nxCG@ 10	nxCG @ 25	nxCG @ 50
baseline	0.3653	0.3163	0.2575	0.2078
1000	0.3674	0.3151	0.2562	0.2072
500	0.3633	0.3125	0.2523	0.2041
250	0.3622	0.3128	0.2516	0.2029
100	0.3622	0.3128	0.2514	0.2053
50	0.3622	0.3136	0.2563	0.2254
25	0.3646	0.3188	0.2824	0.255
10	0.3766	0.3506	0.3126	0.2542
5	0.3963	0.372	0.318	0.2149
1	0.3376	0.2652	0.1711	0.1017

Table 19: Slope = 0.12, Pivot = 38, Task = Focused, Assessment version = 5
Overlap = off

N	nxCG @ 5	nxCG @ 10	nxCG @ 25	nxCG @ 50
baseline	0.3625	0.3134	0.2538	0.2042
1000	0.3645	0.3122	0.2526	0.2036
500	0.3605	0.3098	0.2487	0.2005
250	0.3595	0.3124	0.2481	0.1993
100	0.3595	0.31	0.2479	0.2019
50	0.3595	0.3108	0.2529	0.2221
25	0.3619	0.3159	0.2791	0.2518
10	0.3736	0.3477	0.3104	0.2508
5	0.3933	0.3684	0.3135	0.2111
1	0.3321	0.261	0.1686	0.0992

Table 20: Slope = 0.12, Pivot = 38, Task = Focused, Assessment version = 4
Overlap = on

N	nxCG @ 5	nxCG @ 10	nxCG @ 25	nxCG @ 50
baseline	0.3233	0.2795	0.2275	0.196
1000	0.3254	0.2774	0.2262	0.1956
500	0.3213	0.2753	0.2222	0.1916
250	0.3202	0.2755	0.2211	0.1902
100	0.3202	0.2755	0.2209	0.1916
50	0.3202	0.2764	0.2247	0.1987
25	0.3226	0.2797	0.2354	0.1966
10	0.3311	0.2835	0.2109	0.1625
5	0.3256	0.2576	0.1911	0.126
1	0.2223	0.1481	0.0842	0.0498

Table 21: Slope = 0.12, Pivot = 38, Task = Focused, Assessment version = 5

Overlap = on

N	nxCG @ 5	nxCG @ 10	nxCG @ 25	nxCG @ 50
baseline	0.3241	0.2787	0.222	0.1815
1000	0.3261	0.2766	0.2207	0.1811
500	0.3221	0.2745	0.2169	0.1776
250	0.3211	0.2748	0.2159	0.1764
100	0.3211	0.2748	0.2157	0.1779
50	0.3211	0.2756	0.2197	0.1854
25	0.3234	0.2789	0.2307	0.1846
10	0.3333	0.2852	0.2105	0.1542
5	0.3303	0.2601	0.1894	0.118
1	0.2249	0.1505	0.0843	0.0467

Table 22: Slope = 0.2, Pivot = 110, Task = Focused, Assessment version = 4

Overlap = off

N	nxCG @ 5	nxCG @ 10	nxCG @ 25	nxCG @ 50
baseline	0.3511	0.3171	0.2519	0.2061
1000	0.3498	0.3088	0.2492	0.2025
500	0.3452	0.3044	0.2442	0.1991
250	0.3452	0.3060	0.2443	0.1995
100	0.3452	0.3069	0.2468	0.2028
50	0.3450	0.3093	0.2540	0.2236
25	0.3501	0.3178	0.2800	0.2546
10	0.3633	0.3491	0.3153	0.2542
5	0.3945	0.3745	0.3192	0.2149
1	0.2816	0.2332	0.1570	0.0952

Table 23: Slope = 0.2, Pivot = 110, Task = Focused, Assessment version = 5
Overlap = off

N	nxCG @ 5	nxCG @ 10	nxCG @ 25	nxCG @ 50
baseline	0.3469	0.3136	0.2489	0.2023
1000	0.3456	0.3052	0.2458	0.1991
500	0.3412	0.301	0.2409	0.1957
250	0.3412	0.3026	0.241	0.1961
100	0.3412	0.3034	0.2435	0.1993
50	0.341	0.3066	0.2508	0.2202
25	0.3459	0.3149	0.2767	0.2514
10	0.3606	0.3453	0.3131	0.2507
5	0.3915	0.3709	0.3147	0.2111
1	0.2775	0.2296	0.154	0.0926

