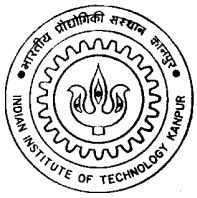
A Web-based Search Engine for Indian Languages

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

Manoj Kumar Malviya



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur
March, 1999

Certificate

This is to certified that the work contained in the thesis entitled "*A Web-based Search Engine for Indian Languages*", by *Manoj Kumar Malviya*, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Rajat Moona)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

(Dr. T. V. Prabhakar)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

March, 1999

Abstract

In this thesis, a Web based Search Engine for Indian languages is implemented. This software allows full-text indexing and searching of a database of HTML documents written in any Brahmi-based Indian Language and English. Gatherer, Indexer and Search Processor are the basic components of this search engine. Gatherer retrieves HTML documents and gathers information from the documents which is then used by the indexer to create the document index. The search Processor contains the logic of document searching based on the search query. It is capable of searching all phonetically equivalent words and different forms of a keyword of any Indian language. Indexing and searching of Hindi and English text has been tested on it.

Acknowledgments

I would like to express my deep sense of gratitude to my thesis supervisors, Dr T. V. Prabhakar and Dr. Rajat Moona for their expert guidance and encouragement throughout this thesis work. Inspite of their hectic schedule they were always approachable and took their time off to attend to my problems and give the appropriate advice. I am greatly indebted to Dr Rajeev Sangal and Dr. Vineet Chaitanya for their guidance and support throughout my work. I thank all my classmates and friends specially Farhan, Guptaji, Kapil, Prasad, Bepari, Prasanna, Nihal, Atul, Srikar, Uma, Saleem, Mahanta, and Sishir for their help and support. I wish I could express my thankfulness to all my old friends for their love, support and encouragement. I thank my parents and my sisters for their love and affection I have been receiving. Finally I thank God for being kind to me and driving me through this journey.

Contents

Chapter 1

Introduction

With the rapid growth of Internet, the availability of information becomes less of a problem as large amounts of digitally stored information is readily available on the Internet. But this information is so much that it becomes increasingly difficult and time-consuming for the users to find the information relevant to their needs. This explosive growth of information on the Internet has greatly increased the need for information retrieval systems. Yahoo, Infoseek, Excite, Altavista, are some search systems (search engines) that offers extensive coverage by trying to index on the entire World Wide Web (WWW). Some of these search engines (like Altavista) have multilingual search capabilities, but it is confined to some European languages. None of these support Indian languages. Hence we found the need for a search engine capable of searching Indian language documents.

1.1 Multilingualism on WWW

A wide number of languages are spoken by human beings in the world, and most of the people prefer to have information in their own language . The WWW, being the largest repository of information, should have the information in such a way so that maximum people can take benefit of it. But unfortunately, this information is not reaching to the widest of the masses. This is primarily because of the sweeping dominance of English and western-European languages on the web.

However this situation is beginning to change, because most of the new Internet

users will not have English as a mother-tounge, and almost everyone wants to use Internet in his or her native language. We can predict that, in several years, we will have a situation similar to the one in publishing regarding the representation of different languages.

The arrival of the languages other than English, points out the need of the support in Internet based applications like HTML, HTTP server, Browser, Search Engines, etc. The main hindrance in achieving this support on the Web is poor standards and protocols in terms of multilingual text representation and rendering. Of late, this has been realized and newer standards and protocols have been proposed and implemented. Now HTML has several tags which allow one to specify fonts and language attributes of a particular section of text. The newer version(1.1) of HTTP also has several new tags especially to aid language negotiation and thereby to achieve multilingualism on Web. A character coding scheme, UNICODE[13], has already been designed to support the interchange, processing, and display of text in many languages of the modern world. Unicode is a large character set that includes most of the world languages. Moreover, there are search engines that index and search text in many European languages other than English.

1.2 Motivation

As the Internet is becoming popular in India, the number of the documents written in Indian languages is also increasing day by day. Many news-papers and magazines of the Indian languages are now available on the Internet. There are search engines which provide capability of searching the documents of various European languages. The most popular search engine 'Altavista' can search the documents of many European languages like French, German, Greek, etc. But there is no such searching facility exists for any of the Indian language. The motivation of this thesis was to develop a search engine which could index and search the documents of Indian languages available on the Internet. Though, we have provided the search engine for Hindi Language, the same design can be applied to search the documents of any other Brahmi-based Indian language.

1.3 Organization of Thesis

Rest of the thesis is organized as follows. Chapter 2 introduces the terms and concepts of the search engines and the Indian Languages . Chapter 3 discusses the design and implementation details of different parts of our search engine. Results of testing, Features of the software, and future work are discussed in chapter 4. User manual of the search engine is provided in the appendix.

Chapter 2

Background

2.1 Search Engines

A Search Engine is a program (CGI, server module or separate server) that accepts the request from the HTML form, searches the indexed database and returns the result page to the user. In this thesis, an attempt has been made, to develop a Search Engine for Indian languages which provides the capability of indexing and searching of the text written in any of the Indian language as well as in English. In order to fully comprehend the capabilities of such a search engine, it is essential, to understand the basic functionality of the search engines and computer representation of Indian languages. This chapter provides the background which will be useful later when we discuss the design of our search engine. For rest of the discussion Hindi is chosen, because all the Indian languages originating from Brahmi have a common structure, hence all arguments for Hindi are also applicable to other Brahmi-based Indian languages[4].

2.1.1 Types of Search Engines

The increasing sophistication of search engines makes the categorization of search engines problematic. But the search engines can be divided into four categories based on the original type of services offered by them [1].

- *Full-Text Search Engines*

Full-text search engines (also known as free-text search engines) analyze the contents of the documents in such a way as to allow users to search for any string of text they wish to find. A good full-text search engine will incorporate some form of relevance weighting mechanism, so that items that have a higher level of relevance are displayed first. The relevance can be based on the factors such as the number of times the search word occurs in the test, the position of the search word in the documents, etc. Altavista and Lycos are full-text search engines.

- *Catalogue-based Search Engines*

Catalogue-based search engines (also known as index search engines) use some form of classification system. This system classifies the documents in different categories. These categories could be like entertainment, Internet, medicine, country, etc. Efficiency of searching in catalogue-based search engines depends on the way in which the items are categorized. The number of categories at each level determine the number of levels that need to be traversed. Many catalogue-based search engines also offer full-text searching of files. Yahoo is a good example of this category.

- *Meta-Search Engines*

Meta-search engines (also known as multi-search engines) allow users to search for the same keywords using more than one search engine, either sequentially or simultaneously. MetaCrawler and Find it! comes in this category.

- *Specialist Search Engine*

The search engines that are specifically designed to provide responses relevant to specific areas of knowledge, comes in this category. This doesn't include those search engines run by individual companies. It restrict itself to a few examples of wide ranging database search tools that cover the needs of particular user communities. The example of this category is Interactive Movie Database Search(IMDb).

2.1.2 Parts of a Search Engine

The search engines have three major elements [2].

1. Gatherer
2. Indexer, and
3. Search Processor

Gatherer

A search engine finds information for its database by accepting listing sent in by authors wanting exposure or by getting the information from their gatherer. It is also called the "web crawler", "spider" or "robots". These are the programs that roam the Internet and store links and information about each page they visit. These agents normally start with a historical list of links, such as list of most popular or best sites, and follow the links on these pages to find more links to add to the database. A Web Crawler could send back just the title and URL of each page, it visits, or just parse some HTML tags, or it could send back the entire text of each page.

Indexer

Everything the spider finds goes into the second part of a search engine, the index. The index, sometimes called the catalogue, is prepared by the program called indexer. The index is like a giant book containing a copy of every web page that the spider finds. If a web page changes, then this book is updated with new information. The purpose of indexing is to process the documents to be searched and to extract appropriate information. This information is stored in a data structure that will allow fast searching of the text. During Indexing, the search engine processes documents in a number of steps, including word extraction, stop word removal, word stemming and term weight calculation.

- **Word Extraction**
  For indexing purposes, it is necessary to convert the document text from a long stream of characters into a stream of words. This process is often called *word*

breaking, *word segmentation*, or *lexical analysis*. According to the language of the document text, the indexer uses a different algorithm to extract words from the text.

- **Stop Word Removal**
  In some languages such as English, functional words (e.g. "the", "a", "and", "that") are useless for indexing purposes. Similarly there are many words (like वह, मैं, वो, ये etc.) occurring in Hindi as well. These words occur in almost every document of the language, and therefore do not help in distinguishing between documents that are about different topics. For this reason, these functional words are removed and are not indexed. The process of removing these functional words is called stop words removal, and the functional words being removed are called stop words.

- **Word Stemming**
  In many languages, including Hindi and English, a word may exist in a number of morphological variants. For example, the English word "compute" may also exists in its other morphological variants such as "computing", "computed", "computer" or "computers". Similarly the Hindi word लड़का may also exists in other morphological variants such as लड़के, लड़कों etc. While these morphological variants are different word forms, they represent the same concept. For indexing purposes, it is generally desirable to combine these morphological variants of the same word into one canonical form .This process is called word stemming and this canonical form is called *root-word* or *base-word*. Without stemming, document containing word लड़के may not be returned for the query लड़कों since the indexer treats the word लड़के and लड़कों as two different words.

