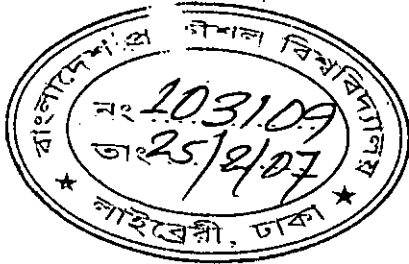
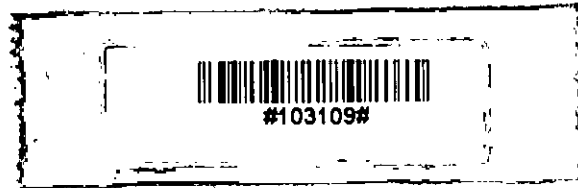


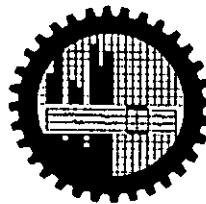
# An Efficient Information Retrieval System for Bangla Text Database



By  
Mohammad Amir Sharif  
Student No. 040305031



A thesis submitted to the Department of Computer Science and Engineering in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

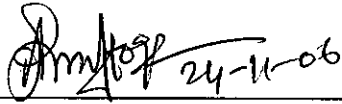


Department of Computer Science and Engineering  
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY


November 2006

The thesis “An Efficient Information Retrieval System for Bangla Text Database”, submitted by Mohammad Amir Sharif, Roll No. 040305031P, Session: April 2003, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Master of Science in Engineering (Computer Science and Engineering) and approved as to its style and contents for the examination held on November 24, 2006.

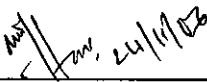
### Board of Examiners

-   
\_\_\_\_\_

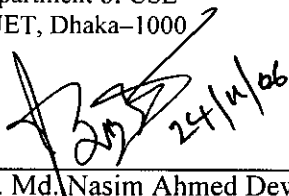
1. Dr. Abu Sayed Md. Latiful Hoque  
Associate Professor  
Department of CSE  
BUET, Dhaka-1000

Chairman  
(Supervisor)
-   
\_\_\_\_\_

2. Dr. Muhammad Masroor Ali  
Professor and Head  
Department of CSE  
BUET, Dhaka-1000

Member  
(Ex-officio)
-   
\_\_\_\_\_

3. Dr. Masud Hasan  
Assistant Professor  
Department of CSE  
BUET, Dhaka-1000

Member
-   
\_\_\_\_\_

4. Dr. Md. Nasim Ahmed Dewan  
Associate Professor  
Department of EEE  
BUET, Dhaka-1000

Member  
(External)

---

# Declaration

---

It is hereby declared that the work presented in this thesis or any part of the thesis has not been submitted elsewhere for the award of any degree or diploma.

Signature

*Mohammad Amir Sharif*  
-----  
*24/11/06*

Mohammad Amir Sharif

---

# Table of Content

---

Table of Content	i
List of Tables	iv
List of Figures	v
Acknowledgement	vi
Abstract	vii
<b>CHAPTER 1: INTRODUCTION</b>	<b>1</b>
1.1 Information Retrieval Systems and Related Issues	1
1.2 Characteristics of Information Retrieval System	2
1.3 Problem of Information Retrieval in Bangla	2
1.4 Objective of the Thesis	3
1.5 Organization of the Thesis	3
<b>CHAPTER 2: LITERTURE REVIEW</b>	<b>5</b>
2.1 Introduction	5
2.2 Different Information Retrieval Systems	6
2.2.1 Boolean Model	6
2.2.2 Probabilistic Model	7
2.2.3 Vector Space Model	9
2.2.3.1 Building Term Vectors in Document Space	10
2.2.3.2 Computation of Similarity between Documents and Query	12
2.2.3.3 Latent Semantic Indexing	15
2.2.3.4 IR System in Bangla using Vector Space Model	18
2.3 Comparison of Different IR systems	19
2.4 Indexing of Documents	19
2.5 Information Retrieval on the Web	22

2.6 Relevance Feedback	23
2.7 Evaluation of IR Performance	24
2.7.1 Precision and Recall	25
<b>CHAPTER 3: INFORMATION RETRIEVAL SYSTEM</b>	<b>27</b>
<b>FOR BANGLA TEXT DATABASE</b>	
3.1 System Architecture	27
3.1.1 Database Initialization and Processing	28
3.1.1.1 Document Database Creation	28
3.1.1.2 Creation of Stoplist	29
3.1.1.3 Keyword Processor	29
3.1.1.4 Font Handler	29
3.1.1.5 Morphological Analyzer	30
3.1.1.6 Creation of Synonymlist	30
3.1.1.7 New Document Addition	30
3.1.2 Storage	30
3.1.2.1 Storage of Synonymlist	31
3.1.2.2 Storage of Term Information	32
3.1.3 Query Execution	32
3.2 Analytical Representation of the Architecture	33
3.2.1 Analysis of the Initialization and Processing	33
3.2.2 Conversion of Non-unicode Documents into Unicode	34
3.2.3 Finding the Root of the Terms in a Document	36
3.2.4 Setup of the IR System	41
3.2.5 Methodology for New Document addition into the Database	45
3.2.6 Information Retrieval Query Processing on Bangla Text Database	48
3.2.6.1 Creation of Vector Space Representation	48
3.2.6.2 Finding the Relevance of Documents	49
3.2.7 Complexity Analysis	50

3.2.7.1 Creation of Index	50
3.2.7.2 Update Operation	51
3.2.7.3 Querying the Database	51
<b>CHAPTER 4: EXPERIMENTAL RESULTS AND DISCUSSION</b>	<b>52</b>
4.1 Experimental Setup	52
4.2 Postfixes Statistics	54
4.3 Creation of Stoplist for the Dataset	57
4.4 Stemming Performance	58
4.5 The Effect of Number of Occurrences of Postfixes	59
4.6 The Effect of Number of Terms in Query Expression	61
4.7 The Effect of Consideration of Document Length	63
4.8 The Effect of Morphological Analysis	64
4.9 Comparison with Other IR Systems	66
<b>CHAPTER 5: CONCLUSIONS AND FUTURE RESEARCH</b>	<b>69</b>
5.1 Conclusion	69
5.2 Suggestions for Further Research	70
<b>Bibliography</b>	<b>71</b>

---

## List of Tables

---

<b>Table No.</b>	<b>Title</b>	<b>Page No.</b>
2.1	Document term matrix	17
2.2	Document term matrix for two principal components	17
2.3	Example (Inverted File)	20
2.4	Relation between relevant and non-relevant document	25
3.1	Synonyms	31
3.2	Bangla characters with unicode and ascii value	35
3.3	Postfix list	37
4.1	Document information	52
4.2	Postfix statistics	55
4.3	List of stopwords with document frequency	57
4.4	Stemming results	58
4.5	Precision and recall for frequent postfix without morphological analysis	60
4.6	Precision and recall for frequent postfix with morphological analysis	60
4.7	Precision and recall (one word)	61
4.8	Precision and recall (two word)	62
4.9	Precision and recall (three word )	62
4.10	Retrieval time for query terms	62
4.11	Precision and recall ( not considering document length)	63
4.12	Precision and recall (considering document length)	63
4.13	Relevance for different query expression	65
4.14	Precision and recall ( not considering morphology)	65
4.15	Precision and recall (considering morphology)	65
4.16	Different steps of different IR system	66
4.17	Precision and recall(not considering morphology and document length)	67
4.18	Precision and recall(considering morphology and document length)	67

---

## List of Figures

---

Figure No.	Title	Page No.
2.1	Cosine distance of two documents for a query	14
2.2	Inverted indexing file	21
3.1	System architecture for Bangla text retrieval	28
3.2	Storage of term information	32
3.3	Algorithm to create document database	33
3.4	Algorithm to convert non-unicode supported text to unicode supported text	34
3.5	Algorithm for conversion of non-unicode document into unicode	36
3.6	Algorithm for finding the root of a word	38
3.7	Character sequence of postfixes	39
3.8	Flowchart for the set up of the database for information retrieval (first scan)	43
3.9	Flowchart for database initialization (second scan)	44
3.10	Algorithm to update occur table	45
3.11	Flowchart for new document addition	46
3.12	Algorithm for updating the synonym table for new document	47
3.13	Algorithm to add term information from temporary table to word table	47
3.14	Query execution process	48
3.15	Vector space representation of document and query vector	49
3.16	Algorithm to display relevant document	50
4.1	Relation between document number and required stemming	58
4.2	Relation between document number and stemming time	59
4.3	Precision-recall curve for frequent postfix	61
4.4	Effect of number of query term in query expression	62
4.5	Consideration of document length	64
4.6	Precision-recall curve for morphological analysis	66
4.7	Precision-recall curve for morphological analysis and document length	68



---

## Acknowledgement

---

The name that comes at first with the sincere gratitude from the core of my heart is my thesis supervisor Dr. Abu sayed Md. Latiful Hoque for his invaluable contribution to make the research as an art of exploiting new idea and technology in the field of information retrieval. He made me enthusiastic to find new ideas through excellent guideline and moral courage. His profound knowledge and expertise in the field of information retrieval have made me able to know many things about information retrieval.

I acknowledge Dr. Muhammad Masroor Ali, Head of the Department of Computer Science and Engineering, BUET for his kind co-operation to complete the thesis.

I would like to express my heartiest gratitude to my parents who always inspired me to complete the thesis work. I would also thank my colleagues of Mawlana Bhashani Science and Technology University, Tangail who helped me cordially to continue my research.

---

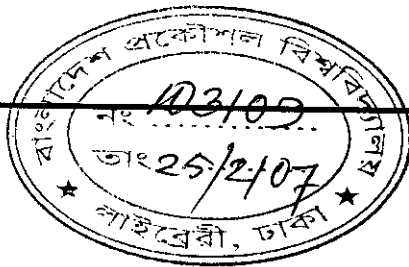
## Abstract

---

The amount of information available in electronic form is growing exponentially, making it increasingly difficult to find the desired information. Information retrieval is primarily concerned with the storage and retrieval of information. Thus, along with the growth of the World Wide Web, information retrieval systems gain importance since they are often the only way to find the few documents actually relevant to a specific query in the vast quantities of text available. Although information retrieval systems mainly deal with natural language, linguistic methods are rarely used. The mechanical cutting off of inflectional and derivational suffixes to better match index terms to query terms is called stemming. Since most research on information retrieval is done for English, which has a relatively weak morphology, this is not regarded as problematic for stemming. Stemming and more linguistically motivated methods show a positive impact on retrieval performance for language such as Dutch, German, Italian, or Bangla, which are morphologically richer than English.

There is so much variation of words in Bangla having similar meaning. So stemming is required to find the root of the words having similar meaning by doing morphological analysis. IR system's performance is affected due to synonyms. This problem is even worse in Bangla than English. Existing Bangla text database contains both unicode and non-unicode texts. It is difficult to search uniformly the database with both the types of text.

We have developed an efficient information retrieval system with morphological analysis to stem the word. The experimental results show that up to 20% better precision with 14% better recall can be achieved for Bangla by using around 150 non-intuitive stemming rules. We have developed a dictionary-based synonym handling technique to store the synonyms and access the database with the consideration of the synonyms. We have developed a technique to access the database irrespective of the type of encoding of the text.



# Chapter 1 Introduction

## 1.1 Information Retrieval Systems and Related Issues

Although the term “information retrieval” seems to be very wide, information retrieval generally focuses on narrative information. In information retrieval systems, information is organized into documents and it is assumed that there is a large number of documents. Data contained in the documents is unstructured, lacking any associated schema. The process of information retrieval consists of locating relevant documents, on the basis of user input, such as keywords or example documents.

Sometimes information retrieval (IR) is used as a more general term, covering all kinds of retrieval tasks; document retrieval or text retrieval. Very often, however, information retrieval and document retrieval are used synonymously as document retrieval is the prevalent area of research.

The amount of information available in electronic form is growing exponentially, making it increasingly difficult to find the desired information. Along with the growth of information, information retrieval (IR) system gains importance since they are often the only way to find the few documents actually relevant to a specific query in vast quantity of text available.

The broad ranges of applications of information retrieval system are:

- i. Research in libraries by making bibliographic data available for searching,
- ii. Retrieval of information about similar cases from large databases of legal decisions to help decision making by lawyers and judges,
- iii. Providing vendors with information about how many units of a particular product are on stock,
- iv. Helping empirical linguists by making it possible to manage, search and evaluate large corpora,
- v. Finding the desired information from the Internet.

---

## 1.2 Characteristics of Information Retrieval System

For any information retrieval system, there are generally four main things. Firstly, there must be a process by which the documents will be represented that is the whole document collection,  $D$  will be a collection of formal representation of documents  $\{d_j\}$ . Here a document may be represented as a vector of keywords present in the document. Secondly, the user information need (queries) must be represented as a formal way that is the query expression  $Q$ , must have the possible formal representation  $\{q_i\}$ , in terms of query terms. Thirdly, after the representation of query expression and document collection, there must be a framework  $F$ , by which we can make a modeling with these two. Lastly, we need a ranking function by which we can find the similarity (relevance) of the documents with respect to query expression. After getting the relevance, the documents are sorted in descending order with the value of the relevance. Then the top document in the sorted list is marked as most relevant with respect to the query.

## 1.3 Problem of Information Retrieval in Bangla

Finding the relevant documents from the Bangla text database has significantly different characteristics than the same for English text database. In Bangla, there are so many variations of words. By just adding some postfixes, many different words can be formed. But these varied words have similar meaning. So unless any mechanism is adopted, the varied words will be considered as distinct words. But as all the words have similar meaning, so all these words are relevant for query with any of the words. Otherwise, queries with these different words will give different relevance value, which will degrade the retrieval efficiency. For this reason it is very much important to find the root of the varied words having similar meaning. These root words are used everywhere instead of all the varied words to keep necessary information for calculating relevance with the query. So it is absolutely necessary to find the roots of the words from the variations of the words by doing morphological analysis for efficient information retrieval. But in English there is a little variation of words. So it is not necessary to find the root of the words by doing similar analysis.

On the other hand, there are many synonyms of words exists in Bangla. That means there are different words having different roots and the same meaning. So an efficient technique is required to manage the synonyms for efficient information retrieval. But if they are not managed in an efficient way then the system will treat all the synonyms of same meaning as distinct words. This will degrade the efficiency of the information retrieval system.

Existing Bangla text database contains both unicode and nonunicode texts. It is difficult to search uniformly the database with both the type of text. But the required information may be found in any of the type of text. So it is very much necessary to make a mechanism to handle both of the types of text uniformly.

## **1.4 Objective of the Thesis**

The objective of the thesis is to find an efficient information retrieval system with retrieval accuracy and retrieval cost. To do so the main outcome will be

- ❑ A morphological analysis will be done to find the root of the word for accurate retrieval of information.
- ❑ A synonym handling technique will be developed which will manage the vast amount synonym found in Bangla.
- ❑ We will also develop a mechanism by which we can manage the unicode and non-unicode text irrespective of their type for efficient retrieval purposes.

## **1.5 Organization of the Thesis**

The structure of this thesis can be outlined as follows:

Chapter 1 depicts some introductory description of information retrieval.

Chapter 2 gives an overview about different kinds of information retrieval systems dealing with information and different theoretical aspects.

Chapter 3 discusses about the full process and implementation of information retrieval of Bangla.

Chapter 4 describes the experimental results and evaluation of the system developed.

Chapter 5 presents the conclusions.

---

## Chapter 2

# Literature Review

---

### 2.1 Introduction

The need to store and retrieve written information became increasingly important over centuries, especially with inventions like paper and the printing press. Soon after computers were invented, people realized that they could be used for storing and mechanically retrieving large amounts of information. In 1945 Vannevar Bush published a ground breaking article titled "As We May Think" that gave birth to the idea of automatic access to large amounts of stored knowledge [1]. In the 1950s, this idea materialized into more concrete descriptions of how archives of text could be searched automatically. Several works emerged in the mid 1950s that elaborated upon the basic idea of searching text with a computer. One of the most influential methods was described by H.P. Luhn in 1957, in which he proposed using words as indexing units for documents and measuring word overlap as a criterion for retrieval [2]. Several key developments in the field happened in the 1960s. Most notable were the development of the SMART system by Gerard Salton and his students, first at Harvard University and later at Cornell University [3] and the Cranfield evaluations done by Cyril Cleverdon and his group at the College of Aeronautics in Cranfield [4]. The Cranfield tests developed an evaluation methodology for retrieval systems that is still in use by IR systems today. The SMART system, on the other hand, allowed researchers to experiment with ideas to improve search quality. A system for experimentation coupled with good evaluation methodology allowed rapid progress in the field, and paved way for many critical developments. In 1970s and 1980s many developments were built on the advances of the 1960s. Various models for document retrieval were developed and advances were made along all dimensions of the retrieval process. These new models/techniques were experimentally proven to be effective on small text collections (several thousand articles) available to researchers at the time. However, due to lack of availability of large text collections, the question whether these models and techniques

---

would scale to larger corpora remained unanswered. This changed in 1992 with the inception of Text Retrieval Conference, or TREC [5]. TREC is a series of evaluation conferences sponsored by various US Government agencies under the auspices of National Institute of Standards and Technology, which aims at encouraging research in IR from large text collections. With large text collections available under TREC, many old techniques were modified, and many new techniques were developed (and are still being developed) to do effective retrieval over large collections. TREC has also branched IR into related but important fields like retrieval of spoken information, information retrieval in non-English, information filtering, user interactions with a retrieval system, and so on. The algorithms developed in IR from 1996 to 1998 were the first ones to be employed for searching the World Wide Web. Web search, however, matured into systems that take advantage of the cross linkage available on the Web, and is not a focus of the present article. Some good IR resources for the evolution of modern textual IR systems are found in [6, 7, 8].

