

Impact of Untagged Text in Dynamic Element Retrieval

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

Nachiket M. Kamat

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

August 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 master's thesis by

Nachiket M. Kamat

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

Carolyn J. Crouch

Name of Faculty Advisor

Signature of Faculty Advisor

Date

GRADUATE SCHOOL

Acknowledgements

I would like to thank Dr. Carolyn Crouch for giving me a wonderful opportunity to work in the field of information retrieval. She has immense knowledge in this field and I have learnt the difference between a master's thesis and a project. She was very helpful and has guided me throughout my master's program.

I would like to thank Dr. Donald Crouch and Dr. Douglas Dunham for their valuable feedback on my work.

I am very grateful to Vishal Bakshi and GMVS Murthy for their valuable guidance and for sharing their knowledge related to the flexible retrieval system. Special thanks to my colleagues Aditya Mone and Vikram Malik for being supportive co-workers as well as good friends throughout the completion of this thesis.

I would like to thank the staff at the Department of Computer Science - Lori Lucia and Linda Meek for their continuous help. Special thanks to the system administrator - Jim Luttinen for helping me with the system related issues.

Finally, I would like to thank my dearest friend Shruti Pandey, my family members and all my friends for encouraging and supporting me throughout my life.

Abstract

Traditional information retrieval systems concentrated on retrieving information from unstructured documents. But with the arrival of the World Wide Web (WWW) and Extensible Markup Language (XML) as a preferred standard for storing documents, the focus now changes to retrieving information from structured documents. The most difficult aspect of structured retrieval is retrieving at the appropriate level of granularity, which we refer to as *flexible retrieval*.

Web retrieval deals with both structured and semi-structured documents. The difference between them arises from the presence of untagged text in the semi-structured documents. Our flexible retrieval system, called Flex [Khanna, 2005], was designed to handle structured documents. We have devised a method to adapt Flex to semi-structured documents.

In this thesis, we analyze the importance of links and untagged text (which are characteristics of a semi-structured document collection like Wikipedia) in the implementation of Flex. We use the Smart retrieval system [Salton, 1971] for basic element retrieval. We report on experiments regarding the impact of the number and type of terminal nodes fed to Flex. We also discuss the design and implementation of an algorithm, based on Flex, for passage retrieval.

Table of Contents

1. Introduction	1
2. Background	4
2.1 Document Collection.....	4
2.2 Query Collection.....	6
2.3 Ad-hoc XML Retrieval Tasks.....	7
2.4 Relevance Assessments.....	9
2.5 Evaluation Metrics.....	10
2.6 Smart Retrieval Engine.....	11
3. The Flexible Retrieval System	12
3.1 Pre-processing and Indexing.....	12
3.2 Term Weighting and Pivoted Normalization.....	16
3.3 Flex.....	19
4. Importance of Untagged Text in Flex	21
5. Passage Retrieval	24
5.1 What is Passage Retrieval?.....	24
5.2 Implementation of Passage Retrieval.....	25
6. Experiments and Results	30
6.1 Experiments.....	30
6.1.1 Collection Links.....	31
6.1.2 Leaf Nodes Input to Flex.....	38
6.2 Results.....	42
7. Conclusions and Future Work	44
References	45
Appendix A.....	46
Appendix B.....	50
Appendix C.....	56

List of Tables

Table 3.1 Tags used for indexing the Wikipedia collection.....	14
Table 3.2 Slope and Pivot values for INEX 2006 Wikipedia collection.....	18
Table 6.1 Task: Thorough, Relevance-Assessments: v4, Case 1.....	32
Table 6.2 Task: Thorough, Relevance-Assessments: v5, Case 1.....	32
Table 6.3 Task: Focused, Relevance-Assessments: v4, Overlap: OFF, Case 1.....	33
Table 6.4 Task: Focused, Relevance-Assessments: v5, Overlap: OFF, Case 1.....	33
Table 6.5 Task: Focused, Relevance-Assessments: v4, Overlap: ON, Case 1.....	34
Table 6.6 Task: Focused, Relevance-Assessments: v5, Overlap: ON, Case 1.....	34.
Table 6.7 Flex Tree Generation Statistics for Case 1.....	35
Table 6.8 Task: Thorough, Relevance Assessments: v4, Case 2.....	35
Table 6.9 Task: Thorough, Relevance Assessments: v5, Case 2.....	36
Table 6.10 Task: Focused, Relevance Assessments: v4, Overlap: OFF, Case 2.....	36
Table 6.11 Task: Focused, Relevance Assessments: v5, Overlap: OFF, Case 2.....	36
Table 6.12 Task: Focused, Relevance Assessments: v4, Overlap: ON, Case 2.....	37
Table 6.13 Task: Focused, Relevance Assessments: v5, Overlap: ON, Case 2.....	37
Table 6.14: Flex Tree Generation Statistics for Case 2.....	37
Table 6.15 Average number of mts ignored per query for Case 3.....	39
Table 6.16 Flex Tree Generation Statistics for Case 3.....	39
Table 6.17 Task: Thorough, Relevance Assessments: v4, Case 3.....	40
Table 6.18 Task: Thorough, Relevance Assessments: v5, Case 3.....	40
Table 6.19 Task: Focused, Relevance Assessments: v4, Overlap: OFF, Case 3.....	40
Table 6.20 Task: Focused, Relevance Assessments: v5, Overlap: OFF, Case 3.....	41
Table 6.21 Task: Focused, Relevance Assessments: v4, Overlap: ON, Case 3.....	41
Table 6.22 Task: Focused, Relevance Assessments: v5, Overlap: ON, Case 3.....	41
Table A.1 Query by query analysis when exactly 100 paragraphs seeded to Flex.....	47
Table B.1 Query by query analysis when at most 100 paragraphs seeded to Flex.....	51
Table B.2 Tree Generation Statistics when at most 100 paragraphs seeded to Flex.....	54
Table B.3 Task: Thorough, Relevance Assessments: v4, at most 100 paragraphs seeded to Flex.....	54

Table B.4 Task: Thorough, Relevance Assessments: v5, at most 100 paragraphs seeded to Flex.....	54
Table B.5 Task: Focused, Relevance Assessments: v4, Overlap: OFF, at most 100 paragraphs seeded to Flex.....	54
Table B.6 Task: Focused, Relevance Assessments: v5, Overlap: OFF, at most 100 paragraphs seeded to Flex.....	55
Table B.7 Task: Focused, Relevance Assessments: v4, Overlap: ON, at most 100 paragraphs seeded to Flex.....	55
Table B.8 Task: Focused, Relevance Assessments: v5, Overlap: ON, at most 100 paragraphs seeded to Flex.....	55
Table C.1 Task: Thorough, Relevance Assessments: v4, collection links included.....	56
Table C.2 Task: Thorough, Relevance Assessments: v5, collection links included.....	56
Table C.3 Task: Focused, Overlap: OFF, Relevance Assessments: v4, collection links included.....	56
Table C.4 Task: Focused, Overlap: OFF, Relevance Assessments: v5, collection links included.....	57
Table C.5 Task: Focused, Overlap: ON, Relevance Assessments: v4, collection links included.....	57
Table C.6 Task: Focused, Overlap: ON, Relevance Assessments: v5, collection links included.....	57
Table C.7 Tree Generation Statistics, collection links included.....	58

List of Figures

Fig 2.1 Structure of a typical IEEE document (INEX 2005).....	5
Fig 2.2 Structure of a typical Wikipedia document (INEX 2006).....	6
Fig 2.3 Typical INEX 2006 topic.....	7
Fig 3.1 Original XML document before ignoring the collection link tag.....	13
Fig 3.2 XML Document after ignoring the collection link tag.....	13
Fig 3.3 Retrieval using the All-Element Index.....	15
Fig 3.4 Pivoted Normalization (from [Singhal, Buckley, Mitra, 1996]).....	17
Fig 3.5 <i>Lnu</i> weighting formula (from [Singhal, Buckley, Mitra, 1996]).....	17
Fig 3.6 <i>ltu</i> weighting formula (from [Singhal, Buckley, Mitra, 1996]).....	18
Fig 3.7 Retrieval using Flex.....	20
Fig 4.1A typical semi-structured Wikipedia Document.....	21
Fig 4.2 A completely structured Wikipedia document.....	22
Fig 5.1 A passage containing only one element.....	24
Fig 5.2 A passage containing two elements.....	24
Fig 5.3 A passage starting on mt and ending on an element.....	25
Fig 5.4 A passage starting and ending on mt.....	25
Fig 5.5 passage retrieval algorithm.....	25
Fig 5.6 passage retrieval using Flex.....	26
Fig 5.7 A snippet from paragraph retrieval.....	27
Fig 5.8 Paragraph retrieval sorted on Smart Id.....	27
Fig 5.9 A typical passage schema.....	28
Fig 5.10 A typical passage retrieval output using Flex.....	29
Fig 5.11 Final passage retrieval output (Thorough).....	29

1. Introduction

Information retrieval involves organizing data and retrieving useful information from it. Over the years, computer users have created data in the form of documents, email, dictionaries, web-pages, etc. We need well-organized techniques to search through this massive data and retrieve useful information when needed. These techniques should allow a ranked retrieval because in many cases you want the “best” answer among many documents that contain certain words.

Traditional information retrieval concentrated on retrieving information from unstructured documents. The user was supposed to fire a query and get back a set of documents satisfying that query. For instance, if a user were interested in finding information on computer animation, he could fire the query “*computer animation*” on a search engine and retrieve a ranked list of documents related to his query. He then had to peruse those documents to find the information he was interested in. But with the advent of the World Wide Web (WWW) and Extensible Markup Language (XML) as a preferred standard for storing documents, an interest arise in retrieving information from structured documents. Structuring allows a document to be processed in terms of its smaller, component elements.

A typical XML document consists of units of data known as entities. Every entity is marked by an XML tag. In a typical XML document, the parent entity is labeled by a tag, say *<article>*, which would itself contain a number of entities, each tagged *<section>*.

Each section might include several smaller entities tagged *<paragraph>*. In this way, XML provides a rich format for sharing documents over the web. Using XML documents, a search engine can retrieve parts of a document in addition to the entire document.

An information retrieval system based on structured documents may allow users to utilize structure in their queries. A conventional information retrieval system can be extended to handle such queries by first structuring elements and then filtering the results ensure that all elements satisfy the structural constraints. The difficult part of the task in structured retrieval is retrieving at the appropriate level of granularity (i.e., the element level) which we refer to as *flexible retrieval* [Khanna, 2005].

There are two types of structured documents with which we have worked, namely completely structured documents such as the INEX IEEE collection and semi-structured documents as exemplified by Wikipedia. The differences between these collections are described in Chapter 2.

The aim of this thesis is to analyze the importance of collection links and untagged text (characteristic of semi-structured documents like Wikipedia documents) in the implementation of flexible retrieval [Khanna, 2005] and to develop a new algorithm for passage retrieval (as opposed to the element retrieval).

The University of Minnesota Duluth (UMD) is a participant in INEX (Initiative for Evaluation of XML Retrieval) [1]. Participating organizations compare the retrieval effectiveness of their XML retrieval systems to others, and in doing so they contribute to the construction of the XML test collection. The test collection consists of a set of XML documents, topics (queries) and relevance assessments.

In Chapter 2, we discuss the INEX test collection used for experiments. Chapter 3 describes the flexible retrieval system which was developed at UMD. Chapter 4 explains the importance of untagged text in Flex. We discuss our approach to Passage Retrieval in Chapter 5. Chapter 6 describes the experiments designed and the results achieved. Finally, conclusions and suggestions for future research are provided in Chapter 7.