Table 24: Slope = 0.2, Pivot = 110, Task = Focused, Assessment version = 4
Overlap = on

N	nxCG @ 5	nxCG @ 10	nxCG @ 25	nxCG @ 50
baseline	0.3163	0.2779	0.2244	0.1967
1000	0.3132	0.2686	0.2204	0.1918
500	0.3068	0.2643	0.2151	0.1879
250	0.3068	0.2659	0.2144	0.1878
100	0.3068	0.2664	0.2178	0.1887
50	0.3066	0.2687	0.2226	0.1959
25	0.3116	0.2767	0.233	0.1965
10	0.3214	0.2792	0.2112	0.1633
5	0.322	0.261	0.1923	0.126
1	0.1861	0.1359	0.0781	0.048

Table 25: Slope = 0.2, Pivot = 110, Task = Focused, Assessment version = 5
Overlap = on

N	nxCG@ 5	nxCG @ 10	nxCG @ 25	nxCG @ 50
baseline	0.3155	0.2762	0.2196	0.1819
1000	0.3124	0.267	0.2155	0.1775
500	0.3063	0.2628	0.2102	0.174
250	0.3063	0.2643	0.2099	0.174
100	0.3063	0.2652	0.2123	0.1748
50	0.306	0.2683	0.2175	0.1826
25	0.311	0.2758	0.2284	0.1843
10	0.3238	0.2801	0.2108	0.1548
5	0.3268	0.2634	0.1906	0.118
1	0.1897	0.1375	0.0772	0.0445

Observations are indicated below:

1. The results for Thorough and Focused [refer Tables 14 to 25] runs show that Flex performs better than the all-element index at various values of n (especially the lower values of n).
2. Flex tends to reach its best value at a smaller value of n and degrade as the value of n is increased thereafter.
3. Even though the same terminal nodes are used to seed Flex, different results are generated with different values of slope and pivot. Better results are obtained by using the tuned values.

Conclusions are indicated below:

Flex produces better results than the all-element index at lower values of n , the number of terminal nodes used to seed Flex. Even though Flex and all-element retrieval produce the identical correlations for each query and the element vector, Flex produces a rank-ordered list of elements which is better than that produced by the all-element retrieval.

The reason for the superior performance of Flex is explained as follows. The document set generated by Flex is determined by the set of top- n -ranked terminal nodes from the initial retrieval. The higher the similarity score of the terminal node with the query the higher is its possibility of being relevant. Flex then generates only the containing elements from the documents identified by the highly ranking terminal nodes.

It can therefore be said that Flex produces a subset of the elements produced by the all-element retrieval. But the basis for this subset is the set of initial terminal nodes. All-element retrieval gives us a global view of the entire document set. All-element retrieval produces a rank-ordered list of the most highly correlating elements in the entire set of elements. But as our results show, some elements in this list may not in fact be relevant. Flex, on the other hand, is limited to a smaller set of elements, because these elements can only come from documents which contain elements (terminal nodes) with the highest possible correlation with the query. Our results show that Flex, at lower values of n (i.e., $n = 100$), consistently produces more relevant elements in the top N elements than does the corresponding all-element retrieval.

5.5 Time and Space Requirement of Flex

The following results show the time and space requirements of Flex. The time to run Flex is the time taken by Flex to compute the non-terminal element vectors, compute correlations and generate the ranked order list of elements (averaged over all queries).

Results:

When Flex is run as described in Section 5.3, the tree generation statistics are computed. These statistics are presented in Tables 26 and 27. In these Tables n = number of terminal nodes (paragraphs + mt) used to seed Flex, Flex is asked to generate 2500 elements per query, and all mt elements are removed from the output. This guarantees a result set of 1500 elements.