- **Term Weight Calculation**
  When given a search query containing keywords, any search engine will return the documents containing those keywords. This simple retrieval algorithm is ineffective especially when some of the query keywords are very common, in which case a large number of documents will be returned, most of which are not even relevant to the query submitted. For advance search engines, each document will be associated with a score indicating the relevance of the document to the query. In this way, the documents are ranked or sorted based on the score so that documents having highest scores (most relevant) are displayed first. In order to calculate the relevance score, each document term or keyword

must be weighted during indexing to indicate its importance within the document. Keywords are weighted based on a number of factors, such as the number of times the keyword appears in the document, the position of the keyword in the document, whether the keyword appears within the document title or in meta-keyword tag or in some heading tag.

Search Processor

Search processor is the third part of a search engine. This is the program that sifts through the millions of pages recorded in the index to find matches to a search and rank them in order of what it believes is most relevant. This search processor could be a CGI program, search server, or a Java servlet. The interface of this program is an HTML form. When the form is submitted, the search processor takes its values and performs the actual search. Each search engine has their own way of deciding what to do about approximate spellings, plural variations, and truncations. The user's input goes through the following steps before searching in the database, depending on the implementation of the search logic of the search processor.

- **Query Processing**
  The first task of the search processor is to extract the keywords from the input given by the user. If the search engine supports boolean commands (like AND, OR, NOT), the input could be a complex boolean expression of the keywords. The query processing routine parses the boolean expression and extract the keywords from it. After searching each keyword, the results are combined according to the boolean expression given in the query and displayed to the user.

- **Stop Word Removal**
  Search engines may remove the stop words from the query and search rest of the keywords, because this speeds the search.

- **Searching Algorithms**
  Each search engine uses its own logic for searching different forms of a word. Some search engines use fuzzy logic to generate all forms of a keyword and then search these in the database. Some engines use thesaurus to generate the

synonyms of the keywords and search those in the database. A good search engine may tolerate some phoneme errors.

2.1.3 Other Issues

Following are some of the issues which are important in determining the quality of search engines:

- *Recall and Precision*

The success of a information retrieval system is typically quantified in terms of *recall* and *precision*. Recall refers completeness. This is the degree in which a search engine returns all the matching documents in a collection . When we obtain a large number of "hits" from a search, this is known as high recall. Precision refers to the system's ability to find only the relevant documents. High precision means that the retrieved documents are highly relevant to the subject of the query [3]. Following example will clear the idea of Recall and precision.

There may be 100 matching documents for a query, but a search engine may only find 80 of them. it would then list these 80 and have a recall of 80% . Similarly if a search engine lists 80 documents found to match a query but only 20 of them are relevant to the user, then the precision would be 25%.

It may be noted that the recall and precision measured depend on a prior knowledge of what is relevant in the collection.

- *Search Result Ranking and Listing*

Search engines assign each document, they find, some measure of the quality of match . This measure is called relevance score. In order to calculate the relevance score, they follow a set of rules, involving the location and the frequency of keywords on a web page. Search engines will check to see if the keyword appears in the title, if it appears at the beginning of the document or it appears in the heading . Frequency of a word in the document is the major factor through which search engine determines relevancy. Documents with a higher frequency of a word are often considered more relevant than other documents [2].

2.2 Indian Languages and Coding Standards

India is a multilingual country having 15 officially recognized languages, written in various scripts. These existing scripts are derivative of ancient Brahmi and Perso-Arabic scripts. Urdu, Sindhi, Kashmiri are primarily written in Perso-Arabic scripts. All other Indian languages have evolved from the ancient Brahmi-script .The Northern scripts are Devanagri, Punjabi, Gujarati, Oriya, Bengali, and Assamese, while the Southern scripts are Telugu, Kannada, Malayalam, and Tamil . Different standards have been envisaged for languages which originate from Perso-Arabic scripts, and for languages which originates from Brahmi scripts. The standards for Brahmi-based Indian scripts are reviewed below [3] .

2.2.1 Nature of Indian Languages Alphabet

All Brahmi-based Indian scripts are phonetic in nature. The alphabet in each may vary somewhat, but they share a common phonetic structure. The difference between scripts primarily are in their written forms, where different combination rules get used. Hence all arguments for Devanagari are also applicable to other Brahmi-based Indian scripts. Also for simplicity, elsewhere, the term Indian scripts implies Brahmi-based Indian scripts [4].

Devanagari character set can be categorized into vowels, consonants, matras, modifiers, numerals, punctuation and some special symbol like halant and nukta .

- **The Consonants, Vowels and Matras**
  Indian script consonants have an implicit अ vowel included in them. they have been categorized according to their phonetic properties. There are 5 Varg (groups) and non-Varg consonants. Each Varg contains 5 consonants the last of which is a nasal one. The first four consonants of each Varg constitute the Primary and Secondary pair. The second consonant of each pair is the aspirated counterpart (has additional ह sound) of the first one. Table 2.1 and 2.2 shows the sets of Vowels and consonants used in Devanagari script.

  Note that the consonants श and ष are pronounced identically today. Each vowel except अ has a corresponding matra which can be attached to a consonant to form composite characters.

	Primary		Secondary		
varg1	क	ख	ग	घ	ङ
varg2	च	छ	ज	झ	ञ
varg3	ट	ठ	ड	ढ	ण
varg4	त	थ	द	ध	न
varg5	प	फ	ब	भ	म
non-varg	य	र	ल	व	श	ष	स

Table 2.1: Consonants of Devanagari

Vowels	अ	आ	इ	ई	उ	ऊ	ऋ
Matra		ा	ि	ी	ु	ू	ृ
Vowels	ए	ऐ	ओ	औ	अं	अः	
Matra	े	ै	ो	ौ	ं	ः	

Table 2.2: Vowels of Devanagari

The vowels ऐ and औ are actually diphthongs (i.e. a compound vowel character, in which the articulation begins as for one vowel and moves onto another) although in Hindi they get pronounced as longer vowel form of ए and ओ respectively.

- **Anuswar**
  Anuswar ं indicates a nasal consonant sound. When an Anuswar comes before a consonant belonging to any of the 5 Vargs, then it represents the nasal consonant belonging to the Varg (see Table 2.3). Before a non-Varg consonant however the anuswar represents different nasal sound.

- **Chandrabindu**
  This ँ denotes nasalization of the preceding vowel (can be implicit अ vowel within a consonant). e.g.

अङ्क=अंक	पङ्ख=पंख	गङ्गा=गंगा	सङ्घ=संघ
मञ्च=मंच	पञ्छी=पंछी	पञ्जा=पंजा	साञ्झ=सांझ
घण्टा=घंटा	कण्ठ=कंठ	झण्डा=झंडा	ढूण्ढ़=ढूंढ़
सन्त =संत	पन्थ=पंथ	बन्द=बंद	गन्ध=गंध
चम्पा =चंपा	गुम्फ=गुंफ	खम्बा=खंबा	स्तम्भ=स्तंभ

Table 2.3: Some examples

अँक, पाँच, हैं, मैं .

In devanagari script it often get substituted with Anuswar, as latter is more convenient for writing.

- **Visarg** (ः) comes after a vowel sound and represents a sound similar to ह.
- **Nukta and Halant** The nukta (.) is used along with some consonants (like क़, ख़, ग़, ज़, ड़, ढ़, फ़) and is mostly used to represent some foreign sound.

  A special sign Halant (्) is needed to indicate that the consonant does not have the implicit अ vowel in it.
- All **punctuation marks** used in Indian scripts are borrowed from English except for the full-stop, instead of which viram is used.

2.2.2 ISCII standard

Since the 70s, different committees of the Department of Official Languages and the DOE (Department of Electronics) have been evolving different code and keyboard, which could cater to all the Indian scripts due to their common phonetic structure. In 1980s the ISCII code (Indian Script Code for Information Interchange) was recommended, and it is widely used for internal representation of Indian scripts.

- ISCII character set
  ISCII character set [4] is a super-set of all the characters required in ten Brahmi-based Indian scripts. For convenience, the alphabet of the official Devanagari

has been used in the standard. The ISCII code contains only the basic alphabet required by the Indian scripts, and all the composite characters are formed by the combination of these basic characters.

ISCII code has the advantage that there is only one way of typing a word. The spelling of a word is the phonetic order of the constituent basic characters. This provides a unique spelling for each word, which is not affected by the display rendition.

- Eight-bit ISCII code
  In this section ISSCII-8 [4] (Indian Script Standard Code for Information interchange) as standardized by DOE in 1986, is reviewed. The lower 128 characters of the 8-bit table contain the ASCII character set, while upper half of the table is used for Indian script code. This coding scheme allows Roman characters to be freely mixed with Indian scripts.

- Seven-bit ISCII code
  Seven-bit coding scheme is recommended for those computers and packages which do not allow the use of 8-bit codes. In 7-bit coding 128 positions are available for representing all the characters of the script. This coding has the disadvantage that Roman scripts cannot be mixed with Indian scripts.

2.2.3 Other Standard

Another popular representation, published by NCST (National Centre for Software Technology), is pure consonant based coding. In this representation the consonants are always in their pure form i.e. with halant. Vowels when added to consonants results in the corresponding matra symbol on the consonant. The coding table is 7-bit table where some of the ASCII codes are replaced by the Indian script characters. This coding facilitates automatic alphabetization in perfect order of Devanagari. Also it does not disturb basic ASCII codes of most of the signs which are common in Devanagari and Latin.

2.2.4 Why ISCII ?

For the implementation of our **search engine for Indian languages**, we are using **eight-bit ISCII encoding** because of the following reasons :

1. ISCII has been designed as per ISO norms and has found acceptance in Unicode[13].
2. ISCII codes allow a complete delinking of the codes from the displayed fonts[4].
3. The **eight-bit ISCII code** retains the standard ASCII code. This makes it feasible to use Indian script along with existing Engilsh computers and software, so long as 8-bit character codes are allowed [4].

