

1. Introduction

Information retrieval is a dynamic field for research and development. Information retrieval systems search for relevant information in large amounts of data (like the web) and retrieve this information in response to a user's query. With the advent of XML, data can be structured and represented as *elements*, which enables the retrieval system to return potentially relevant elements to the user.

In our research, we use the extended vector space model [Fox, 1983], which allows us to handle the different types of information that may be present in a document. Documents and queries are converted to vector form. The terms in the vectors are weighted using an appropriate weighting scheme. Each subvector (ctype) may also be weighted, depending upon its significance. The correlation of a document with a query is calculated using a similarity measure such as cosine or inner product.

Flexible retrieval is a method of retrieving the elements of a document at various levels of granularity [Khanna, 2005]. This process enables the user to retrieve the potentially relevant parts of a document (rather than for the document as a whole) at the time the query is processed.

We first create a base case to use for comparison purposes by indexing the document collection at various element levels (e.g., paragraphs, subsections, sections, and articles). The resulting all-element index is composed of vectors

representing paragraphs, subsections, sections and articles. These vectors are then weighted and retrieval is performed.

In dynamic retrieval, the document collection is indexed at the level of its smallest element (i.e., the paragraph). An initial retrieval is done against this index. Then using tree structures of the XML documents (which have been previously stored) along with the paragraphs, vectors for the parent elements are created and their similarities with the query are calculated. The result of each approach is a ranked list of elements, which is then compared to those produced by the base case. See [Khanna, 2005] for details.

This thesis focuses on the issues associated with moving flexible retrieval into a new domain, the Wikipedia collection. The major question is: Can our system of flexible retrieval, designed for the dynamic retrieval of elements in a specific environment (i.e., scientific articles such as IEEE publications with the very specific formats typical of such articles) be effectively applied in a very different domain (i.e., Wikipedia)? We use the Smart experimental retrieval system [Salton, 1971], and our own system for dynamic element retrieval, called Flex [Khanna, 2005]. The process begins by first identifying elements to be indexed at various levels, then parsing the XML collection into text elements (that can be indexed by Smart) and generating tree schemas for the documents (to be used for flexible retrieval) [Khanna, 2005].

2. Background

This section covers details about the document collection, query collection, evaluation metrics, retrieval engine and other software used for our research.

2.1 Wikipedia Document Collection

The document collection used in this research is the Wikipedia XML Corpus provided by INEX. The collection is available in eight different languages. We use the English language collection for our research. The size of this collection is 4.6 GigaBytes, with a total number of 659,388 documents. The number of categories for the collection is 113,483, with a mean number of categories for each document of 2.2849 (e.g., an article on “American Red Cross” would belong to 2 categories, namely, America and Red Cross). The average number of XML nodes present in an article is 161.35. The average depth of an element is 6.72. [Denoyer, Gallinari, 2006]

2.2 Topic Collection

The topic collection consists of 125 CO + S (Content Only + Structure) topics. [Larsen, Trotman, 2006] Each topic has six components. These components are listed in Table 3. We use the <title> part of the topic for Content Only tasks and the <castitle> part for the Content And Structure tasks. Figure 1 contains an example topic from the 2006 topic collection

Description of topic components

<title>	Contains Content Only (CO) query
<castitle>	Contains Content and Structure (CAS) query
<description>	this field enables derivation of NLP queries
<narrative>	describes what is relevant and non-relevant to the query
<ontopic_keywords>	Contains terms that might be present in relevant documents
<offtopic_keywords>	Contains terms that might be present in non-relevant documents