**Table 26: Tree generation statistics, Paragraph retrieval: Slope = 0.12, Pivot = 18;
Flex: Slope = 0.12, Pivot = 38**

N	Average number of trees per query	Range of trees built	Average time per query (seconds)	Number of MT elements removed
1000	731.73	41-905	1.504	30588
500	370.94	41-461	0.784	30827
250	187.06	41-241	0.408	27224
200	150.13	41-193	0.328	24338
150	112.87	41-146	0.246	19818
100	076.16	41-97	0.184	14605
50	038.68	23-49	0.096	8419
25	019.57	12-25	0.048	4745
10	008.14	3-10	0.024	2399
5	004.28	1-5	0.016	1423
1	001.00	1-1	0.000	447

**Table 27: Tree generation statistics, Paragraph retrieval: Slope = 0.12, Pivot = 18;
Flex: Slope = 0.2, Pivot = 110**

N	Average number of trees per query	Range of trees built	Average time per query (seconds)	Number of MT elements removed
1000	731.73	41-905	1.488	28857
500	370.94	41-461	0.808	30056
250	187.06	41-241	0.400	27050
200	150.13	41-193	0.328	24245
150	112.87	41-146	0.248	19797
100	076.16	41-97	0.184	14605
50	038.68	23-49	0.096	8419
25	019.57	12-25	0.040	4745
10	008.14	3-10	0.024	2399
5	004.28	1-5	0.016	1423
1	001.00	1-1	0.000	447

The space requirement for the all-element and flexible retrieval is shown in Table 28.

Table 28: Data-space requirements

	All-element retrieval	Flexible retrieval
Dictionaries	0.860G	0.86G
Inverted Index	4.100G	5.50G
Doc vectors	4.100G	2.00G
Collstats	2.653G	2.100G
n_stats_vectors	-	0.400G
Total	11.710G	10.860G

Observations and Conclusions:

1. From Tables 26 and 27, it can be seen that the time required for performing flexible retrieval is such that it can reasonably be used in an operational environment.
2. The time required by Flex increases as the number of terminal nodes seeded to Flex increases.
3. The space requirement for flexible retrieval is less than for all-element retrieval, hence we conclude that Flex is more efficient than the all-element retrieval in terms of space requirements.

6. Future Work

In this section, we make suggestions for improvements and future work. First, an improved method of computing the tuned values of slope and pivot may be tried. The entire setup used in the experimentation can be made automatic with a single script for the entire process. Different weighting schemes may be implemented in Flex and their performances may be measured.

Second, improved approach using the DOM (Document Object Model) APIs may be developed for the parsing of documents. The DOM approach is not only bound to improve the accuracy of parsing at various levels of granularity but will also make the process of parsing really simple. And lastly, an approach of generating the required document schema on the fly may be developed. Such an approach will reduce the space required for storing the schemas and make the overall process more compact. This approach may be easily developed by integrating the DOM code in Flex.

References

- [1] Connolly, T., and Begg, C. *Database Systems*, 4th ed. Harlow, England: Pearson Education Ltd., 2005. 1004-1144.
- [2] Initiative for the Evaluation of XML Retrieval (INEX) <http://inex.is.informatik.uni-duisburg.de/2006>
- [3] Khanna, S. Design and Implementation of a Flexible Retrieval System, MS Thesis, University of Minnesota Duluth, 2005.
- [4] Salton, G., Wong, A., and Yang, C. A vector space model for information retrieval. *JASIS*, 18(11):613-620, 1975.
- [5] Crouch, C. Dynamic element retrieval in a structured environment. *ACM TOIS*, 24(4): 437-454, 2006.
- [6] Ganapathibotla, M. Query Processing in a Flexible Retrieval Environment, MS Thesis, University of Minnesota Duluth, 2005.
- [7] Crouch, C., Khanna, S., Potnis, P., Doddapaneni, N. The dynamic retrieval of XML elements. *Advances in XML Information Retrieval and Evaluation: 4th International Workshop of the Initiative for the Evaluation of XML Retrieval*, INEX 2005, Dagstuhl, Germany, 28-30, Springer-Verlag, LNCS 3977, 268-281, 2006.
- [8] Bakshi, V. Flexible Retrieval for the Semi-Structured Documents. MS Thesis, University of Minnesota Duluth, 2006.
- [9] Singhal, A., Buckley, C., Mitra, M. Pivot document length normalization. In *Proceedings of the 19th Annual International ACM Special Interest Group in Information Retrieval (SIGIR) Conference*. Zurich, Switzerland. 19-21, 1996.

- [10] Malik, V. Impact of Terminal Node Processing on Element Retrieval. MS Thesis, University of Minnesota Duluth, 2007.
- [11] Kazai, G; Lalmas, M; Piwowarski, B. INEX 2004 Relevance Assessment Guide. INEX 2004 Workshop Preproceedings, Schloss Dagstuhl, December 6-8, 2004.
- [12] G. Kazai and M. Lalmas : INEX 2005 Evaluation Metrics. INEX 2005.