2.3 Morphological Analyzer

Since we are using a morphological analyzer in our search engine, it is necessary to have a brief look at it. Morphology is concerned with the internal make-up of words. Morphological analyzer analyzes words grammatically. It determines class (i.e noun, adjective, verb etc.) of the words and extract the root words.

Morphological theory can be divided into two categories: inflectional morphology and derivational morphology.

- **Inflectional morphology** is concerned with the manner in which the lexical stem(root word or base word) is combined with grammatical markers (like े, ो, याँ, etc in hindi and s, es, etc. in english) for things like plurality and tense.

  e.g. words लड़के and लड़कों are made from the root word लड़का by combining matras े and ो respectively. Morphological Analyzers which support inflectional morphology gives the root word of the input word by removing any grammatical markers if present. For example, if word नदियाँ is given as input to such a morphological analyzer, then it will return the root word नदी.

- Where inflectional morphology is concerned with the combination of stems with grammatical markers, **derivational morphology** deals the construction of stem themselves. Typical cases of derivational morphology involves the class changing suffixes which form adjectives from nouns, verb from nouns, noun from verbs and so on, as the following examples illustrate.

मूर्ख (noun)	—>	मूर्खता (adjective)
दूध (noun)	—>	दूधवाला (adjective)
धर्म (noun)	—>	धार्मिक (adjective)

Table 2.4: Examples of derivational morphology

The resulting lexical stem are again subject to appropriate inflectional morphology, so we get form such as दूधवाले from दूधवाला .

The morphological analyzer[8], we are using in our search engine, supports only inflectional form of morphology.

Chapter 3

Design and Implementation

This chapter discusses the design and implementation of our search engine for Indian languages. In the implementation of our search engine, we are using the Gatherer of an existing search engine **htdig**[6]. Moreover, some other libraries like basic data structure library, Common Gateway Interface library, and user interface library of the htdig are also being used in our search engine. These libraries are not discussed here. Instead the main stress is laid on the implementation of the features which are important for the search engines, in the context of Indian languages.

3.1 Design Issues

In designing information retrieval engines it is very important to consider the human factors - how people search, how they make decisions. Normally, users specify some keywords of a subject for searching the documents. Sometimes they want to restrict the search criteria by specifying some boolean operators (like AND, NOT etc.) with the keywords and sometimes they want more number of matches. Features of the language for which the search engine is being made, also plays a major role in search engine design. Following are some Indian languages specific features which should be present in a search engine of any Indian language :

- Different forms of the words
  Almost in every language, grammatical markers (like s, es, ing, etc. in English and े, ो, याँ in Hindi) are used with a word stem (or root word) and new words

are made. These new words, called morphological variants of the stem, present the same concept but just differ in tense, plurality etc. For example लड़के, लड़कों are morphological variant of root word लड़का.

It may be possible that the document contains a root word, say लड़का and the user gives any morphological variant of that root word, say लड़के for searching. The search engine should be able to handle such conditions and should give the matches of all the morphological variants of the word.

- Phonetic Tolerance
  As we saw in section 2.2, Indian languages alphabet contains many characters (◌ं, ◌ं, ◌ॅ etc.) which give same sound i.e they are phonetically equivalent. These characters are used interchangeably in many modern Indian languages. e.g. झंडा, झण्डा, झन्डा and झँडा are phonetically equivalent words. Section 2.2 has provided the detailed description of all characters which are phonetically equivalent.

  User may use any phonetically equivalent word of a keyword for the searching. So the search engine should be able to support some form of phonetic tolerance. If the user gives word झन्डा for searching, then the search engine should give matches for all its phoneme equivalent words झंडा, झण्डा, झँडा.

- Font Independence
  Due to lack of standardization in display technologies, Indian language documents on the Web are being written in different fonts. Each web author uses their own font encoding in the HTML documents. There is no universally accepted font or display encoding. Therefore a search engine for Indian languages should be able to incorporate different font encodings. It should be independent of the font encoding used in the documents.

- Boolean operators
  User may want to search more than one keyword at a time. He may want to combine the keywords according to some boolean expression, to restrict or generalize the search. Therefore a good search engine should allow the use of boolean operators(like AND, OR, NOT, etc.)

- User Controllable search
  Sometimes people may want the match on all the phoneme equivalents of a

keyword and sometimes they may want an exact match. A good search engine should provide the facility to control the searching criteria to the search engine user.

3.2 General Design of our Search Engine

Our search engine is primarily designed to index and search Indian language documents available on the Web. However, it can index and search English documents simultaneously. Its design is language independent. Only one module is language specific, and that is the morphological analyzer. We are using a morphological analyzer of Hindi language in the implementation. This Hindi morphological analyzer can easily be replaced by the morphological analyzer of any other Indian language. With this replacement, our search engine can work for any other Indian language.

At the top level, our search engine can be divided into three parts : gatherer, indexer, and search processor. Gatherer collects the documents from the Web and passes them to the indexer. The main task of the indexer is to build an index of the documents submitted by the gatherer. During indexing, indexer parses HTML documents, collects information about each word in the document, generates root words, removes stop words, and at last builds index on the keywords and its root words (if any) using the collected information. This index is used by the search processor for the searching of the documents.

The detail design is shown in figure 3.1. We now discuss each part in detail.

3.3 Gatherer

Gatherer works as a regular Web user, sends HTTP request to the remote server for collecting the documents. It starts its journey of the Web from the starting urls specified in the configuration file of htdig . It follows all hyper links that it comes across. Since we are using the gatherer of htdig in our search engine, we are not discussing it anymore.

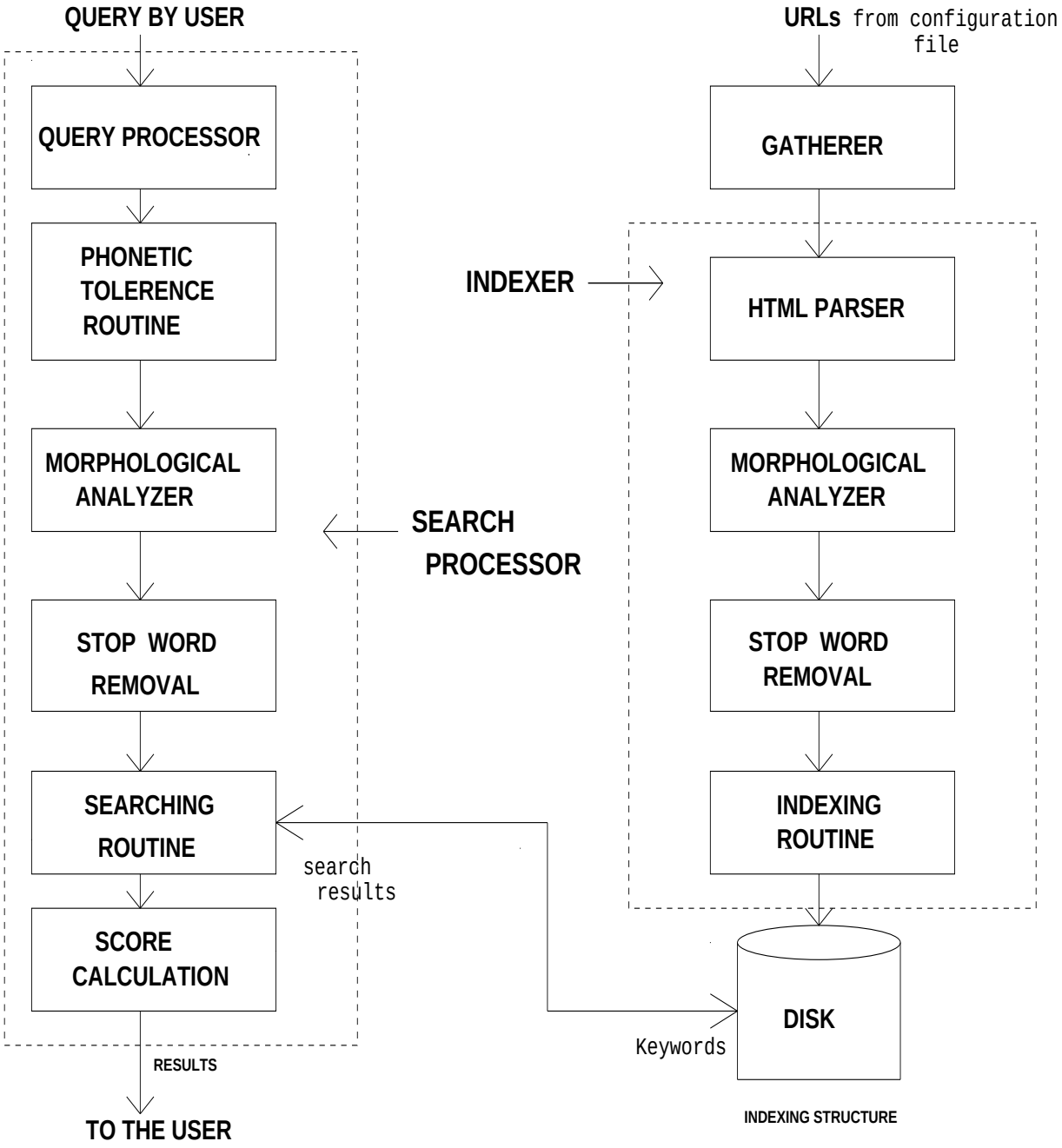


Figure 3.1: The Design of our Search engine

3.4 Indexer

Everything the gatherer finds goes into the indexer. Indexer of our search engine works in two phases. First phase runs with the gatherer. It takes documents from the gatherer, parses those documents, and collects the information about each word of the document. Root word generation and stop word removal are also performed in the first phase. The information collected in the first phase is stored locally for the use of second phase. The first phase creates two files. The first one is the list of all words and the second one is database of URLs. The main task of the second phase of the indexer is to build the indexing structure.