Table 1

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="290" ct_no="9">
<title>"genetic algorithm"</title>
<castitle>//article[about(., "genetic algorithm")]</castitle>
<description>Find information about the history, algorithm, function, data structures and
implementation of genetic algorithms.</description>
<narrative>I am doing an experiment which needs to tune more than four parameters. I
was told that the genetic algorithm is a suitable method for this. I want to have an
overview of this type of algorithms, and especially I am interested in the algorithms,
functions, data structures and implementations of genetic algorithm. Relevant elements
should mention any of the above information of genetic algorithms. </narrative>
<ontopic_keywords>genetic algorithm; GA; algorithm</ontopic_keywords>
</inex_topic>

```

Figure 1. 290.xml

2.3 Relevance Assessments

Our system is evaluated using relevance assessments. This process is accomplished by distributing the topics and document collection among the INEX participants. Participants manually mark those elements of the document collection as relevant which correlate with the topic assigned to them. The relevance of an element is determined by two features: exhaustivity and specificity.

Exhaustivity is the extent to which an element covers the user's need in response to his/her query. An element that discusses precisely the solution to query is considered to be highly exhaustive, whereas an element that talks generally/vaguely about the query is partially exhaustive. At INEX 2005 a user can assign following exhaustivity values to an element [Kazai, Lalmas, 2005]:

Levels of exhaustivity

Exhaustivity = 2 means that element is highly exhaustive.
Exhaustivity = 1 means that element is partially exhaustive.
Exhaustivity = 0 means that element is not exhaustive.
Exhaustivity = ? means that element is exhaustive but too small to be considered relevant.

Table 2

Specificity refers to the length of the portion of the element that correlates with the answer to a query. An element whose entire content correlates with the solution to the query is considered to be highly specific. At INEX, specificity is a continuous variable ranging from 0 to 1. Specificity is determined automatically by INEX depending upon the length of an element marked by a user as relevant. An element whose entire content is marked relevant by the user has a specificity value of 1 and is considered highly specific. An unmarked element has a specificity value of 0. Hence, every document in the collection has values of exhaustivity and specificity for each specific query.

2.4 Evaluation Metrics

One of the metric used to evaluate results is the Recall-Precision metric *Inex-eval* (similar to *Smart-eval*). The metric gives precision at various recall points, where $\text{Precision} = (\text{number of relevant elements retrieved}) / (\text{number of elements retrieved})$ and $\text{Recall} = (\text{number of relevant elements retrieved}) / (\text{total number of relevant elements})$.

Inex-eval is executed using the *Evalj* software provided by INEX. The software requires relevance assessments (provided by INEX) and the retrieval results as input. (*Smart-eval* is executed using *Smart's* utility. It requires retrieval results and *qrels* as input. *Qrels* are the files created using relevance assessments. *Qrels* are in the format that is required by *Smart* to execute *Smart-eval*).

The second metric used for result evaluation is *XCG* (extended cumulative gain). It includes two measures: *nxCG* (normalized extended cumulative gain) and