## 2.2 Different Information Retrieval Systems

Information retrieval system may be classified as

- 1) Boolean model
- 2) Probabilistic model
- 3) Vector space mode

### 2.2.1 Boolean Model

The *Boolean model* represents documents by a set of index terms, each of which is viewed as a Boolean variable and valued as True if it is present in a document. No term weighting is allowed. Queries are specified as arbitrary Boolean expressions formed by linking terms through the standard logical operators: AND, OR, and NOT. Retrieval status value (RSV) is a measure of the query-document similarity. In the Boolean model, RSV equals 1 if the query expression evaluates to True; RSV is 0 otherwise. All documents whose RSV evaluates to 1 are considered relevant to the query. This model is



---

simple to implement and many commercial systems are based on it. User queries can employ arbitrarily complex expressions, but retrieval performance tends to be poor. It is not possible to rank the output since all retrieved documents have the same RSV, nor can weights be assigned to query terms. The results are often counter-intuitive. For example, if the user query specifies 10 terms linked by the logical connective AND, a document that has nine of these terms is not retrieved. User relevance feedback is often used in IR systems to improve retrieval effectiveness. Typically, a user is asked to indicate the relevance or irrelevance of a few documents placed at the top of the output. Since the output is not ranked, however, the selection of documents for relevance feedback elicitation is difficult. The *fuzzy-set model* is based on fuzzy-set theory, which allows partial membership in a set, as compared with conventional set theory, which does not. It redefines logical operators appropriately to include partial set membership, and processes user queries in a manner similar to the case of the Boolean model. IR systems based on the fuzzy-set model have proved nearly as incapable of discriminating among the retrieved output as systems based on the Boolean model. The strict Boolean and fuzzy-set models are preferable to other models in terms of computational requirements, which are low in terms of both the disk space required for storing document representations and the algorithmic complexity of indexing and computing query-document similarities.

### **2.2.2 Probabilistic Model**

This family of IR models is based on the general principle that documents in a collection should be ranked by decreasing probability of their relevance to a query. This is often called the probabilistic ranking principle (PRP) [9]. Since, true probabilities are not available to an IR system, probabilistic IR models *estimate* the probability of relevance of documents for a query. This estimation is the key part of the model, and this is where most probabilistic models differ from one another. The initial idea of probabilistic retrieval was published in 1960 [10]. Since then, many probabilistic models have been proposed, each based on a different probability estimation technique.

It is not possible to discuss the details of all these models here. However, the following description abstracts out the common basis for these models. We denote the probability of relevance for document  $D$ , by  $P(R|D)$  since this ranking criteria is monotonic under log-odds transformation, we can rank documents by  $\log\left(\frac{P(R|D)}{P(\bar{R}|D)}\right)$ , where  $P(\bar{R}|D)$  is the probability that the document is non-relevant. This, by simple bayes transformation, becomes  $\log\left(\frac{P(D|R) \cdot p(R)}{P(D|\bar{R}) \cdot p(R)}\right)$ . Assuming that the prior probability of relevance, i.e.,  $P(R)$  is independent of the document under consideration and thus is constant across documents,  $P(R)$  and  $P(\bar{R})$  are just scaling factors for the final document scores and can be removed from the above formulation (for ranking purposes). This further simplifies the above formulation to:  $\log\left(\frac{P(D|R)}{P(D|\bar{R})}\right)$ .

Based on the assumptions behind estimation of  $P(D|R)$  different probabilistic models start diverging at this point. In the simplest form of this model, we assume that terms (typically words) are mutually independent (this is often called the independence assumption), and  $P(D|R)$  is re-written as a product of individual term probabilities, i.e., probability of presence/absence of a term in relevant/non-relevant documents:

$$P(D|R) = \prod_{t_i \in Q, D} p(t_i | R) \cdot \prod_{t_j \in Q, \bar{D}} (1 - P(t_j | R))$$

which uses probability of presence of a term  $t_i$  in relevant documents for all terms that are common to the query and the document, and the probability of absence of a term  $t_j$  from relevant documents for all terms that are present in the query and absent from the document. If  $p_i$  denotes  $P(t_i | R)$ , and  $q_i$  denotes  $q(t_i | R)$ , the ranking formula

$$\log \frac{P(D|R)}{P(D|\bar{R})} \text{ reduces to: } \log \frac{\prod_{t_i \in Q, D} p_i \cdot \prod_{t_j \in Q, \bar{D}} (1 - p_j)}{\prod_{t_i \in Q, D} q_i \cdot \prod_{t_j \in Q, \bar{D}} (1 - q_j)}$$

For a given query, we can add to this a constant  $\log\left(\prod_{i \in Q} \frac{1 - q_i}{1 - p_i}\right)$  to transform the

ranking formula to use only the terms present in a document  $\log \prod_{i \in Q, D} \frac{p_i \cdot (1 - q_i)}{q_i \cdot (1 - p_j)}$  or

$$\prod_{i \in Q, D} \log \frac{p_i \cdot (1 - q_i)}{q_i \cdot (1 - p_j)}$$

Different assumptions for estimation of  $p_i$  and  $q_j$  yield different document ranking functions. E.g., in [11] Croft and Harper assume that  $p_i$  is the same for all query terms and  $\frac{p_i}{1 - p_i}$  is a constant and can be ignored for ranking purposes. They also assume that

almost all documents in a collection are non-relevant to a query (which is very close to truth given that collections are large) and estimate  $q$  by  $\frac{n_i}{N}$ , where  $N$  is the collection size and  $n_i$  is the number of documents that contain term- $i$ . This yields a scoring

function  $\sum_{i \in Q, D} \log \frac{N - n_i}{n_i}$ , which is similar to the inverse document frequency

function. Notice that if we think of  $\log \frac{p_i \cdot (1 - q_i)}{q_i \cdot (1 - p_i)}$  as the weight of term- $i$  in document  $D$ ,

this formulation becomes very similar to the similarity formulation in the vector space model with query terms assigned a unit weight.

### 2.2.3 Vector Space Model

Vector space model is the mostly used technique for information retrieval. Here documents are represented as the vector of keywords, where weights of these keywords are given nonbinary weights for getting better performance. Query expression is also represented as a vector of query terms. Finally, a similarity measuring technique is used for finding the relevance of the documents with the query. Some phases of vector space model are given in the following sections.

---

### 2.2.3.1 Building Term Vectors in Document Space

One common approach to document representation and indexing for statistical purposes is to represent each textual document as a set of terms [12]. Most commonly, the terms are words extracted automatically from the documents themselves, although they may also be phrases, n-grams or manually assigned descriptor terms. Of course, any such term-based representation sacrifices information about the order in which the terms occur in the document, syntactic information, etc. Often, if the terms are words extracted from the documents, “stop” words (i.e., “noise” words with little discriminatory power) are eliminated. We can apply this process to each document in a given collection, generating a set of terms that represents the given document. Then the union of all these sets of terms are taken to obtain the set of terms that represents the entire collection. This set of terms defines a “space” such that each distinct term represents one dimension in that space. Since each document is represented as a set of terms, this space can be viewed as a document space.

A numeric weight is assigned to each term in a given document, representing an estimate (usually but not necessarily statistical) of the usefulness of the given term as a descriptor of the given document, i.e., an estimate of its usefulness for distinguishing the given document from other documents in the same collection. It should be stressed that a given term may receive a different weight in each document in which it occurs; a term may be a better descriptor of one document than of another. A term that is not in a given document receives a weight of zero in that document. The weights assigned to the terms in a given document  $D_i$  can then be interpreted as the coordinates of  $D_i$  in the document space. In other words,  $D_i$  is represented as a point in document space. Equivalently, we can interpret  $D_i$  as a vector from the origin of document space to the point defined by  $D_i$ 's coordinates.

In document space, each document  $D_i$  is defined by the weights of the terms that represent it. Sometimes, it is desirable to define a “term space” for a given collection. In term space, each document is a dimension. Each point (or vector) in term space is a term

---

in the given collection. The coordinates of a given term are the weights assigned to the given term in each document in which it occurs. As before, a term receives a weight of zero for a document in which it does not occur.

The “document space” and “term space” can be combined perspectives by viewing the collection as represented by a document-by-term matrix. Each row of this matrix is a document (in term space). Each column of this matrix is a term (in document space). The element at row  $i$ , column  $j$ , is the weight of term  $j$  in document  $i$ .

A query may be specified by the user as a set of terms with accompanying numeric weights. Or a query may be specified in natural language. In the latter case, the query can be processed exactly like a document; indeed, the query might *be* a document, e.g., a sample of the kind of document the user wants to retrieve. A natural language query can receive the usual processing, i.e., removal of “stop” words, stemming, etc., transforming it into a set of terms with accompanying weights. Hence, the query can always be interpreted as another document in document space. Note: if the query contains terms that are not in the collection, these represent additional dimensions in document space.

An important question is how weights are assigned to terms either in documents or in queries. A variety of weighting schemes have been used. Given a large collection, manual assignment of weights is very expensive. The most successful and widely used scheme for automatic generation of weights is the “term frequency “ inverse document frequency” weighting scheme, commonly abbreviated “ $tf*idf$ ”. The “term frequency” ( $tf$ ) is the frequency of occurrence of the given term within the given document. Hence,  $tf$  is a document-specific statistic; it varies from one document to another, attempting to measure the importance of the term within a given document. By contrast, inverse document frequency ( $idf$ ) is a “global” statistic;  $idf$  characterizes a given term within an entire collection of documents. It is a measure of how widely the term is distributed over the given collection, and hence of how likely the term is to occur within any given document by chance. The  $idf$  is defined as “ $\ln\left(\frac{N}{n}\right)$ ” where  $N$  is the number of documents

---

in the collection and  $n$  is the number of documents that contain the given term. Hence, the fewer the documents containing the given term, the larger the *idf*. If every document in the collection contains the given term, the *idf* is zero. This expresses the commonsense intuition that a term that occurs in every document in a given collection is not likely to be useful for distinguishing relevant from non-relevant documents. Or, what is equivalent, a term that occurs in every document in a collection is not likely to be useful for distinguishing documents about one topic from documents about another topic. To cite a commonly-used example, in a collection of documents about computer science or software, the term "computer" is likely to occur in all or most of the documents, so it won't be very good at discriminating documents relevant to a given query from documents that are non-relevant to the given query. (But the same term might be very good at discriminating a document about computer science from documents that are not about computer science in another collection where computer science documents are rare.)

Computing the weight of a given term in a given document as  $tf*idf$  says that the best descriptors of a given document will be terms that occur a good deal in the given document and very little in other documents. Similarly, a term that occurs a moderate number of times in a moderate proportion of the documents in the given collection will also be a good descriptor. Hence, the terms that are the best document descriptors in a given collection will be terms that occur with moderate frequency in that collection. The lowest weights will be assigned to terms that occur very infrequently in *any* document (low-frequency documents), and terms that occur in most or all of the documents (high frequency documents).

### **2.2.3.2 Computation of Similarity between Documents and Query**

Once vectors have been computed for the query and for each document in the given collection, e.g., using a weighting scheme like those described above, the next step is to compute a numeric "similarity" between the query and each document. The documents

can then be ranked according to how similar they are to the query, i.e., the highest-ranking document is the document most similar to the query, etc. While it would be too much to hope that ranking by similarity in document vector space would correspond exactly with human judgment of degree of relevance to the given query, the hope is that the documents with high similarity will include a high proportion of the relevant documents, and that the documents with very low similarity will include very few relevant documents. Ranking of course, allows the human user to restrict his attention to a set of documents of manageable size, e.g., the top 20 documents, etc. There are several similarity measuring techniques to find similarity between document vector and query vector.

The usual similarity measure employed in document vector space is the "inner product" between the query vector and a given document vector. The inner product between a query vector and a document vector is computed by multiplying the query vector component (i.e., weight),  $QT_i$  for each term  $i$ , by the corresponding document vector component weight,  $DT_i$  for the same term  $i$ , and summing these products over all  $i$ . Hence the inner product is given by:

$$\sum_{i=1}^N QT_i \times DT_i$$

where  $N$  is the number of descriptor terms common to the query and the given document. If both vectors have been cosine normalized, then this inner product represents the cosine of the angle between the two vectors; hence this similarity measure is often called "cosine similarity". The maximum similarity is one, corresponding to the query and document vectors being identical (angle between them zero). The minimum similarity is zero corresponding to the two vectors having no terms in common (angle between them is 90 degrees). One problem with cosine similarity is that it tends to produce relatively low similarity values for long documents, especially when the document is long because it deals with multiple topics. Many techniques have been developed to solve the problem. Figure 2.1 shows Cosine distance of two documents for a query.

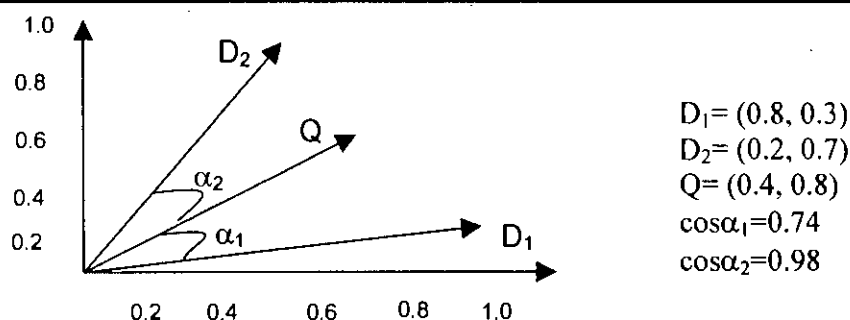


Figure 2.1: Cosine distance of two documents for a query

Apart from such distance metrics, there is a host of similarity formulas that “normalize” by avoiding term frequencies altogether, i.e., functions that only count the number of terms that match and (sometimes) the number of terms that don’t match. One such popular function is Dice’s coefficient [27]:

$$Dice = \frac{2w}{n_1 + n_2}$$

where  $w$  is the number of terms common to vectors  $D_1$  and  $D_2$ ,  $n_1$  is the number of non-zero terms in  $D_1$ , and  $n_2$  is the number of non-zero terms in  $D_2$ . Note that the denominator here performs a kind of normalization, so that a short document  $D_1$  will get a high score relative to a short topic description  $D_2$  to which it is relevant. A long document  $D_3$  relevant to  $D_2$  will get a lower Dice score provided that the additional text in  $D_3$  contains terms that are not in  $D_1$  and also not in the topic description (greater  $n_1$ , same  $w$ ). This could happen if  $D_3$  contains long sections not relevant to  $D_2$ . It could also happen if  $D_3$  contains additional discussion of the topic described by  $D_2$ , but this additional discussion uses terms that were overlooked by the user who specified topic  $D_2$ . On the other hand, if  $D_3$  and  $D_1$  contain most of the same topic-relevant terms that  $D_2$  contains, but  $D$  uses them more frequently and uses few additional terms that  $D_1$  doesn’t use, then  $D_3$  and  $D_1$  will receive similar Dice scores despite their difference in length.

Another common similarity function is Jaccard’s coefficient [27]:

$$Jaccard(D_1, D_2) = \frac{w}{N - z}$$



---

where  $w$  (as before) is the number of terms common to vectors  $D_1$  and  $D_2$ ,  $N$  is the total number of distinct terms (not term occurrences!) in the vector space (union of *all* document and topic vectors), and  $z$  is the number of distinct terms (not term occurrences!) that are neither in  $D_1$  nor in  $D_2$ . In other words,  $N-z$  is the total number of distinct terms that occur in  $D_1$  or  $D_2$  or both. Note that the value of the Jaccard function is lower, the more distinct terms are either in  $D_1$  but not  $D_2$  or vice versa. It doesn't matter whether the mismatch is caused by non-relevance, or difference in document length. On the other hand, it doesn't matter how frequently a mismatching term (or a matching term) occurs in either  $D_1$  or  $D_2$ .

### 2.2.3.3 Latent Semantic Indexing

Documents are represented as a  $T$ -Dimensional vector of term weights where  $T$  means number of distinct terms present in the document set. A criticism of the term-based approach is that user may pose queries using different terminology than the terms used to index a document. For example, from a term similarity viewpoint, the term data mining has nothing directly in common with the term knowledge discovery. However, semantically these two terms have much in common and if we posed a query with one of these terms, we would consider documents containing the other to be relevant.

An interesting and useful alternative methodology goes by the name of Latent semantic indexing (LSI). The name suggests that hidden semantic structure is extracted from text rather than just term occurrences. What LSI actually does is to approximate the original  $T$ -Dimensional term space by the first  $k$  principal component directions in this space, using  $N \times T$  document-term matrix to estimate directions. The first  $k$  principal component directions provide the best set of  $k$  orthogonal basis vectors in terms of explaining the most variance in the data matrix. The principal components approach will exploit redundancy in the terms, if it exists. There are such redundancies very often in practice. For example, terms such as database, SQL, indexing, query optimization can be expected to exhibit redundancy in the sense that many database-related documents may contain all four of these terms together. The intuition behind principal components is

that a single vector consisting of a weighted combination of the original terms may be able to approximate quite closely the effect of a much larger set of terms. Thus the original document-term matrix of size  $N \times T$  can be replaced by matrix of size  $N \times K$  where  $K$  may be much smaller than  $T$  with little loss in information. From a text retrieval perspective, for fixed recall, LSI can increase precision compared to the simple vector space model.

An interesting aspect of the principal component representation for the document-term matrix is that it captures relationships among terms by creating new terms that may more closely affect the semantic content of the document. For example, if the terms database, SQL, indexing, query optimization are effectively combined into a single principal component term, we can think of this new term as defining whether the content of a document is about database concepts. Thus, for example, if the query is posed using the term SQL, but the database-related documents in the set of documents contain only the term indexing, that set of database documents will nonetheless be returned by LSI method (but would not return strictly term based approach).