3.4.1 Document Parsing and Weight calculation

Document parsing involves the separation of text from the HTML tags. Text of Hindi and English languages is also separated by the parser.

Words occur with different HTML tags in the HTML documents. We are assuming that the words coming in headings, title, and keyword tag represent the subject of the document. So the words occurring with these tags are assigned a higher weighting factor. In this way, the weighting factor tells about the HTML tag in which the word has occurred. After determining the weighting factor the next task is to determine the language of the word. In this process, program checks the parameters of the <FONT FACE> tag. If word is surrounded by a <FONT FACE> tag with font name as some Hindi font ('xdvng' in our case), then it means the word belongs to Hindi language otherwise some other language. If the word belongs to Hindi, it is converted into ISCII encoding.

Now the weight of the word is calculated using following formula :

$$weight = (1000 - location) * weighting\_factor$$

where $location = offset * 1000/content\_lentgth$, and
weighting_factor is weight of the tag in which the word is coming,
content_length is the size of the document and
offset is the offset of that word in the document.

The sum of the weight of each occurrence of the word gives the weight of the word in that particular document. Weight, document id, and word frequency are the fields of the data record which have to be stored in database with the word as indexing key.

This word is passed to the morphological analyzer for further processing.

3.4.2 Morphological Analyzer

The primary task of morphological analyzer is to return the root word of a given input word. This is necessary to search the different forms of a word. For example, if the word लड़के appears in the document then its root word लड़का and original word लड़के will be stored in the database. Now, if user gives word लड़कों for searching, then its root word लड़का will also be searched in the index, which is already available in the database. In this fashion, different forms of the words are supported by the search engine.

After generation of the root word from the given input word, the stop word list is checked for these words. If any of the words exist in the stop word list, both these words (root word and original word) are not included in the database, and the next word is extracted from the document. The morphological analyzer used in our search engine does input/output in wx-keyboard scheme (see Appendix), therefore each word is converted from ISCII to wx-keyboard scheme before passing to morphological analyzer, and returned root word is converted back into ISCII from wx-keyboard scheme.

The first phase ends with the creation of two intermediate files. The first file stores the list of all words and second file stores the database of URLs.

3.4.3 Indexing Data Structure

The second phase of the indexer creates a URL database and word database from the intermediate files that were created in the first phase. This word database is accessed via an index of the keywords. The index file is organized as a variant of basic tries[9] referred as compressed tries[5].

A basic trie is an m-ary tree (m is the size of alphabet set) in which the root node points to another node for each of the possible alphabets a word may have. Each of these nodes, likewise, contain a pointer to a node for each possible second alphabet and so forth. The basic trie is space inefficient, therefore we are using compressed tries, which is discussed below.

Compressed Tries

The storage model of the word database is developed in two levels. At the topmost level we have indexing structure of compressed trie and at the lower level we have the database which contains record for each word (see figure 3.2). Search processor first need to traverse the indexing structure to find the required record in the database.

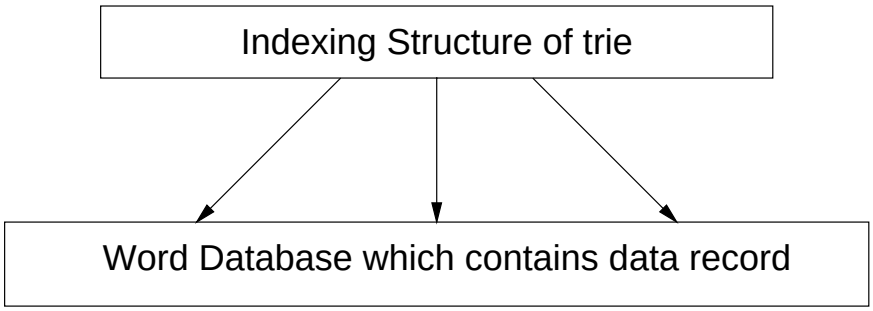


Figure 3.2: Organization of Word Database

We are using a two level structure at each branching level in the indexing structure. The first level is known as a *LEVEL A NODE* and leads to a *LINKED LIST* of *LEVEL B NODES* at the second level. At the leaf level we have *DATA RECORD* stored in the word database. One other structure *SKIP ARRAYS* also exists in the database.

The A level node is essentially an array of elements, which correspond to character ranges. These array elements contain pointer pointers to linked lists of level B nodes. Search through a level A node occurs by mapping the current character to one of the elements in the level A node and following the pointer in it to the corresponding linked list of level B nodes. Figure 3.3 illustrate a level A node, which has six elements and has the character ranges as shown. Choice of the character range and the number of elements in the level A node depends on the application. Distribution of the character ranges should be done such that the linked lists of level B nodes have approximately equal length once the record have been inserted. This will ensure that the difference of maximum and minimum search time is not high. Figure 3.4 shows the structure of level A node in our implementation. In the figure character ranges are written in the form of 8-bit ISCII code.

The B level node is the node that distinguishes the path for one attribute from another in the same range of the A level node. These nodes are created on demand

and maintained as lexicographically ordered linked lists. Level B nodes lead to other A level nodes or to Data records.

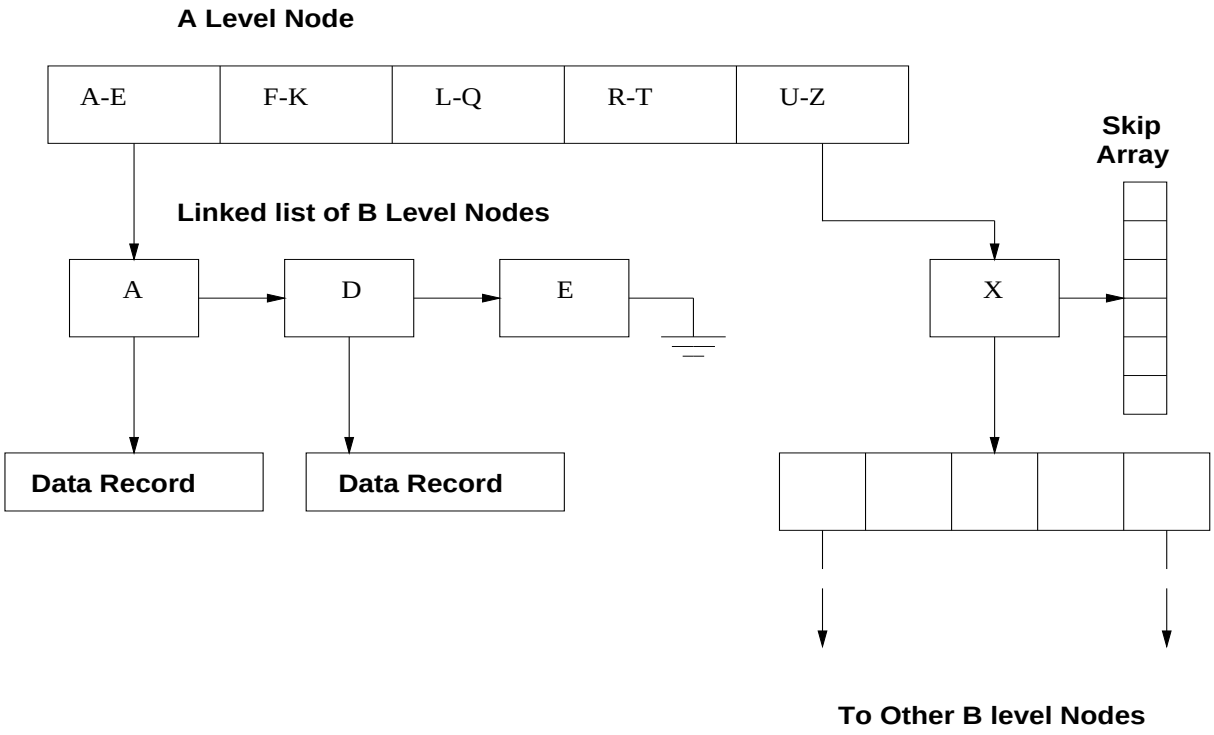


Figure 3.3: Generalized View of the Data Structure Involved in the Tries

0-127	128-179	180-195	196-207	208-214	215-255

Figure 3.4: Character range distribution in A level Node of our implementation

Skip arrays are structures that are being used to minimize the space utilization by LEVEL A and LEVEL B nodes (figure 3.5). The common part of the key is kept in a skip array, which occupy much lesser space as compared to the Level A and Level B nodes. For example, if a database contains only two records with the indexing key beginning with 'M' and the keys are Mario and Mariah, then it is evident that the path for both the records is decided by the last letter in Mario i.e. 'o' and by the second last in Mariah i.e. 'a'. It would be unwise to have the two level node structure for the letters 'a', 'r', and 'i'. Thus, the substring "ari"is kept in a Skip array structure.