ep/gr (effort-precision/gain-recall) [Kazai, Lalmas, 2005]. For a ranked list of elements, xCG at rank i is: $xCG[i] = \sum_{n=1}^i xG[n]$, where xG[i] is relevance score of the element at rank i. Therefore for a ranking list vector $xG = \langle 1,3,0,3,2,1,2,0 \rangle$, xCG vector will be $\langle 1,4,4,7,9,10,12,12 \rangle$. For every result an ideal gain vector (xI) can be calculated by placing the elements in decreasing order of relevance. Hence for the above list vector, $xI = \langle 3,3,2,2,1,1,0,0 \rangle$, the ideal cumulative gain vector is $xCI = \langle 3,6,8,10,11,12,12,12 \rangle$. The nxCG at a rank i is: $nxCG[i] = xCG[i]/xCI[i]$. Therefore value of nxCG denotes the gain accumulated by a system at rank i with respect to the ideal gain.

The ep/gr metric is used to evaluate the effort required by the system to achieve a particular value of gain, with respect to the ideal run. Therefore if the ideal system attains a gain value of k at rank r1, and our system attains same gain value at rank r2, the ep at gain k is $ep[k] = r1/r2$.

An ep value of 1 represents ideal performance. The values of gr at a rank i is computed as $gr[i] = xCG[i]/xCI[n]$, where n is the total number of relevant documents in the collection. The value of ep at a particular gr denotes the number of ranks a system needs to go through in order to achieve that value of gain when compared against an ideal ranking system.

The precision of the results can be evaluated using two quantizations: strict and generalized. For strict quantization, the relevance assessments and the qrels fed to the metric contain only highly correlated elements as relevant. Hence only

elements with an exhaustivity value of 2 are considered for evaluating results. This quantization is used to check the system's capability to produce the best elements at the top ranks. For evaluating results obtained from generalized quantization, the relevance assessments and the qrels fed to the metric must have elements with exhaustivity values of either 2 or 1. This quantization is used to check the system's capability to produce highly correlated as well as lower correlated elements at the appropriate ranks.

2.5. Smart

The retrieval system used for our research is Smart [Salton, 1971]. It provides us with the basic functionalities of indexing, term weighting, and retrieval. It supports the extended vector space model.

2.6. Flex

Flex [Khanna, 2005] is software that works from a single (paragraph) index created by Smart and produces results similar to that of a Smart retrieval against an all-element index. It also creates appropriate correctly weighted query vectors from the initial paragraph level query vectors [Murthy, 2006].

3. Processing the Document Collection

This chapter discusses the process of identifying indexable elements (and subvectors) present in the document collection. It also addresses the converting of XML documents to textual elements (that can be handled by Smart) and creating tree structures for the documents (as required by Flex).

3.1 Identification of Elements to be Indexed

Initially, the entire document collection is parsed to determine all the tags present in it along with their occurrence counts. In Wikipedia 1258 such tags were found. These tags were sorted and all tags with counts of less than 100 were discarded. The remaining tags were analyzed manually to determine those important in retrieval process. (See Appendix A.1 for a list of these tags along with their descriptions and counts in the document collection.)

Upon inspection three types of tags were removed:

- (i) Formatting tags which are used to represent data in some particular format (e.g., ``, `<i>`, etc.). So for an element `<i> america </i>`, we filter out tag `<i>` (that is used for italicizing the text).
- (ii) Small element tags which contain data but the content size is too small to be considered relevant and hence is not retrieved (e.g., `<item>`, `<row>`, `<column>`, etc.). Hence for an element: `<item> america </item>`, that is a child of a `<list>` element, we filter out tag `<item>` and retain its content in the parent element.
- (iii) Link tags which pointed to some other element in/outside the collection (e.g., `<collectionlink>`, `<outsidelink>`, `<unknownlink>`, etc.). Lets consider a `<collectionlink>` element:

<collectionlink xmlns:xlink=<http://www.w3.org/1999/xlink> xlink:type="simple" xlink:href="23040.xml"> political philosophies </collectionlink>. For the above element, the <collectionlink> tag is filtered out and its content is preserved in the body of all its parent elements.

Tags retained for indexing are listed in Table 3:

Tags retained to be indexed

paragraph level tags	<p>, <normallist>, , , <code>, <numberlist>, <caption>, <dl>, <dd>, <definitionlist>, <definitionitem>, <figure>, <table>, <image>, <title>, <name>
section level tags	<section>
article level tags	<body>

Table 3

See Table 4 for a description of all the tags that are retained for indexing.

3.2. Identification of Subvectors

The extended vector space model requires that all subvectors (ctypes) associated with an element be identified. Upon examining all the tags present in the collection, we conclude that there are two ctypes associated with every element.

These are:

- (i) <body> subvector that is composed of the content of the element.
- (ii) <name> subvector that is the title of the document to which the element belongs.

In the wikipedia collection, there were no other elements outside <body> except for <name>. Therefore, the number of subvectors was limited to two.

Description of all the tags that are indexed for Wikipedia collection

TAG	TAG DESCRIPTION
<body>	contains all the content of an article except article title
<section>	section in an article
<p>	paragraph in an article
<normallist>	List of items
	Ordered list
	Unordered list
<code>	Contains codes of programs
<numberlist>	List of numbers
<caption>	Caption of a figure or table
<dl>	Defines a definitionlist.
<dd>	Defines a term in a definitionlist.
<definitionlist>	List of Definitions
<definitionitem>	Items in a definitionlist.
<figure>	Encapsulates an image and its caption
<table>	Table
<image>	Image
<title>	Title of a section
<name>	Article title

Table 4

3.3 Generation of Tree Structures for Documents

In order to implement flexible retrieval using Flex, we require tree structures for the documents (schemas). A file must be created for every document that

contains all the elements of the document in a pre-order, parsed tree structure. Every element has associated with it the number of its siblings and children. Flex uses these tree structures to construct a document when a paragraph is fed to it as input. Figure 2 represents the tree generation process. See [Khanna, 2005] for details.

3.4 Parsing the XML Document Collection

In the final step of document processing, the collection is parsed into textual units (elements) that are fed to Smart. This is done using the parser written by us. The parser takes as input (i) The XML Document collection, and (ii) The Configuration file (defined below).

The document collection contains an XML file for each document to be parsed. The configuration file contains the tags of the list of elements to be indexed (e.g., <p>, <section>, <body>, etc.). The list of elements in the configuration file may vary depending upon the type of index required (e.g., paragraph, section, article and all-element index).

The parser works in two stages: file formatting and file parsing. In the file formatting stage, all unnecessary tags are removed from the XML file, but their data is retained. The output of this stage is a temporary text file containing flat text from XML documents for only the elements that need to be indexed. The file parsing stage takes as input the text file created by the file formatter. It scans all the elements in the input file, and stores the xpath of each element along with its text content in a separate file.

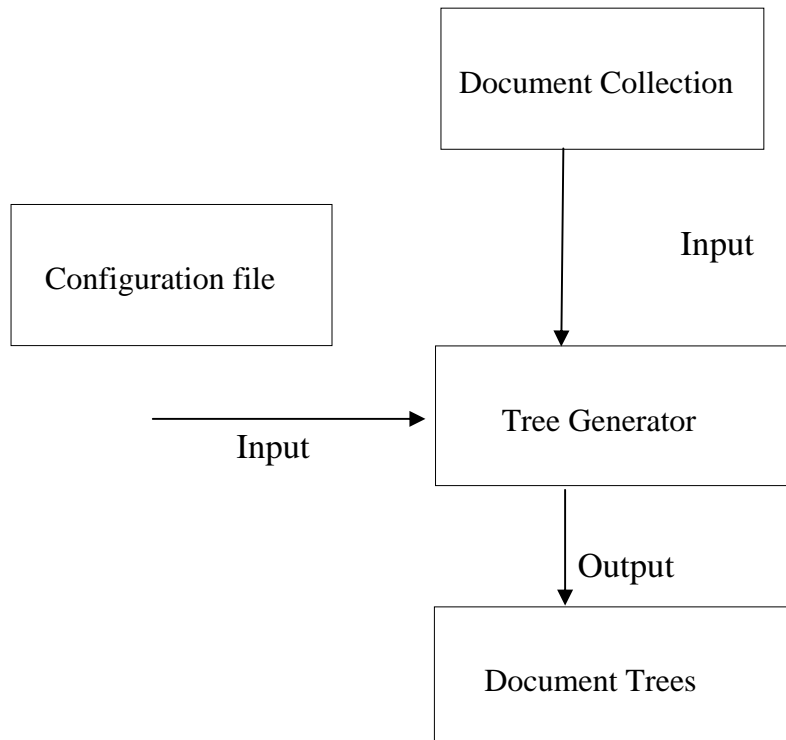


Figure 2. Tree generation

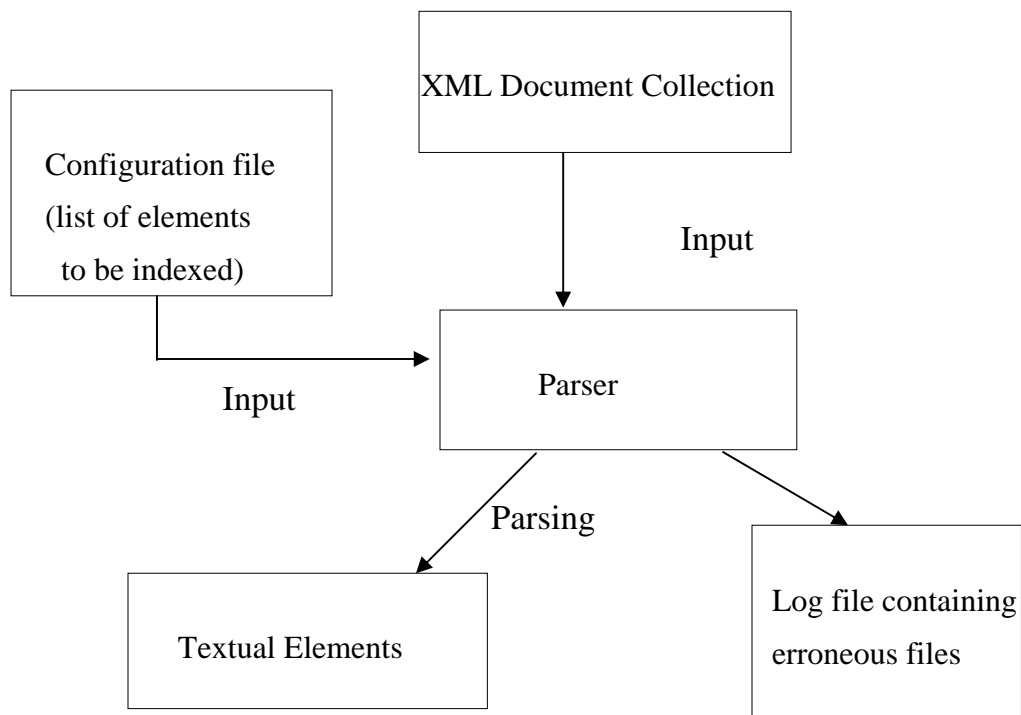


Figure 3. Parsing process

These files are used as input to our retrieval engine, Smart. Figure 3 provides a description of parsing process. Figure 4 shows example of an element created as a result of the parsing process.