2. Background

This section covers details about the document collection, query collection, ad-hoc retrieval tasks, relevance assessments, evaluation metrics used in INEX and the Smart retrieval engine.

2.1 Document Collection

In 2005, the INEX document collection consisted of IEEE articles published between 1995 and 2002. These IEEE articles were completely structured; i.e., these were XML documents and they conformed to a single DTD (Document Type Definition). Figure 2.1 shows the overall structure of a typical IEEE document. More details can be found in [Khanna, 2005].

In 2006, a new, semi-structured document collection was defined. This is the Wikipedia XML Corpus which is around 4.6GB in size. This collection is huge compared to the earlier IEEE collection; it does not have a DTD. It is composed of 659,388 documents, divided into 113,483 categories. The average number of XML nodes per article is 161.35, and the average depth of an element is 6.72. See [Denoyer, Gallinari, 2006] for details on the Wikipedia Corpus.

Retrieving useful information from semi-structured documents is more difficult from our view than returning from completely structured documents. The difficulty arises due to

the following: (1) There is no standard hierarchy defined for the data elements. For example, a section can occur inside a leaf node. Such an hierarchy was not present in the

```
<article>
  <fm>
    <atl> text </atl>
  </fm>
  <bdy>
    <sec>
      <ss>
        <p> text </p>
      </ss>
      <p> text </p>
      .....
    </sec>
    .....
  </bdy>
  <bm>
    <bib> text </bib>
  </bm>
</article>
```

Fig 2.1 Structure of a typical IEEE document (INEX 2005)

IEEE collection because it conformed to a DTD and the DTD said that a section cannot occur inside a leaf node because it is the parent of that leaf node. If a search engine expects a paragraph to be the leaf node, then a section occurring inside that paragraph would violate the DTD. (2) Some text is not labeled with appropriate XML tags. For example, a particular section may consist of a set of paragraphs and some text without paragraph tags. It is obvious that the untagged text should be labeled as a paragraph, but

it lies inside its parent section tag. This untagged text cannot be retrieved at a granular level as expected. These cases are evident from Figure 2.2 which outlines the structure of a typical Wikipedia document.

```
<article>
<name> text </name>
<body>
  text
  <section>
    text
    <p> text </p>
    <p> text </p>
    .....
  </section>
  <p> text </p>
  text
  .....
</body>
</article>
```

Fig 2.2 Structure of a typical Wikipedia document (INEX 2006)

2.2 Query Collection

There are around 125 CO+S (Content Only + Structure) queries (also know as topics) in the INEX 2006 collection. Each topic has six components, i.e., title, castitle, description,

narrative, ontopic keywords and offtopic keywords [Larsen, Trotman, 2006]. We use *title* for Content Only tasks and *castitle* for Content and Structure tasks. Figure 2.3 shows a typical topic from INEX 2006 topic collection.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic query_type="CO+S" ct_no="95">

<InitialTopicStatement>I am interested in information about operating systems that
have microkernel. Studying operating systems for last few years inspired my interest in
the topic.</InitialTopicStatement>

<title>microkernel operating systems</title>

<castitle>//article[about(.,microkernel operating systems)]</castitle>

<description>Find Information about operating systems with microkernel</description>

<narrative>I worked as teaching assistant for operating systems. This inspired my
interest in microkernel operating systems. To be relevant an element should talk about
operating systems with microkernel. An element talking only about operating systems or
microkernel should be considered irrelevant. </narrative>

<ontopic_keywords>microkernel; operating systems; mach</ontopic_keywords>

<offtopic_keywords>automobiles</offtopic_keywords>

</inex_topic>
```

Fig 2.3 Typical INEX 2006 topic

2.3 Ad-hoc XML Retrieval Tasks

The main task to be performed in INEX 2006 is the ad-hoc retrieval [5]. There are four sub-tasks in the ad-hoc track [Clarke, Kamps, Lalmas, 2006]:

- Thorough Task - The aim of this task is to find all relevant elements ranked in order of relevance. Overlapping elements are allowed in this task, i.e., the system can return more than one element along the same path. In our implementation of the thorough task, we return the output of flexible retrieval.
- Focused Task - The aim of this task is to return a ranked list of elements, where no element overlaps with any other element. If there is more than one element along a path with the same correlation score, then the smaller one is returned as it is believed that the smaller elements are more focused towards the query. In our implementation of the focused task, we take the output of flexible retrieval, and remove all overlapping elements from a path except the one which best correlates with the query.
- Relevant in Context Task – The aim of this task is first to identify relevant articles (the fetching phase) and then to identify the relevant elements within those articles (the browsing phase). In our implementation of the relevant in context task, we take the output of flexible retrieval and fetch a ranked list of articles from it. We then sort this list in descending order. For each article, we browse through the output to retrieve all the elements from the article. We remove all overlapping elements from that article and return the highly correlating elements. Thus we have a list of non-overlapping elements for each article.
- Best in Context Task - The aim of this task is first to identify relevant articles (the

fetching phase) and then to identify the element corresponding to the best entry point for each article (the browsing phase). In our implementation of the best in context task, we take the result of relevant in context task and remove all elements from each article except the one that best correlates with the query. The final result is a list of best elements per article in response to a query.

2.4 Relevance Assessments

Relevance assessments are carried out to evaluate the XML retrievals submitted by participating organizations. The topics and an identified document collection set are distributed by INEX, and it is the participant's responsibility to manually assess the documents. A relevance assessor is supposed to understand the information need specified in the topic and then read the documents, highlight the text which he finds relevant to the topic, and mark a best entry point for every relevant document.

Previous to INEX 2006, relevance was defined by two dimensions, specificity and exhaustivity [7]. The latter describes the extent to which the document component discusses the topic of request. For the year 2006, only specificity is used. Specificity describes the extent to which the document element focuses on the topic of request. Specificity is a continuous variable ranging from 0 to 1. An element whose entire content is marked relevant by the user has a specificity value of 1 and is considered highly specific where as an unmarked element has a specificity of 0. If part of an element is marked as relevant, then depending on the size of the relevant portion and the size of the

element, the element would get a specificity score between 0 and 1.

There are two sets of Relevance Assessments in INEX 2006: (1) relevance assessments with *collection links*, and (2) relevance assessments without *collection links*. These versions of relevance assessments are added to understand the impact of collection links in XML retrieval. There are a huge number of collection links present in the Wikipedia collection. Collection links contain important keywords and their presence drastically impacts retrieval, as is evident from the experiments described in Chapter 5.

2.5 Evaluation Metrics

Submissions in INEX are evaluated using two types of metrics: (1) XCGEval, which evaluates runs with the xCG (extended cumulative gain) metric, and (2) EvalJ, which evaluates submissions with PR (also known as *inex-eval*), PRNG (also known as *inex-eval-ng*) and PRUM metrics [GR, PRUM and EPRUM].

xCG contains nxCG (normalized extended cumulative gain) and ep/gr (effort-precision/gain-recall). These new metrics aim to provide an evaluation framework that allows the consideration of the dependency that exists among XML document components and, in particular, incorporation of mechanisms to reward the retrieval of so-called near-misses and to address issues of overlap [5].

For a ranked list of elements, xCG at rank i is given by: $xCG[i] = \sum xG[i]$, where $xG[i]$ is

relevance score of the element at rank i . For every result an ideal gain vector (xI) can be calculated by placing the elements in decreasing order of relevance. The value of $nxCG$ at rank i is given by: $nxCG[i] = xCG[i] / xCI[i]$ where xCI is the ideal cumulative gain vector. Therefore the value of $nxCG$ denotes the gain accumulated by a system at rank i with respect to the ideal gain. More information on the xCG metric can be found in [9].

ep/gr is a metric used to evaluate the effort required by the system to achieve a particular value of gain, with respect to the ideal run. The ep at gain k is given by $ep[k] = r1/r2$ where $r1$ is the rank at which the ideal system attains a gain value of k and $r2$ is the rank at which our system attains the gain value. An ep value of 1 represents ideal performance. The value of gr at rank i is given by $gr[i] = xCG[i]/xCI[n]$, where n is the total number of relevant documents in the Wikipedia 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.

2.6 Smart Retrieval Engine

We use Smart 13.0 [Salton, 1971] as our basic retrieval system. Smart uses the Vector Space Model (VSM) in which each document and query is represented as a weighted vector of terms. The distance between the document vector and the query vector in vector space determines the relationship between the two. Smart provides us with the basic functionalities of indexing a document collection, weighting the document/query vectors and retrieving highly correlating document elements.

3. The Flexible Retrieval System

Flexible Retrieval refers to the task of retrieving specific elements that satisfy the query's information need [Khanna, 2005]. This means the system must be able to retrieve at the appropriate level of granularity.

3.1 Pre-processing and Indexing

Preprocessing begins by parsing the document collection. Parsing is done at three levels article, section or paragraph. Wikipedia contains 1258 different XML tags. Consider tags with a count of 100 or more. The remaining tags were analyzed to determine their importance in the retrieval process. After a careful inspection, we excluded formatting tags (, <i>, etc), link tags (<collection link>, <unknown link>, etc), miscellaneous tags which contain data but the content size is too small to be considered relevant (<row>, <column>, etc).

Refer to [Bakshi, 2006] for the list of tags which were discarded during the parsing process. When discarding these tags, we ignore the XML tag but keep the text contained in the tag. This text is then included in the immediate parent tag. For example, Fig 3.1 shows a typical XML document. There are two elements in the *section* element, i.e., a *paragraph* and a *collection link*. In the parsing process, we ignore the collection link but keep the text contained in that tag. So in this case, *data2* becomes part of its immediate

parent, which is *section*. Fig 3.2 shows the structure of the parsed XML document. In this way, we eliminate unnecessary tags and yet do not lose any useful data.