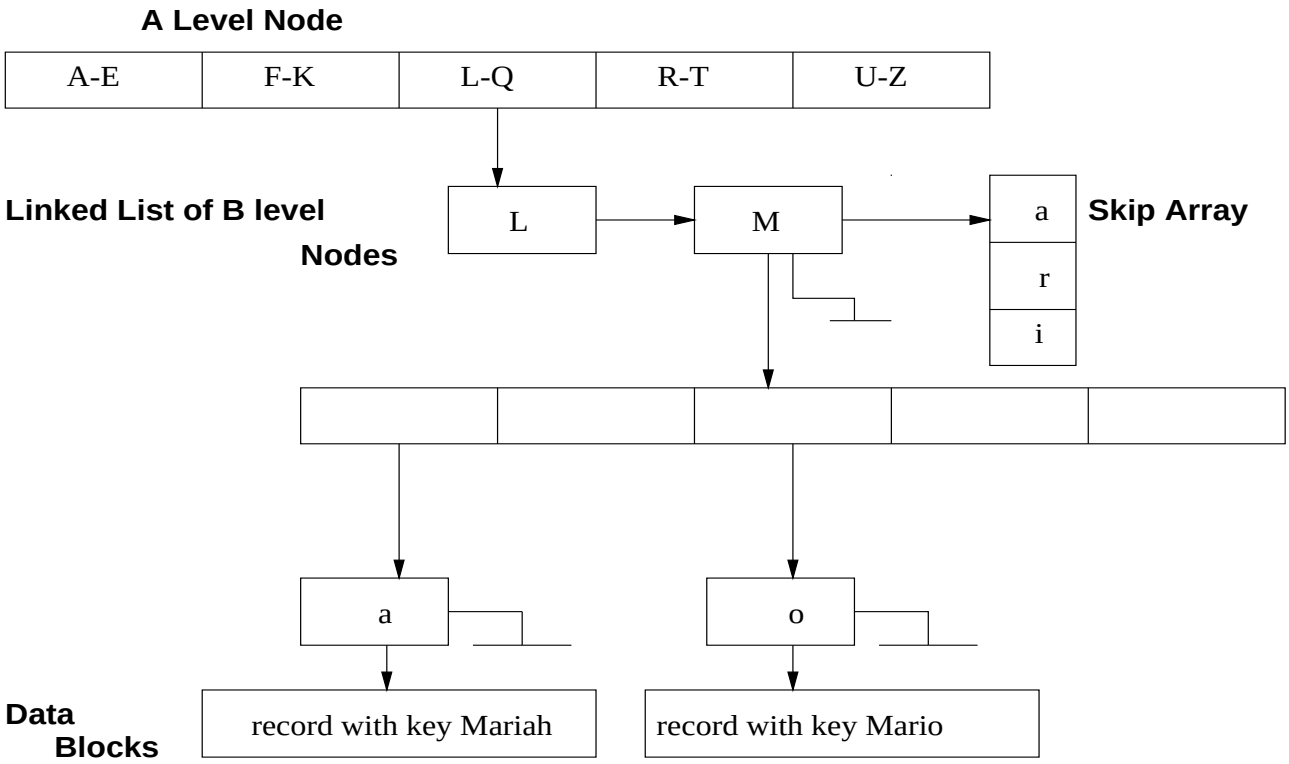


Figure 3.5: Compressed Tries with skip array

- *Why Tries?*

Following are some reasons for the selection of compressed trie.

- Trie is most suitable for retrieving all the words which have common prefix.
- To retrieve a data record, the number of comparisons does not depend on the number of keys indexed. Instead it depends on the length of the key.
- An insertion into tries is localized and does not propagate to higher levels in the indexing structure. Insertion only causes the expansion of the trie structure.

3.5 Search Processor

Search processor is the actual search engine of the search system. Here it is implementation as a CGI program which is expected to be invoked by an HTML form. It

will accept both GET and POST methods[17] of passing data to the CGI program.

HTML form is the primary interface of the search processor. When the form is submitted, the search program will take values from the form and perform the actual search in the indexing structure. The HTML form contains an input text field where user will enter the search words. This form has some other option for controlling the search criteria and for formatting of the results. The Searching process is divided into the following four modules, which execute in the given sequence.

3.5.1 Query Processor

Search processor gets a list of words from the HTML form that invoked it. Query processor takes this list and if the searching program was invoked with the boolean expression parsing enable, then it does syntax check on that list. If there are syntax errors, it displays the syntax error. If the boolean parser was not enabled, the list of words is converted into a boolean expression by putting either "and"s or "or"s between the words. This depends on the search type specified by the user.

In both cases each of the words in the list is converted into ISCII encoding, if it belongs to the Hindi language. The language of the words is specified in the HTML form by the user. Now, the query processor passes each of these words to the phonetic tolerance routine.

3.5.2 Phonetic Tolerance Routines

The phonetic tolerance routine expand the list of words by generating their phoneme equivalent words. If the user specified 'the exact match' search criteria then no additions are made by this routine. This routine is divided into two parts. First part involves the generation of the new words in which only the nasalized characters (ँ, ं, ॅ, etc. see Table 2.3) are replaced by their phoneme equivalent characters. The second part uses phoneme rules specified in the phoneme rules file. This file contains the pairs of other phonetically equivalent characters of Indian language alphabet, and an integer value called matching percentage. For example,

[GA] -> [GA] [NUKTA] -> 70

indicates that if ग occurs in a word then generate a new word in which ग will be replaced with ग़. And if this new word is later found in the index, its final weight

will be 70% of the weight value stored in the data record. This file can be changed according to the requirements of various Indian languages.

3.5.3 Morphological Analyzer

The words generated in the last step are passed to the morphological analyzer for generation of the root words. Since, among all the phoneme equivalent words, only one word is grammatically valid, so the root word will be returned only for that valid word. Morphological analyzer ignores other words. Here a grammatically valid word means the word which is present in the dictionary. e.g. अंको , अँको, and अन्को are phonetically equivalent words but word अंको is grammatically valid. So morphological analyzer recognizes only this word and returns root word (अंक) for it.

3.5.4 Stop Word Removal

Before searching the words in the indexing structure, it would be better if we remove all the stop words from the list. This improves the searching time as non-relevant words are not searched.

3.5.5 Searching in Database

Now each word of the list is searched in the indexing structure. Search begins at the root. The current character say keyword[i] is mapped to a field in the A level node. The search proceeds by following the pointer to the linked list of B level nodes associated with it. If a B level node exists in the list with the same keyword as keyword[i], then the search continues by following the pointer to the node at the next level associated with the B level node, else the search stops and returns failure. If a valid pointer to an A level is found then the search continues by repeating the above steps. This is repeated each time an A level node is reached by the search. If a data record is reached, then it is stored in the 'result list', and search continues for the next keyword. After finishing the searching of all the keywords, the 'result list' is used for result ranking and display.

3.5.6 Results Ranking and Display

Before ranking, the results are combined according to the boolean expression of the query. Now the results are ranked. The rank of the match is determined by the weight of the word that caused match. The document in which the keyword has higher weight, is given higher rank. A Word's weight is determined by the importance of the word (as explained in the section 2.1.2) and by the phoneme rules (see section 3.5.2). For example words in the title of a document have a much higher weight than words at the bottom of the documents.

Finally when the document ranks have been determined, the resulting matches are displayed in the order of ranks.

3.6 User Interface

The user Interface of the search engine is made up of an HTML from, through which user gives keywords for searching. Users are expected to enter Indian language (Hindi in current implementation) keywords in wx-keyboard encoding scheme. A list of transcription rules for Hindi language which uses this encoding scheme has been provided in the appendix. A hyper link to these transcription rules has also been provided in the HTML form for the convenience of the user.

Chapter 4

Conclusion and Future Work

4.1 Experiments

This software has been tested on a database of 780 Hindi documents spanning 14 MB of disk space. We found that the index size is 113% of the total documents size. It is reasonable because this index allows fast searching of the database. Some snapshots of the user and search engine interaction are given in the following pages. wx-keyboard scheme has been used for input and output of Hindi language, while ASCII code has been used for entry and display of English text.

Figure 4.6 shows the search results of all the morphological variants of the words प्यारे, दिलों and मन्दिर. This task is accomplished by searching the root words प्यारा, दिल and मन्दिर. This figure also depicts the use of boolean expression in the query. Figure 4.7, displays the matches for all the phonetically equivalent words. Fig 4.8 shows the use of English words in the search engine.

4.1.1 Recall and Precision of our Search Engine

In order to measure the recall and precision of our search engine, following tests were conducted.

- We indexed 34 Hindi documents, out of which 21 documents were relevant to a keyword आँखें and 11 documents were relevant to a query (चांद or चांदनी)[14]. When the keyword 'आँखें is given for searching, our search engine retrieved 18

documents, out of which 17 documents were relevant to that keyword, i.e. 80% recall and 94% precision. Our search engine retrieved 9 documents for the query (चांद or चांदनी), out of which 8 documents were relevant to this query, i.e. 72% recall and 89% precision.

- In the second test, total indexed documents were 106, out of which 21 documents were relevant to the keyword आँखें and 13 documents were relevant to the query (चांद or चांदनी). Search engine retrieved 22 documents for the keyword आँखें, out of which 17 documents were relevant. This gave 80% recall and 77% precision. For the query (चांद or चांदनी), our search engine retrieved 19 documents, out of which 11 documents were relevant, i.e. 80% recall and 57% precision.

Our search engine missed some relevant documents because it doesn't support synonym search.

4.2 Features Summary

The salient features of our search engine are as follows:

- It gives the match on all the morphological variants of the given input word.
- It supports phonetic tolerance.
- The search engine is independent to the font encodings. It can be easily configured for different font encodings.
- Users can control the search criteria.
- Searching of English documents is also supported by our search engine.

