ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

DESIGN AND DEVELOPMENT OF TIGRIGNA SEARCH ENGINE

By: Hailay Beyene Berhe

Advisor: Solomon Atnafu (PhD)

A Thesis Submitted to the School of Graduate Studies of the Addis Ababa University in partial fulfillment for the Degree of Master of Science in Computer Science

March 1, 2013

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

DESIGN AND DEVELOPMENT OF TIGRIGNA SEARCH ENGINE

By: Hailay Beyene Berhe

Advisor: Solomon Atnafu (PhD)

APPROVED BY:

EXAMINING BOARD:

1. Dr. Solomon Atnafu, Advisor _____
2. _____ _____
3. _____ _____

Acknowledgement

First and for most I would like to thank my creator-'God' for giving me health and patience in completing my thesis work. Next to this, I wish to express my sincere gratitude to my advisor Dr. Solomon Atnafu for his continuous and constructive advice and guidance throughout this work. Since the period period of this thesis work, he has been showing me the way I have to go through and has also provided me important resources, such as open source tools and related research papers. He has been also tackling with me in finding contacts when we encounter challenges in relation to tools. I am also highly thankful to Tessema Mindaye for his advice to work on this problem area and support in providing me important resources, Hymanot (Secretary of the department of computer science) for her cooperation in searching and mailing asked resources from the department's collection, my classmate –Alelgn Tefera, for his technical support and advice, and Salahadin Seid for his love. I also need to express my special thanks to Ato Tsegay W/mariam-staff of Tigrigna language department, Addis Ababa University, for working with me in the study of Tigrigna language characteristics. I am deeply grateful to my father, mother, brothers and sisters for their patience and love.

Since I cannot list all, I thank all who helped me in the accomplishment of my work directly or indirectly.

Table of Contents

LIST OF TABLES

LISTINGS

LIST OF FIGURES

Abstract

The web is one of the applications that run over the Internet and that stores information in different formats, such as image, text, audio and video. In order for the user to get the information of his/her interest, search engines play a great role. However, the general purpose search engines fail to consider the specific features of languages other than the English language for effective information retrieval. So, to ensure the efficiency of the IR system, local languages' search engines are very important. Particularly, Tigrigna language is not well supported by general search engines. Due to this, we have designed and developed a search engine that takes into account the specific properties of Tigrigna language. We proposed a general architecture for Tigrigna search engine. As any other search engine, the Tigrigna search engine we designed has a crawler, indexer and query engine as major components. The crawler downloads and identifies Tigrigna language web documents and stores them in the Tigrigna pages' repository to be used by the indexer component for further processing. The indexer performs the analysis tasks and finally creates an index which is suitable for searching. The search engine provides an interface in which the user writes his/her queries using Ethiopic script and an interface in which the result is displayed to the user. In this component, after the query phrase is composed in the query field, it is analyzed, searched from the index and finally the documents that are most relevant to this query are displayed based on their rank order (descending).

Our Tigrigna search engine is evaluated using 96 documents and some randomly selected queries that can demonstrate the effectiveness of the search engine using the precision-recall matrix.

Key words: *Information Retrieval, Tigrigna Search Engine, Tigrigna language Identifier, Tigrigna Analyzer.*

Chapter One: Introduction

Information is a processed data that can be used to make decisions and conclusions on an organization's business affairs. It is a data that is proved to be accurate, up-to-date, specific and organized for a purpose. It is described in a context that gives it meaning and importance to increase understanding and certainty. Its value should be able to affect the behavior, decision or outcome of an organization. A piece of information is valueless if the things remain the same after receiving it [62]. It is an important ingredient of a day to day life for an individual or organization. Since a raw data cannot be used for decision making, it must be processed and organized so that it can increase the knowledge of the organization or the person that uses it. So information is the important asset like money and energy [4].

There are a lot of sources of information, such as TV, radio, and different publications, but the web (World Wide Web) is the dominant one nowadays. The web is a huge repository of information which runs over the Internet. The web stores information in the form of text, audio, video, image, etc. The volume of information on the web is exponentially increasing and is expected to continue in the future. To retrieve the information available on the web, search engines play a great role [1].

A search engine is a tool (program) that enables us to search web documents that are more relevant to the query terms and returns the documents where the query terms are found. The term search engine is used to describe systems, such as, Google, AltaVista and Excite that enable users to search for documents on the web although it is a general class of programs. Each search engine uses its own algorithm to create its indices so that, but ideally, only meaningful results are returned for each query [2].

Starting from the invention of the web, the English language was the dominant language of the technology. Web documents in the other languages were not considered on the web until the last few years. Google, Yahoo, AltaVista and MSN are among the general search engines on the web used for searching, locating and retrieving information on the web. As an important quality of these engines, if we write English queries on their query interface, they handle more or less in the same way. But they handle non-English language queries differently from those that are designed for the specific languages. Therefore, the non-English language queries are best handled by language specific (search engines of the target language) search engines [3, 4, 5].

Some general purpose search engines, like Google provide local versions with local interfaces. Google, for example, supports some local languages of Ethiopia (Amharic, Affan Oromo, Tigrigna and Somali). Other general purpose search engines, like Yahoo, Bing, AltaVista and MSN also allow users to search in some local languages although they do not provide local interfaces. But their effectiveness cannot be evaluated because there is no standard local search engine that considers the special characteristics of the languages to

compare with [5]. Therefore, in this thesis work, we designed and developed a search engine for Tigrigna texts on the web that considers the specific characteristics of the language.