```
<article>
.....
<section>
<p> data 1 </p>
<collection link> data 2<collection link>
</section>
.....
</article>
```

Fig 3.1 Original XML document before ignoring the collection link tag

```
<article>
<section>
<p> data 1 </p>
data 2
</section>
.....
</article>
```

Fig 3.2 XML Document after ignoring the collection link tag

Table 3.1 shows a list of tags which were indexed for the Wikipedia collection. The last row in Table 3.1 describes our own user-defined tag which we refer to as `<mt>`. This is not a valid XML tag; we have added it to overcome the limitation of semi-structured

documents when implementing flexible retrieval. The method used is described in Chapter 4.

Table 3.1 Tags used for indexing the Wikipedia collection

Tag	Level	Description
<body>	article	contains the whole article except for article name
<section>	section	section in an article
<p>	paragraph	paragraph in an article
<normallist>	paragraph	list of items
	paragraph	ordered list
	paragraph	unordered list
<numberlist>	paragraph	list of numbers
<definitionlist>	paragraph	list of definitions
<figure>	paragraph	contains an image and its caption
<name>	paragraph	article title
<table>	paragraph	Table
<mt>	paragraph	magic text

To produce a base case against which flexible retrieval is evaluated, after parsing the collection, we use Smart to index it. The result is known as all-element index. Fig 3.3 describes retrieval using the all-element index. As seen in Fig 3.3, the document collection is parsed at three different levels, which are then combined and fed to Smart for indexing. It is observed that a *paragraph's* text is present in its parent *section* and a *section's* text is present in its parent *article*. It is clear that the all-element index is highly

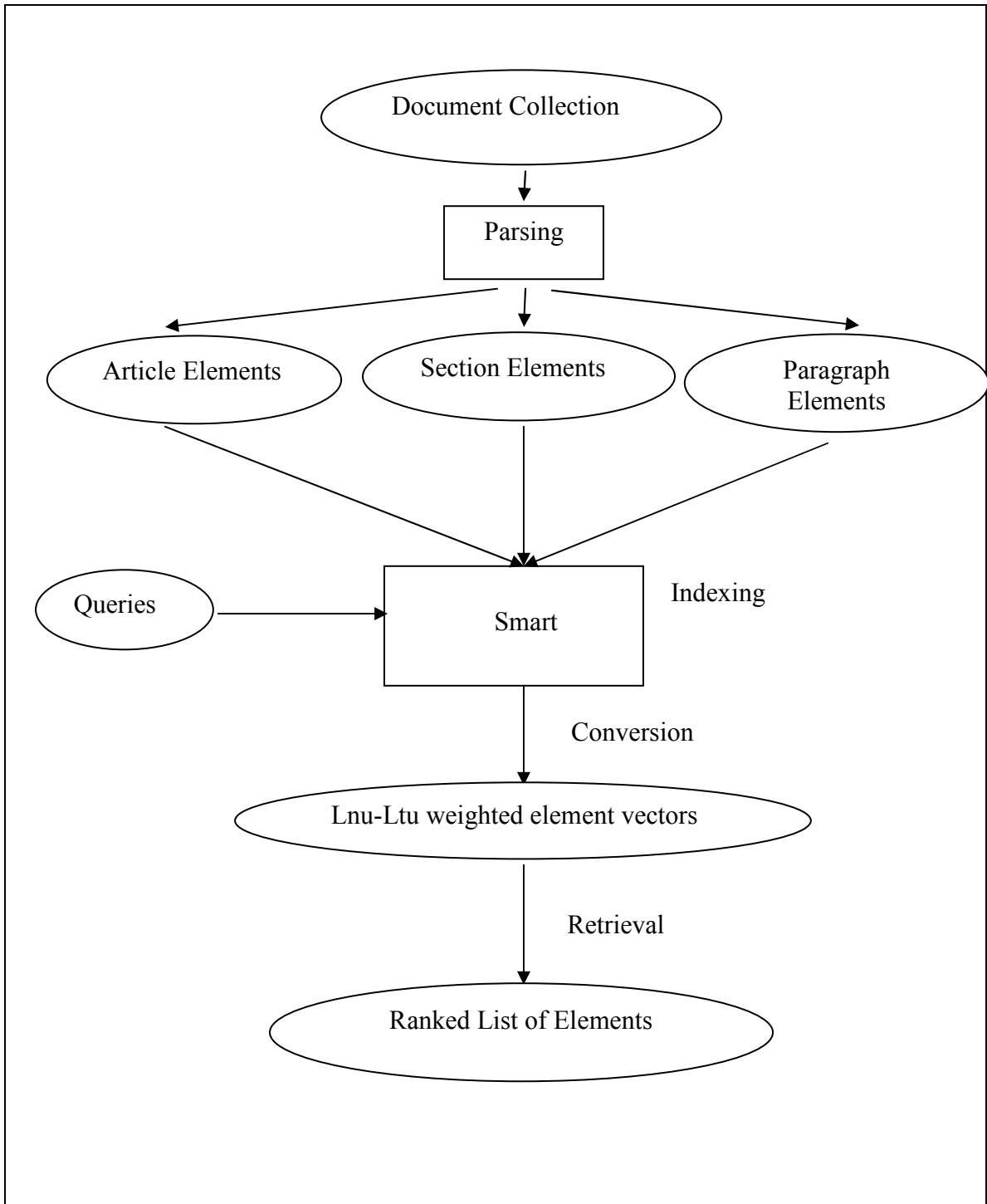


Fig 3.3 Retrieval using the All-Element Index

redundant; as a result, it takes up a lot of disk space. Flexible retrieval uses another approach where we index only the paragraphs and then build the respective section(s) and

article from the paragraphs. This process is described in the following section.

3.2 Term Weighting and Pivoted Normalization

Our Smart index of the leaf node uses the *nnn* weighting scheme to weight the terms. In this weighting scheme, the weight assigned to a term is equal to its frequency in the document element. This weighting scheme takes only the term frequency into account. We need a weighting scheme that takes the element length into consideration and normalizes term weights based on it.

Failure to consider length and normalization issues produces scores biased towards larger documents. Larger documents have more terms and terms with higher term frequency. Thus the probability of retrieval is higher than the probability of relevance for larger documents and vice-versa for smaller elements.

We know that the probability of retrieval of a document is inversely proportional to the value of the normalization factor for it [Singhal, Buckley, Mitra, 1996]. Thus we want to increase the normalization factor for the larger elements and decrease it for smaller elements. We achieve this normalization using two constants, slope and pivot. Pivot is the document length for which the probability of relevance is equal to the probability of retrieval. The normalization is now pivoted at pivot and tilted so that the documents on one side of pivot get larger normalization factors and the others get smaller normalization factors. This process of tilting is known as slope. Fig 3.4 illustrates this concept.

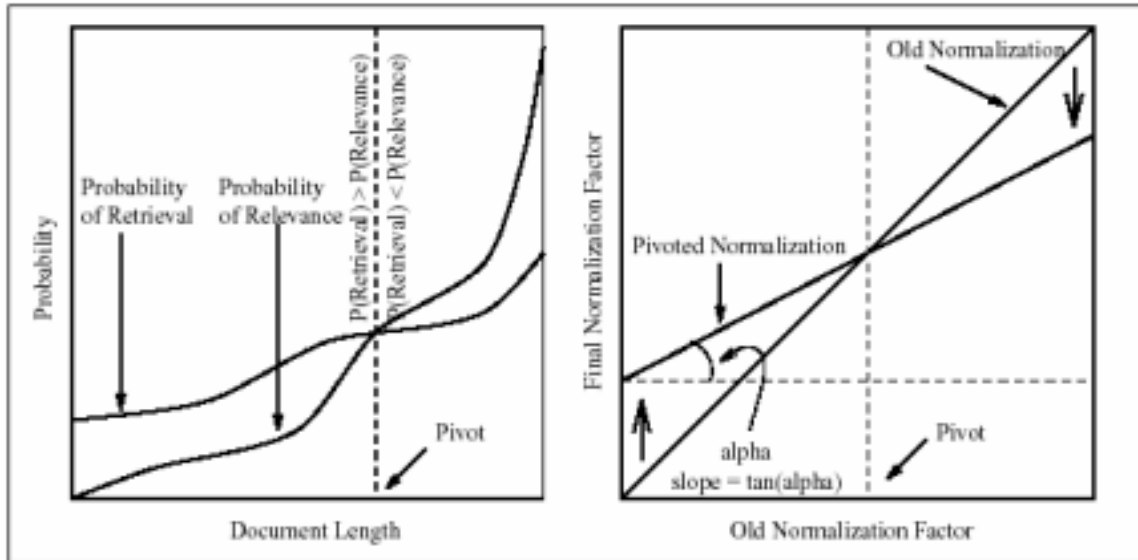


Fig 3.4 Pivoted Normalization (from [Singhal, Buckley, Mitra, 1996])

We convert the *mmn* weights on element vectors to *Lnu* weights for achieving useful term weights. The *Lnu* formula is shown in Fig 3.5

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

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

Fig 3.5 *Lnu* weighting formula (from [Singhal, Buckley, Mitra, 1996])

Slope and pivot are experimentally determined constants. The pivot value varies from collection to collection and slope needs to be calculated accordingly. The values of slope and pivot for the INEX 2006 Wikipedia collection are given in Table 3.2.

Table 3.2 Slope and Pivot values for INEX 2006 Wikipedia collection

Element Level	Slope	Pivot
Article	0.04	120
Paragraph	0.12	18
All-element	0.12	38

The query vectors are weighted using the *ltu* weighting scheme. Fig 3.6 shows the *ltu* formula.

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

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

Fig 3.6 *ltu* weighting formula (from [Singhal, Buckley, Mitra, 1996])

The *Lnu* weighted document vectors are correlated with the *ltu* weighted query vectors using inner product and a score is assigned to each document. The documents are then sorted according to the score which produces the final list of ranked elements. This scheme is applied to element vectors and used for both all-element and flexible retrieval.

3.3 Flex

Flex [Khanna, 2005] is software that works from a paragraph index created by Smart and produces results identical to that of a Smart retrieval against the all-element index. It creates both correctly weighted element vectors and query vectors [Ganapathibhotla, 2006]. The vectors are created and retrieval takes place dynamically.

Fig 3.7 shows the working of Flex. Initially, the document collection is parsed at the paragraph level. These paragraph elements and queries are indexed and *Lnu-ltu* weighted using Smart. The *Lnu*-weighted vectors are correlated with the *ltu* weighted query vectors using inner product and a ranked list of documents elements is generated. We also scan the document collection and produce a document schema for each and every document. This schema contains the x-path of each and every element present in the document. We then select top *n* paragraphs from the Smart retrieval and feed them to Flex.

Flex generates trees for all those documents which have at least one paragraph in the highly correlating set. The tree population process works as follows. The *nnn* weights of the paragraph vectors are converted to *Lnu* weights. These are then correlated with the *ltu*-weighted query vectors and a score is assigned to the elements based on the inner product. Child elements are joined to form the *nnn* vectors at the section level. Again these are *Lnu* weighted and correlated with *ltu* weighted query vectors. The process continues at the article level. All the elements are then sorted by score and a final, ranked list of elements is returned.

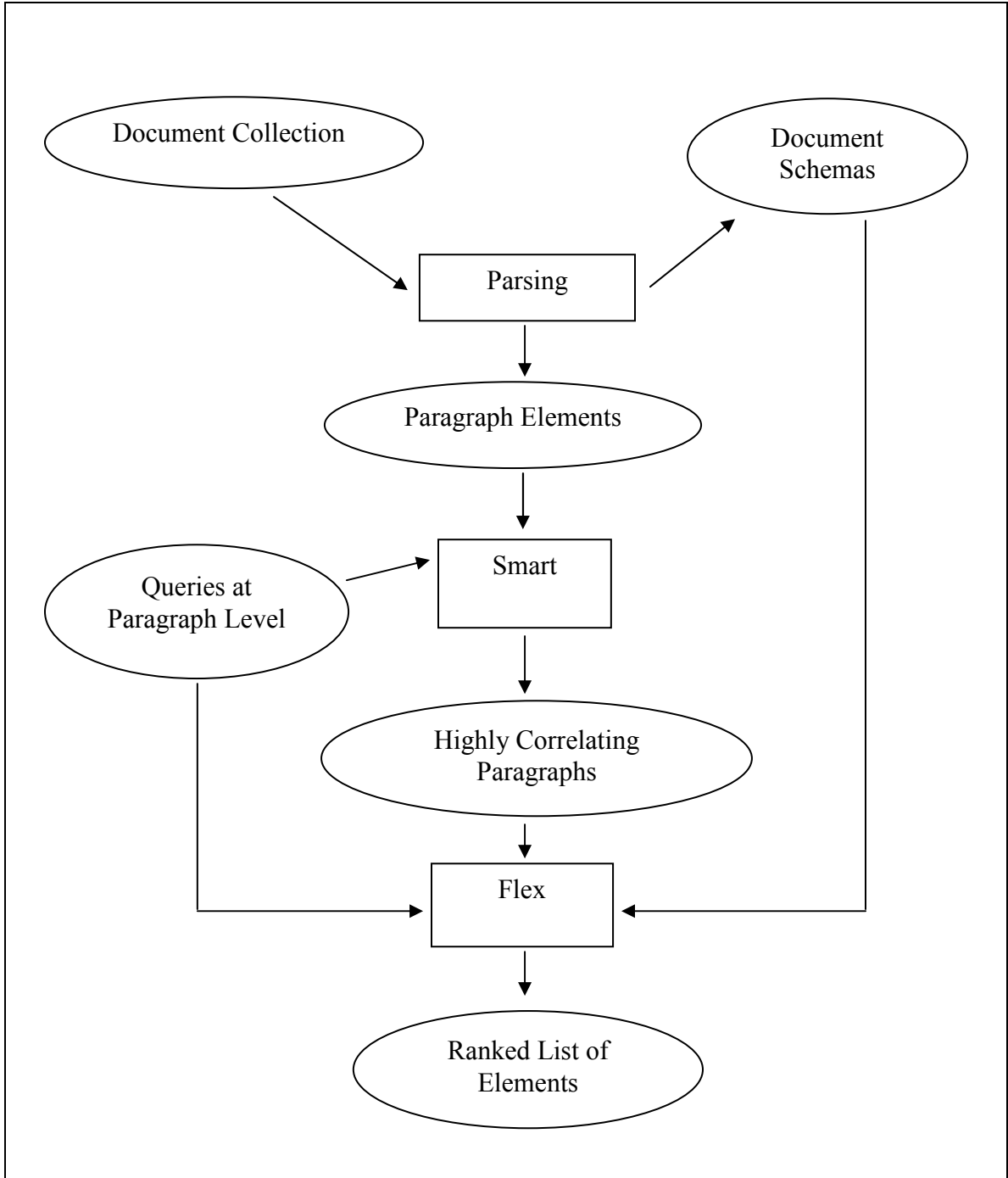


Fig 3.7 Retrieval using Flex

4. Importance of Untagged Text in Flex

Chapter 3 mentions the problem of the untagged text present in the Wikipedia collection. Fig 4.1 shows the structure of a typical semi-structured Wikipedia document. There is text present in the *section* which is not labeled with a *paragraph* tag. The presence of this untagged text causes problems in Flex. Flexible retrieval requires the presence of each discrete leaf node so that an exact copy of the parent element (*section* in our example) can be generated. Flex cannot generate the equivalent vector unless the untagged text can also be inserted in it. This is not a problem in all-element retrieval since any untagged text is automatically included in the parent element and indexed as part of it.

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

Fig 4.1A typical semi-structured Wikipedia document

We have designed an approach for solving this problem. Whenever untagged text is encountered, we create *magic tags* (labeled *<mt>*) and place the untagged text within these tags. Thus all the untagged text lying within an element is combined and labeled as an *<mt>* element. Thus each *article* or a *section* can have at most one magic element.

This magic element starts with `<mt>`, contains all the untagged text within that element and ends with `</mt>`. Fig 4.2 shows how we convert the semi-structured document of Fig 4.1 into a completely structured document.