We can calculate a singular value decomposition (SVD) for the document term matrix  $M$ . That is we find decomposition  $M = U \times S \times V^T$ . Here  $U$  is a  $10 \times 6$  matrix shown in Table 2.1 as for example where each row is vector of weights for a particular documents.  $S$  is a  $6 \times 6$  diagonal matrix of eigenvalues for each principal component directions, and the column of the  $6 \times 6$  matrix  $V^T$  provide a new orthogonal basis for the data, often referred to as the principal component directions. The  $U$  matrix, for example, may be stated as the document term matrix shown in Table 2.1.

The  $S$  matrix for  $M$  has diagonal elements

$$\lambda_1, \dots, \lambda_6 = \{77.4, 69.5, 22.9, 13.5, 12.1, 4.8\}$$

In agreement with our intuition, most of the variance in the data is captured by the first two principal components. In fact, if we were to retain only these two principal components (as two surrogate terms instead of the six original terms), the fraction of

variance that our two-dimensional representation retains  $(\lambda_1^2 + \lambda_2^2) / \sum_{i=1}^6 \lambda_i^2 = 0.925$ ; i.e., only 7.5% of the information has been lost (in mean square sense).

Table 2.1: Document term matrix

	T1	T2	T3	T4	T5	T6
D1	24	21	9	0	0	3
D2	32	10	5	0	3	0
D3	12	16	5	0	0	0
D4	6	7	2	0	0	0
D5	43	31	20	0	3	0
D6	2	0	0	18	7	16
D7	0	0	1	32	12	0
D8	3	0	0	22	4	2
D9	1	0	0	34	27	25
D10	6	0	0	17	4	23

If we represent the documents in the new two dimensional principal component spaces, the co-efficient for each document correspond to the first two columns of the U matrix. The matrix is shown in Table 2.2.

Table 2.2: Document term matrix for two principal components

D1	30.8998	-11.4912
D2	30.3131	-10.7801
D3	18.0007	-7.7138
D4	8.3765	-3.5611
D5	52.7057	-20.6051
D6	14.2118	21.8263
D7	10.8052	21.9141
D8	11.5080	28.0101
D9	9.5259	17.7666
D10	19.9219	45.0751

Latent semantic indexing is done in query expression also and the space for the principle component direction is also found for the query expression. Then the similarity is calculated with the new space of document vector and query vector.

From a computational viewpoint, directly computing the principal component vectors (by seeking the eigenvectors of the correlation or covariance matrix, for example) is usually neither computationally feasible or numerically stable.

Many other techniques have been developed over the years and have met with varying success. **Cluster hypothesis**[28] states that documents that cluster together (are very similar to each other) will have a similar relevance profile for a given query. Document clustering techniques were (and still are) an active area of research. Even though the usefulness of document clustering for improved search effectiveness (or efficiency) has been very limited, document clustering has allowed several developments in IR, e.g., for browsing and search interfaces. **Natural Language Processing (NLP)**[29] has also been proposed as a tool to enhance retrieval effectiveness, but has had very limited success. Even though document ranking is a critical application for IR, it is definitely not the only one. The field has developed techniques to attack many different problems like information filtering, topic detection and tracking (TDT), speech retrieval, cross-language retrieval, question answering, and many more.

#### **2.2.3.4 IR System in Bangla using Vector Space Model**

An information retrieval system in Bangla using vector space model has been developed by **Islam et al. [26]**. They have developed a corpus based information retrieval and summarizer for Bangla. Term frequency and document frequency is used to give term weighting of the keywords and cosine distance formula is used to find the relevance of the documents with respect to a query. No morphological analysis is done to find the root of the terms. They showed a sentence ranking process to find a summary of the relevant documents. They did not show any experimental result of their experiment.

---

## 2.3 Comparison of Different IR Systems

Boolean model is very simple and it has a very clear semantic and neat formalism. But the problem is that it retrieves too many or too few. In these methods all the retrieved documents will have same ranking. That is all the retrieved documents are equally relevant. There is no way of more or less similarity. In probabilistic model theoretical adequacy can be achieved because here by assigning manual probability the ranking of the retrieved documents can be changed. But the problem is that the system requires guessing initial ranking. Another problem is that it gives binary weight to the terms, which degrades retrieval accuracy. But the vector space model has many advantages. It improves quality by giving term weighting, it also allows approximate matching by giving partial matching and it is very simple and fast. Most benefit is that it gives more or less relevant documents by using similarity measurement function. It has some problem also. It does not consider term dependency. In spite of this vector space model is used in my system for the facilities mentioned earlier.

## 2.4 Indexing of Documents

No large database can be searched without indexes. There may be primary and secondary indexes. Elaborated data structures are also needed to hold the index to support rapid queries. An index of an information retrieval system allows finding the documents matching a particular query without having to look at the documents themselves. This speeds up the searching considerably (by several orders of magnitude).

Traditionally, IR systems use specialized data structures such as tries. However, most of these structures are character-based, while the word forms of natural languages are composed of allomorphs. Character-based structures may not be really appropriate to store and search natural-language texts.

One of the problems of the character-based method is that they go to great lengths to make it possible to do rapid searches for patterns such as "c.t" where "." is a wild card representing any single character. In English, this would return the items, "cat", "cot"

and “cut” which do not form any natural set of interest. Wild cards fail in this example by returning too much and in other cases by returning too little (no simple variant of “mouse” returns “mice” nor does “is” return “are”).

A common index type is *inverted files*. In this system each document in the collection is assigned a list of attributes, which are supposed to represent the document. The most common type of attributes is keywords. The inverted file is then the sorted list of keywords of all documents, where each keyword has links to the documents that contain that keyword. An effective index structure is important for efficient processing of queries in an information retrieval system. Rather than update existing inverted lists when adding new documents, many IR systems simply rebuild the inverted file by adding the new documents to the existing collection and indexing the entire collection from scratch. This technique is expensive in terms of time and disk space, resulting in update costs proportional to the size of the total collection after the addition. On the other hand inverted indexing is not satisfactory at handling synonymy and polysemy. Inverted indexing is used mostly for rapid reindexing for addition, deletion and update. Here size of the index is an important issue for indexing.

Inverted File Index consists of

- term, keywords of interest
- lexicon, list of all terms occurring in the text
- index [term] = document1, document2, . . .

### **An Example of inverted index**

Document	Text
1	Pease porridge hot, pease porridge cold
2	Pease porridge in the pot
3	Nine days old
4	Some like it hot, some like it cold
5	Sole like it in the pot
6	Nine days old

Lexicon and the inverted index of the above text are given in Table 2.3 and figure of inverted indexing is given in Figure 2.2.

Table 2.3: Example (inverted file)

Number	Term	Documents
1	cold	1, 4
2	days	3, 6
3	hot	1, 4
4	in	2, 5
5	it	4, 5
6	like	4, 5
7	nine	3, 6
9	old	3, 6
10	pease	1, 2
11	porridge	1, 2
12	pot	2, 5
13	some	4, 5
14	the	2, 5

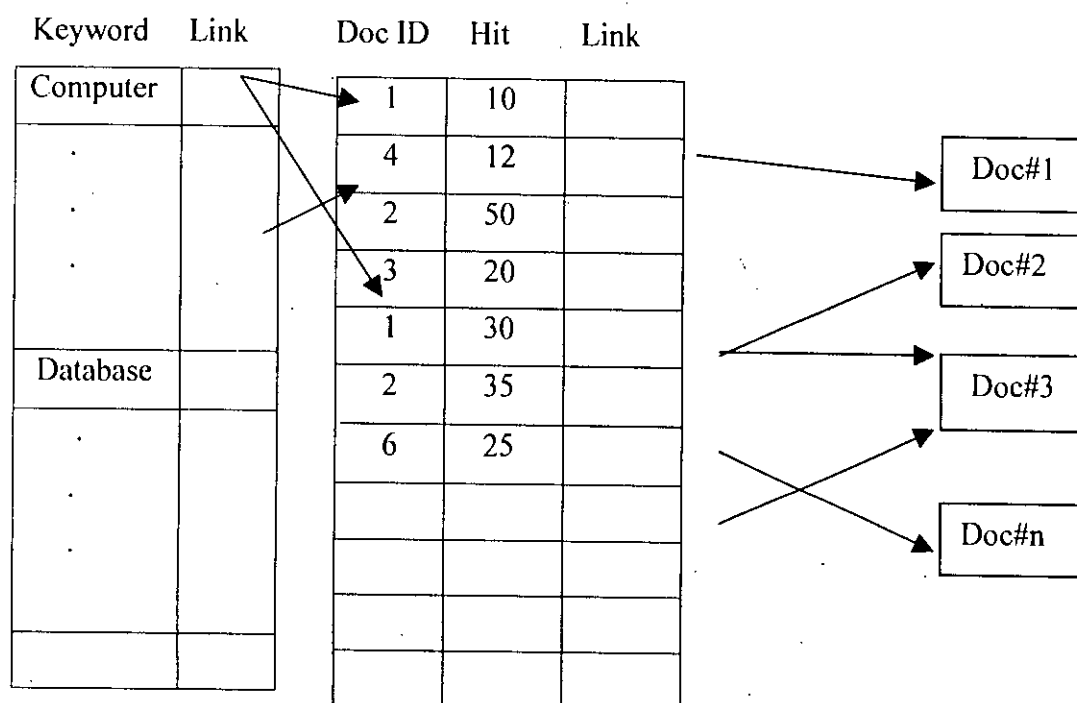


Figure 2.2: Inverted indexing file

A signature file is a file that stores a signature record for each document in the database. Each signature has a fixed size of  $b$  bits representing terms. A simple encoding scheme goes as follows. Each bit of a document signature is initialized to 0. A bit is set to 1 if the term it represents appears in the document. A signature  $S_i$  matches another signature

$S_2$  if each bit that is set in signature  $S_2$  is also set in  $S_1$ . Since there are usually more terms than available bits, there may be multiple terms mapped into the same bit. Such multiple-to-one mapping makes the search expensive since a document that matches the signature of a query does not necessarily contain the set of keywords of the query.

Relational database management systems (RDBMS) have further advantages besides using proven technology with a solid theoretical background. They offer flexible structures, so that attributes can be easily added and removed, and they allow the easy addition, deletion, and updating of entries, something which is very difficult with other approaches, where the addition of new documents to a collection usually imply a reindexing of the whole collection. It gives a nice management of synonyms found in Bengali and also gives the accuracy of searching.

## **2.5 Information Retrieval on the Web**

The World Wide Web contains a vast amount of information on every imaginable subject. The vastness and exponential growth of the Web make the necessity to search a precompiled index built and updated periodically for finding necessary information. The index is a searchable archive that gives reference pointers to Web documents. In order to search the Web for information, or to simply gather documents to build an index database, one needs to run a program that retrieves documents from the Web in much the same way as a surfing user retrieves documents by means of the browser. These special programs are called *robots* or *spiders*.

Generating a comprehensive index requires systematic traversal of the Web to locate all documents. The Web's structure is similar to that of a directed graph, so it can be traversed using graph-traversal algorithms. Because Web servers and clients use the client-server paradigm to communicate, it is possible for a robot executing on a single computer to traverse the entire Web.



There are currently three traversal methods:

- Providing the robot a “seed URL” to initiate exploration. The robot indexes the seed document, extracts URLs pointing to other documents, then examines each of these URLs recursively in a breadth-first or depth-first fashion.
- Starting with a set of URLs determined on the basis of a Web site’s popularity and searching recursively. Intuitively, we can expect a popular site’s home page to contain URLs that point to the most frequently sought information on the local and other Web servers.
- Partitioning the Web space based on Internet names or country codes and assigning one or more robots to explore the space exhaustively. This method is more widely used than the first two.

When user submits a query, the query expression is checked in the index of the document database. The result of the query expression may be rank ordered in a list of clickable URLs that includes a short summary of the documents and a relevance score. Results may also include the title, a short abstract, size, and date of the last modification for each retrieved document.

In Internet, query written in different languages should retrieve the relevant documents of different language. Automatic training of the IR system in multilingual environment can be performed using machine translation facility.

## **2.6 Relevance Feedback**

Information retrieval is an unsupervised process. The performance of IR can be improved by some indication of relevant and irrelevant items to use in ranking the documents. Relevance feedback accomplishes this by adding extra iterations to the retrieval process. IR system retrieves documents when a given query matches. Then retrieved documents are marked as relevant or not. These newly marked documents are used to achieve better performance. IR system uses these relevant documents as new

queries into the database. Other documents that are relevant to these good documents are then returned. According to Rocchio algorithm [12] new query  $q_r$  is constructed:

$$q_r = \frac{1}{|R|} \sum_{D_i \in R} D_i - \frac{1}{|S|} \sum_{D_i \in S} D_i$$

where  $R$  and  $S$  are known relevant and irrelevant documents respectively and  $D_i$  is the document term representation. The query  $q_r$  points towards the components that separate the relevant documents from the non-relevant documents. In practice, the negative components are removed from  $q_r$ . Further performance is improved by re-centering the new vector around the original,  $q_0$ :

$$q_r = \alpha q_r + \beta q_0$$

## 2.7 Evaluation of IR Performance

The evaluation of information retrieval systems has been a concern of IR research from the start. Question is why information retrieval systems have to be evaluated, what should be evaluated, and how it can be done.

There are basically two types of evaluations: tests for *efficiency*, and tests for *effectiveness*. The effectiveness of an information retrieval system depends on its ability to provide its users with the information they need, while the efficiency is determined by the time and resources needed to perform a specified task.

In most areas of computer science, it is possible to consider only efficiency, because the task of the system is exactly specified, and the complete and correct solution is an absolute precondition.

In this respect the evaluation of information retrieval systems poses challenges similar to the evaluation of natural language processing systems, like morphologic analyzers.

The quality of an information retrieval system obviously depends on both its efficiency and its effectiveness, and both have to be evaluated. However, the retrieval effectiveness is often of greater importance than the efficiency of the system.

---

The next question is then, what to evaluate?

Six main measurable quantities have been listed here:

1. The *coverage* of the collection, that is, the extent to which the system includes relevant matter;
2. the *time lag*, that is, the average interval between the time the search request is made and the time an answer is given;
3. the *form* of presentation of the output;
4. the *effort* involved on the part of the user in obtaining answers to his search requests;
5. the *recall* of the system, that is, the proportion of relevant material actually retrieved in answer to a search request;
6. the *precision* of the system, that is, the proportion of retrieved material that is actually relevant.

The first four of the previous criteria are relatively easy to assess. Recall and precision, however, pose a problem because they rely on the notion of *relevance*.

In information retrieval, the notion of *relevance* is usually interpreted as a logical property between two texts, the query and a document. A document is considered relevant if contains material, which is appropriate to the requirements stated in the query. This could be described as an “objective view” of relevance. A subjective view of relevance would not only have to consider the content of a document, but it would also have to take into account the knowledge of the user, and what documents they already know about at the time of the query. Although, this interpretation of relevance is probably closer to its meaning in everyday language, this is not (yet) possible, since the system cannot know what a user already knows about a topic.

### 2.7.1 Precision and Recall

Let  $E$  be a set of documents, the *collection*, and let  $A, B \in E$ , where  $A$  is the set of relevant documents with respect to some query, and  $B$  being the set of actually retrieved documents in response to that query.

Table 2.4: Relation between relevant and non-relevant document

	Relevant	Non Relevant	
Retrieved	$A \cap B$	$A' \cap B$	B
Not Retrieved	$A \cap B'$	$A' \cap B'$	B'
	A	A'	

Many effectiveness measures can be derived from this table, the most important being recall and precision. The *precision*  $P$  of a retrieval system for some query can be defined as:

$$P = \frac{|A \cap B|}{|B|}$$

That is, precision is the proportion of retrieved documents that are relevant. The *recall*  $R$  of a retrieval system for some query can then be defined as:

$$R = \frac{|A \cap B|}{|A|}$$

That is, recall is the proportion of relevant documents that are retrieved. Precision and recall is the standard technique for measuring effectiveness of an IR system. Precision and Recall are inversely related. Recall and precision are related in such a way that the higher the recall, the lower the precision, and vice versa. One IR system is considered to have performed better than another when its precision-recall curve is above and to the right of the other.

---

## Chapter 3

# Information Retrieval System for Bangla Text Database

---

This chapter describes the details of the information retrieval system developed for Bangla text database. There are mainly two sections in this chapter. First section is about the system architecture of the information retrieval system developed and the second section is about the analytical representation of the information retrieval system. In first section, different modules of the architecture and relationship among them are described briefly. Among the main modules of the architecture like database initialization and processing, storage and query execution, this section gives elaborated description of the storage module. Second section of the chapter gives the details of the developed system with different algorithms, flowcharts and tables required for information retrieval.

### 3.1 System Architecture

The system architecture for the retrieval of relevant documents from Bangla text database comprises three main modules: database initialization and processing module, storage module and query execution module. The database initialization and processing module sets up the database from an initial set of documents for information retrieval. The storage module manages the storage of the text database and related information. Query execution module performs all information retrieval queries with different options. The architecture is shown in the Figure 3.1. The single directional arrows represent the direction of next sub-module to be executed in a module and the double directional arrows represent the relationship of a sub-module with sub-modules from which the sub-module gets help for processing. The modules are described in the following sub-sections.

### 3.1.1 Database Initialization and Processing

Database initialization and processing module consists of the sub-modules: document database creation, new document addition, creation of stoplist, keyword processor, font handler and morphological analyzer, creation of synonymlist and addition of new document. The relationships among the sub-modules are shown in Figure 3.1.