1.1 Motivation

As it is stated by Yonas Fissha [58], Tigrigna language is one of the Ethio-Semitic languages which belong to Afro-Asiatic super family. Although it is mainly spoken in Tigray region of Ethiopia and central Eritrea, there are also immigrant communities which speak Tigrigna language in Sudan, Saudi Arabia, the USA, Canada, Germany, Italy, UK, Sweden, and others [57]. Therefore, there are more than six million Tigrigna language speakers worldwide [57]. Tigrigna is written in the Ge'ez script which these days is called Ethiopic script and originally developed for Ge'ez language. Besides, Tigrigna is the working language of Tigray region in Ethiopia and one of the official languages in Eritrea.

Since information is very important in humans' day to day activities, the web provides large number of references for information that we are in need. The information over the web is accessible by the help of search engines. Most of the time people want to get the information they need in the language that they can easily understand. As it is mentioned above, Google provides an Ethiopic version interface that enables searching documents written in Tigrigna language though it misses the Tigrigna language features, as it can be demonstrated below.

Tigrigna language is a morphologically rich language. It is common to create changes in number, gender, possession, etc. by adding affixes to the stem of the words. Besides, new words in terms of meaning and category can also be created by adding the language's affixes. So, morphology can be either derivational or inflectional. Inflectional morphology deals with changes of gender, pluralisation and possession whereas derivational morphology deals with changes of meaning and category due to the addition of affixes [58]. Google Ethiopia[1] provides an interface for searching Tigrigna language web documents and returns different number of results. For example, for the short form "ቤት ት/ቲ" and expanded form "ቤት ትምህርቲ" of the same phrase in Tigrigna language, different results are returned. But when a user makes a query of these phrases, he/she must obtain the same or nearly the same result. So, in order to have the same or nearly the same results of such queries, the words have to be considered as they are talking about the same thing –school- so that the efficiency of the retrieval system can be improved [7]. While searching, term queries may be used in different forms to indicate sex, number, possession, condition, time, etc, like "ቤት ትምህርቲ" and "ብቤት ትምህርቲ". Since these terms belong to the same category or meaning, the inflectional richness should not affect the results of the query. Such types of queries are highly affected due to the reason that general search engines, including Google, do not incorporate the language's specific characteristics. The above cases are

[1] www.google.com.et

well illustrated in Table 1 for three general purpose search engines using the same Tigrigna queries and they return different number of results.

Table 1-1: Selected Tigrigna queries and their results using three search engines

Tigrigna Queries	Google's Results	MSN's Results	Yahoo's Results
ቤት ትምህርቲ	14,200	6,070	7,390
ቤት ት/ቲ	5,460	970	8,400
ገዛ	31,200	12,106	8,850
ገዛይ	1,230	306	554
ገዛውቲ	1,360	495	773
ገዛኻ	918	344	200
ገዛና	2,810	900	832
ገዛኺ	400	61	365
ገዛኹም	460	155	129
ገዛኽን	26	63	7
መፅሓፍ	3,810	145	262
መፃሕፍቲ	84	36	27

Therefore, as the above table indicates, the morphological variations of Tigrigna language are not supported by the search engines. As a result, the researcher is motivated to undertake this work because there is no other local search engine for Tigrigna language that considers the language's specific characteristics.

1.2 Statement of the problem

As it is stated above, General purpose search engines are inefficient when they are used for non-English language queries including Tigrigna because they are mainly designed to handle English language queries and they also lack to consider the specific characteristics of the language. To consider the morphological variations of a language, search engines that are designed for specific languages are more effective at handling queries of that language than general purpose search engines [6].

As Tigrigna search engine should consider and integrate the language's specific features into its searching technique, this thesis work aims to study these features and develop a search engine for the language.

1.3 Objectives

General Objective

The general objective of this research work is to design and develop a web search engine for Tigrigna language.

Specific Objectives

The specific objectives of the work are to:

- Study the language's features that can affect the retrieval of the language's documents on the web,
- Explore the existing language identifying techniques for documents on the web and choosing the one that fits to Tigrigna language web documents or developing an algorithm that can do this task.
- Design the general architecture of Tigrigna search engine,
- Develop a Tigrigna search engine,
- Demonstrate the effectiveness of the developed search engine.

1.4 Scope of the Study

Tigrigna pages on the web might be different types (text, audio, video, image, etc) and in different encodings. However, this work considers only texts in Unicode based fonts of Tigrigna language web documents.

1.5 Methodology

The methods that we have used in conducting this thesis work include:

- Different literatures on information retrieval are reviewed,
- Related research works that were developed for other languages are studied,
- Open source tools that were designed for search engines are identified and reviewed. These tools were investigated if they could be customized for Tigrigna language, and those that fit for our purpose are chosen,
- Precision and recall matrix are used to measure the retrieval quality of the system.

1.6 Application of Results

Since Tigrigna search engine is language specific, it is used for retrieving documents on the web that are written in Tigrigna language by considering the unique features of the language. As a result, the search engine can be used by the users of the language and those who want to use Tigrigna language documents on the web for their purpose.