```
<article>
<fno>902</fno>
<doi>/article[1]/body[1]/p[1]</doi>
<body>
An atom is a submicroscopic structure found in all ordinary
matter.
<name>
Atom
```

Figure. 4 Example of a resultant element from parsing

In addition to the parsing process, the parser also creates a log file that contains all the erroneous XML files encountered during parsing process. These errors may occur due to certain incomplete tag pairs in the XML files.

4. Processing the Topic Collection

In this chapter we discuss the processing of the query before it is fed as input to Smart. This involves query categorization, query decomposition, and query manipulation.

4.1 CO (Content Only) Processing

There are two key factors for CO query processing:

(1) Handling “+” and “-“ in a query

Lets take as an example query 378.xml from the INEX 2006 topic collection.

The <title> of the query is as follows:

```
<title>+rules "team sports" +indoor +ball world -football -basketball -handball  
-volleyball</title>.
```

The query can be interpreted in two ways: vague interpretation and strict interpretation [Doddapaneni, 2005].

In vague interpretation, terms followed by “-” are ignored and terms followed by “+” are retained in the query as hints. Hence on vaguely interpreting the above example query, the resulting query would be:

```
<title>+rules “team sports” +indoor +ball world</title>
```

In strict interpretation, our system decomposes the original query into children queries. Separate retrievals are run for each query, and these individual results are manipulated to get final results. Query 378.xml be decomposed into following sub-queries:

sub-query 1: <title>“team sports”</title>

sub-query 2: <title>rules</title>

sub-query 3: <title>indoor</title>

sub-query 4: <title>ball world</title>

sub-query 5: <title>football</title>

sub-query 6: <title>basketball</title>

sub-query 7: <title>handball</title>

sub-query 8: <title>volleyball</title>

Upon getting the results for all the queries, we take the results of query 1, retain all the elements which are present in the results of queries 2, 3 and 4, and filter out all the elements which are present in the results of queries 5, 6, 7 or 8.

(2) Handling phrases (denoted by quotation marks) in a query

Lets take as an example query 332.xml from INEX 2006 topic collection:

<title>NCAA basketball tournament "march madness"</title>.

The query can be interpreted in two ways: “with decomposition” and “without decomposition” [Doddapaneni, 2005]. When handling a query without decomposition, we do not modify the query and simply retrieve using all query terms of the original query. When handling a query using decomposition, we decompose the original query in to sub-queries. For the query 332.xml, the resulting sub-queries would be:

sub-query 1: <title>NCAA basketball tournament</title>

sub-query 2: <title>march</title>

sub-query 3: <title>madness</title>

After doing initial Smart retrieval on each of the sub-queries, a manipulation is done on individual results. For example, a single result file can be created using the elements present in all of the original result files (i.e., the common intersection). Based on earlier query processing of this type [Doddapaneni, 2005], we conclude that “vaguely” interpreted queries without decomposition produce the best results (largely due to the directions for relevance assessment provided by INEX).

4.2 CAS (Content And Structure) Processing

There are 4 ways to handle a CAS query: VVCAS, SVCAS, VSCAS and SSCAS [Lalmas, 2005]. The first letter in the CAS definition denotes “what to return” and the second letter denotes “where to search”.

(1) VVCAS

In VVCAS (Vague Vague CAS), we look for the query terms anywhere in the article, and return any type of elements from the article. Lets consider, for example, query 293.xml from the INEX 2006 topic collection:

```
<castitle>//article[about(.,wifi)]//section[about(.,wifi security
encryption)]</castitle>
```

This query asks the system to return those articles about “wifi” that have sections about “wifi security encryption”. Hence the system is supposed to look for information in sections and articles and return articles. But in VVCAS we look for information in all kind of elements and return all kinds of elements.

(2) SVCAS

In SVCAS (Strict Vague CAS), we look for query terms in all elements, but return only those elements that the user specifies in the query. Therefore, for the above example, the system looks for information everywhere but returns only articles.

(3) VSCAS

In VSCAS (Vague Strict CAS), we look for information in only the elements specified by the query but return all type of elements. Therefore for the above example, the system will look for information only in articles and sections but returns all types of elements.

(4) SSCAS

In SSCAS (Strict Strict CAS), the system looks for information only in the elements specified by the query and returns only those elements that are asked for in the query. Therefore, for the above example, the system will look for the information in the sections and the articles and return only articles to the user.

Results of the query processing [Doddapaneni, 2005] show that VVCAS produces the best results of all the 4 interpretations of CAS. Hence we use VVCAS for the 2006 topic collection.

5. Implementing Flexible Retrieval

In this chapter we discuss the two approaches to element retrieval: (1) Using the all-element index (the base case) and (2) flexible retrieval based on the paragraph Index.

5.1 Using the All-element Index

In this approach, we input all elements of the document collection (produced by parsing at the paragraph, section and the article levels) to Smart. This collection, along with the query vectors, is indexed by Smart. The resulting “all-element index” is term-weighted using the Lnu-ltu weighting scheme [Singhal, Mitra, Buckley, 1996]. The values of slope and pivot used for this collection are 0.2 and 110 respectively (Here we use Smart’s default values since relevance assessments are not available to tune for slope and pivot). The last step is retrieval. In the retrieval process, the similarity of elements to the query is calculated using the inner product. In this step, extended subvector weighting [Fox, 1983] is also applied. A weight of 1 is applied to “body” subvector and weight of 0 is applied to “name” subvector. This is based on extended vector experiments on the INEX 2004 and 2005 document collections [Murthy, 2006]. The result of this retrieval is a ranked list of 15000 elements.

5.2 Flexible Retrieval based on the Paragraph Index

In this approach the input to Smart is the document collection parsed at the

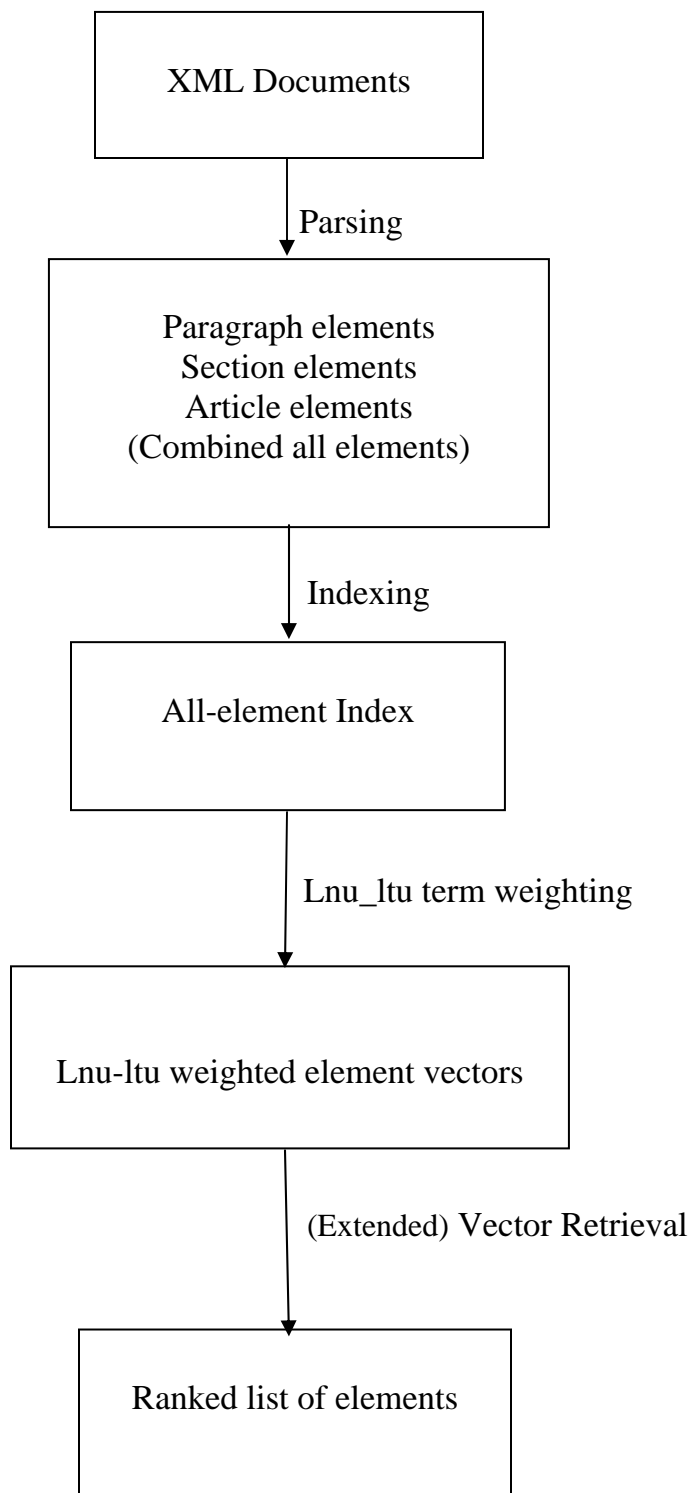


Figure 5. Retrieval using the all-element index

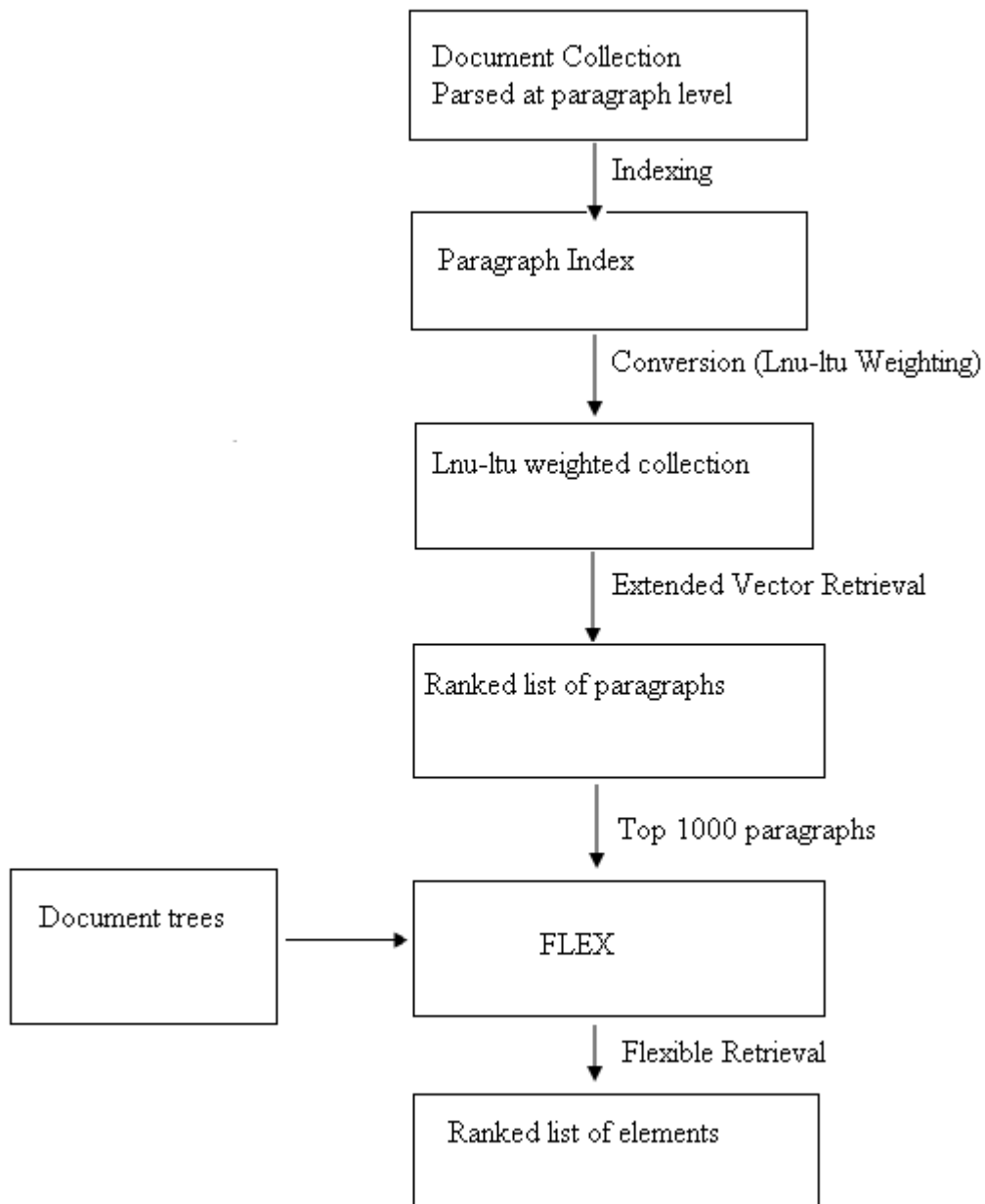


Figure 6. Retrieval using paragraph index and Flex