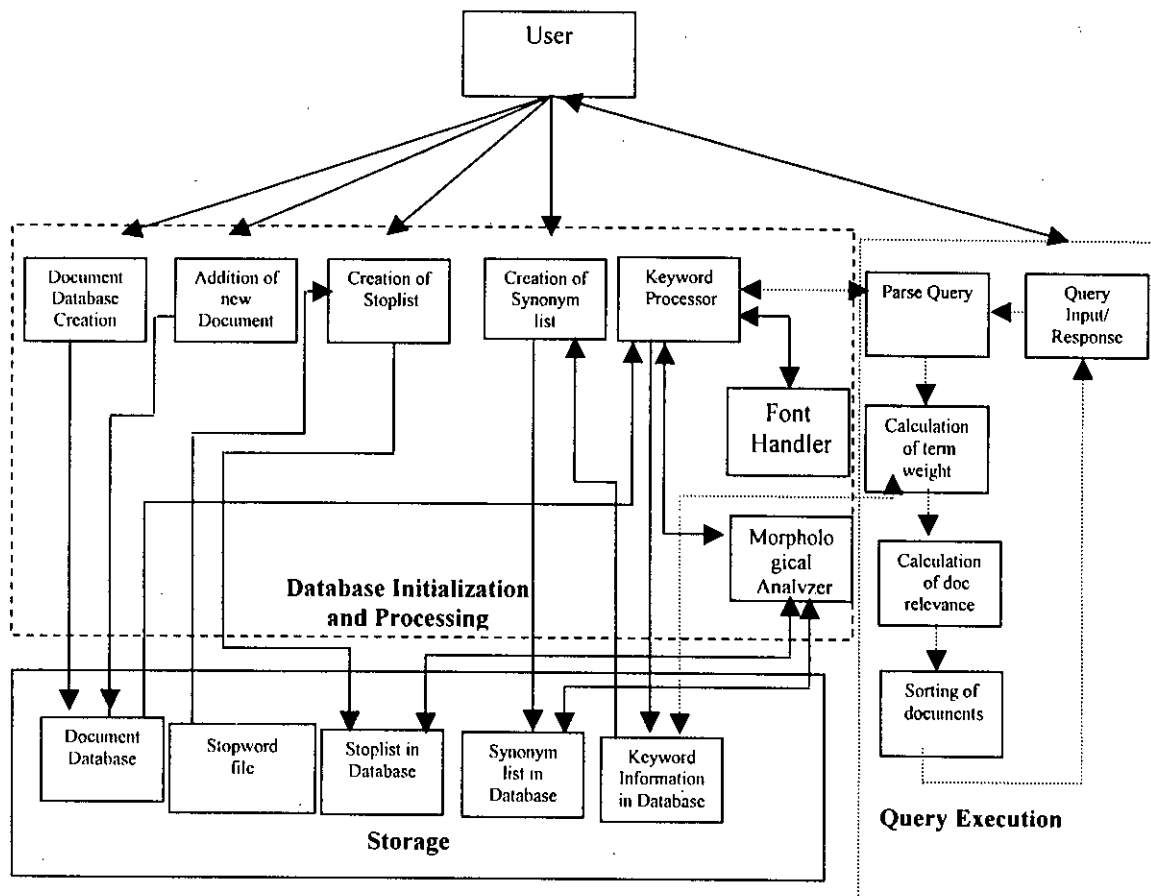


Figure 3.1: System architecture for Bangla text retrieval

#### 3.1.1.1 Document Database Creation

This module stores initial document set to be retrieved, from file system to database system. The document database is created with the information document ID, document title and size of the document for each of the documents. Each of the documents can be stored as a document object in the database as BLOB (Binary Large Object) but query

---

processing and management will be difficult as BLOBs are part of the database. BFILE data type has been used to store the document database in a file system.

### **3.1.1.2 Creation of Stoplist**

After document database creation, next submodule is creation of stoplist. This module stores some most frequently used terms of Bangla like কে, সেখানে, এর, এই etc. in the database. These words are auxiliary terms of Bangla and used most frequently. These are called stopwords. The list of stopwords for the IR system is called stoplist.

There are two types of stopwords in our system. Some are user defined and some are database specific. The above-mentioned auxiliary terms form the user-defined stopwords. Scanning the document set and having the frequency count of words found some additional stopwords. Among these words, some words had excessively high inverted document frequency count and considered to be stopword. All these stopwords are kept in a file and stoplist creation module stores the stopwords in the database tables. It requires faster checking to verify whether any word is stopword or not. For this reason stopwords are stored in database table for faster access. When a new stopword is required to be added, it is just appended in the table.

### **3.1.1.3 Keyword Processor**

Keyword processor module converts all non-unicode Bangla text documents into unicode supported with the help of font handler. Keyword processor also extracts the root of every word using morphological analyzer. The root words are checked against stoplist. If the root word is not found in the stoplist, the keyword processor stores all related information of the root word in the database.

### **3.1.1.4 Font Handler**

The terms that are extracted from the document set may consist of unicode or non-unicode supported font. The font handler performs necessary transformation to convert any non-unicode text to unicode text. If the integer equivalents of the characters of the terms are in the range of unicode value then they are considered as unicode supported.

Otherwise, it is considered as non-unicode supported. In such case, this module performs a character mapping to make conversion of non-unicode supported Bangla fonts into unicode supported. As for example the word কলম consists of three characters. The ascii equivalent of ক, ল, ম are 75, 106 and 103, respectively, and the unicode value are 2453, 2482 and 2478, respectively. This module makes the substitution of the ascii values of the characters by unicode values and gets the corresponding unicode equivalent of the string. All the characters of non-unicode fonts including “sanjukta akhar” (Composite Letter) of Bangla are transformed into unicode equivalent in this module.

### **3.1.1.5 Morphological Analyzer**

After the font conversion, the terms of the document files are stemmed to find the root of the terms, which may be referred as keyword. For this reason a morphological analysis was done to stem গুলি (GOLI), গুলা (GOLA), টি (TEE), টা (TAA), দেব (DER) etc. from the end of the terms. To stem these postfixes from the terms of the document set, the terms were checked against a postfix list. The morphological analyzer takes a word as input and gives the root of the word as an output.

### **3.1.1.6 Creation of Synonymlist**

This module stores the synonym of words as the structure shown in Table 3.1. The synonyms are stored in the database scanning the words of document database excluding stop words.

### **3.1.1.7 New Document Addition**

After scanning all the initial documents, when new document is required to be added this module adds the document in the document database and performs similar tasks for each keyword in that document as described in previous sections.

## **3.1.2 Storage**

This module handles the storage of the document database and different necessary information for retrieval purposes. The stoplist is stored accordingly as described in



stoplist creation sub-module. The most critical part of the storage module is the storage of synonyms of words and the term information.

### 3.1.2.1 Storage of Synonymlist

A word having the same meaning as another in the same language is called synonym. Sometimes closely associated words are also treated as synonyms. In any information retrieval system it is very much important to manage the synonyms efficiently. Because if the synonyms having same meaning are not treated as same words rather different words, then there will be a great degradation of retrieval accuracy. Here a very little relevant information will be retrieved for any query. But if synonyms are considered, any query with any word will retrieve all the necessary information regarding the synonyms of that word also. Searching uniformly over a text database considering all the synonyms is a difficult problem. A special synonym handling mechanism is used to store the synonyms.

Table 3.1: Synonyms

Synonym_ID	Sequence	Word
101	1	বই
101	2	পুস্তক
101	3	অভিধান
102	1	কলম
102	2	লেখনী

The document database is scanned twice to create the synonymlist. After the first scan of the database initialization without the synonym effect, the occur table of the database as shown in Figure 3.2 contains different words of the document database. This table also contains words with their synonyms also. So this table is checked to find the group of words having same meaning. With the group of words a synonym table is created according to Table 3.1. In the structure the words having same meaning will have same synonym id that forms a group of synonyms. In each group of synonyms, a sequence number is maintained according to the importance. During second scan of the database

initialization the synonym table is used to keep the effect of synonyms. For storing or ranking of term during retrieval, the term having sequence number 1 is used.

### 3.1.2.2 Storage of Term Information

This module also manages the storage of the terms in such a way that the IR system can run uniformly overcoming the synonym-handling problem. Storage module stores term information and necessary related information in the database according to Figure 3.2.

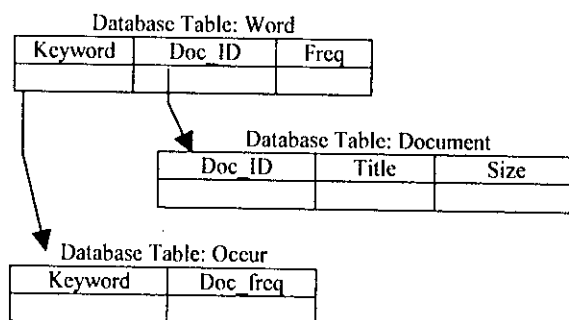


Figure 3.2: Storage of term information

In Figure 3.2 keyword and Doc\_ID of the word table represents which keyword is present in which document and Freq field shows how many times the keyword occurs in that document. The Doc\_ID, Title and Size fields of the document table represent document ID of documents, Title and number of terms present in that document, respectively. In occur table the field Keyword and Doc\_freq is used to represent in how many documents a keyword occurs.

After stemming the postfix of a term by morphological analysis, the root is stored in the word table of the database with the corresponding document ID. All the keywords of all documents are stored in the word table. After the creation of word table, the occur table is created from word table. A record in document table is inserted after completion of scanning of each document.

### 3.1.3 Query Execution

In Query Execution module the sub-modules that are related to process a query are- query input/response sub-module, parse query sub-module, calculation of term weight

sub-module, calculation of document relevance sub-module and sorting of relevant documents sub-module as shown in Figure 3.1. Query input or response sub-module takes the query expression from the user and returns relevant documents to the user for that query. "Parse query" sub-module identifies different roots from the query expression written in Bangla with the help of "keyword processor" sub-module. "Calculation of term weight" sub-module calculates the weights of every query terms for each document. "Calculation of document relevance" sub-module calculates the relevance of each document against the query expression. "Sorting of relevant documents" sub-module sorts the documents according to their relevance and then sends the relevant documents to "Query processor" sub-module.

## 3.2 Analytical Representation of the Architecture

This section gives an analytical description of the system architecture given in previous sections.

### 3.2.1 Analysis of the Initialization and Processing

In document database creation module set of documents are represented as a database of documents. The algorithm for storing the document set in database file is given in Figure 3.3.

```

Algorithm Document_DB ()
{ 1. Create a File object of the directory name of the directory where the
   document files exist.
  2. Create table named document with the field Doc_ID, Title, Size having the
   data type as number, BFILE and number respectively.
  3. Initialize counter variable as 1.
  4. While counter is not greater than directory length of the directory
     4.1 Insert the file name of directory [counter] into Title field of Document
        table.
     4.2 Increment the value of counter by 1.
}

```

Figure 3.3: Algorithm to create document database

The algorithm in Figure 3.3 stores all the documents in the database as BFILE. The first sql query creates a path from which the BFILE should keep the reference. The second sql command creates a table declaring document title as BFILE according to the syntax. When new document is required to be added, the insert statement is executed for that document file.

In stoplist creation module, the stoplist is created from a file where the stopwords are stored manually. For finding the frequent word the document set is checked incrementally to find the number of occurrence of terms in document set.

### 3.2.2 Conversion of Non-unicode Documents into Unicode

Bangla documents containing non-unicode character contain the ascii values as shown in Table 3.2. At the same time, unicode supported Bangla text documents contain the unicode value as shown in the same. The algorithm shown in Figure 3.4 converts a non-unicode string to unicode. To do this it checks all the characters of the string sequentially. If the integer equivalent value of the character is within unicode range then it requires no transformation. If its value is within ascii range then corresponding

```

Pseudocode string_convert (str) // str is non-unicode string to be converted
{set i:=0,j:=0,flag1:=0, word:= null, p:= length (str);
while(j<p)
{c:= str[j]; l:= (int)c;
if (l= integer equivalent of bangla character which have unicode value) then
i:= unicode value of that character. ;
else if (l=integer equivalent of bangla 'sanjukta' akhar ) then //do for all 'sanjukta'
{ word:= word + first unicode character of 'sanjukta'.
word:=word + 'jafalaa' in unicode.
word:= word + first unicode character of 'sanjukta'.
if (flag=1)
{ word:= word +y; y:= null; flag1:=0;
else if ((l= integer equivalent of second part of bangla 'sanjukta' akhar) and
( (int) word[length(word)-1]= integer equivalent of first part of bangla
'sanjukta' akhar)) //do for all such 'sanjukta' akhar
{ word:= word + first unicode character of 'sanjukta'.
word := word + 'jafalaa' in unicode.
word:= word + first unicode character of 'sanjukta'.
if (flag=1) { word:= word +y; y:= null; flag1:=0;
else word:= word + c;
if ((i>=2432)AND(i<=2559)) // within bangla unicode range
{ if (i=2495 or i=2503 or i=2504) //for aekar, rishikar, oikar
{y:=(char)i; flag1:=1;
if (i= 2494) // for ookar or oukar
{ if ((int) word[length(word)-1]=(2503 or 2504))
word:= word- word [length(word)-1];
word:= word + char (2507 or 2508) ; flag1:=0;
else { word:= word +(char) i;
if (flag=1)
{word:= word +y; y:= null; flag1:=0;
j:=j+1;
return (word); // end of string_convert

```

Figure 3.4: Algorithm for conversion of non-unicode supported text to unicode supported text.

Table 3.2: Bangla characters with unicode and ascii value

Character	Unicode value	Non Unicode ASCII Value
অ	2437	65
আ	2438	
ই	2439	66
ঈ	2440	67
উ	2441	68
ঊ	2442	69
ঋ	2443	70
এ	2447	71
ঐ	2448	72
ও	2451	73
ঔ	2452	74
ক	2453	75
খ	2454	76
গ	2455	77
ঘ	2456	78
ঙ	2457	79
চ	2458	80
ছ	2459	81
জ	2460	82
ঝ	2461	83
ঞ	2462	84
ট	2463	85
ঠ	2464	86
ড	2465	87
ঢ	2466	88
ণ	2467	89
ত	2468	90
থ	2469	95
দ	2470	96
ধ	2471	97
ন	2472	98
প	2474	99
ফ	2475	100
ব	2476	101
ভ	2477	102

Character	Unicode value	Non Unicode ASCII Value
ম	2478	103
য	2479	104
র	2480	105
ল	2482	106
শ	2486	107
ষ	2487	108
স	2488	109
হ	2489	110
়	2494	118
ি	2495	119
ী	2496	120
ৎ	2497	121
ূ	2498	126
ৃ	2499	8222
ৄ	2503	8225
৅	2504	8240
৆	2507	
ে	2508	
ৈ	2509	38
৉	2524	111
৊	2525	112
ো	2527	113
ৌ	2434	115
্	2443	70
ৎ	2534	48
৏	2536	49
৐	2537	50
৑	2538	51
৒	2539	52
৓	2540	53
৔	2541	54
৕	2542	55
৖	2543	56
ৗ	2544	57

non-unicode value of that character is substituted by the unicode value with a look up of Table 3.2. If any “sangukta akhar”(composite letter) comes, as in unicode no “sangukta akhar” has any specific value, so it is substituted by two unicode characters by which the “sangukta akhar” is formed. For joining of two characters in unicode, “jafala” is used as connective of two characters. In this way the algorithm works for all the characters of Bangla.

The algorithm for conversion of a non-unicode document into a unicode document is given in Figure 3.5. The algorithm scans each character of the document file and appends it with a null string until a new line or space is found. During scanning the characters is checked, if the integer equivalent value of the character is within the unicode range or ascii range. If any character is within ascii range and the new line or space is encountered the string is converted by string converting algorithms and then processed. If all the characters of the string are within unicode range then the string is processed directly.

```
Pseudocode Font_handling (document_name)
{
  set flag:= 0;
  read (document_name);
  For j =1 to character number of document_name do{
    c:= read(char[j]);
    p:=(int) c;
    if ((value of p is within unicode range of Bangla characters) OR (p=13 OR p=32)) {
      if ((flag=0)AND(p=13 OR p=32)) {
        process the word;
        word := null;
        flag:= 0;
      }
      if ((flag=1)AND (p=13 OR p=32)) {
        convert word using string_convert algorithm
        process the word;
        word := null;
        flag:= 0;
      }
      else word := word +c;
    }
    else if (value of p is within ascii range of Bangla characters){
      word := word + c;
      flag:=1;
    }
  }
}
```

Figure 3.5: Algorithm for conversion of non-unicode document into unicode

### 3.2.3 Finding the Root of the Terms in a Document

Bangla words can have many difficult variations by adding the postfixes with the root of the word. These variations have significant impact on the information retrieval of Bangla

text database. Finding the root of a word is called stemming. To stem the postfixes from the terms of the document set, the morphological analyzer checks the terms against a postfix list according Table 3.3. This list is created from Bangla grammar books.