Chapter Two: Literature Review

2.1 Information Retrieval

The challenges of information retrieval and storage came into existence in the early times [8]. Although the amount of information has increased, the accuracy and speedy access were ever more difficult. As a result, the awareness of individuals has increased towards these challenges. With the increase of computers, a great focus has been given to using them to provide rapid and intelligent retrieval systems. For example, many libraries that have an information storage and retrieval problems, such as cataloguing and general administration, have successfully been taken over by computers. However, the problem of effective retrieval remains largely unsolved.

Information storage and retrieval seems simple because if there is a document and the user requests, he/she can obtain the information by reading all the documents in the store to satisfy his/her information need. But this solution is impractical because the user either does not have time or does not need to spend his/her time reading the entire collection. It may also be impossible to physically move to read for the user. Later, high speed computers emerged and many thought that the computers would be able to read the entire document collection to extract relevant documents. In addition to this, using the natural language text of a document caused input, output and storage problems. As a solution, hardware development was supposed to make natural language input and storage more feasible. But software attempts to duplicate the human process of reading was very sticky problem. Since reading involves attempting to extract information (both semantic and syntactic) from the text and using it to decide whether each document is relevant or not to a particular request, the difficulty was not only knowing how to extract the information but also how to use it to decide relevance [9].

A typical information retrieval system can be illustrated as follows.

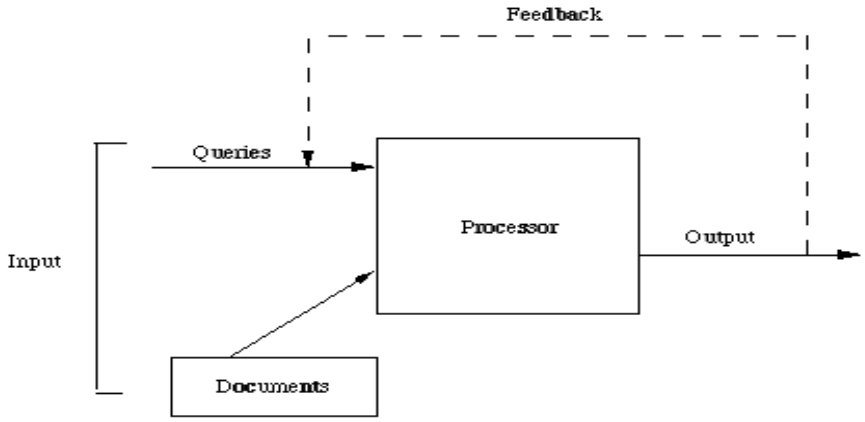


Figure 2-1. Typical IR system ([10]).

The above diagram shows three components: input, processor and output. The main problem with the input side is to obtain a representation of each document and query suitable for a computer to use. Because most computer-based retrieval systems store only a representation of the document. This is to mean that the text of a document is lost if the document is processed for the purpose of generating its representation. For example, the document representative can be a list of extracted words that are considered to be significant [10].

The user can change his/her request during one searching session to improve the subsequent retrieval run. This procedure is called a feedback as it can be seen from the diagram above.

The processor, as the part of the IR system, deals with the retrieval process. The process involves structuring the information (for example classifying) in some way appropriate for searching and performing the actual information retrieval function, such as, executing the search strategy in response to a query.

Finally, the output component displays the documents which satisfy the query and the retrieval ends here in an operational IR system whereas in an experimental system it leaves the evaluation to be done.

Therefore, information retrieval deals with the representation, storage, organization and access to information items [10]. The representation and organization of the information items should provide the user with easy access to the information in which he/she is interested in.

In the past times, IR (information retrieval) was made to remain the area of interest for librarians and information experts [11]. But nowadays, IR researches including modeling, document classification and categorization, systems architecture, user interfaces, data visualization, filtering and natural languages are conducted due to the dissemination of computers and other IR tools for multimedia and hypertext applications. This was possible due to the introduction of the web in the late 90s [11]. Since this time, the web is used as a universal repository for the human knowledge. The web is important to users due to the user interface provided to access the knowledge by hiding the communication details such as protocols, machine location and operating systems. It is also possible to create web documents and make them point to other web documents if there exists a need to link with other web documents. Therefore, the introduction of the web is a big revolution because it enabled people to make home shopping and home banking without any geographic limitation. However, the web has introduced its own problems. The serious problem was the difficulty and tediousness of finding useful information from it. When a user wants to search for information of interest, he/she should navigate the space of the web links (hyper space). Moreover, because the hyper space is vast and almost unknown, the searching process became inefficient. It might also be more severe, especially, for the illiterate users. This was true due to the absence of well defined underlying data model for the web. The promising solution for this problem was employing IR systems and its techniques.

2.1.1 Text analysis

Before computerized IR systems started to operate to retrieve some information, the information must be stored in the form of documents. However, the computer cannot store the complete text of each document that is written in all the natural languages. Instead, a document representative which may have been produced from the documents either manually or automatically. The text analysis process may start from a complete document text, an abstract, the title only or a list of words only in which a document representative can be produced in a form which the computer can handle [12,13].

The document representative consists of a list of class names in which each name representing a class of words occurring in the total input text. As a result, a document may be indexed by a name if one of its significant words occurs as a member of that class. So, such a system usually consists of removal of high frequency words, suffix stripping and detecting equivalent stems [13]. Stop word removal helps as a means to remove non-significant words as not to interfere during retrieval and reduce the size of the total document file. Suffix stripping is more complicated. As a result, a standard approach is to have a complete list of suffixes and to remove the longest possible one. While removing suffixes, since context free removal leads to some significant error rate, context rules are devised so that the suffix will be removed only if the context is satisfied [14, 15]. The context rules may emphasize:

- The length of the remaining stem exceeds a given number
- The stem ending satisfies a certain condition

For example, if we want to say N is removed from AMERICAN, not from SUDAN, we have to use some context rule (identifier). Each stem has one class and a class name is assigned to a document if and only if one of its members occurs as a significant word in the text of the document. A document representative then becomes a list of class names. These are often referred to as the document's index terms or keywords [16].