```
<article>
  <section>
    <mt> text </mt>
    <p> text </p>
  </section>
</artcile>
```

Fig 4.2 A completely structured Wikipedia document

We now have a completely structured document. During paragraph indexing, a magic text element is treated in the same way as any other paragraph element. At this point, all discrete textual elements in the document are represented in vector form. We also generate the document schemas from the document collection. Magic text elements are included in the schemas in the same way that paragraph elements are included.

Magic elements are treated as normal paragraphs and retrieval is done using the paragraph index (which actually, in this case, contains both paragraphs and *mts*). We then select the top n elements this paragraph retrieval and seed them to Flex. Flex generates trees only for those documents which have at least one element in the top n . After getting the final list of ranked elements from Flex, a filtration process is carried out in which all

the magic text elements are removed. These elements do not exist in the original XML collection and thus cannot be returned, but we do return their immediate parent nodes. Chapter 6 describes the experiments which were carried out to ascertain the importance of magic text elements in the operation.

The output from both all-element retrieval and Flex is a ranked list of elements along with their similarity score. INEX expects the output in XML format. Certain parameters need to be included along with the ranked list of elements (participant ID, run description, name of the task, starting query ID, number of elements in the output, etc). These runs are then evaluated using the ep/gr and nxCG metrics described in Section 2.5.

5. Passage Retrieval

5.1 What is Passage Retrieval?

In earlier competitions, only XML elements were retrieved but as of INEX 2007, passages can be returned instead of pure XML elements. A passage can start anywhere within an element and end anywhere within the same or another element occurring later in document order [Clarke, Kamps, Lalmas, 2007].

There are various cases to be considered during passage identification. Fig 5.1 refers to an example where a passage contains only one element. A passage can contain more than one element; this is shown in Fig 5.2. A passage can start on an mt element (untagged text) and end on an element and vice-versa; this is evident in Fig 5.3. Finally, a passage can start and end on an mt element as seen in Fig 5.4. Passages are marked in bold in Figures 5.1 to 5.4.

```
<sarticle>
<section>
<p>
text
</p>
</section>
</article>
```

Fig 5.1 A passage containing only one element

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

Fig 5.2 A passage containing two elements

```
<article>
<section>
<mt> text </mt>
<p> text </p>
</section>
</article>
```

Fig 5.3 A passage starting on an mt element and ending on an element

```
<article>
.....
<section>
<mt> text </mt>
<p> text </p>
<mt> text </mt>
</section>
</article>
```

Fig 5.4 A passage starting and ending on an mt element

In INEX 2007 there is no separate passage retrieval task, but for all three tasks (Focused, Relevant in Context and Best in Context) arbitrary passages may be returned instead of elements. There will be a comparison on a query by query basis, to see if passage retrieval performs better than element retrieval.

5.2 Implementation of Passage Retrieval

Refer to Fig 5.5 for the algorithm to be used in passage retrieval and Fig 5.6 an overview of the process involved. The passage retrieval algorithm is explained in detail as follows.

1. Sort the elements (from initial retrieval) on the Smart-id.
2. Identify passages from consecutive elements.
3. Input the passage elements to Flex.
4. Filter out the terminal elements from the final output.

Fig 5.5 passage retrieval algorithm

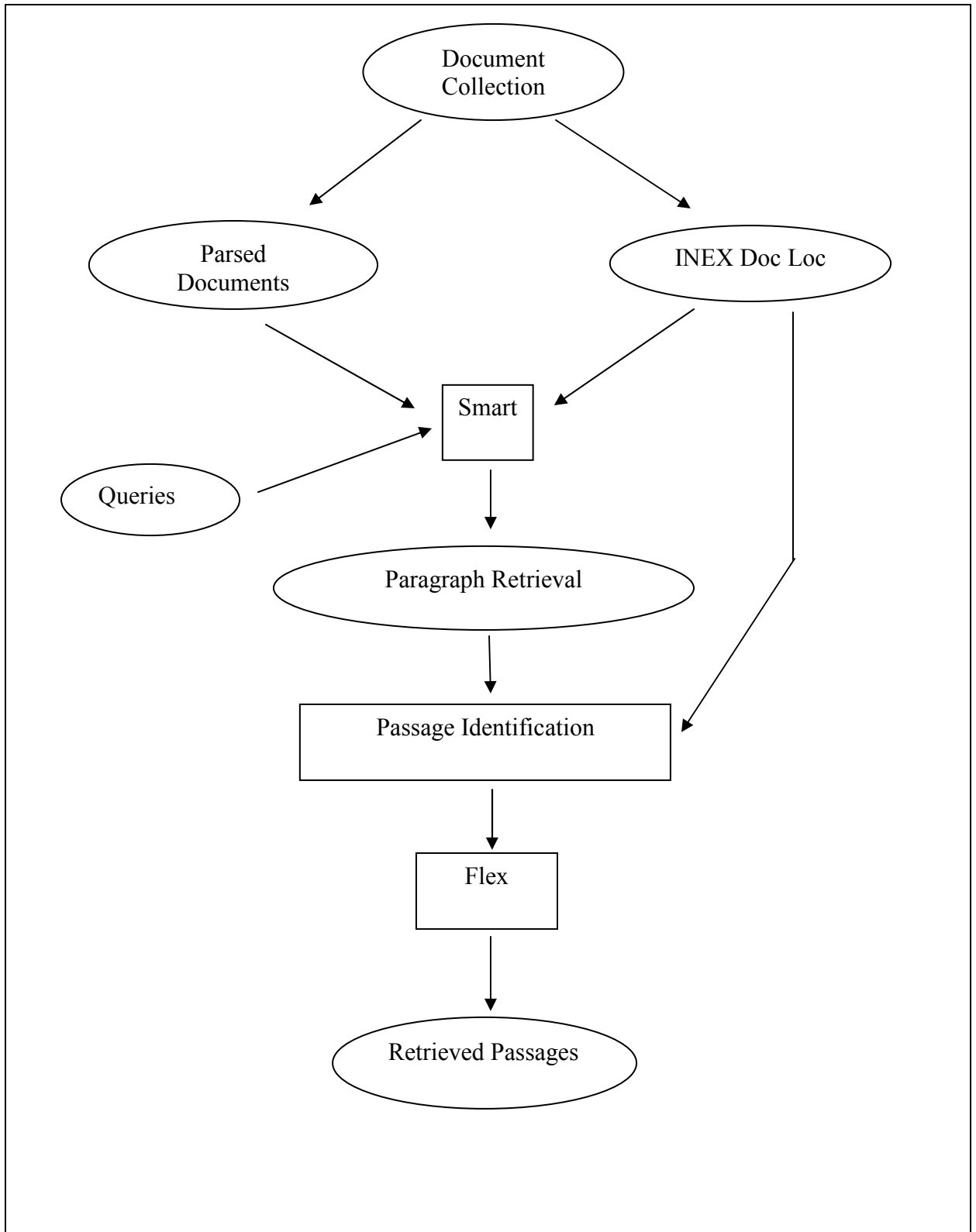


Fig 5.6 passage retrieval using Flex

First we sort the paragraph elements produced by Smart retrieval on their smart-id. Fig 5.7 shows a snippet from paragraph retrieval. Typical paragraph retrieval output contains

```

1 4905/article[1]/body[1]/p[5] 37.9339
1 632889/article[1]/body[1]/p[2] 36.9787
1 2281211/article[1]/body[1]/p[2] 36.8946
1 140367/article[1]/body[1]/section[1]/section[10]/p[2] 36.7723
1 62519/article[1]/body[1]/section[6]/normallist[1] 36.6526
1 258783/article[1]/body[1]/section[1]/section[1]/p[2] 36.3380
1 4905/article[1]/body[1]/p[2] 35.9491
1 288363/article[1]/body[1]/section[3]/p[2] 35.5694
1 734679/article[1]/body[1]/section[1]/p[4] 35.4655
1 22936/article[1]/body[1]/section[2]/p[4] 35.2657
.....

```

Fig 5.7 A snippet from paragraph retrieval

the query-id, x-path of the element, and similarity score. After sorting the elements on Smart-Id, the elements look like Fig 5.8. The sorted file contains the query-id, smart-id of the element and x-path of the element.