Table 3.3: Postfix list

S.N	Postfix	S.N	Postfix	S.N	Postfix	S.N	Postfix
1	টি	38	নিচয়	75	ছিলি	112	ইলেন
2	টির	39	সমূহ	76	ছিলে	113	ইলাম
3	টা	40	গর্ব	77	ছিলেন	114	ইলেম
4	টার	41	বর্গ	78	ছিলাম	115	ইলুম
5	টুকু	42	আবলী	79	ছিলুম	116	ইত
6	টুকুর	43	সমুদয়	80	ছিলেম	117	ইতে
7	খানা	44	গুচ্ছ	81	ল	118	ইতাম
8	খানার	45	গ্রাম	82	লে	119	ইতেন
9	খানি	46	বৃন্দ	83	লি	120	ইতিস
10	খানির	47	পুঞ্জ	84	লাম	121	ইতেছ
11	গাছা	48	মণ্ডলী	85	লুম	122	ইতেছে
12	গাছি	49	শ্রেণী	86	লেম	123	ইতেছি
13	গুলো	50	র	87	লেন	124	ইতেছেন
14	গুলা	51	রা	88	এ	125	ইতেছিস
15	গুলি	52	রে	89	এন	126	ইতেছিল
16	গুলোর	53	এর	90	এছ	127	ইতেছিলাম
17	গুলার	54	এরা	91	এছে	128	ইতেছিলে
18	গুলির	55	ব	92	এছি	129	ইতেছিলি
19	দের	56	বা	93	এছেন	130	ইতেছিলেন
20	দিগের	57	বে	94	এছিস	131	ইতেছিলেম
21	গণ	58	বেন	95	এছিল	132	ইতেছিলুম
22	জন	59	বি	96	এছিলে	133	ইয়াছ
23	দল	60	ত	97	এছিলি	134	ইয়াছে
24	চয়	61	তে	98	এছিলেন	135	ইয়াছি
25	সব	62	তেম	99	এছিলেম	136	ইয়াছেন
26	মহল	63	তেন	100	এছিলুম	137	ইয়াছিস
27	পটল	64	তাম	101	এছিলাম	138	ইয়াছিলে
28	জাল	65	তুম	102	ই	139	ইয়াছিলি
29	দাম	66	তিস	103	ইও	140	ইয়াছিল
30	পাল	67	উন	104	ইস	141	ইয়াছিলেন
31	সকল	68	উক	105	ইবে	142	ইয়াছিলেম
32	কুশ	69	ছ	106	ইব	143	ইয়াছিলাম
33	যুথ	70	ছে	107	ইবি		
34	মালা	71	ছি	108	ইবেন		
35	রাজি	72	ছেন	109	ইল		
36	রাশি	73	ছিস	110	ইলে		
37	নিকর	74	ছিল	111	ইলি		

The algorithm to find a root of the term is given in Figure 3.6.

```

Word_Stem (str) {
  barna=0, i=0,p=0, f=0;
  i=str.length () -1;
  While (true) {
    if (barna== 0)
      barna = (int) str.charAt (i);
    switch (barna) { // barna is ascii value of each
      case Ascii_barna1: { // char of all possible postfix
        if (i-1>=0)
          barna=(int) str.charAt (i-1);
          if (barna==100){i--; break;}
          else if (barna==dantana){i--;break;}
          .....
          .....
        else if ((i+2<=str.length ()-1)&&(str.charAt (i+1)
          ==ackar )&& (str.charAt (i+2)==dantana))
          {str = str.substring (0,i);barna=1;break;}
        else if ((i+1<=str.length ()-1)&&(str.charAt (i+1)
          ==dantasha))
          {str =str.substring (0,i); barna=1;break;}
          barna=1;
          break;
          default: barna=2;
        if ((barna==1)||(barna==2))
          break;
      }
    }
    return (str); // end of Word_Stem
  }
}

```

Figure 3.6: Algorithm for finding the root of a word.

The algorithm takes a word as input and gives the root of the words as output. To find a matching of postfix, each character of terms from last is checked with the last character of any postfix. If it is matched then next matching is checked until a complete matching is found. If a complete matching is found, the matched part is discarded from the term and the root is extracted. If no matching is found then the scanning is discarded and next term is considered for checking. A simple case-switch statement is used to perform this operation. Each case option contains some decisions for an alphabet. The decisions may be the next possible alphabets of different postfixes to which next matching occurs, whether the current alphabet is the end of any postfix matching etc. The analysis that requires selecting the right postfixes for the right word is called morphological analysis.

The character sequence of postfixes is given in Figure 3.7.





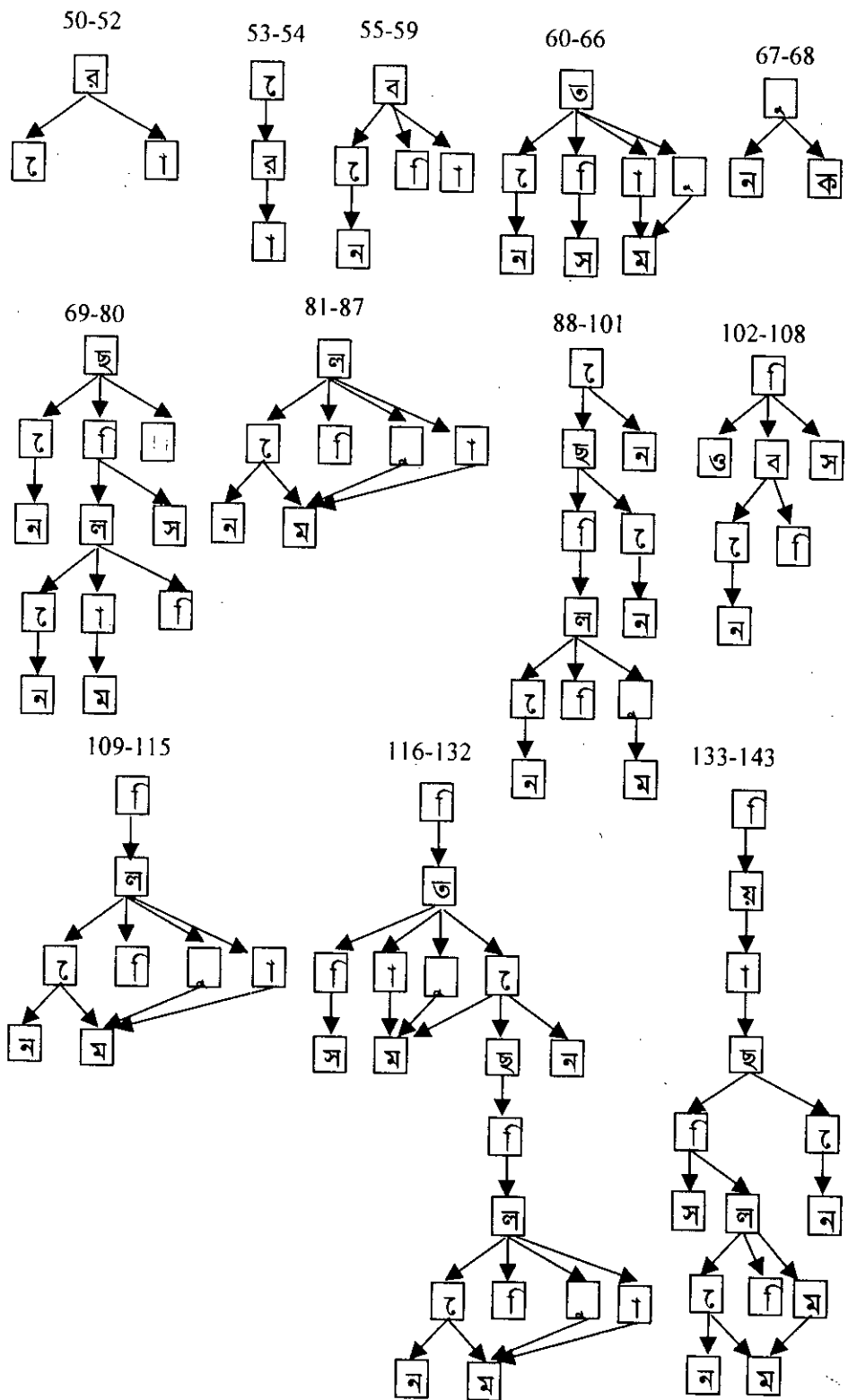


Figure 3.7: Character sequence of postfixes (cont.)

In Figure 3.7 we find the character sequence of different postfixes in which they occurs. Table 3.3 contains the postfixes of Bangla with a serial number. Figure 3.7 also shows the character sequence of a postfix with a corresponding serial number. As during morphological analysis all the terms will be in unicode supported, so it will follow the phonetic sequence of characters for their appearance in the terms. Figure 3.7 shows the phonetic sequences of the postfixes. To find any matching it is required to scan the terms according to any possible sequence of postfixes given in Figure 3.7. For example the postfix of serial number 15 is গুলি. Normally, in non-unicode format the sequence of appearance of the characters of গুলি is গ+ু+লি, but the sequence that is followed in unicode format is গ+ু+ল+ি, as given in Figure 3.7.

### 3.2.4 Setup of the IR System

The flowchart for the setup of the information retrieval system for Bangla text database is given in Figure 3.8. and 3.9. Figure 3.8 describes the first scan of the document database to make a synonym table for the information retrieval system.

First scan starts with the creation of a document database. After this, a stoplist is created. Then all the documents are scanned using a counter *i*. Each word file is scanned until end of the file. A term is extracted from that file and is converted to unicode-supported text, if necessary. Then the term is checked whether it is in stoplist or not. If it is in stoplist then the term is skipped and next term is considered. If it is not in the stoplist then its root is found by morphological analysis. After finding the root of the term, the root is checked whether it is present in the word table for that document or not. If it is not present then the root word is stored in word table with other information. If it is present in the word table, the value of "Freq" is incremented by 1 for that term of the document. If the end of file occurs after scanning the whole document, the document table is updated with the "Size" value. When all the documents are scanned, the occur table is created with the necessary information.

After the first scan, the occur table of the database contains different words of the document database. But this table contains words with their synonyms. So this table is checked to find the groups of words having same meaning. With the group of

words a synonym table is created according to Table 3.1. In the structure all words having same meaning will have same synonym id that forms a group of synonyms. In each group of synonyms, a sequence number is maintained according to the importance. For ranking of documents, the term having importance 1 is used.

The occur table is used to find the terms to add in the stoplist which occurs in many documents. These terms are added in the stopword file for adding in the stoplist.

In the second scan of the database initialization, the synonym table which has been developed from the occur table is used. Before running the second scan, all the tables of storage of term information is deleted for adding new data with the effect of synonym. In the second scan, first of all, a document database is created, then a stoplist is created from stopwords. Each term of each document is converted into unicode, if needed, then checked in stoplist, and the equivalent synonym of that term is found from the synonym table. After finding equivalent term, the term is checked whether it is present in the word table for that document or not. If it is not present then the term is stored in word table with other information. If it is present in the word table, the value of "Freq" is incremented by 1 for that term of the document. After scanning all the documents the occur table is newly created.

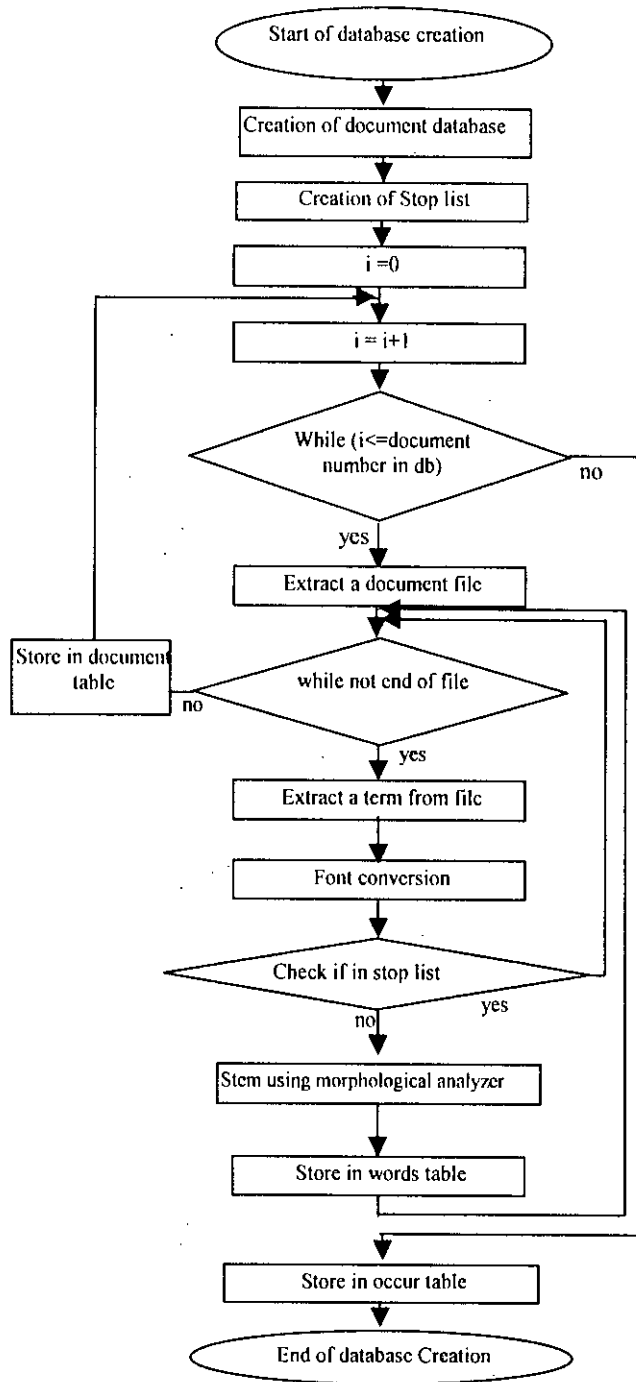


Figure 3.8: Flowchart for the set up of the database for information retrieval  
(first scan)

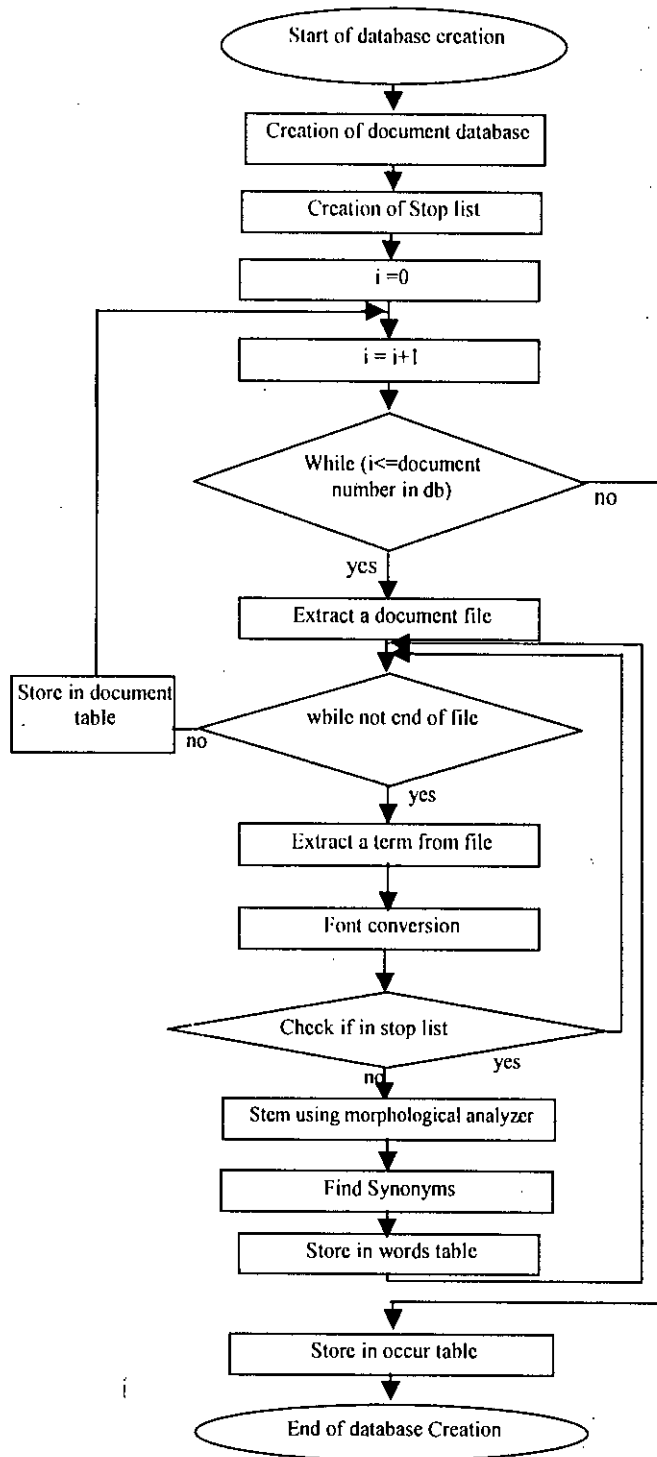


Figure 3.9: Flowchart for database initialization (second scan)

### 3.2.5 Methodology for New Document Addition into the Database

The flowchart for new document addition is given in Figure 3.11. First of all, the document to be added is selected. Then the maximum document ID is obtained from the document table. The new ID is incremented by one for the new document. Then the document's terms are scanned one by one until the end of the file is reached. For each term it is converted into unicode supported and checked whether it is in stoplist or not. If it is in the stoplist, it is just skipped. If it is not in the stoplist morphological analysis is done to find the root of the word.