2.1.2 Indexing

Although it is controversial to choose the best index language for document retrieval, index language is used to describe documents and requests [17]. Index languages can be defined as pre-coordinate or post-coordinate. In pre-coordinate, the terms are coordinated at the time of indexing whereas in post-coordinate, the terms are coordinated at the time of searching. This is to mean that, in pre-coordinate indexing a logical combination of any index terms may be used as a label to identify a class of documents, whereas in post-coordinate indexing the same class would be identified at search time by combining the classes of documents labeled with the individual index terms.

Index term weighting

Exhaustivity and specificity of the index language are the most important factors that govern the effectiveness of an index language even though their exact meaning is debatable by different scholars. According to [18], index exhaustivity is defined as the number of different topics indexed, and the index language specificity is the ability of the index language to describe topics precisely. These authors [8] have also defined index specificity as the level of precision with which a document is actually indexed. Since both have an effect on the effectiveness of the retrieval system, according to [19], a high level of exhaustivity of indexing results high recall and low precision. Conversely, a low level of exhaustivity leads to high precision and low recall. In indexing specificity, high specificity leads to high precision and low recall. Taking this into account, weighting is applied based on the way the index terms are distributed in the entire collection. If we count the number of documents in which each index term occurs and plot them according to rank order, then we obtain the usual hyperbola shape. [20] Showed experimentally that if there are N documents and an index term occurs in n of them then a weight **log (N/n) + 1** leads to more effective retrieval than those that are unweighted. Therefore, a term with high total frequency of occurrence is not very useful in retrieval irrespective of its distribution. Middle frequency terms are useful particularly if the distribution is skewed. Rare terms with a skewed distribution are likely to be useful, but less than those of the middle frequency ones.

2.1.3 Classification Techniques in IR

The word classification is used to describe the process by which a classifying system is constructed. Classification is used for some purpose. One of the purposes in which classification is used may be to group the documents in such a way that retrieval will be faster or it may be used to construct a thesaurus automatically. Whatever the purpose, the goodness of the classification can only be measured by its performance during retrieval.

Therefore, the two main areas of application of classification methods in IR are [21]:

- Keyword clustering
- Document clustering

Keyword clustering by definition is a group of related phrases that are applied to a website. For example, we can have the cluster of medical words, technology words, etc. These keywords should be words that an audience will be entering in search engines. Each keyword cluster has a primary keyword that identifies the whole cluster. All other keywords in a specific cluster will be related to the primary keyword.

Document clustering is the grouping of documents or representations of documents which can be handled as a unit. Although documents that are related in some sense are grouped together, but more basically,

documents are grouped because they are likely to be wanted together. The likelihood among documents is measured by a logical relationship [22].

Logical organization is achieved in two ways.

- Through direct classification of documents
- By calculating the measure of closeness between documents..

2.1.4 Measure of association

Most classification methods are based on a binary relationship between objects. This relationship enables the classification method to construct a system of clusters. The relationship can be similarity, association and dissimilarity [24]. Similarity and association are to mean almost the same thing except that association is reserved for the similarity between objects characterized by discrete-state attributes. Measure of similarity is used to quantify the likeness between objects. Information retrieval systems commonly use five measures of association [25]. Since documents and requests in IR are represented by term or keyword lists, it is also assumed that objects are represented in a similar way and the counting measure gives the size of the set. The five association methods are:

- |X n Y| -Simple matching coefficient-

is the number of shared index terms, but it does not take into account the sizes of X and Y. The rest four (mentioned below) are used in document retrieval to take into consideration the information provided by the sizes of X and Y.

- $2\frac{|X \cap Y|}{|X|+|Y|}$ –Dice's Coefficient
- $\frac{|X \cap Y|}{|X \cup Y|}$ – Jaccard's coefficient
- $\frac{|X \cap Y|}{\sqrt{|X|} * \sqrt{|Y|}}$ – Cosine Coefficient
- $\frac{|X \cap Y|}{\min(|X|,|Y|)}$ – Overlap coefficient

Where X and Y are arbitrary real vectors (weighted keyword lists).

2.1.5 Cluster hypothesis

Cluster hypothesis identifies that closely associated documents tend to be relevant to the same request. In IR systems, relevant documents are more like to one another than the non-relevant documents. This is illustrated by computing the association between all pairs of documents that are [23]:

- Both relevant to a request and
- One is relevant and the other is non-relevant

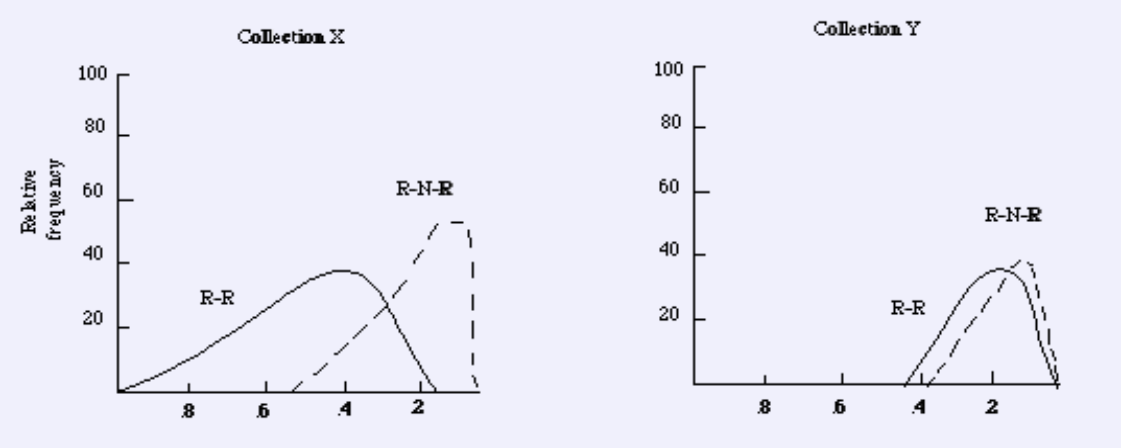


Figure 2-2. R-R is the distribution of relevant-relevant associations, and R-N-R is the distribution of relevant-non-relevant associations ([23]).

Therefore, document clustering can lead to more effective retrieval than a linear search because linear search ignores the relationship which exists between documents [23]. If the hypothesis is satisfied for a particular collection, it is clear that structuring the collection in such a way that the closely associated documents appear in one class, will not only speed up the retrieval, but may also make it more effective, since a class once found will tend to contain only relevant documents.

2.1.6 Search strategies

As it is mentioned above, in the case of document retrieval the information is the subset of documents which are decided to be relevant to the query. Although the search strategies are based on comparison between the query and the stored documents, the comparison is only achieved indirectly when the query is compared with clusters or more precisely with the profiles representing the clusters. The most common search strategies are discussed below.

2.1.6.1 Boolean search strategy

This search strategy uses the logical operators (AND, OR and NOT) to retrieve documents which are true for the query (documents that satisfy the query). In order this strategy to be meaningful, the queries must be

expressed in terms of keywords and combined by either one or all of the above mentioned logical operators. If we take the query Q= (A AND B) OR(C AND (NOT D)) then the Boolean search will retrieve the documents indexed by A and B in addition to the documents indexed by C that are not indexed by D [27]. For example,

A = {D1, D2, D3, D4}

B = {D1, D2}

C = {D1, D2, D3}

D = {D1} and

Q = (A AND B) OR (C AND (NOT D)) then the result that satisfies this query is the set {D1, D2, D3} and it is found by intersecting the A and B lists to satisfy the (A AND B) part and subtracting D list from the C to satisfy the (C AND (NOT D)) part. The OR is satisfied by taking the union of the two sets of documents obtained for the parts.

2.1.6.2 Matching Function

It is a means by which many of the search strategies are implemented [28]. It is similar to association measure although matching function measures the association between a query and a document or cluster profile and association measure is applied to objects of the same kind. These two functions have the same properties mathematically except their interpretations.

If M is the matching function, D the set of keywords representing the document and Q is the set representing the query, then:

$$M = 2\frac{|D \cap Q|}{|D| + |Q|}$$

is a matching function. According to cosine correlation, a document and a query are represented as a numerical vector on t-space. This means if Q= (q1, q2, …, qt) and D=(d1, d2, …, dt), qi and di are numerical weights associated with the keyword i [28].

2.1.6.3 Serial search

It gives a convenient demonstration of the use of matching functions although specifying the threshold is difficult. The serial search calculates N values of M (Q, Di) to determine the set of documents to be retrieved if there are N documents Di in the system. This can be done using two ways [29]:

- A suitable threshold is given to the matching function and documents above this threshold are retrieved whereas the ones below this threshold are discarded. If T is the threshold, then the retrieved set B is the set {Di | M (Q, Di) > T}.

- If r (i) is the rank position assigned to the document Di and R is a rank position chosen as a cut-off value, the documents below this rank are retrieved because the documents are ranked in an increasing order of matching function value. So B = {Di |r (i) < R}.

2.1.6.4 Cluster representatives

As it is tried to define above, queries must match with clusters. For this purpose clusters are represented by a profile called cluster representative which summarizes and characterizes the cluster of documents. The representative should be able to discriminate the relevant from the non-relevant documents for the requested query [30].

The main reason for constructing the cluster representative is to lead a search strategy to relevant documents and to guide a search to documents meeting some condition on the matching function. The drawback of cluster representative is that it treats the distribution of keywords in the cluster as independent [31].

2.1.6.5 Interactive search formulation

Always a user expresses his/her information need in trial and error process in which he/she formulates his/her query in the light of what the system can tell him/her about his/her query. As a result, the kind of information that the user is likely to want to use for the reformulation of the query is [29]:

- The frequency of occurrence in the database of his/her search terms;
- The number of documents likely to be retrieved by his/her query;
- Alternative and related terms to be the ones used in his/her search;
- A small sample of the citations likely to be retrieved, and
- The terms used to index the citations in.

The above mentioned information can be provided to a user during his/her search session by an interactive retrieval system. If the user's request occurs frequently, he/she may want to make it more specific by consulting a hierarchic dictionary which may probably tell him/her what his/her options are. In addition to this, if the user's query is also to retrieve too many documents he/she can make it more specific. While using this strategy, the user modifies his/her query in the light of this sample retrieval. The process in which the user modifies his/her query based on the actual search results is described as a form of feedback.