```

1 3 87731/article[1]/body[1]/p[1]
1 8 87731/article[1]/body[1]/p[6]
1 2546 80861/article[1]/body[1]/mt[1]
1 2547 80861/article[1]/body[1]/p[1]
1 2549 80861/article[1]/body[1]/section[1]/normallist[1]
1 4157 112414/article[1]/body[1]/section[2]/section[1]/mt[1]
1 4666 88326/article[1]/body[1]/section[2]/p[6]
1 4673 88326/article[1]/body[1]/section[4]/normallist[1]
1 7055 89605/article[1]/body[1]/p[2]
1 7100 81456/article[1]/body[1]/section[4]/p[2]
.....

```

Fig 5.8 Paragraph retrieval sorted on Smart Id

Passages are identified from consecutive elements. We can form $n*(n+1)/2$ number of passages from n number of consecutive elements. Fig 5.9 shows that for three consecutive elements with Smart-ids from 1189653 to 1189655, 6 ($3 * 4 / 2$) passages can be formed. The sections in bold are examples of passages that can be formed from one,

three and two consecutive elements. The passage schema is very similar to the document schema. See [Khanna, 2005] for the details of document schemas.

```

178328/article[1] 1 0
178328/article[1]/body[1] 9 0
178328/article[1]/body[1]/passage[1] 1 1
178328/article[1]/body[1]/p[1] 0 0 1189644 0
178328/article[1]/body[1]/passage[2] 1 1
178328/article[1]/body[1]/section[3]/p[1] 0 0 1189653 0
178328/article[1]/body[1]/passage[3] 2 1
178328/article[1]/body[1]/section[3]/p[1] 0 1 1189653 0
178328/article[1]/body[1]/section[3]/p[2] 0 0 1189654 0
178328/article[1]/body[1]/passage[4] 3 1
178328/article[1]/body[1]/section[3]/p[1] 0 1 1189653 0
178328/article[1]/body[1]/section[3]/p[2] 0 1 1189654 0
178328/article[1]/body[1]/section[3]/p[3] 0 0 1189655 0
178328/article[1]/body[1]/passage[5] 1 1
178328/article[1]/body[1]/section[3]/p[2] 0 0 1189654 0
178328/article[1]/body[1]/passage[6] 2 1
178328/article[1]/body[1]/section[3]/p[2] 0 1 1189654 0
178328/article[1]/body[1]/section[3]/p[3] 0 0 1189655 0
178328/article[1]/body[1]/passage[7] 1 1
178328/article[1]/body[1]/section[3]/p[3] 0 0 1189655 0

```

Fig 5.9 A typical passage schema

The passage retrieval process is described as follows. The passage element components can be fed to Flex for generating the passages vectors. For example, in Fig 5.9, when passage element components with Smart-ids 1189653, 1189654 and 1189655 are fed to Flex, passage vector for *passage[4]* can be generated. Similarity scores for the resultant passage vectors and query element vectors can be computed. Currently, we are using all-element query vectors which are approximations to the passage query vectors. (To get exact passage query vectors, we should have the correct values of N (total number of passages) and n_k (number of passages in which a particular term occurs).) After all the

elements are correctly scored, top ranking elements can be retrieved. A typical passage retrieval output might look like Fig 5.10.

```
1 211490/article[1]/body[1]/passage[8]
1 211490/article[1]/body[1]/passage[9]
1 13439/article[1]/body[1]/passage[26]
1 211490/article[1]/body[1]/passage[78]...
```

Fig 5.10 A typical passage retrieval output using Flex.

Finally, passage retrieval output can be mapped with the passage schemas to obtain the start and end of a passage so that the passage can be returned in terms of the document elements. The final output of passage retrieval might look like Fig 5.11. This output can be submitted for the Thorough task. The INEX 2007, there are also Focused, Relevant-in-context and Best-in-context tasks. Overlapping passages can be removed from the Through output to get the result for the Focused task. For the Relevant-in-context task, we can first return a ranked list of articles and for each article, we can return an unranked list of passages, each covering the relevant material in that article. Overlap is not permitted in this task. Finally, for the Best-in-context task, we can return a ranked list of articles and for each article, we can return a single passage, representing the best entry point for that article with respect to the topic.