The root word is checked in the synonym table. If it is present, then the equivalent term is checked in the word table for that document whether it is present or not. If it is not present then one record will be inserted into the word table with values equivalent synonym as keyword, the new document id as Doc\_ID and frequency as 1. If it is present with the new document id then the Freq value of the term for that term will be incremented by 1. If the root word is not found in the synonym table, it is inserted into a temporary table with the same structure as the word table. In this way the scanning of the new document is completed. If no term is found in the temporary table after the complete scan of the file the occur table is processed from word table according to the algorithm given in Figure 3.10.

```
Algorithm update_occur_tbl_new_document ()
{
  1. Select maximum document id from the word table.
  2. Get all the terms from the word table where document id = maximum document id.
  3. For each selected term do the following
     If the term is present in the occur table
       Increment the doc_freq value of that term by 1 in the occur table
     Else
       Insert the term into occur table with doc_freq value by 1.
}
```

Figure 3.10: Algorithm to update occur table

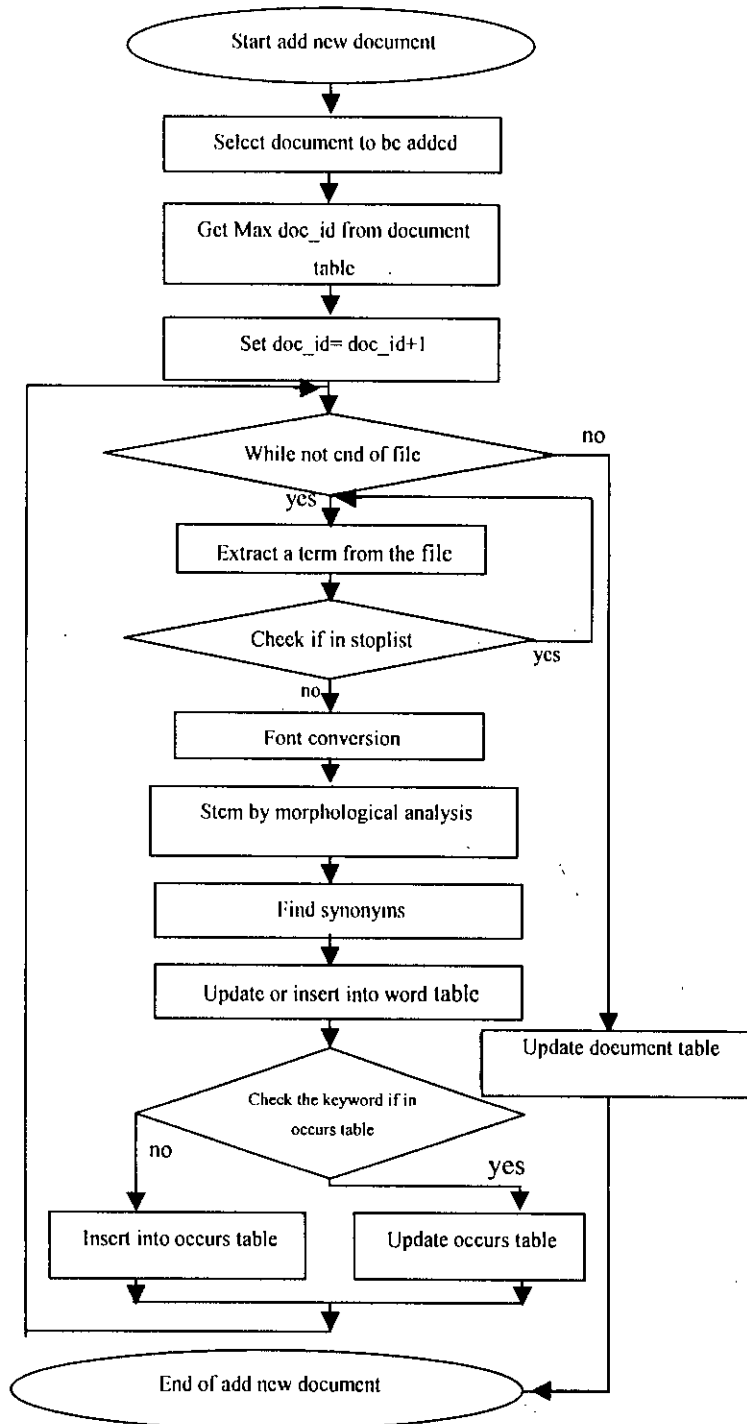


Figure 3.11: Flowchart for new document addition



If there are some records in the temporary table then it is considered that the terms occurring in that records have no entity in the synonym table. As a result no effect of synonyms is considered for those terms. So synonym table is updated for the terms in temporary word table. The algorithm for this synonym table updating is given in Figure 3.12.

```

Algorithm update_synonym_tbl_new_document ()
{
    1. Get all the terms from the temporary word table.
    2. Mark those terms as unmarked.
    3. For each unmarked term do the following
        3.1 Select maximum synonym id from the synonym table
        3.2 Increment the selected id by 1.
        3.3 Insert the unmarked term into synonym table with id 1 and sequence number 1.
        3.4 Scan all the other unmarked terms to find the synonyms of the previous term.
        3.5 If some terms are found then do the following
            For each such term insert the term into synonym table with the same id and
            sequence number as sequence number +1;
        3.6 Mark all the terms as marked having the generated synonym id
}

```

Figure 3.12: Algorithm for updating the synonym table for new document

The algorithm given in Figure 3.12 shows how the synonym is generated for the newly added document. Now it is required to add the terms from temporary word table to the word table considering the effect of synonyms. The algorithm to add the terms from temporary word table to word table is given in Figure 3.13.

```

Algorithm insert_word_tbl_new_document ()
{
    1. Select document id for the new document.
    2. Get all the terms from the temporary word table.
    3. For each selected term do the following
        3.1 Find the term's equivalent synonym from the synonym table
            having sequence number 1.
        3.2 Check this equivalent synonym in word table with same doc_id
        3.3 If it is present then increment the freq value of that term for the new document
            by 1.
        3.4 If it is not present then insert a record in the word table with the equivalent
            synonym having the new document id and freq value as 1.
}

```

Figure 3.13: Algorithm to add term information from temporary table to word table.

After inserting the term information into word table, the occur table is updated according to algorithm given in Figure 3.10. Then the document table is updated for the new document with its size. In this way all the information regarding the new document is stored in the database.

### 3.2.6 Information Retrieval Query Processing on Bangla Text Database

The steps of information retrieval query processing on Bangla text database are given in Figure 3.14. The query manager receives the user query through an interface and transfers the query to the parser. Query manager also presents the result of the query to the user. The parser translates the user query consisting of Bangla words and selects a query plan for execution. Firstly, the words are checked if they are in the stoplist. If the word is found in the stoplist, then the word is skipped and next term is considered for relevance calculation. If the word is not a stopword, it is included into the query vector,  $Q$ . For all  $q_i \in Q$ , the equivalent synonym with sequence number 1 is found. The query vector is generated using these equivalent synonyms. This is done with the help of synonyms table of the database.

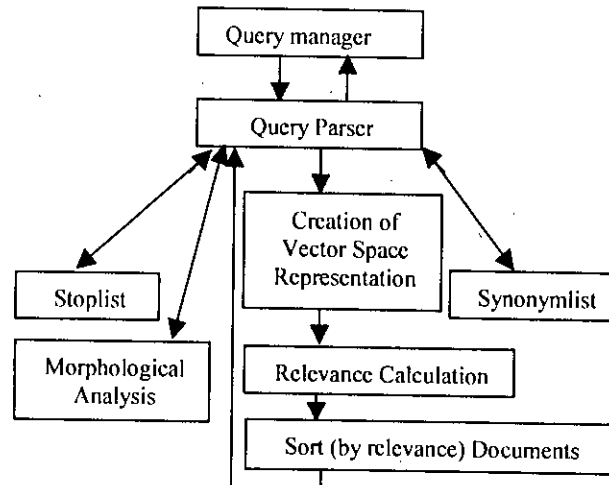


Figure 3.14: Query execution process

#### 3.2.6.1 Creation of Vector Space Representation

The document vector  $D_i$  for  $i^{\text{th}}$  document is created in the same way as the query vector  $Q$ , but the weighting factor for  $D$  is not same as the query vector. The weight of a term  $t$  for a document  $D$  is represented as

$$W(t, D_i) = T_f \times \log \left( \frac{N}{n} \right) \times \frac{\bar{X}}{DL_t}$$

where,

$T_f$  = Frequency of the term  $t$  in document  $D$ ,

$N$  = Number of documents in the database,  
 $n$  = Number of documents the term  $t$  occurs,  
 $\bar{X}$  = Mean document length of the database, and  
 $DL_i$  = Number of terms in  $i^{\text{th}}$  document.

A vector space representation of each of the documents with the weight of the keywords is found as shown in Figure 3.15. For weighting the keywords, document length is also brought into consideration. This gives much more preference to the shorter document having keyword than longer one.

	$t_1$	$t_2$	$\dots$	$t_n$		$t_1$	$t_2$	$\dots$	$t_n$
$d_1$	6	0		2		1	0		1
$d_2$	1	2		0					
$\dots$									
$d_n$	2	1		3					

Document space
Query space

Figure 3.15: Vector space representation of document and query vector

### 3.2.6.2 Finding the Relevance of Documents

Cosine distance formula is used by this sub-module to find the rank of each document with respect to the query expression. Cosine distance formula is used for relevance ranking because it emphasizes the relative contributions of individual terms resulting in good performance. The formula is given as,

$$d_c(Q, D_j) = \frac{\sum_{i=1}^n Q_i \times D_{ij}}{\sqrt{\sum_{i=1}^n (D_{ij})^2 \times \sum_{i=1}^n (Q_i)^2}}$$

Here,  $Q$  and  $D$  are the vector space representation of the query expression and documents, respectively, and  $n$  is the cardinality of the query vector. Cardinality of query vector is considered instead of the cardinality of document vector to reduce the computational cost. After getting the relevance of all the documents, this module sorts the documents in the descending order of relevance. The highest ranked documents will be at the top of the list as the most relevant documents for any particular query.

In query execution module when the relevant documents are listed they can be accessed from front end, because all the documents stored in the database are as BFILE. The algorithm is given in Figure 3.16. When the corresponding relevant document is needed to view, its corresponding Doc\_ID is selected and the word file of that address bearing the Doc\_ID is displayed.

```

Algorithm select_BFILE ()
{
    Statement stmt = Connection.createStatement ();
    ResultSet rset = stmt.executeQuery ("select Title from document where
                                        doc_ID='n'");

    Oracle.sql.BFILE bfile = null;
    bfile= ((oracleResultSet)rset).getFile (1);
    bfile.openFile ();
    InputStream instream = bfile.getBinaryStream ();
    Gui.displayFile (instream);
}

```

Figure 3.16: Algorithm to display relevant document

Here in the algorithm we find that rset variable stores the resultset for the select query statement where Doc\_ID is equal to n. Here n is the Doc\_ID of the selected document. Then the address of the document having n as Doc\_ID is assigned in BFILE variable. Then the BFILE is opened and it is stored in an input stream variable instream. This instream is then displayed by a graphical user interface. In this way any relevant document can be viewed.

### 3.2.7 Complexity Analysis

The Information retrieval system developed has two parts for the time concern, one for storing the necessary information in index structure and another one is query time for any particular query. In most of the information retrieval system indexing time is sacrificed for the fast response against a query.

#### 3.2.7.1 Creation of Index

The time complexity for the indexing may be described as follows,

Let the number of words in a file be  $N$  and number of roots in a file be  $R$ .

Then the complexity is  $O(N) + O(R \times (Q + U))$

where,  $Q$ =database query time for each root

$U$ = database insertion time for each root

If the database size becomes large then the complexity will be  $O(R \times Q)$  because other terms will be negligible in comparison with it.

### 3.2.7.2 Update Operation

For creating an inverted index the system scans all the documents with the complexity  $O(N) + O(R \times (Q + U))$ , which is proportional to the document number. So if the document number is so big then this time will be very long. RDBMS structure is used here so that we can update the index by updating one document each time rather making the re-indexing of whole index.

### 3.2.7.3 Querying the Database

The time complexity for searching may be described as follows,

Lct,  $n$  be the number of keyword in the query expression,

$D$  be the number of document in the collection.

$Q$  be the time required to get necessary information about a keyword.

So time required for querying all the keyword in a query expression is  $O(nQ)$ ,

time required for calculating relevance of  $D$  number of document is  $O(D)$ ,

and, time required for sorting the relevant document is  $O(D \log D)$ .

So total time complexity for searching is  $O(nQ) + O(D \log D)$ .

## Chapter 4

# Experimental Results and Discussions

Precision and recall is the standard technique for measuring effectiveness of any information retrieval system. In our experiment we used precision recall for performance measurement of information retrieval system in Bangla. By using precision recall curve we evaluated the performance of our system with various options. We have also shown the effect of number of terms present in query expression by using precision recall curve. The effect of morphological analysis for the query terms with the most frequently found postfixes is also shown by precision recall curve.

### 4.1 Experimental Setup

The information retrieval system has been developed on a machine having the operating system Windows 2000 Professional, 1.6 GHz Pentium IV Processor with 256 MB memory. The system was implemented in Jbuilder-8 in the front and in Oracle 9i DBMS in the back-end for storing the text database and related information.

We developed a corpus in Bangla with 63 documents as given in Table 4.1. These documents were collected from different departments of a public university, where the documents were created for different application domain. According to Table 4.1, we see that there are two types of documents in the dataset: non-unicode supported and unicode supported. We have developed the system such that information retrieval queries can be processed as the dataset irrespective of the type of the document.

Table 4.1: Document Information

S.N	Document Title	Word count	Type	S.N	Document Title	Word count	Type
1	Aad. Commitee	3816	Non-unicode	3	Alama Khatun 2	1265	Non-unicode
2	Admission form	317	Non-unicode	4	Alama khatun	558	Non-unicode

S.N	Document Title	Word count	Type
5	ALOWA	210	Non-unicode
6	ANNA_PO	320	Non-unicode
7	attendance	195	Non-unicode
8	Backup of fd	175	Non-unicode
9	bihar.doc	396	Non-unicode
10	Backup of ma	171	Non-unicode
11	bari sir l	734	Non-unicode
12	baten	61	Non-unicode
13	Bill of admission l	1483	Non-unicode
14	BOU	4541	Non-unicode
15	BOU_2	4107	Non-unicode
16	BOU_77	4111	Non-unicode
17	BOU00	4527	Non-unicode
18	BU	2937	Non-unicode
19	budgetU	1358	Non-unicode
20	Chairman l	1172	Non-unicode
21	Chairman pad	724	Non-unicode
22	CIVIX_1	2045	Non-unicode
23	CIVIX_2	2229	Non-unicode
24	coaching	334	Non-unicode
25	computer generation unicode	41	unicode
26	credit	72	Non-unicode

S.N	Document Title	Word count	Type
27	database	27	unicode
28	databaseSystem unicode	36	unicode
29	dormitory l	258	unicode
30	dormitory rule	134	Non-unicode
31	dormitory l_ alama	805	Non-unicode
32	Eivei	2580	Non-unicode
33	Exam BILL	905	Non-unicode
34	exam	245	Non-unicode
35	history	2854	Non-unicode
36	Inquiry Accounts	430	Non-unicode
37	inquiry physical	284	Non-unicode
38	inquiry Administration	638	Non-unicode
39	inquiry library	1658	Non-unicode
40	inquiry SP	1407	Non-unicode
40	inverted index unicode	24	unicode
42	iptv	36	unicode
43	khairul Alam	3629	Non-unicode
44	Internet	32	unicode
45	Message	465	Non-unicode
46	mv_APP	1241	Non-unicode
47	Proctor office	281	Non-unicode
48	regent board	279	Non-unicode
49	relational database unicode	24	unicode

S.N	Document Title	Word count	Type
50	retrievalunicode	32	unicode
51	RU	62	Non-unicode
52	serverunicode	21	unicode
53	SHAMAJ_7	9822	Non-unicode
54	sociology1part	9095	Non-unicode
55	sociology2part	8282	Non-unicode
56	stud	152	Non-unicode
57	suggestion	664	Non-unicode
58	Tax	1751	Non-unicode
59	Testimonialcse	1417	Non-unicode
60	twotire	32	unicode
61	VC Sir	982	Non-unicode
62	Webserverunicode	45	Unicode
63	webunicode	28	Unicode

## 4.2 Postfixes Statistics

We analyzed the syntax of various words using the dictionary published by Bangla Academy of the Government of Bangladesh. We found altogether 168 postfixes that are used to form different words by adding with the root of Bangla words as shown in Table 4.2. There are three columns in the table: first column contains the postfixes, the second column contains the sequence in which they appear with the root words and the third column shows the frequency count of the postfixes in the document set. Some of the postfixes that are listed in the first column, are for “Shadhu Bhasha” and some are for “Chalti Bhasha”. As we used our document set written in “Chalti Bhasha”, so there are some postfixes whose frequency count is zero in the document set. In our experiment we found that our data set requires stemming of around 72 different kinds of postfixes, where the system can handle around 168 postfixes. The



frequency count column of Table 4.2 contains value other than zero for these 72 postfixes only.