4.3 Future Work

- Our search engine does not support 'Sandhi'. For example, it cannot give match for the word राम्, which is coming in the composite word रामेश्वर. This feature can be incorporated in the search engine later. This will require the replacement of the morphological analyzer with a new one which supports 'Sandhi' and derivational kind of morphology.

- In our search engine users can not use both Hindi and English words for searching, simultaneously. Search engine can be extended to provide this feature.
- The user interface of our search engine can be modified to provide input and output in Devanagari script.
- Our search engine does not support synonym search. It can be extended to support this feature thus allowing the searching of original words as well as their synonyms.

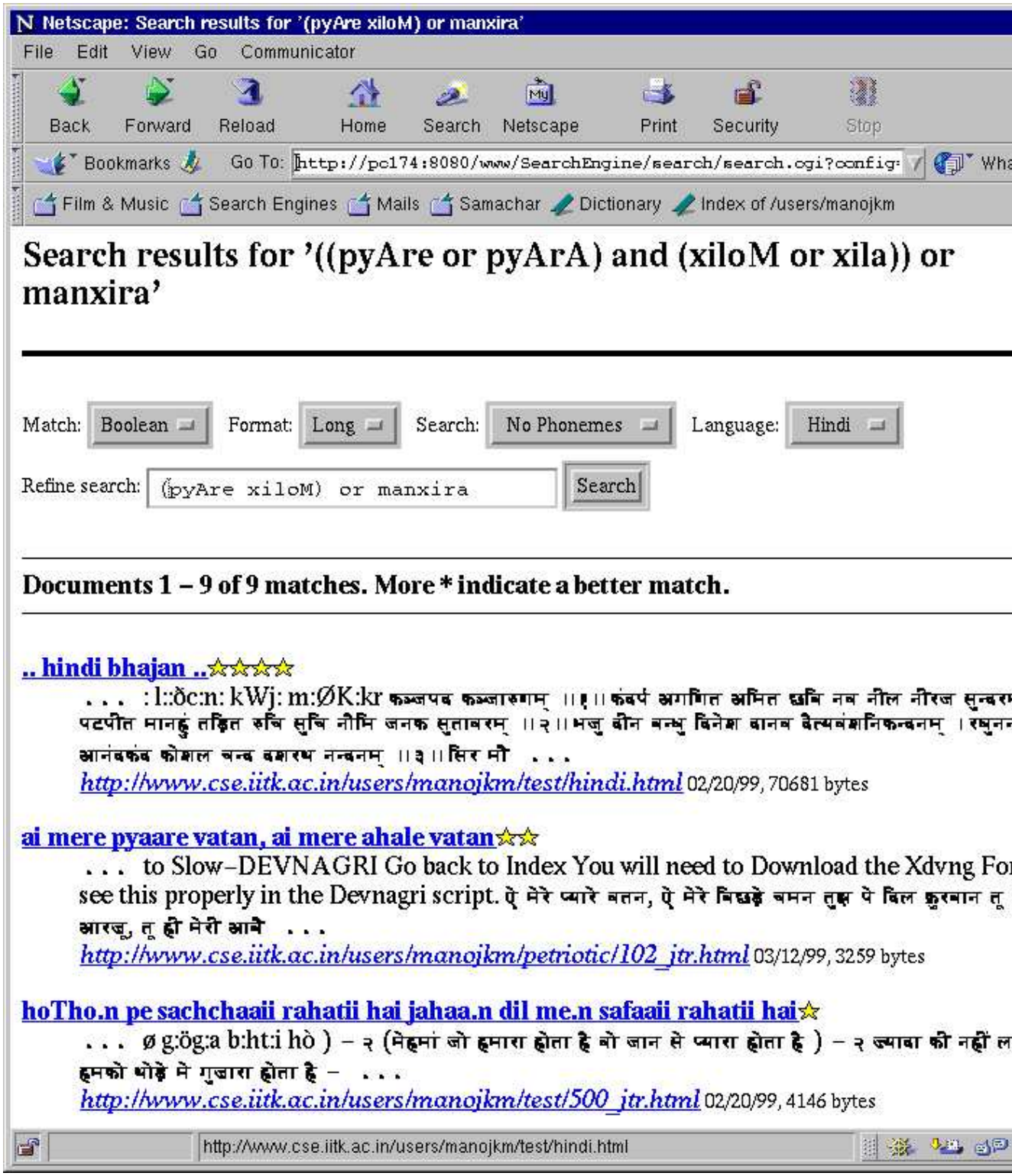


Figure 4.6: Example showing the searching of different forms of the words

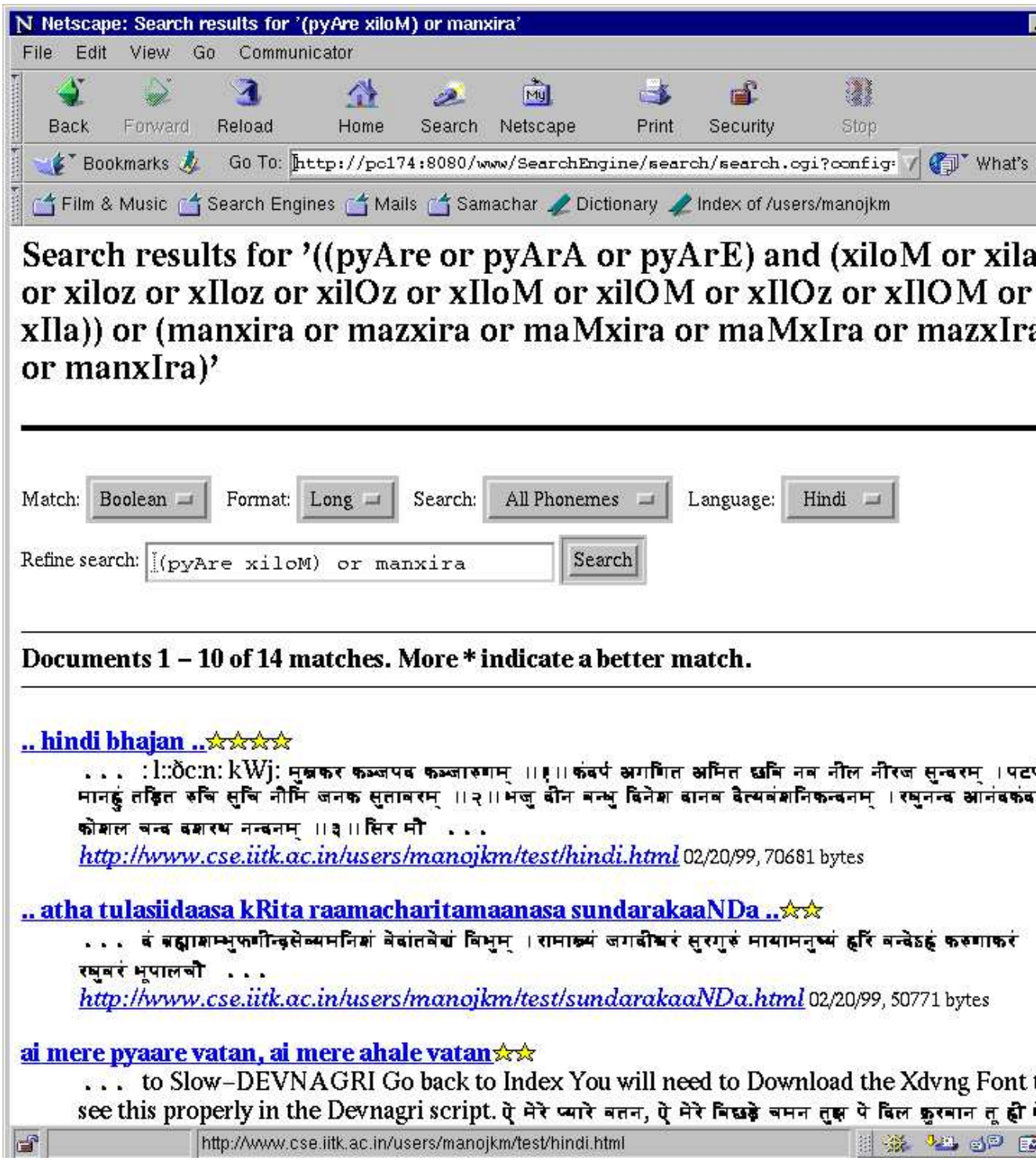


Figure 4.7: Example showing the searching of phonetically equivalent words

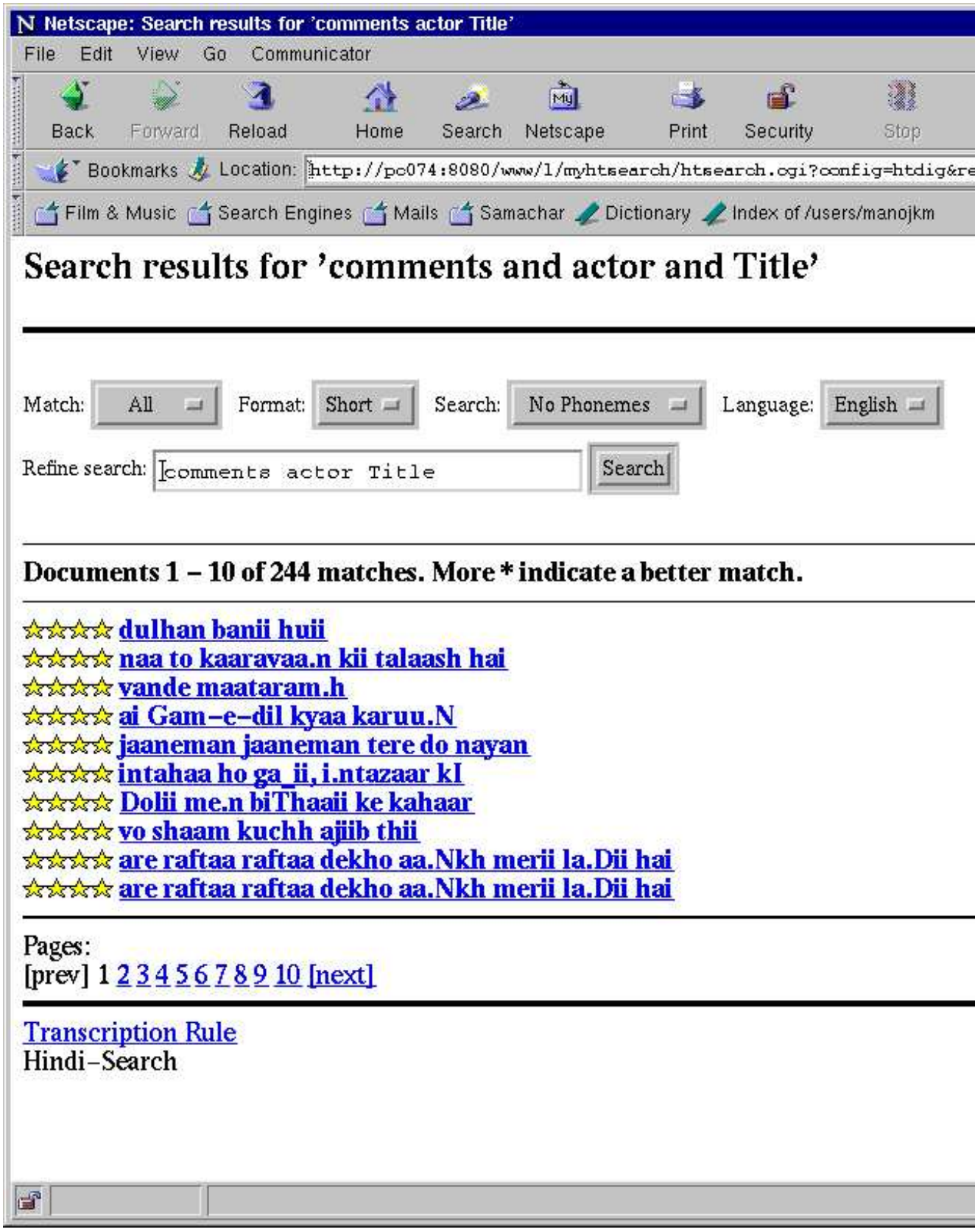


Figure 4.8: Example showing the use of English words

Appendix A

User Manual

The search engine implemented in this thesis is a complete world wide indexing and searching system which support indexing and searching of any Indian language. It is made as three executable modules. These are :

- Gatherer
- Index Builder and
- Search Processor

The gatherer and index builder are executed by the search engine administrator on periodic basis. The interface of both of these program is command line driven. The search processor is invoked by an HTML form. The details of each module is listed in the following paragraphs.

A.1 Gatherer

Command to run gatherer :

gather [options]

Options

-h Restrict the gatherer to documents that are at most *maxhops* links away from the starting documents. this only works if **-i** is also given.

-i Initial. Do not use any old database.

Files

${CONFIG_DIR}/htdig.conf - The htdig configuration file.
${DATABASE_DIR}/db.wordlist - The intermediate word database file created by gatherer.
${DATABASE_DIR}/db.docdb - The intermediate documents url database file created by gatherer.
${CONFIG_DIR}/config/ - Directory containing the language and font related information.

A.2 Index Builder

Command to run index builder :

build

Files

${CONFIG_DIR}/htdig.conf - The htdig configuration file.
${DATABASE_DIR}/db.wordlist - The intermediate word database file created by gatherer.
${DATABASE_DIR}/db.docdb - The intermediate documents url database file created by gatherer.
${DATABASE_DIR}/trie/ - The Directory containing the indexing structure and the word database.
${DATABASE_DIR}/db.docs.index - The documents url index.

A.3 Search Processor

It is a CGI program (named 'search') that is invoked by an HTML form.

Inputs

Match Type of search.

- **All** Default boolean operator is "and".
- **Any** Default boolean operator is "or".
- **Boolean** Use boolean expression given in query.

Format Result Format

- **Long** Detailed display of results.
- **Short** Brief display of results.

Search The Search criteria.

- **No Phonemes** Give matches only for words and there different forms.
- **Valid Phonemes** Give matches for words, there different forms and the phoneme equivalents in which only nasalized characters are replaced with their equivalents.
- **All Phonemes** Give matches for all phoneme equivalent words.

Language Language of the keywords.

- **Hindi** Language of the keywords is Hindi.
- **English** Language of the keywords is English.

Files

${CONFIG_DIR}/htdig.conf - The htdig configuration file.
${DATABASE_DIR}/trie/ - The Directory containing the indexing structure and the word database.
${DATABASE_DIR}/db.docs.index - The documents url index.

${DATABASE_DIR}/db.docdb - The intermediate documents url database file created by gatherer.
${CONFIG_DIR}/config/ - Directory containing the language and font related information.

A.4 Configuration Files

Configuration Files are divided into two categories. The files of first category specify the attributes for a particular environment. This category has only one file ${CONFIG_DIR}/htdig.conf. The languages and fonts related configuration files come in second category.

A.4.1 ${CONFIG_DIR}/htdig.conf

This is the main runtime configuration file for all programs that make up search engine. Each line in this file is either a comment or contains an attribute. Comment line are blank lines or lines that start with a '#'. Attributes consist of a variable name and an associated value:

`<name>:<whitespace><value><newline>`