```
query= 1 passage= 211491/article[1]/body[1]/passage[8]
start= 211491/article[1]/body[1]/section[1]/section[1]/p[1]
end= 211491/article[1]/body[1]/section[1]/section[2]/p[2]
query= 1 passage= 211490/article[1]/body[1]/passage[9]
start= 211490/article[1]/body[1]/section[1]/section[1]/p[1]
end= 211490/article[1]/body[1]/section[1]/section[1]
query= 1 passage= 13439/article[1]/body[1]/passage[26]
start= 13439/article[1]/body[1]/section[6]/normallist[1]
end= 13439/article[1]/body[1]/section[6]/p[3]
.....
```

Fig 5.11 Final passage retrieval output (Thorough)

6. Experiments and Results

We have conducted a number of experiments to fine tune the parameters used by our system as well as to evaluate its performance. We have used the document collection and queries available under INEX 2006 for our experiments since 2006 relevance assessment were available. The system is tested using the INEX evaluation metrics described in Section 2.5. We have submitted our runs for INEX 2007 and the relevance assessment will be out later during the year.

6.1 Experiments

We conducted many runs to find the best values for slope and pivot at various levels. We also conducted experiments to determine the importance of collection links in XML retrieval and experiments to determine the number of input nodes for Flex. Following is a description of these experiments along with the results obtained.

Slope and Pivot

We conducted numerous runs to calculate the best values of slope and pivot. We evaluated our runs using EvalJ software provided by INEX so that we could compare them. See Table 3.2 for the best INEX 2006 slope and pivot values. We also use a base run for all-element retrieval with a slope of 0.2 and pivot of 110. This is useful for

comparing various runs; these values were used in some of the early TREC experiments [Singhal, Buckley, Mitra, 1996].

6.1.1 Collection Links

A collection link is a link to a document in the collection. There are 17,021,858 collection links in the Wikipedia collection [Bakshi, 2006]. These constitute the largest number of elements in the whole collection (2GB in size). Table 3.1 lists the elements which we use for indexing the Wikipedia collection. Our original thoughts on the importance of collection links was that although they were useful for linking the collection, they did not contain enough information to be useful from a retrieval viewpoint. Since we index only meaningful elements, collection links are absent in Table 3.1. INEX 2006 has two types of Relevance Assessments, i.e., one without collection links (v4) and another which includes collection links (v5). We wanted to experiment with collection links and observe their importance in the retrieval. We filtered all the collection links from the Wikipedia collection and carried out the following experiments. The results are compared with identical runs which include collection links.

There are two cases. The first case uses tuned values of slope and pivot, second uses the TREC slope and pivot values. Both these cases are evaluated against the two versions of Relevance Assessments i.e., v4 and v5.

Case 1:

Collection: Wikipedia collection without collection links

All-element slope and pivot used: **0.06** and **28** respectively (tuned values) [base case].

Paragraph slope and pivot used: **0.11** and **15** respectively (tuned values) [initial retrieval].

Tables 6.1 and 6.2 show the results for the Thorough task. These results are evaluated using MAep [Kazai, Lalmas, 2006]. The columns show the MAep value for 10, 20, 50, 100, 500 and 1500 output elements. The first row describes the results for the all-element run whereas the subsequent rows describe the runs when Flex is seeded with 1000, 500,....., 1 leaf nodes, respectively. It is observed that Flex performs better than the all-element run in most of the cases. This is consistent with results obtained by [Mone, 2007]. See [Mone, 2007] for the rationale explaining why Flex performs better than the all-element run at some values of n.

Table 6.1 Task: Thorough, Relevance-Assessments: v4, Case 1

n	MAep@10	MAep@20	MAep@50	MAep@100	MAep@500	MAep@1500
all-el	0.0066	0.0092	0.0135	0.0165	0.0218	0.0238
1000	0.0074	0.0103	0.0150	0.0182	0.0239	0.0261
500	0.0074	0.0104	0.0150	0.0182	0.0240	0.0269
250	0.0074	0.0104	0.0159	0.0182	0.0244	0.0287
100	0.0074	0.0103	0.0149	0.0181	0.0264	0.0300
50	0.0073	0.0102	0.0148	0.0184	0.0282	0.0293
25	0.0074	0.0104	0.0154	0.0205	0.0281	0.0281
10	0.0073	0.0105	0.0176	0.0219	0.0248	0.0248
5	0.007	0.0113	0.0182	0.0207	0.0214	0.0214
1	0.0085	0.0123	0.0159	0.0163	0.0163	0.0163

Table 6.2 Task: Thorough, Relevance-Assessments: v5, Case 1

n	MAep@10	MAep@20	MAep@50	MAep@100	MAep@500	MAep@1500
all-el	0.0034	0.0048	0.0070	0.0086	0.0113	0.0123
1000	0.0039	0.0054	0.0079	0.0095	0.0125	0.0136
500	0.0039	0.0054	0.0078	0.0095	0.0125	0.0140
250	0.0039	0.0055	0.0078	0.0095	0.0127	0.0150
100	0.0038	0.0054	0.0078	0.0094	0.0138	0.0157
50	0.0038	0.0054	0.0078	0.0096	0.0148	0.0153
25	0.0039	0.0055	0.0080	0.0107	0.0146	0.0147
10	0.0038	0.0056	0.0093	0.0116	0.0130	0.0130
5	0.0038	0.0060	0.0096	0.0109	0.0112	0.0112
1	0.0043	0.0062	0.0080	0.0082	0.0082	0.0082

Tables 6.3 to 6.6 show the results for the Focused task. These results are evaluated using the nxCG value [Kazai, Lalmas, 2006]. nxCG is calculated at 5, 10, 25 and 50 output elements. The first row describes the all-element run whereas the following rows describe the results when Flex is seeded with 1000, 500, ..., 1 leaf nodes. Again, it is observed that Flex performs better than the All-element runs and results degrade if we input more than 10 leaf nodes to Flex.

Table 6.3 Task: Focused, Relevance-Assessments: v4, Overlap: OFF, Case 1

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	0.2810	0.2345	0.1892	0.1498
1000	0.2717	0.2254	0.1793	0.1452
500	0.2717	0.2254	0.1789	0.1455
250	0.2717	0.2263	0.179	0.1455
100	0.2717	0.2257	0.1802	0.1477
50	0.2715	0.2265	0.1847	0.1573
25	0.2781	0.2308	0.201	0.1925
10	0.2829	0.2474	0.2443	0.2064
5	0.2837	0.2895	0.2638	0.1939
1	0.3126	0.2625	0.1768	0.1076

Table 6.4 Task: Focused, Relevance-Assessments: v5, Overlap: OFF, Case 1

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	0.2715	0.2215	0.1834	0.1476
1000	0.2645	0.2195	0.1743	0.1402
500	0.2645	0.2195	0.174	0.1404
250	0.2645	0.2204	0.1741	0.1405
100	0.2645	0.2198	0.1752	0.1426
50	0.2643	0.2206	0.1797	0.1519
25	0.2708	0.2248	0.1955	0.1863
10	0.2755	0.2408	0.2377	0.1999
5	0.2763	0.2819	0.2566	0.1873
1	0.3043	0.2556	0.1719	0.1038

Table 6.7 describes the tree statistics for Flex with 1000, 500, ..., 1 leaf nodes. We note the average number of trees built per query in seconds, range of trees built, average time

per query and number of mts present in the output of 2500 elements. When we seed n leaf nodes to Flex, we can generate a minimum of 1 and a maximum of n trees. That is, all n leaf nodes might belong to a single document or to n different documents. We also observe that the more the leaf nodes fed to Flex, the more mts present in the output.

Table 6.5 Task: Focused, Relevance-Assessments: v4, Overlap: ON, Case 1

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	0.2235	0.1892	0.1515	0.1312
1000	0.2224	0.1839	0.1477	0.1257
500	0.2224	0.1839	0.1473	0.1260
250	0.2224	0.1848	0.1474	0.1258
100	0.2224	0.1843	0.1486	0.1270
50	0.2222	0.1851	0.1507	0.1324
25	0.2290	0.1876	0.1612	0.1445
10	0.2320	0.1937	0.1606	0.1276
5	0.2253	0.2034	0.1478	0.1086
1	0.217	0.1540	0.0905	0.0561

Table 6.6 Task: Focused, Relevance-Assessments: v5, Overlap: ON, Case 1

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	0.2208	0.1912	0.1429	0.1267
1000	0.2190	0.1800	0.1424	0.1154
500	0.2190	0.1800	0.1420	0.1156
250	0.2190	0.1809	0.1421	0.1155
100	0.2190	0.1803	0.1433	0.1164
50	0.2189	0.1811	0.1454	0.1215
25	0.2253	0.1835	0.1560	0.1335
10	0.2282	0.1903	0.1548	0.1171
5	0.2203	0.2003	0.1423	0.0988
1	0.2112	0.1495	0.0866	0.0505

Case 2:

Collection: Wikipedia collection with collection links

All-element slope and pivot used: **0.2** and **110** respectively (TREC values)

Paragraph slope and pivot used: **0.11** and **15** respectively (tuned values).

Table 6.7 Flex Tree Generation Statistics for Case 1

n	Average no of trees/query	Range of trees built	Average time per query in seconds	No. of mts in 2500 retrieved elements.
1000	642.48	15-913	6.608	25896
500	324.056	15-462	4.512	28016
250	162.816	15-234	2.136	25660
200	129.68	15-189	2.08	22660
150	97.176	15-142	1.928	18424
100	65.184	15-96	1.792	13264
50	32.856	15-50	1.128	7486
25	16.344	9-25	0.744	4245
10	6.752	2-10	0.312	2239
5	3.616	2-5	0.272	1367
1	0.888	1-1	0.2	436

Case 2 is similar to Case 1; the only difference is the all-element slope and pivot value, where TREC as opposed to tuned slope and pivot values are used. Tables 6.8 to 6.14 have the same experimental set up as Tables 6.1 to 6.7. We observe a similar trend in Case 2 as observed in Case 1.

Table 6.8 Task: Thorough, Relevance Assessments: v4, Case 2

n	MAep@10	MAep@20	MAep@50	MAep@100	MAep@500	MAep@1500
all-el	0.0066	0.0092	0.0135	0.0164	0.0218	0.0237
1000	0.0074	0.0103	0.0150	0.0182	0.0239	0.0261
500	0.0074	0.0104	0.0150	0.0182	0.0240	0.0269
250	0.0074	0.0104	0.0150	0.0182	0.0244	0.0287
100	0.0074	0.0103	0.0149	0.0181	0.0264	0.0300
50	0.0073	0.0102	0.0148	0.0184	0.0282	0.0293
25	0.0074	0.0104	0.0154	0.0205	0.0281	0.0281
10	0.0073	0.0105	0.0176	0.0219	0.0248	0.0248
5	0.0072	0.0113	0.0182	0.0207	0.0214	0.0214
1	0.0085	0.0123	0.0159	0.0163	0.0163	0.0163

Table 6.9 Task: Thorough, Relevance Assessments: v5, Case 2

N	MAep@10	MAep@20	MAep@50	MAep@100	MAep@500	MAep@1500
all-el	0.0034	0.0048	0.0070	0.0086	0.0113	0.0123
1000	0.0039	0.0054	0.0079	0.0095	0.0125	0.0136
500	0.0039	0.0054	0.0079	0.0095	0.0125	0.0140
250	0.0039	0.0055	0.0078	0.0095	0.0127	0.0150
100	0.0038	0.0054	0.0078	0.0094	0.0138	0.0157
50	0.0038	0.0054	0.0078	0.0096	0.0148	0.0153
25	0.0039	0.0055	0.0080	0.0107	0.0146	0.0147
10	0.0038	0.0056	0.0093	0.0116	0.0130	0.0130
5	0.0038	0.0060	0.0096	0.0109	0.0112	0.0112
1	0.0043	0.0062	0.0080	0.0082	0.0082	0.0082

Table 6.10 Task: Focused, Relevance Assessments: v4, Overlap: OFF, Case 2

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
All-el	0.3312	0.2856	0.2293	0.1782
1000	0.3206	0.2727	0.2028	0.1637
500	0.3188	0.2718	0.2032	0.1637
250	0.3206	0.2727	0.2041	0.1639
100	0.3209	0.2714	0.2076	0.1667
50	0.3207	0.2732	0.216	0.1868
25	0.3267	0.2809	0.2445	0.2226
10	0.3297	0.3038	0.2911	0.2293
5	0.3596	0.3582	0.3007	0.2081
1	0.3675	0.2931	0.1901	0.1133

Table 6.11 Task: Focused, Relevance Assessments: v5, Overlap: OFF, Case 2

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
All-el	0.3248	0.2873	0.3245	0.1846
1000	0.3121	0.2655	0.1972	0.1582
500	0.3104	0.2646	0.1976	0.1581
250	0.3121	0.2655	0.1985	0.1584
100	0.3125	0.2643	0.2019	0.1611
50	0.3123	0.266	0.2101	0.1806
25	0.3181	0.2735	0.2378	0.2156
10	0.3211	0.2958	0.2832	0.2222
5	0.3501	0.3487	0.2924	0.2011
1	0.3578	0.2854	0.1847	0.1093

Table 6.12 Task: Focused, Relevance Assessments: v4, Overlap: ON, Case 2

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
All-el	0.2615	0.2315	0.1729	0.1487
1000	0.2641	0.2203	0.1659	0.1403
500	0.2623	0.2194	0.1663	0.1401
250	0.2641	0.2203	0.1672	0.1402
100	0.2645	0.2200	0.1699	0.1416
50	0.2643	0.2209	0.1759	0.1525
25	0.2703	0.2262	0.1898	0.1590
10	0.2734	0.2303	0.1852	0.1389
5	0.2867	0.2535	0.1626	0.1153
1	0.2467	0.1675	0.0961	0.0568

Table 6.13 Task: Focused, Relevance Assessments: v5, Overlap: ON, Case 2

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	0.2619	0.2218	0.1754	0.1391
1000	0.2596	0.2155	0.1601	0.1293
500	0.2579	0.2146	0.1605	0.1291
250	0.2596	0.2155	0.1614	0.1292