Table 4.2: Postfix statistics

Postfix	Character sequence	Number of occurrence	Postfix	Character sequence	Number of occurrence
টি	ট+ি	১২৮১	রাশি	র+া+শ+ি	০
টা	ট+া	২২৪	ই	ই	৩৯৭৮
টির	ট+ি+র	২২৭	ইতেছি	ই+ত+ে+ছ+ি	০
টার	ট+া+র	৯৭	ছি	ছ+ি	৯০
খানা	খ+া+ন+া	১	ইস	ই+স	১৩০
খানার	খ+া+ন+া+র	৩	এন	এ+ন	৭৭
খানি	খ+া+ন+ি	০	ইতেছ	ই+ত+ে+ছ	০
খানির	খ+া+ন+ি+র	০	ইতেহিস	ই+ত+ে+ছ+ি+স	০
গাছা	গ+া+ছ+া	০	এ	এ	৪৮১৮
গাছি	গ+া+ছ+ি	২	ইতেছে	ই+ত+ে+ছ+ে	০
রা	র+া	১৪২০	ইতেছন	ই+ত+ে+ছ+ন	০
এরা	এ+র+া	২৭	ছ	ছ	০
এর	ে+র	৪৫১৬	হিস	ছ+ি+স	০
আর	া+র	৩৮৪৪	ছেন	ছ+ে+ন	১৪
রে	র+ে	১২৭৮	ছে	ছ+ে	৪২৫
র	র	৫১৬০	ইয়াছ	ই+য়+া+ছ	০
গুণো	গ+ু+ণ+ো	১১৬	ইয়াছিস	ই+য়+া+ছ+ি+স	০
গুণা	গ+ু+ণ+া	২	ইয়াছেন	ই+য়+া+ছ+ে+ন	০
গুণি	গ+ু+ণ+ি	৩৩	এছ	এ+ছ	০
দের	দ+ে+র	১০৯০	এছিস	এ+ছ+ি+স	০
দিগের	দ+ি+গ+ে+র	০	এছেন	এ+ছ+ে+ন	২০
কুল	ক+ু+ল	৭০	ইয়াছে	ই+য়+া+ছ+ে	০
গণ	গ+ণ	৬৬	ইতে	ই+ত+ে	১৬৩
জন	জ+ন	২৫২	ইতিস	ই+ত+ি+স	০
দল	দ+ল	১১১	ইতেন	ই+ত+ে+ন	০
সকল	স+ক+ল	১৭৬	তে	ত+ে	১২৬৩
গম্বুহ	স+ম+ব+হ	৮	তিস	ত+ি+স	০
সমুদয়	স+ম+ব+দ+য়	১	তেন	ত+ে+ন	১
গর্ব	গ+র+ব	২	ইত	ই+ত	১১২৮
শ্রেণী	শ+্র+ণ+ে+ণ+ী	০	ত	ত	২৫৩৮
পাল	প+া+ল	১৪	ইতেছিলাম	ই+ত+ে+ছ+ি+ল+া+ম	০
বন্দ	ব+ন+দ	০	ছিলাম	ছ+ি+ল+া+ম	১৩
বর্গ	ব+র+গ	৩	ছিলুম	ছ+ি+ল+ু+ম	০
মণ্ডলী	ম+ণ+ড+ল+ী	০	ছিলেম	ছ+ি+ল+ে+ম	০
মহল	ম+হ+ল	০	ইতেছিলে	ই+ত+ে+ছ+ি+ল+ে	০
যুথ	য+থ	০	ইতেছিলি	ই+ত+ে+ছ+ি+ল+ি	০
আবলী	আ+ব+ল+ী	৭১	ইতেছিলেন	ই+ত+ে+ছ+ি+ল+ে+ন	০
গুচ্ছ	গ+ু+চ্ছ	০	ছিলে	ছ+ি+ল+ে	০
গ্রাম	গ+্র+া+ম	৫৯	ছিলি	ছ+ি+ল+ি	০
চয়	চ+য়	২৫	ছিলেন	ছ+ি+ল+ে+ন	২২
জাম	জ+া+ম	১	ইতেছিল	ই+ত+ে+ছ+ি+ল	০
দাম	দ+া+ম	০	ছিল	ছ+ি+ল	১৭৯
নিকর	ন+ি+ক+র	০	ইয়াছিলাম	ই+য়+া+ছ+ি+ল+া+ম	০
নিচয়	ন+ি+চ+য়	০	এছে	এ+ছ+ে	৭৩
পটল	প+ট+ল	০	উন	উ+ন	১২৯
পুঞ্জ	প+ু+ঞ্জ	০	উক	উ+ক	২০
মালা	ম+া+লা	৮	ইলাম	ই+ল+া+ম	৩
রাজি	র+া+জ+ি	১	লাম	ল+া+ম	১১৪

Postfix	Character sequence	Number of occurrence
মুম	ল+ <sub>u</sub> +ম	৭
লেম	ল+ <sub>e</sub> +ম	০
ইলে	ই+ল+ <sub>e</sub>	৯
ইলি	ই+ল+ <sub>i</sub>	০
ইলেন	ই+ল+ <sub>e</sub> +ন	৭০
লে	ল+ <sub>e</sub>	৬৬৩
লি	ল+ <sub>i</sub>	৮৩
লেন	ল+ <sub>e</sub> +ন	০
ইল	ই+ল	১৫৫
ল	ল	১৭২৯
ইতাম	ই+ত+ <sub>i</sub> +ম	০
তাম	ত+ <sub>i</sub> +ম	৫
তুম	ত+ <sub>u</sub> +ম	০
তেম	ত+ <sub>e</sub> +ম	০
এছিলাম	এ+ছ+ <sub>i</sub> ল+ <sub>i</sub> +ম	২
এছিলেম	এ+ছ+ <sub>i</sub> ল+ <sub>e</sub> +ম	০
এছিলুম	এ+ছ+ <sub>i</sub> ল+ <sub>u</sub> +ম	০
ইয়াছিলে	ই+য়+ <sub>i</sub> +ছ+ <sub>i</sub> ল+ <sub>e</sub>	০
ইয়াছিলি	ই+য়+ <sub>i</sub> +ছ+ <sub>i</sub> ল+ <sub>i</sub>	০
ইয়াছিলেন	ই+য়+ <sub>i</sub> +ছ+ <sub>i</sub> ল+ <sub>e</sub> +ন	০
এছিলেম	এ+ছ+ <sub>i</sub> ল+ <sub>e</sub> +ম	০
এছিলুম	এ+ছ+ <sub>i</sub> ল+ <sub>u</sub> +ম	০
ইয়াছিলে	ই+য়+ <sub>i</sub> +ছ+ <sub>i</sub> ল+ <sub>e</sub>	০
ইয়াছিলি	ই+য়+ <sub>i</sub> +ছ+ <sub>i</sub> ল+ <sub>i</sub>	০
ইয়াছিলেন	ই+য়+ <sub>i</sub> +ছ+ <sub>i</sub> ল+ <sub>e</sub> +ন	০
এছিলেম	এ+ছ+ <sub>i</sub> ল+ <sub>e</sub> +ম	০
এছিলে	এ+ছ+ <sub>i</sub> ল+ <sub>e</sub>	০
এছিলি	এ+ছ+ <sub>i</sub> ল+ <sub>i</sub>	০
এছিলেন	এ+ছ+ <sub>i</sub> ল+ <sub>e</sub> +ন	০
ইয়াছিল	ই+য়+ <sub>i</sub> +ছ+ <sub>i</sub> ল	০
এছিল	এ+ছ+ <sub>i</sub> ল	২৯
ইব	ই+ব	৬৪
ব	ব	২০৬১
ইবে	ই+ব+ <sub>e</sub>	৬
ইবি	ই+ব+ <sub>i</sub>	২
ইবেন	ই+ব+ <sub>e</sub> +ন	০
বে	ব+ <sub>e</sub>	৭১১
বি	ব+ <sub>i</sub>	৭৩১
বেন	ব+ <sub>e</sub> +ন	১৫
ইও	ই+ও	৪

### 4.3 Creation of Stoplist for the Dataset

All Bangla documents contain some auxiliary words like কে, সেখানে, এর, এই etc. These are the stopwords. We found 15 stopwords in the dataset as given in Table 4.3. We also included some terms like বিভাগ, নোটিশ, তারিখ etc. that are not auxiliary but occur in most of the documents of our document set. So these terms will not give considerable significance in measuring the relevance of documents. In our system we added the terms with more than fifty percent occurrence in the document database into stoplist. Terms like these are domain specific that means it varies if we take the document files from different domain. In this way a stoplist was created and then stored in the database.

Table 4.3: List of stop word with document frequency

WORD	OCCURRED DOC_FREQ
এক	40
এবং	39
এর	44
তারিখ	42
ও	52
কর	51
ঘর	31
জন্য	43
নাম	31
বিভাগ	32
ঘবে	35
বিল	38
নোটিশ	39
মানুষ	34
সময়	31
হবেন	User defined

WORD	OCCURRED DOC_FREQ
এটি	User defined
এটা	User defined
টি	User defined
এতে	User defined
তথা	User defined
প্রায়	User defined
হবে	User defined
করা	User defined
যায়	User defined
কে	User defined
সেখানে	User defined
এর	User defined
এই	User defined
যার	User defined
যেটি	User defined
সেটি	User defined

## 4.4 Stemming Performance

Stemming has been described in section 3.2.3. We performed stemming on the dataset initially with ten documents and incrementally added ten word documents in the system. Table 4.4 shows the number of stemming required and the required time for stemming of different number of documents. Figure 4.1 shows that the number of stemming required increases with the increase of document number. Figure 4.2 shows that the stemming time increases almost linearly with the increase of document number.

Table 4.4: Stemming results

Document number	Total No. of words	Words stemmed	Required time (m-sec)
10	11019	7424	3703
20	39411	15581	6578
30	46486	17557	7594
40	59325	21383	8844
50	64933	22859	9875
60	987021	31657	12781

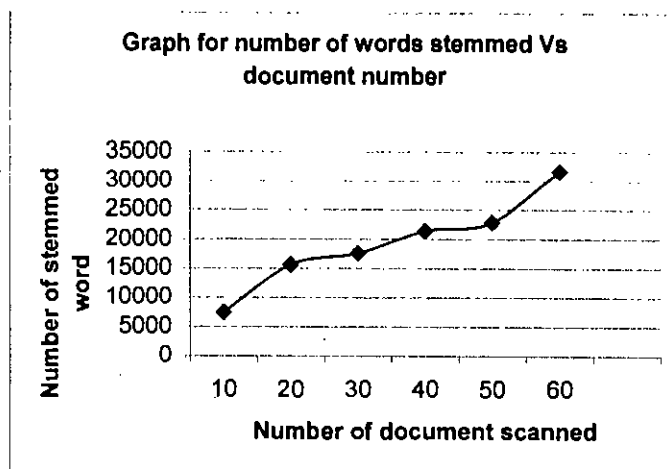


Figure 4.1: Relation between document number and required stemming

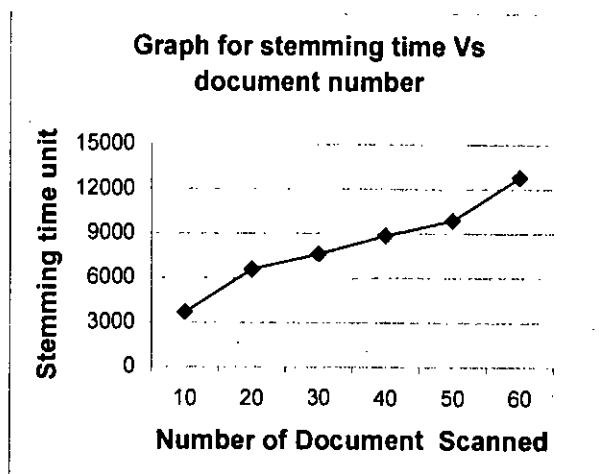


Figure 4.2: Relation between document number and stemming time

## 4.5 The Effect of Number of Occurrences of Postfixes

We applied the morphological analyzer on the Bangla data set and found that total of 72 postfixes were used to form different words. Table 4.2 shows the list of postfixes, their character sequence and number of occurrence in the dataset. The frequency count of the postfixes ranges from 0 to a maximum of about 4500 with an average of 570. The standard deviation of frequency count for different postfixes is very high. It indicates that there are some postfixes with very high occurrence in the dataset and querying with words forming by these postfixes will affect significantly if stemming is not performed.

As mentioned before there are two writing styles in Bangla language – namely “Shadhu Bhasha” and “Chalti Bhasha”. Recent trend in Bangla document writing uses “Chalti Bhasha”. So we used the dataset consisting of “Chalti Bhasha”. We found 55% of 168 postfixes with zero occurrences count in the dataset. From these statistics we can estimate that about half of the postfixes are used for “Chalti Bhasha” and half for the “Shadhu Bhasha”.

We measured the retrieval effectiveness (as shown in Table 4.5 and 4.6) for the postfix with maximum occurrences in the dataset. We found 3 relevant documents out of 8 (Table 4.5) without using morphological analysis. This poor performance result is due to the words with postfixes resembles to different words and not participate in the relevance measurement. We obtained improved performance (6 documents relevant out of 8) by using morphological analysis as shown in Table 4.6. Precision is the proportion of retrieved documents that are relevant. Recall is the proportion of relevant documents that are retrieved. Precision and Recall are inversely related. One IR system is considered to have performed better than another when its precision-recall curve is above and to the right of the other. The precision recall curve for this scenario is given in Figure 4.3.

Table 4.5: Precision and recall for frequent postfix without morphological analysis

Document Title	Rank	Actual relevance	Recall	Precision
SHAMAJ_7.doc	1	yes	0.16	1
Message.doc	2	yes	0.33	1
BOU.doc	3	yes	0.50	1
bari sir1.doc	4	no	0.50	0.75
mv APP.doc	5	no	0.50	0.60
AI.OWA.doc	6	no	0.50	0.50
ANNA_PO.doc	7	no	0.50	0.43
attendance.doc	8	no	0.50	0.38

Table 4.6: Precision and recall for frequent postfix with morphological analysis

Document Title	Rank	Actual relevance	Recall	Precision
AadComittee.doc	1	yes	0.16	1
Alama Khatun2.doc	2	yes	0.33	1
Alama khatun.doc	3	yes	0.50	1
SHAMAJ_7.doc	4	yes	0.67	1
bari sir1.doc	5	yes	0.83	1
sociology1part.doc	6	no	0.83	0.83
BU.doc	7	yes	1.00	0.86
Backup of khairul Alam.doc	8	no	1.00	0.75

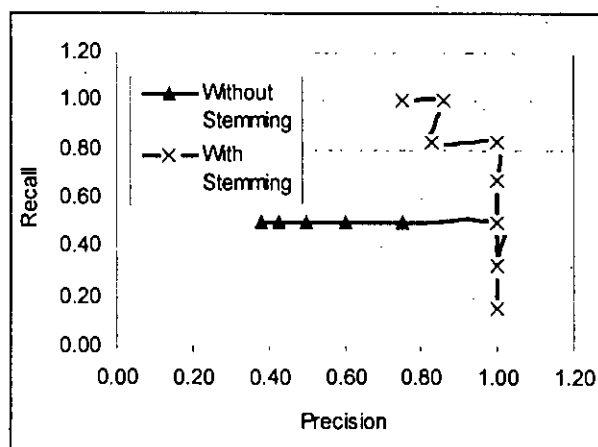


Figure 4.3: Precision-recall curve for frequent postfix

## 4.6 The Effect of Number of Terms in Query Expression

We have studied the system with various query options like single term, two terms, and three terms. The results are shown in Table 4.7, 4.8 and 4.9. We ran the queries with stemming in both query vector and document vector. We found that the ranking of documents changes as more terms are added to the queries. This is because the context of the query changes as more words are added to the query and at the same time, the ranking of the documents also changes. Our system does not consider the proximity of the terms while evaluating the queries. This could be incorporated by using a window of query size. Table 4.10 shows the required retrieval time for one, two and three query terms. The required time increases with the increase of query terms, but it does not increase linearly with the number of term. The effect is shown in Figure 4.4.

Table 4.7: Precision and recall (one word)

Document Title	Rank	Actual relevance	Recall	Precision
AadComittee.doc	1	yes	0.14	1
Alamakhatun.doc	2	yes	0.29	1
bari sir1.doc	3	yes	0.43	1
Exam BILL.doc	4	yes	0.57	1
Chairman1.doc	5	yes	0.71	1
khairul Alam.doc	6	yes	0.86	1
Bill of admission1.doc	7	yes	1.00	1
Eivci.doc	8	no	1.00	0.88

Table 4.8: Precision and recall (two words)

Document Title	Rank	Actual relevance	Recall	Precision
AadComittee.doc	1	yes	0.16	1
Chairman1.doc	2	yes	0.33	1
BOU_77.doc	3	yes	0.50	1
BOU_2.doc	4	yes	0.67	1
BOU .doc	5	yes	0.83	1
Eivei.doc	6	no	0.83	0.83
BOU00.doc	7	yes	1.00	0.86
sociology2part .doc	8	no	1.00	0.75

Table 4.9: Precision and recall (three words)

Document Title	Rank	Actual relevance	Recall	Precision
AadComittee.doc	1	yes	0.17	1
Alama Khatun2.doc	2	yes	0.33	1
Alama khatun.doc	3	yes	0.50	1
bari sir1.doc	4	yes	0.67	1
BOU 77 .doc	5	no	0.67	0.80
SHAMAJ 7.doc	6	yes	0.83	0.83
sociology1part.doc	7	yes	1.00	0.86
sociology2part .doc	8	no	1.00	0.75

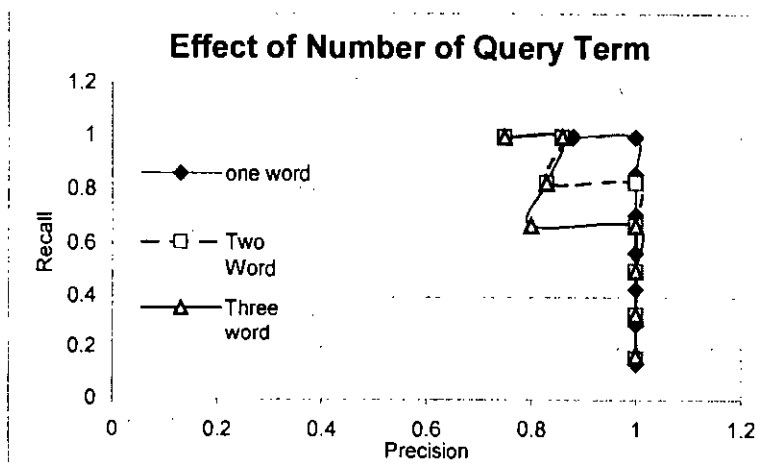


Figure 4.4: Effect of number of query term in query expression

Table 4.10: Retrieval time for query terms

Number of query terms	Retrieval time (m-s)
1	1344
2	2625
3	3141



## 4.7 The Effect of Consideration of Document Length

For weighting the terms of a document, the document length was brought into consideration. If a term occurs in a document which is very small in length then it becomes important. But if it occurs in a document which is large in length then the term will not be as much important as of a small document. We ran the query for a system considering and not considering the document length and found better performance with considering document length, which is shown in Table 4.11 and Table 4.12. The precision recall curve for Table 4.11 and 4.12 are shown in Figure 4.5.