2.1.7 Evaluation of an IR System

IR is the area of study concerned with searching for documents, for information within documents, as well as that of searching structured storage, relational databases, and the web [26]. But this (retrieving what is needed) is not an easy task. To achieve the above mentioned tasks, we should have an efficient retrieval system. A good IR system should retrieve as many relevant documents as possible (i.e. have a high recall),

and it should retrieve very few non-relevant documents (i.e. have high precision) [26]. Therefore, recall and precision are the most commonly used techniques to measure efficiency (relevance) of an IR system. Consider,

I: an information request

R: the set of relevant documents for I

A: the answer set for I, generated by an IR system

RnA: the intersection of the sets R and A

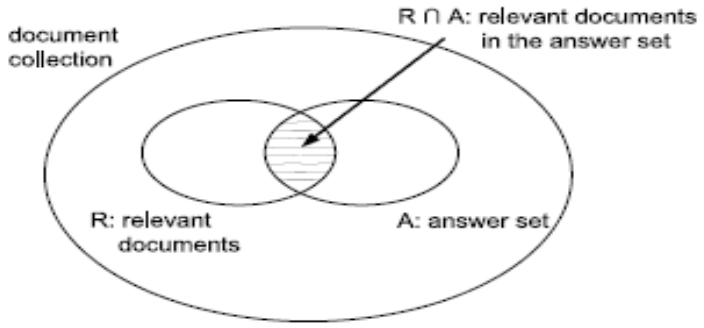


Figure 2-3. Set of a document collection for a query (Source: [26]).

Taking the above set, the recall and precision measures are defined as follows:

- Recall is the fraction of the relevant documents (the set R) which have been retrieved i.e.,

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

- Precision is the fraction of the retrieved documents (the set A) which are relevant i.e.,

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

2.1.8 PageRank

It is common to search documents from the web using search queries. Since all results of the query may not equally satisfy the query term, there must be a mechanism in which the results can be displayed in some order. This is what we call ranking. PageRank is a link analysis algorithm that assigns a numerical weighting to each element of a hyperlinked set of documents with the purpose of measuring its relative importance within the set [32]. The numerical weight that it assigns to any given element E is referred to as the PageRank of E and is denoted by PR(E). Therefore, according to [32] PageRank is defined as follows.

> "We assume page A has pages T_1...T_n which point to it (i.e. are citations). The parameter d is a damping factor which can be set between 0 and 1. We usually set d to be 0.85. Also C(A) is

defined as the number of links going out of page A". The PageRank of a page A is given as follows:

$PR(A) = (1-d) + d(PR(T_1)/C(T_1) + ... + PR(T_n)/C(T_n))$

The damping factor d is the probability that a person continues to click. As it is stated by different studies, damping factor is set to be around 0.85. The damping factor is subtracted from 1 and this term is then added to the product of the damping factor and the sum of the incoming PageRank scores [36] i.e.:

$PR(A) = 1\text{-}d + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + , \ldots\right)$, where A, B, C, D are web pages.

2.2 Search Engines

As search engines are software programs that search documents to satisfy user queries, they look through their own databases of information in order to find what people are looking for. Web search engines search information from the web or FTP servers and the search results are presented in a list of search engine result pages [36]. The information may consist of web pages, images, audio, video and other types of files. Search engines are important to retrieve information on the web. But the accustomed search engines are general purpose search engines which do not consider other languages' morphology other than English. Nowadays, to solve the above problem, language specific search engines are more or less becoming useful and applicable. This is because general purpose search engines are subject to low precision and low coverage whereas language specific search engines return higher-quality references than the general purpose search engines. Search engines create website lists by using spiders that crawl web pages, index their information, and optimally follow that site's links to other pages. Spiders return to already crawled sites on a regular bases in order to check for updates and everything that these spiders find goes into the search engine database. As a result of their complexity, search engines require detail processes and methodologies and are updated all the time. So, although search engines vary in their search results, they search documents (everything depending on the query) by performing the following tasks accordingly.

- The searcher types a query into a search engine
- Search engine software quickly sorts through literally millions of pages in its database to find matches to this query.
- The search engine's results are ranked in order of relevancy

Since the amount of information on the web and new users are dramatically increasing, the web creates challenges for information retrieval.

2.2.1 Components of a Search Engine on the web

All search engines, either general purpose or language specific, have three major components [34].

- Web crawling component
- Indexing component
- Searching (query engine) component

2.2.1.1 Crawling Component

A web crawler, which is most of the time called an ant, automatic indexer, bots, web spider and/or web robot, is a computer program that collects the information or documents (such as audio, text, video, image, etc.) on the web and FTP servers. Web crawlers are used to create a copy of all visited pages for latter processing by a search engine that indexes the downloaded pages to provide fast searches and used for automating maintenance tasks on a website, such as checking links or validating HTML code [35]. Besides this, web crawlers are also used to gather specific information, such as email address, usually for sending spam, from web pages. Web crawlers always start with a list of URLs, called seed URLs, to visit. As crawlers visit these URLs, they identify all the hyperlinks in the page and add them to the list of URLs, called the crawl frontier, to visit.