100	0.2600	0.2151	0.1639	0.1305
50	0.2598	0.2169	0.1700	0.1406
25	0.2656	0.2221	0.1836	0.1475
10	0.2686	0.2269	0.1788	0.1279
5	0.2801	0.2314	0.1568	0.1052
1	0.2402	0.1626	0.0920	0.0512

Table 6.14: Flex Tree Generation Statistics for Case 2

n	Average no of trees/query	Range of trees built	Average time per query in seconds	No. of mts in 2500 retrieved elements.
1000	642.48	15-913	7.464	25883
500	324.056	15-462	5.152	28004
250	162.816	15-234	2.496	25659
200	129.68	15-189	2.456	22661
150	97.176	15-142	2.272	18424
100	65.184	15-96	2.096	13264
50	32.856	15-50	1.304	7486
25	16.344	9-25	0.84	4245
10	6.752	2-10	0.376	2239
5	3.616	2-5	0.312	1367
1	0.888	1-1	0.192	436

6.1.2 Leaf Nodes Input to Flex

Note: In the following experiments, “paragraph” means all the paragraph elements from Table 3.1 except mt unless otherwise stated. The number of leaf nodes to fed Flex is an important factor in our experiments because results vary greatly depending based on it. Not only are the numbers of leaf nodes important, but also the kind of leaf nodes as well. Appendix C describes the experiments when we seed paragraphs and mts to Flex. In Case 3, we input exactly n paragraphs to Flex whereas in Case 4, we select the top n elements from paragraph retrieval (i.e., elements consisting of paragraphs and mts) and then select only the paragraphs which are then seeded to Flex. If we input more leaf nodes to Flex, then more trees are generated. We want n to represent the number of leaf nodes for which we get good results yet still generate a minimum number of trees. From Cases 3 and 4, we are interested in observing the importance of mts in Flex. We believe that results from Appendix C should be better than results in Case 3 and Case 4. Our prediction is based on the fact that mts contain a lot of useful information and ignoring mts means losing that information with respect to retrieval.

Case 3: Exactly n paragraphs input to Flex

Collection: Wikipedia collection with collection links

All element slope and pivot used: **0.12 and 38** respectively (tuned values).

Paragraph slope and pivot used: **0.12 and 18** respectively (tuned values).

Table 6.15 describes the statistics on the number of mts that were ignored when we wanted to select the top n paragraphs from paragraph retrieval. We observe that a lot of

mts are present in the paragraph retrieval. Table 6.16 describes the tree generation statistics for Case 3. Tables 6.17 and 6.18 describe the results for the Thorough task whereas Tables 6.19 to 6.22 describe the results for the Focused task. It is observed that the all-element retrieval performs well when we consider 20 output elements and also for the Focused runs (with Overlap ON). See Appendix A for a more detailed experiment (on a query by query basis).

Table 6.15 Average number of mts ignored per query for Case 3

N	No. of mts ignored	Average no. of mts ignored / per query
1000	29273	234
500	14916	119
250	7614	61
200	6036	48
150	4490	36
100	2979	24
50	1585	13
25	825	7
10	367	3
5	193	2
1	44	0

Table 6.16 Flex Tree Generation Statistics for Case 3

n	Average no of trees/query	Range of trees built	Average time per query in seconds	No. of mts in 2500 retrieved elements.
1000	680.336	37-907	4.032	15748
500	344.880	37-465	2.088	23450
250	174.152	37-240	0.896	24360
200	139.656	37-192	0.616	22546
150	105.504	37-142	0.488	18788
100	70.856	37-92	0.408	13786
50	36.144	20-50	0.240	8003
25	18.336	10-25	0.176	4563
10	7.560	4-10	0.080	2366
5	3.920	1-5	0.064	1373
1	0.912	1-1	0.024	436

Table 6.17 Task: Thorough, Relevance Assessments: v4, Case 3

n	MAep@10	MAep@20	MAep@50	MAep@100	MAep@500	MAep@1500
all-el	0.0085	0.0122	0.0185	0.0238	0.0325	0.0356
1000	0.0084	0.0119	0.0182	0.0232	0.0315	0.0344
500	0.0084	0.0119	0.0182	0.0231	0.0313	0.0353
250	0.0084	0.0119	0.0182	0.0231	0.0315	0.0371
100	0.0085	0.0119	0.0180	0.0230	0.0340	0.0390
50	0.0085	0.0118	0.0178	0.0231	0.0356	0.0373
25	0.0084	0.0115	0.0180	0.0245	0.0341	0.0344
10	0.0082	0.0119	0.0194	0.0248	0.0277	0.0277
5	0.0080	0.0119	0.0181	0.0212	0.0219	0.0219
1	0.0080	0.0114	0.0142	0.0146	0.0146	0.0146

Table 6.18 Task: Thorough, Relevance Assessments: v5, Case 3

n	MAep@10	MAep@20	MAep@50	MAep@100	MAep@500	MAep@1500
all-el	0.0045	0.0065	0.0098	0.0126	0.0171	0.0188
1000	0.0045	0.0064	0.0096	0.0123	0.0166	0.0182
500	0.0045	0.0063	0.0096	0.0122	0.0166	0.0187
250	0.0045	0.0063	0.0096	0.0122	0.0167	0.0196
100	0.0045	0.0063	0.0095	0.0121	0.0179	0.0206
50	0.0045	0.0063	0.0094	0.0122	0.0188	0.0197
25	0.0045	0.0062	0.0095	0.0130	0.0180	0.0182
10	0.0044	0.0063	0.0103	0.0131	0.0146	0.0146
5	0.0043	0.0064	0.0097	0.0112	0.0116	0.0116
1	0.0043	0.0060	0.0074	0.0076	0.0076	0.0076

Table 6.19 Task: Focused, Relevance Assessments: v4, Overlap: OFF, Case 3

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	0.3653	0.3163	0.2575	0.2078
1000	0.3595	0.3090	0.2511	0.2013
500	0.3595	0.3090	0.2509	0.2009
250	0.3603	0.3090	0.2513	0.2007
100	0.3603	0.3090	0.2515	0.2019
50	0.3621	0.3090	0.2546	0.2080
25	0.3639	0.3137	0.2614	0.2425
10	0.3665	0.3222	0.2926	0.2395
5	0.3695	0.3324	0.2810	0.1924
1	0.3482	0.2793	0.1783	0.1030

Table 6.20 Task: Focused, Relevance Assessments: v5, Overlap: OFF, Case 3

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	0.3625	0.3134	0.2538	0.2042
1000	0.3569	0.3063	0.2476	0.1978
500	0.3569	0.3063	0.2474	0.1974
250	0.3576	0.3063	0.2478	0.1972
100	0.3576	0.3063	0.2479	0.1984
50	0.3594	0.3063	0.2513	0.2049
25	0.3611	0.3109	0.2576	0.2386
10	0.3637	0.3199	0.2891	0.2350
5	0.3672	0.3291	0.2771	0.1888
1	0.3424	0.2748	0.1756	0.1004

Table 6.21 Task: Focused, Relevance Assessments: v4, Overlap: ON, Case 3

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	0.3233	0.2795	0.2275	0.1960
1000	0.3175	0.2726	0.2201	0.1870
500	0.3175	0.2726	0.2200	0.1868
250	0.3183	0.2726	0.2203	0.1867
100	0.3183	0.2726	0.2205	0.1873
50	0.3201	0.2726	0.2229	0.1896
25	0.3219	0.2765	0.2258	0.1907
10	0.3227	0.2741	0.2013	0.1550
5	0.3186	0.2348	0.1656	0.1110
1	0.2329	0.1559	0.0864	0.0507

Table 6.22 Task: Focused, Relevance Assessments: v5, Overlap: ON, Case 3

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	0.3241	0.2787	0.2220	0.1815
1000	0.3185	0.2720	0.2147	0.1733
500	0.3185	0.2720	0.2145	0.1734
250	0.3192	0.2720	0.2149	0.1733
100	0.3192	0.2720	0.2150	0.1738
50	0.3209	0.2720	0.2177	0.1765
25	0.3227	0.2757	0.2202	0.1780
10	0.3252	0.2751	0.1992	0.1449
5	0.3217	0.2370	0.1641	0.1033
1	0.2352	0.1581	0.0865	0.0476

Case 4: At most n paragraphs input to Flex

In Case 3, we used the top n paragraphs from the paragraph retrieval for seeding Flex. We now have a different set of experiments. In Case 4, we select the top n elements from paragraph retrieval and then filter out the mts from these elements. The remaining elements are then seeded to Flex. We predict that Case 4 shouldn't perform better than Case 3. The reason is that in Case 3, we have more highly correlating paragraphs (as compared to Case 4) and thus more trees are generated in Case 3. Since these trees contain potentially relevant information, we get good results. Our prediction is supported by the tables in Appendix B. Appendix B describes experiments done on a query by query basis.

6.2 Results

The following are the observations from the experiments carried out in Section 6.1.

1. Results deteriorate when collection links are ignored from the collection.

We compare the results without collection links (i.e., Tables 6.1 to 6.14) with the results including collection links (Appendix C) and observe that the results go down if we ignore collection links. Collection links contain useful information and ignoring collection links means neglecting important keywords with respect to the retrieval. (also, mts contain collection links, so if mts are not seeded to Flex, that collection link information is lost as well).

2. Results deteriorate when exactly n paragraphs (from initial retrieval) are seeded to Flex.

Comparing Tables 6.16 to 6.22 with Appendix C shows that results go down when we input exactly n paragraphs to Flex. There are cases when mts are the only highly correlating elements from an article. We thus ignore articles which are identified by such mts, which results in the deterioration of the retrieval results.

3. Results deteriorate when at most n paragraphs (selected from top n elements of initial retrieval) are seeded to Flex.

Comparing Appendix B with Appendix C shows that results are worse when we consider paragraphs from only a specified number of elements from the paragraph retrieval. Results degrade because we select at most n paragraphs; we lose more articles than those lost when we consider exactly n paragraphs. This is because if more highly correlating paragraphs are seeded to Flex then more trees are generated thus improving the results. From Appendix B, we observe that for 5 queries, identical trees are generated when we consider top 100 paragraphs + mts and at most 100 paragraphs from the paragraph retrieval. This indicates that for 5 queries, mts don't make any difference. For all other queries, differences exist.

4. Performance

From Appendices A, B, and C we observe that results deteriorate as we change the number and type of elements fed to Flex (from top 100 paragraphs + mts, top 100 paragraphs, only paragraphs from top 100 paragraphs + mts).

7. Conclusions and Future Work

The aim of this thesis was to determine the importance of collection links and magic text in dynamic element retrieval. Experiments described in Chapter 6 show that retrieval results deteriorate when we ignore collection links and magic text in Flex. We know that collection links contain useful keywords and that ignoring these collection links, depreciates the results. Magic text, on the other hand, contains untagged text. Since our system expects a fully structured document, ignoring magic text means ignoring a lot of untagged text which actually might be of interest. Thus we conclude that collection links and magic text plays an important role in dynamic element retrieval.

We also develop a new algorithm for carrying out passage retrieval for the INEX 2007 adhoc track. We can compare the results of passage retrieval against element retrieval once the relevance assessments for INEX 2007 are released. This comparison will help us determine if the “best answer” is an element or a passage consisting of relevant text. Presently, in passage retrieval, we are not getting an exact similarity score as we are using all-element’s query for approximating the passage query. A method to determine a more accurate ranking criterion for passages may prove useful. Such a method might consist of two phases: identification of passages followed by the indexing of those passages with retrieval against them. Also, we are returning passages in terms of elements only. We can improve the relevance of these passages by devising new methods using the DOM approach which would define a passage in terms of the actual relevant characters as opposed to an element.

References:

- [1]. Initiative for the Evaluation of XML Retrieval (INEX)
<http://inex.is.informatik.uni-duisburg.de/>
- [2] Khanna, S. *Design and Implementation of a Flexible Retrieval System*, MS Thesis, University of Minnesota Duluth, 2005.
- [3] Denoyer, L; Gallinari, P. The Wikipedia XML corpus, 2006.
http://info.acm.org/sigir/forum/2006J/2006j_sigirforum_denoyer.pdf
- [4] Larsen, B. INEX 2006 Guidelines for Topic Development, 2006.
<http://inex.is.informatik.uni-duisburg.de/2006/inex06/pdf/NEXI.pdf>
- [5] INEX 2006 Adhoc Track
<http://inex.is.informatik.uni-duisburg.de/2006/adhoc.html>