Table 4.11: Precision and recall (not considering document length)

Document Title	Rank	Actual relevance	Recall	Precision
AadComittee.doc	1	yes	0.20	1
Bill of admission.doc	2	yes	0.40	1
Chairman1.doc	3	yes	0.60	1
Chairman2.doc	4	yes	0.80	1
Testimonial csc.DOC	5	no	0.80	0.80
Bu.doc	6	no	0.80	0.67
budgetU.doc	7	no	0.80	0.57
RU.doc	8	no	0.80	0.50

Table 4.12: Precision and recall (considering document length)

Document Title	Rank	Actual relevance	Recall	Precision
AadComittee.doc	1	yes	0.20	1
Credit.doc	2	yes	0.40	1
Credit2.doc	3	yes	0.60	1
Chairman1.doc	4	yes	0.80	1
RU.doc	5	no	0.80	0.80
Bill of admission.doc	6	yes	1.00	0.84
Testimonial.doc	7	no	1.00	0.78
BudgetU.doc	8	no	1.00	0.66

103109

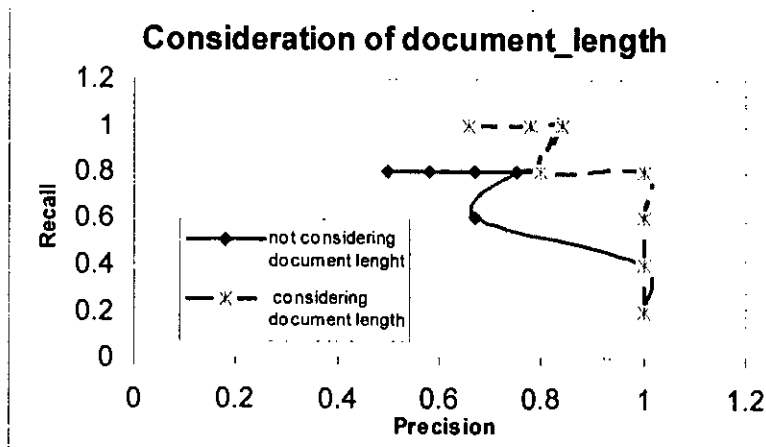


Figure 4.5: Consideration of document length

## 4.8 The Effect of Morphological Analysis

Experimental result shows that the dataset contains a total of 128518 distinct words, where only 41733 words out of them require stemming. This implies that almost one third of the words are not root words. Words that are formed by the addition of postfixes with roots are called extended words. These extended words are considered as different word if stemming is not performed. So the relevance value  $r(q, d)$  is affected and many relevant documents become irrelevant and the performance of the IR system is degraded.

We ran a number of experiments to test the effect of morphological analysis on IR system in Bangla. Table 4.13 shows some relevance value for different query expressions with three query terms. We find from the table that the retrieval may vary, because of the presence of a number of relevant documents. We used the terms বিশ্ববিদ্যালয়ের ভর্তির ফলাফল for measuring precision and recall and the results are shown in Table 4.14 and 4.15. We found 4 relevant documents for the top ranking 8 documents without stemming, which is shown in Table 4.14. Table 4.15 shows that by doing morphological analysis 5 relevant documents are found out of 8 top ranking documents. The precision-recall curve for Table 4.14 and 4.16 are shown in the Figure 4.6. It shows that by using morphological analysis, on the average, 20% better precision with 14% better recall can be achieved.

Table 4.13: Relevance for different query expression

SL. NO.	Query terms	Relevant	Irrelevant
1	উরমিটরীর নিয়মসমূহ পর্যালোচনা	3	5
2	বাজেটে বরাদ্দ খাতগুলো	6	2
3	বাংলাদেশে প্রযুক্তির উন্নতি	5	3

Table 4.14: Precision and recall (not considering morphology)

Document Title	Rank	Actual relevance	Recall	Precision
AadComittee.doc	1	yes	0.20	1
Credit.doc	2	yes	0.40	1
Ru.doc	3	no	0.40	0.67
Bill of admission.doc	4	yes	0.60	0.75
Chairman1.doc	5	yes	0.80	0.80
BudgetU.doc	6	no	0.80	0.67
inquerySP.doc	7	no	0.80	0.58
Bu.doc	8	no	0.80	0.50

Table 4.15: Precision and recall (considering morphology)

Document Title	Rank	Actual relevance	Recall	Precision
AadComittee.doc	1	yes	0.20	1
Credit.doc	2	yes	0.40	1
Credit2.doc	3	yes	0.60	1
Chairman1.doc	4	yes	0.80	1
RU.doc	5	no	0.80	0.80
Bill of admission.doc	6	yes	1.00	0.84
Testimonial.doc	7	no	1.00	0.78
BudgetU.doc	8	no	1.00	0.66

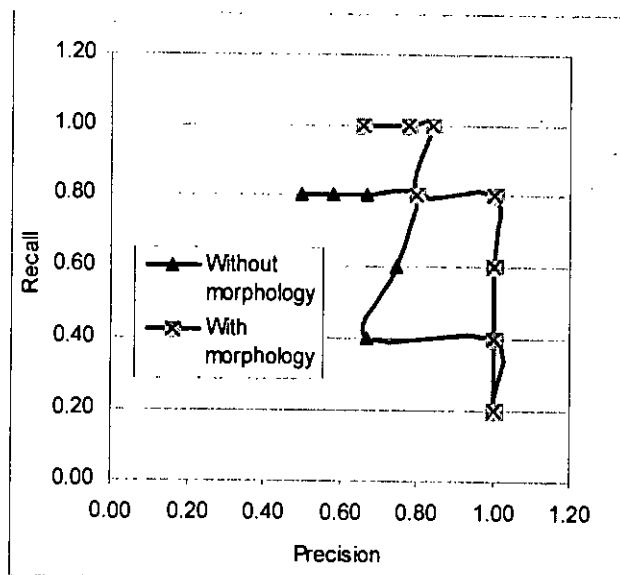


Figure 4.6: Precision-recall curve for morphological analysis

## 4.9 Comparison with Other IR systems

Table 4.16 shows comparison of different information retrieval systems. The information retrieval system developed by **Koudas et al. [15]** works for English language. Here partial matching is used for finding relevance. The weighting technique has been modified for better relevance. They used vector space model and cosine similarity for measuring relevance. But they did not do any kind of morphological analysis for stemming the words.

Table 4.16: Different steps of Different IR system

	<b>Koudas et al.</b>	<b>Jelita et al.</b>	<b>Chowdhury et al.</b>	<b>Islam et al.</b>	<b>Our IR system</b>
<b>Morphological analysis</b>	Not done	Not done	done	Not done	done
<b>Vector space model</b>	Used	yes	no	Used	used
<b>Cosine similarity</b>	Used	Not used	Not used	Used	used

The information retrieval system developed by **Jelita et al. [24]** works for Indonesian language. Instead of cosine similarity measure they used pooling to find the actual relevant documents. They represented documents in a structured way, which is like vector space model for adhoc retrieval. They showed that stemming does not show retrieval accuracy for Indonesian language. The information retrieval system developed by **Chowdhury et al. [25]** works for information retrieval system in Arabic language. The information retrieval system developed by **Islam et al. [26]** works for Bangla language which is based on vector space model and cosine similarity measure. They used term frequency and document frequency for weighting and did not do any morphological analysis for stemming. They did not show any experimental results also. Table 4.17 and 4.18 show that by doing morphological and considering document length better performance can be achieved than that of IR system developed by **Islam et al. [26]**. The precision-recall curve for Table 4.17 and 4.18 are shown in the Figure 4.7.

Table 4.17: Precision and recall (not considering morphology and document length)

Document Title	Rank	Actual relevance	Recall	Precision
AadComittee.doc	1	yes	0.17	1
Bill of admission.doc	2	yes	0.33	1
Exam BILL.doc	3	yes	0.50	1
Chairman1.doc	4	no	0.50	0.75
sociology2part .doc	5	yes	0.67	0.80
Eivei.doc	6	no	0.67	0.67
BOU 77 .doc	7	no	0.67	0.57
Bu.doc	8	no	0.67	0.50

Table 4.18: Precision and recall (considering morphology and document length)

Document Title	Rank	Actual relevance	Recall	Precision
AadComittee.doc	1	yes	0.17	1
Exam BILL.doc	2	yes	0.34	1
Bill of admission.doc	3	yes	0.50	1
khairul Alam.doc	4	yes	0.67	1
Testimonial.doc	5	yes	0.83	1
Chairman1.doc	6	yes	1	1
BOU 77 .doc	7	no	1	0.86
Eivei.doc	8	no	1	0.75

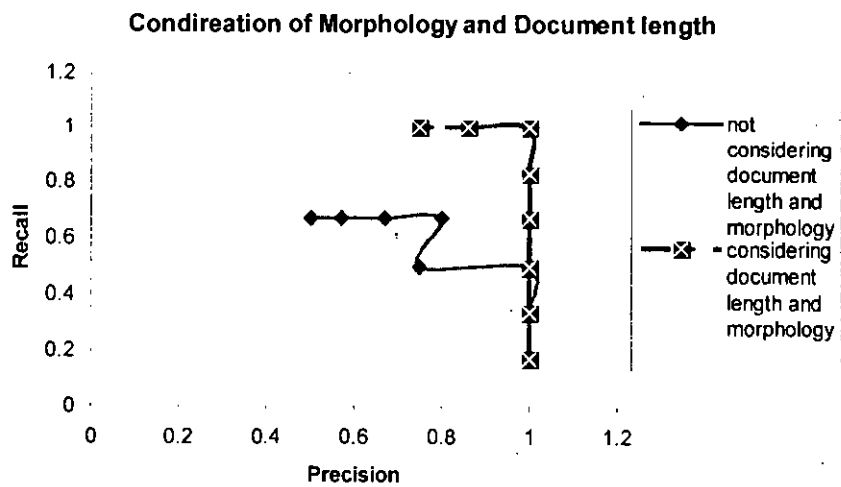


Figure 4.7: Precision-recall curve for morphological analysis and document length

---

## Chapter 5

# Conclusion and Future Research

---

### 5.1 Conclusion

We have developed an IR system for retrieval of most relevant documents to a query from Bangla text database. The problems of IR in Bangla text database are

- i) huge number of words with different postfixes forming different words with same meaning,
- ii) synonym handling and
- iii) unicode and non-unicode format of data.

We have explored the set of postfixes that are added to the root of a word forming different words. There are two major styles of writing Bangla documents: “Shadhu Bhasha” and “Chalti Bhasha”. We have studied the effect of postfixes on the language styles. About 45% of the postfixes are used in “Chalti Bhasha” and the rest are for “Shaddhu Bhasha”.

We have also evaluated the effect of the frequency of occurrence of postfixes and found that the postfixes with higher frequency of occurrence require stemming operation to improve performance of the system.

Efficient IR system in Bangla requires a morphological analysis. We have developed an efficient morphological analyzer for this purpose. Our experimental results show that 20% better precision with 14% better recall can be achieved using the morphological analyzer.

A synonym handling technique have been developed which handles synonyms efficiently. A dictionary-based synonym handling mechanism has been developed. The

IR system can find any equivalent synonym efficiently from the database of synonyms for document database initialization or query processing.

We have also developed a fonthandling mechanism to retrieve information from both unicode and non-unicode type of data uniformly. A substitution method is used to convert non-unicode text into unicode text. If any non-unocode text is found then it is converted into unicode and necessary information regarding that text is stored in the database for information retrieval. For unicode-supported text, it is stored directly in the database.

## **5.2 Suggestions for Further Research**

The document database that is used here are domain specific. The document set was collected from registry section of a public university. Bangla being one of the ten most spoken languages requires the development of full database of Bangla text.

A corpus of Bangla text documents may be constructed fully in future. The performance for Bangla corpus may be tested with the IR system that has been developed in this work.

This thesis work can also be extended in cross language information retrieval system adopting machine translation facility.



---

## Bibliography

- [1] Bush, V., "As We May Think", *Atlantic Monthly*, Vol-176, pp 101-108, July 1945.
- [2] Luhn, H. P., "A Statistical Approach to Mechanized Encoding and Searching of Literary Information", *IBM Journal of Research and Development*, pp 234-244, 1957.
- [3] Salton, G., "The SMART Retrieval System—Experiments in Automatic Document Retrieval", Prentice Hall Inc., Englewood Cliffs, NJ, 1971.
- [4] Cleverdon, C. W., "The Cranfield Tests on Index Language Devices", *Aslib Proceedings*, Vol-19, pp 173-192, 1967.
- [5] Harman, D. K., "Overview of the First Text Retrieval Conference (TREC-1)", In *Proceedings of the First Text Retrieval Conference (TREC-1)*, pp 1-20, NIST Special Publication 500-207, March 1993.
- [6] Salton, G. and McGill, M. J., "Introduction to Modern Information Retrieval", McGraw Hill Book Co., New York, 1983.
- [7] Robertson, S. E., Walker, S. and Beaulieu, M., "Okapi at TREC-7: Automatic Ad-hoc, Filtering, VLC and Filtering Tracks", In *Proceedings of the Seventh Text Retrieval Conference (TREC-7)*, pp 253-264. NIST Special Publication 500-242, 1999.
- [8] Jones, K. S. and Willett, P., "Readings in Information Retrieval", Morgan Kaufmann, 1997.
- [9] Robertson, S. E., "The Probabilistic Ranking Principle in IR", *Journal of Documentation*, Vol-33, pp 294-304, 1977.
- [10] Maron, M. E. and Kuhns, J. L., "On Relevance, Probabilistic Indexing and Information Retrieval", *Journal of the ACM*, Vol-7, pp 216-244, 1960.
- [11] Croft, W. B. and Harper, D. J., "Using Probabilistic Models on Document Retrieval without Relevance Information", *Journal of Documentation*, Vol-35, pp 285-295, 1979.

- 
- [12] Salton, G., Wong, A. and Yang, C. S., "A Vector Space Model for Information Retrieval", *Communications of the ACM*, Vol-18, No. 11, pp 613-620, November 1975.
- [13] Piotrowski, M., "NLP-Supported Full-Text Retrieval", *CLUE Technical Reports-3*, pp 1-55, 2001.
- [14] Kahveci, T. and Singh, A. K., "An Efficient Index Structure for String Databases", *Proceedings of the 27<sup>th</sup> VLD Conference*, pp 351-360, 2001.
- [15] Koudas, N., Marathe, A. and Srivastava, D., "Flexible String Matching against Large Databases in Practice", *Proceedings of the 30<sup>th</sup> VLDB Conference*, pp 1-9, 2004.
- [16] Zobel, J., Moffat, A. and Davis, R. S., "An Efficient Indexing Technique for Full Text Database System", *Proceedings of the 18<sup>th</sup> VLDB Conference*, pp 352-362, 1992.
- [17] Faloutsos, C. and Chritodoulakis, S., "Signature Files: An Access Method for Documents and its Analytical Performance Evaluation", *ACM Transaction on Database Systems*, Vol-2, pp 267-288, 1984.
- [18] Martynov, M. and Novikov, B., "An Indexing Algorithm for Text Retrieval", *Proceedings of the International Workshop on Advances in Databases and Information Systems*, pp 171-175, 1996.
- [19] Tomasic, A. and Molina, H. G., "Query Processing and Inverted Indices in Shared-Nothing Text Document Information Retrieval Systems", *VLDB Journal*, Vol- 2, pp 243-275, 1993.
- [20] Chiueh, T. and Huang, L., "Efficient Real-time Index Updates in Text Retrieval Systems", *ECSL Technical Report 66*, pp 1-10, 1999.
- [21] Finch, S., "Exploiting Sophisticated Representations for Document Retrieval", *Proceedings of the 4th Conference on Applied Natural Language Processing*, pp 65-70, 1994.
- [22] D'Souza, D. J., Thom, J. A. and Zobel, J., "A Comparison of Techniques for Selecting Text Collections", *Australasian Database Conference*, pp 1-10, 2000.
- [23] Losee, R. M., "When Information Retrieval Measures Agree about the Relative Quality of Document Ranking", *Journal of the American Society for Information Science* Vol -51, No. 9, pp 834-840, 2000.

- [24] Asian, J. and Williams, H. E., "A Testbed for Indonesian Text Retrieval", Proceedings of the 9th Australasian Document Computing Symposium, Australia, pp 1-4, 2004.
- [25] Chowdhury, A., Alilayl, M., Jenson, E., Beitzel, S., Grossman, D. and Frieder, O., "Linear Combinations Based Document Structure and Varied Stemming for Arabic Information Retrieval", Information Retrieval Laboratory, Illinois Institute of Technology, Chicago, 2002.
- [26] Islam, M. T. and Masum, S. M. A., "Bhasa: A Corpus-based Information Retrieval and Summarizer for Bengali Text", Proceedings of the 7th International Conference on Computer and Information Technology, pp 627-631, 2004.
- [27] Rijsbergen, V., "Information Retrieval", Second Edition, Online Version, London, Butterworths, 1979.
- [28] Xu, J. and Croft, W. B., "Cluster-based Language Models for Distributed Retrieval", Proceedings of the 22<sup>nd</sup> Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, United States, pp 254-261, 1999.
- [29] Voorhees, E. M., "Information Extraction: Towards Scalable Adaptable Systems". Springer, Berlin, 2004.