paragraph level. The paragraph vectors along with the query vectors are indexed by Smart. The “paragraph index” thus created is term weighted using the Lnu-ltu weighting scheme with the values of slope and pivot at 0.2 and 110, respectively. Then an initial retrieval is performed using the inner product as the similarity measure. The subvector weights applied to the extended vector during the retrieval process (identical to those used in the base-case retrieval with the all-element index) are not utilized until relevance assessments are available. The result of this retrieval is a ranked list of 5000 paragraphs. Of these paragraphs, the top 1500 paragraphs are fed to Flex. Flex also takes as input the paragraph-level, ltu-weighted query vectors and re-weights them as all-element level, ltu-weighted vectors [Murthy, 2006]. Then, using document schemas, Flex builds Lnu-weighted document vectors for all the parent elements of the N paragraphs fed to it as input. In the last step, Flex calculates the similarity scores (using the inner product) for all the element vectors with the query vector. The result is a ranked list of elements virtually identical to that produced by retrieval with the same query against the all-element index [Murthy, 2006].

6. Tasks at INEX

This section discusses various tasks for INEX 2006. It also discusses our techniques for implementation of each of these tasks. There are four such tasks: Thorough, Focused, Relevant in Context, and Best in Context [Clarke, et. al., 2006]. The description of these tasks are given below.

(1) Thorough Task: The thorough task simply asks user to return a ranked list of elements from the collection in order of their correlation to the query. Overlap is permitted for this task. Therefore the system can return more than one element from a document. In our implementation of the thorough task, we report the output of flexible retrieval.

(2) Focused Task: The focused task asks the user to return the most relevant element along a path. Therefore overlap is not permitted among the elements. In case there is more than one element along a path with the same correlation value, the smallest of those elements is returned. In our implementation of this task, we take as input the result of the flexible retrieval and filter out all the elements from a path except for the one which is best correlated to the query. Therefore, if an element is reported for a query, none of its ancestors or descendants can be reported.

(3) Relevant in Context Task: This task asks the system to search for the articles most relevant to a query and then report an unranked list of non-overlapped elements from each such article. In our implementation for this task, we take as

input the output of the flexible retrieval, and from it fetch a ranked list of articles in descending order. Then, for each article, we browse through the input to retrieve all the elements from the article. All the overlapping elements from an article are filtered out, with only the most highly correlating one retained. The result of this approach is a non-overlapping list of elements for each article.

(4) Best in Context Task: This task asks the system to report the “best entry point” to an article, meaning that the system is required to report the element from an article which is the “best starting point” for reading the article [Clarke, et. al., 2006]. In our implementation of this task, we take the result reported for the Relevant in Context Task as input and filter out all the elements from each article except for the most highly-correlating one. The final result is a list of “best elements” per article in response to the query.

7. Experiments, Results, and Conclusions

(1) Issues with Wikipedia

A major difference between Wikipedia and earlier INEX collections is that some elements in Wikipedia have their own untagged content in addition to the content of their children. For example, consider the following figure:

```
<section>  
  text  
  <p>  
  </p>  
  text  
</section>
```

Figure 7. A typical Wikipedia element

The existence of untagged text within an element causes problems for Flex. Flexible retrieval depends on having all discrete textual elements available so that an exact copy of the enclosing element (in this case, section) may be generated. All-element retrieval can generate the section vector properly because the untagged text lies within the section tags. Flex, on the other hand, cannot generate the equivalent vector unless the untagged text can also be inserted in it.

This problem is solved using the following approach. First, when the document is parsed, all untagged text is recognized and enclosed within special tags (`<mt>`, `</mt>`). During the paragraph indexing, an element is generated for each piece of text enclosed by special tags in exactly the same way that all other elements are generated. At this point, all discrete textual elements in the document are represented in vector form. These elements are also represented in the tree schemas. The elements created using the tag `<mt>` are fed to Flex but are filtered out of the final ranked list if retrieved because they do not exist per se in the original XML collection and the all-element index and therefore cannot be returned in the results. Nor are these tags included when values of N and n_k are calculated. As a result, the element vectors created by the all-element index and Flex are identical.

(2) Experiments and Results

For comparing the results of our two flexible retrieval approaches, we perform an all-element retrieval with slope and pivot values of 0.2 and 110, respectively. Dynamic retrieval using Flex is then performed using various values of n . The result of each of these dynamic retrievals is compared against the corresponding all-element retrieval (base case). We use window sizes of 5, 10, 20, 50, 100, 200, 500, 1000 and 1500 to determine the number of common elements between the two results. An average for number of common results is computed across 125

queries. Table 5 presents the results obtained, where n denotes the number of input elements fed to Flex.

Number of elements in Common
(Comparison of Flex and all-element retrieval)

n	Window								
	5	10	20	50	100	200	500	1000	1500
5	2.10	4.48	6.70	8.02	8.68	9.23	9.82	10.26	10.45
10	2.12	4.57	9.59	14.89	16.50	17.70	19.07	19.87	20.29
20	2.15	4.64	9.86	24.64	30.29	33.33	36.19	37.86	38.77
50	2.16	4.86	10.17	27.02	56.27	76.08	86.19	91.91	94.62
100	2.18	4.86	10.19	27.31	57.44	117.16	160.98	175.52	182.28
250	2.27	4.88	10.28	27.57	57.81	119.34	303.89	395.46	420.58
500	2.39	5.03	10.40	27.58	57.83	119.65	317.87	622.60	755.50
1000	2.67	5.74	11.93	29.45	64.60	126.45	335.27	682.45	902.35
1500	2.96	6.30	13.13	34.88	72.55	147.16	371.53	745.31	1115.82
2000	2.98	6.36	13.33	35.51	73.90	150.77	383.10	771.32	1155.68
3000	3.03	6.36	13.33	35.52	73.91	150.82	383.89	776.17	1167.95
5000	3.13	6.36	13.33	35.52	73.92	150.90	384.71	779.34	1176.01

Table 5

(3) Conclusion

At this time, relevance assessments for Wikipedia collection are not available, making it impossible to compute average precision and precision at various ranks.

Table 5 shows the increase in the number of common elements as both n and window size increase. Comparison of the all-element and Flex-generated queries show the two query sets contain identical vectors; corresponding element vectors (those generated by Flex and those produced by the Smart in the all-element index) are identical as well.

It is unknown at this point whether the values of n found to be sufficient for the 2004-2005 INEX collections (for which $n=250$ served as an upper bound) will also prove sufficient for this very different collection. However, it is clear that the method of flexible retrieval can also be applied to semi-structured collections.

Suggestions for future work include application of Flex with respect to the Focused, Relevant in Context, and Best in Context tasks (see Section 6 for the definition of these tasks), comparing the results produced by Flex and all-element retrieval. Further experiments can be performed to determine best values of extended vector weights, once the relevance assessments become available. This collection can also be tuned to find best values of slope and pivot, which will improve the results for both Flex and the all-element retrieval.

References

- [1] Salton, G., editor. The SMART Retrieval System – Experiments in Automatic Document Retrieval, Prentice-Hall, Englewood Cliffs, NJ, 1971.
- [2] Fox, E. Extending the Boolean and Vector Space Models of Information Retrieval with P-norm Queries and Multiple Concept types. Ph.D. Dissertation, Department of Computer Science, Cornell University, 1983.
- [3] Khanna, S. Design and Implementation of a Flexible Retrieval System. MS Thesis, University of Minnesota Duluth, 2005.
- [4] Doddapaneni, N. Effective Structured Query Processing. MS Thesis, University of Minnesota Duluth, 2005.
- [5] Singhal, A; Buckley, C; Mitra, M. Pivoted Document Length Normalization. In Proceedings of the 19th Annual International ACM SIGIR Conference, Zurich, Switzerland, 1996, 21-29.
- [6] Murthy, GMVS. Query Processing in a Flexible Retrieval Environment. MS Thesis, University of Minnesota Duluth, 2006.
- [7] Kazai, G; Lalmas, M. INEX 2005 evaluation Metrics, 2005.
<http://inex.is.informatik.uni-duisburg.de/2005/inex-2005-metricsv6.pdf>
- [8] Lalmas, M. INEX 2005 Retrieval Task and Result Submission Specification.
http://inex.is.informatik.uniduisburg.de/2005/internal/pdf/INEX05_Tasks_v2.pdf
- [9] Larsen, B. INEX 2006 Guidelines for Topic Development, 2006.
<http://inex.is.informatik.uni-duisburg.de/2006/inex06/pdf/NEXI.pdf>
- [10] Clarke, C; Kamps, J; Lalmas, M. INEX 2006 Retrieval Task and Result Submission Specification.
http://inex.is.informatik.uni-duisburg.de/2006/inex06/pdf/INEX06_Tasks_v1.pdf
- [11] Denoyer, L; Gallinari, P. The Wikipedia XML corpus, 2006.
http://info.acm.org/sigir/forum/2006J/2006j_sigirforum_denoyer.pdf

Appendix

A.1

Tags	Counts	Tag Description
collectionlink	17021858	Link to a document in the collection
item	5685190	Items in lists
unknownlink	3948850	Link to an unknown document
cell	3771637	Cell in a table
p	2753451	Paragraph in an article
emph2	2723700	Tag to emphasize text
template	2428121	Contains field that can be edited
section	1610547	Section in an article
title	1592989	Title of a section
emph3	1481252	Tag to emphasize text
normallist	1110506	List of items
row	940056	Row in a table
outsidelink	859290	Link to a document outside the collection
languagelink	789704	Link to a language
body	659478	Contains all the content of an article except article title
article	659472	Contains whole article
name	659407	Article title
conversionwarning	659388	Tag to explain if an article has been modified
br	384038	Forced line break
td	371101	Defines cell in a table
caption	350938	Caption of a figure or table
image	344998	Image
figure	344994	Encapsulates an image and its caption
wikipedialink	176711	Link to a document in wikipedia collection
cadre	149394	Grouping elements together
indentation1	135439	Indentation

Tags	Counts	Tag Description
tr	121963	Defines a row in the table
table	92353	Table
numberlist	81731	List of numbers
emph5	81138	Tag to emphasize text
sup	71522	Used for subscripted text (e.g., +)
math	66901	For mathematical formula
small	61147	Renders as small text
sub	54995	For superscripted text (e.g.,CO₂)
weblink	46059	Link to a website
value	28987	To indicate a location where content can be inserted
term	28971	Term
definitionitem	28968	Items in a definitionlist
font	27137	Font
th	23356	Table head element
i	17939	Italics
definitionlist	15724	List of Definitions
center	15619	Centers the enclosed text horizontally
indentation2	14073	Indentation
div	13118	Defines a division in the document
li	12054	Defines the start of a list item
b	11298	Marks the text as bold
tt	6842	Text is spaced non-proportionally
code	5956	Contains codes of programs
blockquote	4837	Defines the start of a long quotation
u	3527	Underlines the text
big	3213	Renders as big text
ul	3050	Unordered list
span	2592	Brings elements into one line
gallery	2527	Groups pictures together
cite	2154	Used for citations or references
indentation3	1993	Indentation

Tags	Counts	Tag Description
s	1006	Displays the text with strike through
hr	942	Inserts a horizontal rule
emph4	940	Tag to emphasize the text
em	608	Tag to emphasize the text
strong	351	Renders the text as bold
h4	307	Used to specify level 4 headings
hieroglyph	282	Used for hieroglyphic writing (e.g.,<hieroglyph>S29-Z7-U29-G1Z6</hieroglyph>)
h3	231	Used to specify level 3 headings
redirectlink	208	Redirect a link to some address
http:	205	Used to mark http protocol
st3	180	Terms (e.g., Henry VIII)
var	179	Used for variable references within programs
ol	163	Ordered list
dd	163	Defines a term in a definitionlist
h2	144	Used to specify level 2 headings
dl	126	Defines a definitionlist