The name contains any alphanumeric character or underline (_) The <value> can include any character except newline. It is possible to split the <value> across several lines of the configuration file by ending each line with a backslash (\).

If a program needs a particular attribute and it is not in the configuration file, it will use the default value which is defined in `htcommon/default.cc`. Following are some important attributes specified in `htdig.conf` file.

- **DATABASE_DIR**
  This is the directory which contains all database and other intermediate data files.

- **START_URL**
  This is the list of URLs that will be used to start a dig when there was no existing database. Note that multiple URLs can be given here.

- **EXCLUDE_URL**
  The urls which contains any of the pattern specified with this attribute will not be retrieved by the gatherer. This is used to exclude common things such as an infinite virtual web-tree which start with cgi-bin. e.g. exclude_urls : cgi-bin .cgi

- **LIMIT_URLS_TO**
  This attribute limits the scope of the indexing process. The value for this attribute is just a list of string patterns. As long as URLs contain at least one of the patterns it will be seen as part of the scope of the index. e.g LIMIT_URLS_TO : ${START_URL}

- **HTTP_PROXY**
  The URL specified in this attribute points to the host and port where the proxy server resides. e.g. HTTP_PROXY : http://202.54.56.146:3128

A.4.2 Languages and fonts related configuration files

The formats of these files are taken from the configuration files of Iterm[7]. There is a main configuration file or the specification file, whose name is `specs`. It is present in `${CONFIG_DIR}/config` directory. Besides specification file, there are other file which contains information about ISCII encoding, fonts, syllable rules of the language, phoneme rules, and wx-keyboard encoding. These files are listed in specification file. These files are used for the purpose of translation of ISCII code to font code and vice-versa.

All the files have a common format. Blank, newline, and tabs are ignored. Comments can be present between "/*" and "*/". Every string should be followed by a blank and all the strings except where specified can have maximum of 15 characters. Rest of the string is ignored. % is a delimiter which should be present before starting of each new information. : and "-> are used as delimiters.

Various symbols used to explain the format of files are:

- <> indicates that the value conforming to the description in these brackets be specified. The value may be in form of string,characters or decimal values.

- {} means that the value can occur 0 or more number of times.

- [ ] indicates that it is optional.

Specification file format

Various files containing the ISCII coding details, fonts, type, syllable rules, phoneme rules, and wx-keyword scheme are specified in this file. All the entries are in form of string of characters. The maximum number of characters present in the font name and file could be upto 100. Absolute or relative filename may be specified. If relative filename is specified, then path prefix from `specs` file is prepended to the file name. The absolute filename begins with \.

```
%<Default Language>

%<Language name> : <normal font name> <bold font name> :
                   <file which contains ISCII coding detail>
                   <file containing the font details>
                   <file which gives the type map>
                   <file containing the syllable rules>
                   <file containing phoneme rules>
                   <file containing the wx-keyword encoding>
```

%Devanagari
%Devanagari :dvng10 dvng10 :iscii font type rule phonemes morph

Table A.5: Syntax - specification file

ISCII Coding scheme file format

The coding scheme file contains 7 and 8 bit details for Indian scripts. All the characters in the set are given some descriptive names. This name is used refer to the character in all other files. For example:

Corresponding to each character its equivalent 7-bit and 8-bit coding is provided. For each character there can be only 8 bit code, while in 7-bit coding maximum of 5 codes can form one character.

Descriptive Name	Character
%Chandrabindu	ँ
%Visarg	ः
%Aa	आ
%I	इ
%Ka	क
%Kha	ख

Table A.6: Assigning descriptive names to characters

```
%[<Esc Character >]

{%<string description for characters>   ->
 <8 bit coding in decimal>             ->
{<7 bit coding in form of character>}    }
```

%x		/⋆ Escape Character (7-bit) ⋆/
%Chandrabindu	$->$ 161	$->$ A
%Visarg	$->$ 163	$->$ B x
%Aa	$->$ 165	$->$ C k
%I	$->$ 166	$->$ C l
%Ka	$->$ 179	$->$ D
%Kha	$->$ 180	$->$ E

Table A.7: Syntax - code file

Decimal value is to be specified for 8 bit, however, 7-bit code are specified in form of characters.