- [6] Clarke, C; Kamps, J; Lalmas M. INEX 2006 Retrieval Task and Result Submission Specification
http://inex.is.informatik.uniduisburg.de/2006/inex06/pdf/INEX06_Tasks_v1.pdf
- [7] INEX 2006 Relevance Assessment Guide
http://inex.is.informatik.uni-duisburg.de/2006/inex06/adhoc_protected/downloads/Relevance_Assessment2006.pdf
- [8] Salton, G., editor. *The SMART Retrieval System – Experiments in Automatic Document Retrieval*, Prentice-Hall, Englewood Cliffs, NJ, 1971.
- [9] Kazai, G; Lalmas, M. eXtended Cumulated Gain Measures for the Evaluation of Content-Oriented XML Retrieval, *ACM TOIS*, 24(4), 2006.
- [10] Bakshi, V. *Flexible Retrieval for the Semi-Structured Documents*. MS Thesis, University of Minnesota Duluth, 2006.
- [11] Ganapathibhotla, M. *Query Processing in a Flexible Retrieval Environment*. MS Thesis, University of Minnesota Duluth, 2006.
- [12] Singhal, A; Buckley, C; Mitra, M. Pivoted Document Length Normalization. *In Proc. of the 19th Annual International ACM SIGIR Conference* (Zurich, 1996).
- [13] Clarke, C; Kamps, J; Lalmas M. INEX 2007 Retrieval Task and Submission Spec.
http://inex.is.informatik.uni-duisburg.de/2007/inex07/pdf/INEX07_Tasks_v1.pdf
- [14] Mone, A. *Dynamic Element Retrieval for Semi-Structured Documents*. MS Thesis, University of Minnesota Duluth, 2007.

Appendix A

A.1. Exactly 100 Paragraphs Seeded to Flex

Note: In the following experiment “paragraph” means all paragraph elements from Table 3.1 except mt unless otherwise stated. We have observed that Flex performs well when seeded with 100 leaf nodes so for these experiments we use 100 leaf nodes as input to Flex.

In this experiment, top 100 paragraphs were selected for each query from the paragraph retrieval and these were then seeded to Flex. The following are the observations -

Number of INPUT elements to Flex = 100

Number of OUTPUT elements from Flex = 1500

Collection:

Wikipedia with collection links

Slope and Pivot:

All elements Slope and Pivot: 0.12 and 38

Our Slope and Pivot: 0.12 and 18

Relevance Assessments: v5 (with collection links)

Experiments:

Case1: Input to Flex – Paragraphs and mts

Case2: Input to Flex – Paragraphs

Table A.1 Query by query analysis when exactly 100 paragraphs seeded to Flex

Query	# Articles that are present in Case1 only	# Articles present in Case1 only and relevant	# Articles that are present in Case2 only	# Articles present in Case2 only and are relevant
1	11	4	12	3
2	8	4	13	1
3	14	2	14	2
4	12	2	12	0
5	12	3	10	1
6	15	5	16	3
7	15	4	13	5
8	17	13	16	3
9	0	0	0	0
10	18	15	20	8
11	No Assessments	No Assessments	No Assessments	No Assessments
12	6	0	6	4
13	22	4	27	4
14	19	1	22	0
15	15	1	17	0
16	9	2	12	0
17	9	1	11	0
18	0	0	0	0
19	0	0	0	0
20	10	5	17	3
21	13	0	12	0
22	8	1	11	0
23	7	4	9	1
24	13	5	17	5
25	15	5	16	1
26	11	3	13	1
27	9	2	9	0
28	15	0	16	0
29	7	1	8	0
30	4	0	10	0
31	2	1	6	2
32	5	1	4	0
33	16	5	20	2
34	15	2	14	1
35	1	0	4	0
36	5	2	17	4
37	9	5	12	1
38	15	3	12	0
39	0	0	0	0
40	50	0	48	8

41	24	1	23	1
42	2	1	2	1
43	7	0	11	6
44	9	4	9	3
45	6	1	9	0
46	5	3	7	2
47	5	0	7	0
48	12	2	19	0
49	0	0	2	1
50	6	0	9	2
51	9	0	17	0
52	10	1	11	0
53	13	10	16	2
54	10	7	16	5
55	49	23	54	0
56	10	1	13	2
57	8	0	12	0
58	15	0	19	0
59	7	0	7	0
60	7	0	10	0
61	18	1	15	0
62	0	0	0	0
63	3	0	9	0
64	8	6	11	0
65	9	0	8	0
66	23	4	17	1
67	7	1	7	0
68	19	3	24	0
69	7	7	6	6
70	9	2	13	0
71	22	2	17	1
72	13	3	12	1
73	0	0	0	0
74	5	1	0	0
75	22	0	18	1
76	4	0	7	0
77	0	0	1	1
78	12	2	18	2
79	No Assessments	No Assessments	No Assessments	No Assessments
80	22	7	24	4
81	16	1	17	1
82	No Assessments	No Assessments	No Assessments	No Assessments
83	0	0	18	1
84	13	1	12	0

85	11	1	24	1
86	23	1	13	0
87	9	2	8	4
88	4	0	11	1
89	No Assessments	No Assessments	No Assessments	No Assessments
90	0	0	0	0
91	5	0	9	1
92	9	1	7	2
93	21	0	22	0
94	19	6	17	0
95	21	4	21	1
96	18	4	22	1
97	6	3	11	1
98	12	5	8	3
99	16	12	20	9
100	4	2	0	0
101	No Assessments	No Assessments	No Assessments	No Assessments
102	14	1	7	0
103	28	3	25	6
104	12	1	15	2
105	No Assessments	No Assessments	No Assessments	No Assessments
106	7	2	8	2
107	12	2	13	3
108	No Assessments	No Assessments	No Assessments	No Assessments
109	No Assessments	No Assessments	No Assessments	No Assessments
110	No Assessments	No Assessments	No Assessments	No Assessments
111	10	2	15	3
112	0	0	0	0
113	12	1	13	1
114	0	0	0	0
115	0	0	16	2
116	0	0	0	0
117	15	0	16	0
118	11	5	12	1
119	20	1	20	1
120	No Assessments	No Assessments	No Assessments	No Assessments
121	0	0	0	0
122	19	2	20	0
123	4	0	6	0
124	No Assessments	No Assessments	No Assessments	No Assessments
125	0	0	0	0
TOTAL	1220	262	1402	152
AVG	12	3	14	1

Appendix B

B.1. At Most 100 Paragraphs seeded to Flex

Note: In the following experiment “paragraph” means all paragraph elements from Table 3.1 except mt unless otherwise stated. We have observed that Flex performs well when seeded with 100 leaf nodes so for these experiments we use 100 leaf nodes as input to Flex.

In this experiment, paragraphs were separated from the top 100 elements of the paragraph retrieval and they were then seeded to Flex. The following are the observations -

Number of INPUT elements to Flex = 100

Number of OUTPUT elements from Flex = 1500

Collection:

Wikipedia with collection links

Slope and Pivot:

All elements Slope and Pivot: 0.12 and 38

Our Slope and Pivot: 0.12 and 18

Relevance Assessments: v5 (with collection links)

Experiments:

Case1: Input to Flex – Paragraphs and mts

Case2: Input to Flex – Paragraphs

Table B.1 Query by query analysis when at most 100 paragraphs seeded to Flex

Query	# Articles that are present in Case1 only	# Articles present in Case1 only and are relevant	# Articles that are present in Case2 only	# Articles present in Case2 only and are relevant
1	11	4	0	0
2	8	4	0	0
3	14	2	0	0
4	13	3	0	0
5	12	3	0	0
6	16	6	0	0
7	16	5	0	0
8	18	14	0	0
9	21	2	0	0
10	19	16	0	0
11	No Assessments	No Assessments	No Assessments	No Assessments
12	6	0	0	0
13	24	6	0	0
14	20	1	0	0
15	17	2	0	0
16	9	2	0	0
17	9	1	0	0
18	5	1	0	0
19	0	0	0	0
20	10	5	0	0
21	15	0	0	0
22	8	1	0	0
23	7	4	0	0
24	16	7	0	0
25	15	5	0	0
26	11	3	0	0
27	9	2	0	0
28	15	0	0	0
29	7	1	0	0
30	9	1	0	0
31	3	1	0	0
32	5	1	0	0
33	17	5	0	0
34	15	2	0	0
35	1	0	0	0
36	14	2	0	0
37	10	5	0	0
38	16	3	0	0
39	6	1	0	0
40	51	0	0	0

41	24	1	0	0
42	2	1	0	0
43	7	0	0	0
44	9	4	0	0
45	6	1	0	0
46	5	3	0	0
47	5	0	0	0
48	14	3	0	0
49	16	0	0	0
50	6	0	0	0
51	9	0	0	0
52	10	1	0	0
53	14	11	0	0
54	10	7	0	0
55	49	23	0	0
56	16	1	0	0
57	8	0	0	0
58	18	0	0	0
59	7	0	0	0
60	7	0	0	0
61	20	2	0	0
62	10	0	0	0
63	10	1	0	0
64	8	6	0	0
65	11	0	0	0
66	25	4	0	0
67	7	1	0	0
68	20	3	0	0
69	8	8	0	0
70	9	2	0	0
71	23	2	0	0
72	15	4	0	0
73	0	0	0	0
74	12	1	0	0
75	23	0	0	0
76	5	0	0	0
77	8	1	0	0
78	14	2	0	0
79	No Assessments	No Assessments	No Assessments	No Assessments
80	26	8	0	0
81	17	1	0	0
82	No Assessments	No Assessments	No Assessments	No Assessments
83	0	0	0	0
84	12	1	0	0

85	24	1	0	0
86	9	1	0	0
87	4	2	0	0
88	5	0	0	0
89	No Assessments	No Assessments	No Assessments	No Assessments
90	27	3	0	0
91	9	0	0	0
92	9	1	0	0
93	23	1	0	0
94	20	7	0	0
95	23	4	0	0
96	20	5	0	0
97	6	3	0	0
98	12	5	0	0
99	16	12	0	0
100	4	2	0	0
101	No Assessments	No Assessments	No Assessments	No Assessments
102	14	1	0	0
103	29	4	0	0
104	13	2	0	0
105	No Assessments	No Assessments	No Assessments	No Assessments
106	7	2	0	0
107	12	2	0	0
108	No Assessments	No Assessments	No Assessments	No Assessments
109	No Assessments	No Assessments	No Assessments	No Assessments
110	No Assessments	No Assessments	No Assessments	No Assessments
111	10	2	0	0
112	0	0	0	0
113	15	1	0	0
114	0	0	0	0
115	15	3	0	0
116	8	4	0	0
117	15	0	0	0
118	11	5	0	0
119	22	1	0	0
120	No Assessments	No Assessments	No Assessments	No Assessments
121	10	1	0	0
122	21	2	0	0
123	4	0	0	0
124	No Assessments	No Assessments	No Assessments	No Assessments
125	39	2	0	0
TOTAL	1479	303	0	0
AVG	14	3	0	0

Table B.2 Tree Generation Statistics when at most 100 paragraphs seeded to Flex

n	Average no of trees/query	Range of trees built	Average time per query	mt removed / 2500 elements retrieved.
Paragraphs + mt =100	76.166	41-97	0.184	14605
Paragraphs = 100	70.856	37-92	0.408	13786
Paragraphs<=100	56.664	37-89	1.272	11435

Table B.3 Task: Thorough, Relevance Assessments: v4, at most 100 paragraphs seeded to Flex

n	MAep @10	MAep @20	MAep @50	MAep @100	MAep @500	MAep @1500
Paragraphs+mt =100	0.0086	0.0122	0.0185	0.0237	0.0361	0.0415
Paragraphs=100	0.0085	0.0119	0.0180	0.0230	0.0340	0.0390
Paragraphs<=100	0.0085	0.0118	0.0178	0.0228	0.0345	0.0384

Table B.4 Task: Thorough, Relevance Assessments: v5, at most 100 paragraphs seeded to Flex

n	MAep @10	MAep @20	MAep @50	MAep @100	MAep @500	MAep @1500
Paragraphs+mt =100	0.0046	0.0065	0.0098	0.0125	0.0190	0.0219
Paragraphs=100	0.0045	0.0063	0.0095	0.0121	0.0179	0.0206
Paragraphs<=100	0.0045	0.0063	0.0095	0.0120	0.0182	0.0203

Table B.5 Task: Focused, Relevance Assessments: v4, Overlap: OFF, at most 100 paragraphs seeded to Flex

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
Paragraphs+mt=100	0.3622	0.3128	0.2514	0.2053
Paragraphs = 100	0.3603	0.3090	0.2515	0.2019
Paragraphs<= 100	0.3621	0.3090	0.2529	0.2033

Table B.6 Task: Focused, Relevance Assessments: v5, Overlap: OFF, at most 100 paragraphs seeded to Flex

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
Paragraphs+mt=100	0.3595	0.31	0.2479	0.2019
Paragraphs = 100	0.3576	0.3063	0.2479	0.1984
Paragraphs<= 100	0.3594	0.3063	0.2493	0.1997

Table B.7 Task: Focused, Relevance Assessments: v4, Overlap: ON, at most 100 paragraphs seeded to Flex

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
Paragraphs+mt=100	0.3202	0.2755	0.2209	0.1916
Paragraphs = 100	0.3183	0.2726	0.2205	0.1873
Paragraphs<= 100	0.3201	0.2726	0.2216	0.1881

Table B.8 Task: Focused, Relevance Assessments: v5, Overlap: ON, at most 100 paragraphs seeded to Flex

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
Paragraphs+mt=100	0.3211	0.2748	0.2157	0.1779
Paragraphs = 100	0.3192	0.2720	0.2150	0.1738
Paragraphs<= 100	0.3209	0.272	0.2161	0.1748

Appendix C

All element slope and pivot used: **0.12** and **38** respectively

Paragraph slope and pivot used: **0.12** and **18** respectively

Table C.1 Task: Thorough, Relevance Assessments: v4, collection links included

n	MAep@10	MAep@20	MAep@50	MAep@100	MAep@500	MAep@1500
all-el	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 C.2 Task: Thorough, Relevance Assessments: v5, collection links included

n	MAep@10	MAep@20	MAep@50	MAep@100	MAep@500	MAep@1500
all-el	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 C.3 Task: Focused, Overlap: OFF, Rel-Assess: v4, collection links included

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	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 C.4 Task: Focused, Overlap: OFF, Relevance Assessments: v5,
collection links included**

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	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 C.5 Task: Focused, Overlap: ON, Relevance Assessments: v4,
collection links included**

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	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 C.6 Task: Focused, Overlap: ON, Relevance Assessments: v5,
collection links included**

n	nxCG@5	nxCG@10	nxCG@25	nxCG@50
all-el	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 C.7 Tree Generation Statistics, collection links included

N	Average no of trees/query	Range of trees built	Average time per query in seconds	MT removed / 2500 elements retrieved.
1000	731.736	41-905	1.504	30588
500	370.947	41-461	0.784	30827
250	187.061	41-241	0.408	27224
200	150.131	41-193	0.328	24338
150	112.877	41-146	0.246	19818
100	76.166	41-97	0.184	14605
50	38.684	23-49	0.096	8419
25	19.570	12-25	0.048	4745
10	8.1491	3-10	0.024	2399
5	4.2894	1-5	0.016	1423
1	1	1-1	0	447