Since the web is characterized by high rate of change or deletion and large volume of information is added to the web regularly and because the crawler cannot download all the documents on the web, it must prioritize its downloads. When conducting crawls, bandwidth is neither infinite nor free. So, the web crawler must be scalable and efficient and must carefully choose at each step which pages to visit next.

The behavior of a web crawler is the outcome of a combination of the following policies [35].

- A selection policy that states which pages to download.
- A revisit policy that states when to check for changes to the pages.
- A politeness policy that states how to avoid overloading web sites, and
- A parallelization policy that states how to coordinate distributed web crawlers

Selection policy

Due to the hugeness of the web, the web crawler always downloads a fraction of the web pages. As a result, it is highly desirable that the downloaded fraction contains the most relevant pages rather than any random sample. This requires a metric of importance for prioritization of web pages. The importance of a page is a function of its quality, popularity in terms of links or visits and even of its URL. Although designing a good selection policy is an added difficulty, there must be a policy that works with partial information because the complete set of web pages is not known during crawling. Therefore, one possible selection policy can be

crawling web pages with high PageRank from different communities in less iteration in comparison with a crawl starting from random seeds.

Re-visit policy

As we have tried to mention it above, crawling a fraction of the web may take weeks or months and because the web is dynamic in nature, many events, such as creation, updating and deletions could have happened by the time the crawler has finished its crawl. From the search engine's point of view, there are two most commonly used cost functions.

- Freshness and
- Age

Freshness is a measure which deals with whether or not the local copy is accurate. The freshness of a page p in the repository at time t is defined as:

$$F_p(t) = \begin{cases} 1 & \text{if p is equal to the local copy at time t} \\ 0 & \text{otherwise} \end{cases}$$

Age is a measure concerned with how outdated the local copy is. The age of a page p in the repository, at time t is defined as:

$$A_p(t) = \begin{cases} 0 & \text{if p is not modified at time t} \\ \text{t - Modification time of p} & \text{otherwise} \end{cases}$$

As its objective, the crawler should maintain the average freshness of the pages as high as possible and the average age of pages as low as possible.

Politeness policy

As noted by Koster [35], the use of web crawlers is useful for a number of tasks, but comes with a price for the general community. The costs of using web crawlers include:

- Network resources, as crawlers require considerable bandwidth and operate with a high degree of parallelism during a long period of time.
- Server overload, especially if the frequency of accesses to a given server is too high.
- Poorly written crawlers, which can crash servers or routers, or which download pages they cannot handle.
- Personal crawler that, if deployed by too many users, can disrupt networks and web server.

Parallelization policy

A parallel crawler is a crawler that runs multiple processes in parallel to maximize the download rate and minimize the overload. This also uses not to repeat downloading of the same page more than once by having a policy for assigning the new URLs discovered during the crawling process. The standard high level architecture of a web crawler is presented as follows.

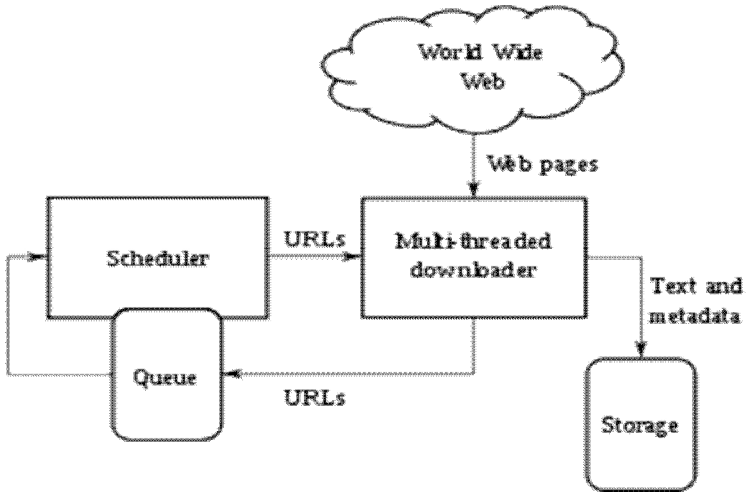


Figure 2-4: High level architecture of a standard web crawler (Source: [35]).

2.2.1.2 Indexing component

Indexing component is one of the major components of any search engine and is responsible for collecting, parsing and storing of data to facilitate fast and accurate information retrieval. Search engines can be categorized as those that focus on the full text indexing of online, natural language documents, Video, Audio and graphics, reuse the indices of other services and do not store a local index and cache based search engines that permanently store the index along with the corpus [36, 37]. Partial text services restrict the depth indexed to reduce index size unlike full text indices. The main focus of dealing with indexing is to optimize the speed and performance of a retrieval system to obtain relevant documents for a query. Without indexing, search engines would scan every document in the corpus and this requires much more time and a computing power.

2.2.1.3 Query engine component

This component allows the user to type his/her query and displays the results of the query to the user. It has two interfaces, namely search interface and display interface. Since it directly interacts with the user, it accepts the user query, analyses it and consults the indexer to display the results to the user by some ranking [5].