Font map file format

Font table and font characters used for determining the number of characters per row are specified in font map file. Also list of characters to be moved to the beginning or to the end of the syllable are present in font map file.

Fonts are basically categorized into various user defined types. Each category contains several mappings. To convert an ISCII code into font code, font table is searched for matching entries. "Conjuncts" is a reserved keyword and all mappings

```
%{ <font characters to be moved to the beginning> }
%{ <font characters to be moved to the end> }

%<font character to be considered as base char - for 7 bit coding>
%<font character to be considered as base char - for 8 bit coding>

{ %<type name>
{ %{<string description for characters>} -> { <font coding in decimal>} } }
```

% 69		/⋆ Characters to be moved to the beginning ⋆/		
% 13		/⋆ Characters to be moved to the end ⋆/		
% 107		/⋆ Base character for 7 bit code ⋆/		
% 107		/⋆ Base character for 8 bit code ⋆/		
% Conjunct				
%Ka	Halant	Hard-Sha	− > 35	
%Ja	Halant	Jna	− > 43	
%Vowel				
% A			− > 97	
% Aa			− > 97	65
%Consonant				
% Ka			− > 107	
% Kha			− > 75	
%Half-Consonant				
% Ka	Halant		− > 63	
% Kha	Halant		− > 72	
%Matra				
% Matra-Aa			− > 65	
% Matra-i			− > 69	
%Reph				
% Ra	Halant		− > 13	

Table A.8: Syntax - font file

specified under conjunct are first searched for. The user may edit the present font file to add more *conjuncts*. The categories defined here are used in listing all the combination rules, details of which are present in rules section.

Type map file format

Word written in any Indian script is composed of syllables. Syllables are a sequence of characters combined according to some rules. The user can provide the rules for finding valid syllables.

To specify the rules first of all a type map is to be provided. This map categorized the character set. For example the set may be categorized into *vowels*, *consonants*, *matras*, etc. A particular character not included in this file is assigned the default type specified by the user. Maximum of 50 different types can be present. "Begin" and "End" are reserved keywords and cannot be used. Refer to Table A.9 for syntax of type_map file.

Syllable rules file format

This file contains the syllable rules which lists the valid syllables. It also contains the combination rules, which specifies the mapping between input ISCII characters and output font codes. The combination rules follow the syllable rules. "Begin" and "End" are reserved keywords and are used to denote the beginning and end of syllable or word.

- **Syllable rules**
  Now using the categories in type map file syllable rules can be specified. There are a number of rules each having some name. User can specify all combination of categories representing valid syllables.

  Every type indicates one character of that type in the character set. If there can be combination of same type, then the type has to be written down required number of times. Some rules are very complicated. To specify these rules, the total combination specifying a syllable can be split in many rules. The user can specify that a particular rule does not indicate a valid syllable but it has to combine with some other rules to form a complete syllable. To specify this he

```
 %<default type>
{%<type>        ->  <string description for character> }
```

%Invalid	
%A	– > Vowel
%Aa	– > Vowel
%I	– > Vowel
%Ii	– > Vowel
%Ka	– > Consonant
%Kha	– > Consonant
%Ga	– > Consonant
%Gha	– > Consonant
%Ra	– > Cons-r
%Chandrabindu	– > Modifier
%Ansuwar	– > Modifier
%Visarg	– > Modifier
%Matra-Aa	– > Matra
%Matra-I	– > Matra
%Matra-U	– > Matra
%Halant	– > Halant

Table A.9: Syntax - type map file

state what all rules can follow this rule.

For example:
R2:Type_consonant Type_Halant -> R3 R4
R3:END
R4:Type_Consonant
If there are no rules present after -> then it indicates that the combinations present in that rule signifies a valid syllable.

- **Combination Rules**

  The input string of characters are combine in some way to form display symbols of certain category in the font table. These can be specified in form of combination rules. These rules basically states that these combination of character from various categories (specified in type map file) in the input string will generate the font codes belonging to certain categories (specified in font map file). This helps in determining the location in the font table where characters

{%<rule number>: {<types>} -> {<rule number>} }

%R0	:Vowel Modifier	– >
%R1	:Vowel	– >
%R2	:Consonant Halant	– > R3
%R3	:Consonant Halant	– > R4
%R4	:Consonant Halant	– > R5 R6 R7 R8
%R5	:End	– >
%R6	:Consonant Matra Modifier	– >
%R7	:Consonant Matra	– >
%R8	:Consonant Modifier	– >
%R9	:Consonant	– >

Table A.10: Syntax - syllable rules

{%{ <character types> } -> {<font character types>}}

%Begin	Cons-r	Halant	End	– > Half-cons
%Begin	Cons-r	Halant		– > Reph
%Halant	Cons-r			– > Rkar
%Consonant	Halant	End		– > Consonant Halant
%Consonant	Halant			– > Half-cons
%Vowel				– > Vowel
%Modifier				– > Modifier
%Matra				– > Matra
%Consonant				– > Consonant
%Numeral				– > Numeral
%Punctuation				– > Punctuation
%Halant				– > Halant
%Nukta				– > Nukta

Table A.11: Syntax - combination rules

of that particular type are present. Also it helps to generate the font characters according to the given context. The string which matches is replaced by font code. Generation of font code are generally context sensitive. This requirement of the language is met by specifying the combination rules.

Phoneme rule file format

This file contains the pairs of phonetically equivalent characters of Indian language alphabet and an integer value called matching percentage. In the rules, character descriptions are enclosed in '[' and ']'. In order to generate the phoneme equivalent words, one side of the characters are replaced by the other side of the characters in the original word. Any no of characters can be specified on both of the sides. According to the requirements of the language, rules can be added and deleted by the users, and matching percentage can be changed. The matching percentage should be an integer.

```
{%{[<string description for characters>]}   ->
{[<string description for characters>]}    ->
matching_percentage    }
```

%[MATRA_I]	$->$ [MATRA_II]	$->$ 30
%[MATRA_EY]	$->$ [MATRA_AI]	$->$ 35
%[Ka]	$->$ [KA] [NUKTA]	$->$ 50
%[Kha]	$->$ [KHA] [NUKTA]	$->$ 50
%[CHANDRABUNDU]	$->$ [ANUSWAR]	$->$ 90

Table A.12: Syntax - phoneme rules file

Wx-keyboard scheme file format

The wx-keyword scheme is specified by indicating the correspondence between the characters in Indian scripts and characters of wx-keyboard encoding. Morphological analyzer used in our search engine accepts input and produces output using this encoding. Moreover, the search engine does input/output with user in this encoding. Each keyword entered by the user is converted into ISCII encoding, using this table.

```
{%<string description for characters> -> { <wx-keyboard encoded chars> } }
```

%II	$->$ i	
%U	$->$ u	
%UU	$->$ U	
%KA	$->$ k	a
%KHA	$->$ K	a
%GA	$->$ g	a

Table A.13: Syntax - wx-keyboard scheme file

Each group of characters on the right hand side of the rule is equivalent to the character whose description is specified on the left hand side of the rule.

Bibliography

[1] *OII - Searching techniques*
http://www2.echo.lu/oii/en/search.html

[2] *A Webmaster's Guide To Search Engines*
http://searchenginewatch.com/webmasters/index.html

[3] *Search Engine Glossary*
http://searchenginewatch.com/facts/glossary.html

[4] *Indian Script Code for Information Interchange - ISCII standard.*
IS 13194 : 1991, Bureau of Indian Standards, Manak Bhawan, 9 Bahadur Shah Zafar Marg, New Delhi, December 1992.

[5] Puneet Chopra, *An Efficient Concurrency Control Model for Compressed Tries*, Department of Computer Science and Engineering, Indian Institute of Technology Delhi,New Delhi 110016

[6] *ht://Dig – Internet Search Engine Software*
http://www.htdig.org/

[7] Nitu Choudhary, *Iterm-Indian Terminal Emulator for X*, Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur (U.P.)

[8] Dr. Vineet Chaitanya and Dr. Rajeev Sangal, *Morphological Analyzer for Anusaraka, Indian Languages Translation Project*, IIT Kanpur Center for Natural Language Processing, University of Hyderabad, Hyderabad (A.P)
ftp://202.41.85.21/

[9] Panos E. Livadas, *File structures, Theory and Practice*, Prentice Hall ISBN 0-13-315094-1

[10] <FONT FACE> considered harmful.
http://www.isoc.org:8080/web_ml/html/fontface.html

[11] Specifying fonts in Web pages.
http://www.microsoft/com/typography/web/designer/font-face.htm

[12] Francois Yergeau. A world-wide World Wide Web.
http://www13.w3.org/International/francois.yergeau.html

[13] *The Unicode Standard.*
http://www.unicode.org/unicode/standard.html

[14] Index of Song Categories.
http://www.cs.wisc.edu/ navin/india/songs/isongs/indexes/categories/index.html

[15] Carrasco Benitez. Web Internationalization.
http://www.dragoman.org/winter/draft0.html

[16] R. Fielding, et. al.. Hypertext Transfer Protocol - HTTP/1.1.
RFC 2068

[17] Thomas Boutell. CGI programming in C and Perl.
Addison-Wesley Publishing Company.