Appendix A

A.1 Configuration File for Flex

Figure 9 shows a typical configuration file for Flex. The parameters which are mentioned in <..> are the variable parameters and can be changed as per the use's need. The other parameters are variable but were kept constant during the experimentation phase for the purpose of this thesis.

```
sim cosine = 1 inner = 2 wt avg = 1 Lnu = 2
DOC_INDEX_PATH <Path to the doc.nnn file being used>
QUERY_OPTION 2
QUERY_PARA_INDEX_PATH <Path to the query.nnn file being used>
OUTPUT_PATH <Path for the output file to be generated along with the file name>
RESULT_TREES_PATH <Path to folder which contains the seeded document trees (schemas)>
NUM_OUTPUT_ELEMS <Total output elements expected>
SIZE_OUTPUT_ELEMS 0
SIM_OUTPUT_ELEMS 0
SIM 2
WT 2
SLOPE_PARA <Paragraph slope>
SLOPE_SEC <Section slope>
SLOPE_BDY <Article slope>
SLOPE_ALLELEMS <All-element slope>
PIVOT_PARA <Paragraph pivot>
PIVOT_SEC <Section pivot>
PIVOT_BDY <Article pivot>
PIVOT_ALLELEMS <All-element pivot>
NUM_CTYPES 1
CTYPE_WTS 1
N_PARA_VALUES <number of paragraph vectors>
N_SEC_VALUES <number of section vectors>
N_BDY_VALUES <number of article vectors>
N_STATS_FOLDER <Folder where the binary n_stat file is present>
N_STATS_FILES ctype0
```

Figure 9: A typical configuration file for Flex

A.2 Configuration Files for EvalJ

The Figure 10 shows the configuration file for evaluating the runs for the Thorough task using EvalJ.

```
TASK: Thorough
METRICS: ep/gr
ALPHA: 1.0
QUANT_FUNCTIONS: gen
OVERLAP: off
ASSESSMENTS_DIR: <Directory containing the rel-assesments>
SUBMISSIONRUNS_DIR: <Directory containing the output files along with the recognisable file extension>
RESULTS_DIR: <Directory for result generation>
INEXDOCCOLL_DIR: <Path of the folder containing the entire corpus in a single folder>
```

Figure 10: A typical configuration file for EvalJ (Thorough subtask)

The Figure 11 shows the configuration file for evaluating the runs for the Focused task using EvalJ.

```
TASK: Focused
METRICS: nxCG
ALPHA: 1.0
QUANT_FUNCTIONS: gen
DCV: 5, 10, 25, 50
OVERLAP: <(off / on) specify according to requirement>
ASSESSMENTS_DIR: <Directory containing the rel-assesments>
SUBMISSIONRUNS_DIR: <Directory containing the output files along with the recognisable file extension>
RESULTS_DIR: <Directory for result generation>
INEXDOCCOLL_DIR: <Path to the folder containing the entire corpus in a single folder>
```

Figure 11: A typical configuration file for EvalJ (Focused subtask)

The parameters given in <..> are the variable parameters and should be changed according to the user's need.

A.3 Sample Output from Flex and All-element Retrieval

The Figure 12 shows a sample all-element output whereas Figure 13 shows a sample Flex output. (The output presented is a partial output.)

```
1 288363/article[1]/body[1]/section[3] 40.7043
1 4905/article[1]/body[1]/p[5] 38.9533
1 734679/article[1]/body[1]/section[1]/p[4] 37.8511
1 288363/article[1]/body[1]/section[3]/p[2] 37.7817
1 140367/article[1]/body[1]/section[1]/section[10]/p[2] 37.6558
1 632889/article[1]/body[1]/p[2] 36.9786
1 140367/article[1]/body[1]/section[1]/section[10] 36.5637
1 22936/article[1]/body[1]/section[2]/p[4] 36.0115
1 62519/article[1]/body[1]/section[6] 35.6326
1 258783/article[1]/body[1]/section[1]/section[1]/p[2] 35.5745
1 2281211/article[1]/body[1]/p[2] 35.0684
1 62519/article[1]/body[1]/section[6]/normallist[1] 34.7183
1 4905/article[1]/body[1]/p[2] 34.2539
1 258783/article[1]/body[1]/section[1]/section[1] 33.5555
1 258783/article[1]/body[1]/section[1] 33.5555
1 2281211/article[1]/body[1] 32.8941
```

Figure 12: Sample all-element output

```
1 288363/article[1]/body[1]/section[3] 40.7043
1 4905/article[1]/body[1]/p[5] 38.9533
1 734679/article[1]/body[1]/section[1]/p[4] 37.8511
1 288363/article[1]/body[1]/section[3]/p[2] 37.7817
1 140367/article[1]/body[1]/section[1]/section[10]/p[2] 37.6558
1 632889/article[1]/body[1]/p[2] 36.9786
1 140367/article[1]/body[1]/section[1]/section[10] 36.5637
1 22936/article[1]/body[1]/section[2]/p[4] 36.0115
1 62519/article[1]/body[1]/section[6] 35.6326
1 258783/article[1]/body[1]/section[1]/section[1]/p[2] 35.5745
1 2281211/article[1]/body[1]/p[2] 35.0684
1 62519/article[1]/body[1]/section[6]/normallist[1] 34.7183
1 4905/article[1]/body[1]/p[2] 34.2539
1 258783/article[1]/body[1]/section[1]/section[1] 33.5555
1 258783/article[1]/body[1]/section[1] 33.5555
1 2281211/article[1]/body[1] 32.8941
```

Figure 13: Sample Flex output

A.4 Results for the All-In-Context Subtask for INEX 2006

After applying the technique discussed in section 3.8, the output for the All-In-Context task was evaluated using the script provided on the INEX 2006 website. The results for the same are given in Tables 29 and 30. The best results at various points are indicated by bold type. Here n is the number of terminal nodes used to seed Flex.

Table 29: Slope = 0.12, Pivot = 38, Task = All-In-Context, Assessment version = 4

n	gP[5]	gP[10]	gP[25]	gP[50]	MAgP
1500	0.3032	0.2440	0.1718	0.1257	0.1250
500	0.3032	0.2440	0.1718	0.1257	0.1250
250	0.3032	0.2440	0.1718	0.1257	0.1242
100	0.3032	0.2440	0.1718	0.1257	0.1162
50	0.3032	0.2440	0.1718	0.1257	0.1033
25	0.3032	0.2440	0.1718	0.0859	0.0850
10	0.3032	0.2440	0.0976	0.0488	0.0613
5	0.3032	0.1516	0.0606	0.0303	0.0429
1	0.0828	0.0414	0.0165	0.0082	0.0140

Table 30: Slope = 0.12, Pivot = 38, Task = All-In-Context, Assessment version = 5

n	gP[5]	gP[10]	gP[25]	gP[50]	MAGP
1500	0.3031	0.2439	0.1717	0.1257	0.1250
500	0.3031	0.2439	0.1717	0.1257	0.1250
250	0.3031	0.2439	0.1717	0.1257	0.1241
100	0.3031	0.2439	0.1717	0.1257	0.1162
50	0.3031	0.2439	0.1717	0.1257	0.1033
25	0.3031	0.2439	0.1717	0.0858	0.0849
10	0.3031	0.2439	0.0975	0.0487	0.0613
5	0.3031	0.1515	0.0606	0.0303	0.0429
1	0.0828	0.0414	0.0165	0.0082	0.0140

A.5 Results for the Best-In-Context Subtask for INEX 2006

After applying the technique discussed in Section 3.8, the output for the Best-In-Context task was evaluated using EvalJ. The results are given in Table 31. The best results at various points are indicated by bold type. Here n is the number of terminal nodes used to seed Flex.

Table 31: Slope = 0.12, Pivot = 38, Task = Best-In-Context, Assessment version = 5

n	BEPD @ 0.01	BEPD @ 0.1	BEPD @ 1.0	BEPD @ 10.0	BEPD @ 100.0
1500	0.1443	0.2246	0.3512	0.5414	0.7357
500	0.1254	0.1985	0.3131	0.4865	0.6658
250	0.1072	0.1701	0.2690	0.4199	0.5792
100	0.0746	0.1233	0.1979	0.3113	0.4358
50	0.0552	0.0936	0.1519	0.2395	0.3370
25	0.0351	0.0620	0.1025	0.1657	0.2381
10	0.0215	0.0376	0.0604	0.0971	0.1442
5	0.0135	0.0250	0.0388	0.0605	0.0891
1	0.0031	0.0064	0.0107	0.0167	0.0244