Chapter Three: Related Works

3.1 Search Engine for Amharic language

Amharic language is one of the Semitic languages. It is the official language of the federal government of Ethiopia and some other regional states such as, Amhara, Benshangul Gumz, and Gambella. It is included in the educational policy of all regional states of the country. This language is widely spoken by more than 17 million people as a first language and by more than 5 million people as a second language [4]. Amharic language is a morphologically rich language. Due to the reason that most search engines are designed for English language, they do not consider the specific natures of other local languages. As a result, Tessema [4] was motivated by this problem and has developed the ሐበሻ Amharic search engine. In this work, he studied the morphological variations of the Amharic language in detail to make the search engine efficient. The morphology that he considered was inflectional morphology in order to make the proposed stemmer polite. As any other search engine, Amharic search engine consists of three major components (crawler, indexer and query engine) that adapt the nature of the Amharic language. These components consist other sub-components such as, normalizer, tokenizer, stemmer, etc. to effectively consider the language's features. The study of the morphology of the Amharic language considers affixes, short forms of compound and single words, repetitive alphabets and encoding issues. The components of the search engine are all language specific and consider only documents that contain Amharic language texts. Therefore, the crawler crawls Amharic language documents, the indexer indexes pre-processed Amharic documents because what is crawled becomes an input for the indexer, and the query engine accepts Amharic user queries and displays Amharic documents to the user. Each subcomponent plays a specific role to achieve the full functionality of the IR system. For example, since the documents to be processed must be Amharic documents, the crawler has a language identifier or categorizer. For each component, Tessema has developed algorithms and implemented them. For implementing the proposed algorithms, the researcher has used lucene as a tool, Java programming language and other assisting tools such as, wordextract, NekoHTML and PDFBOX for text extraction. While crawling, the documents are stored in their file format folder. The indexer uses the crawled Amharic documents for further processing so that the documents will be stored in the repository in the form appropriate for searching. But before indexing takes place, normalization, stop words removal and stemming activities are performed. Then, the words are stored in an inverted index structure. Although a lot of computers and servers can be used for crawling and indexing respectively, the author has used a single personal computer of Intel Pentium IV, CPU with 2.00 GHz speed, 256 MB of RAM and 40 GB of hard disk capacity, with Microsoft windows XP professional operating system, and a bandwidth of 6Mb/sec. The researcher has found out open and closed source crawlers. Of these, the two are SPHINX1 and ht://dig, but none of these were used for crawling Amharic documents.

This is because, as stated by the researcher, SPHINX1 explicitly supports crawlers that are site specific, personal and re-locatable and ht://dig system is a complete indexing and searching system for a domain or intranet. But the Amharic system needs a crawler that can scale to a large portion of the web. As a result, the researcher has developed a language specific (focused) crawler from the scratch. During the course of development of the ሐበሻ search engine, the researcher has faced the problem of getting the seed URLs due to blocking of Amharic websites hosted outside of .et domain and the absence of Amharic websites that contain many Unicode encoded Amharic documents. Having the above mentioned two problems, the researcher has selected am.wikitionary, www.waltainfo.com, www2.dw-world.de/Amharic and www.ethiopiawakeu.net as seed URLs to begin the initial crawling. In the indexing component, words are tokenized using a whitespace, normalized, stop words are removed and stemming is taken place before indexing. For tokenizing the Amharic words, Amharic punctuation marks and white space are used. In addition to this, hyphenated words are tokenized as they are. Short forms of compound and single words (terms) are treated using "/" or "."(dot).

Besides this, "/" is used for formatting dates. To correctly represent the short forms of words, the researcher has first collected these words with the help of a language expert and these words are stored in a data structure, called a HashMap, by storing the beginning letter in the HashMap as a key with its corresponding expanded form as a value. The values of short forms of words in which their key is same are stored in an array list. In removing the affixes from inflectionally rich words, since the document is Unicode encoded, the added morphemes are removed and the last letter of the word is changed to the 6th letter of each alphabet (ፊደል). After completing all the search engine's tasks, the researcher has evaluated the search engine using precision-recall method and some selected queries to assure that the requirements are met. The crawler is also tested using two runs on different days and it has processed 7500 pages. It has selected 1803 Amharic web documents (pages) in the first run. The researcher has also helped the crawler in deleting non-Amharic URLs from the frontier manually. In the second run, the crawling was done on a single host and it has processed 22,276 pages and 12,276 pages were with Amharic content. The rest pages were link pages. Since a time gap between two consecutive downloads is necessary, a 20 seconds gap was used and it was found that the crawler was faster in night time. The categorizer was also tested against Amharic news documents to check whether it can identify Amharic Unicode encoded pages from the document collection and it was found that 99.41% was achieved. Its accuracy against identifying non-Unicode encoded pages was perfect (100%). In this case, the researcher did not test the categorizer against other languages' documents (pages) to assure whether or not it can identify as Amharic pages.

In evaluating the developed search engine, the researcher has used 75 news documents and 11 queries from computer science postgraduate students who are regular followers of news and have knowledge of a search engine. This was evaluated by one journalist and "AND" and "OR" Boolean operators were used. The results of precision and recall using these operators were different. i.e.

- Using “OR” precision 65%, recall 95%.
- Using “AND” precision 99%, recall 52%

Finally, it was concluded that the shortfall of the general purpose search engines is solved. As the modification of the “Habesha” Amharic search engine, Hassen Redwan [39] has increased the efficiency (speed) of the crawler by employing multi-threading. He also converted non-Unicode fonts into Unicode encoding (UTF-8) font, upgraded the normalizer to handle short form words separated by more than one period or slash and incorporated the different Amharic aliases.

3.2 Afaan Oromo Search Engine

Afaan Oromo is one of the major Ethiopian languages. It is an Afro- Asiatic language and is spoken as a mother tongue language by more than 25 million Oromo and neighboring people in Ethiopia and Kenya. It is also used as a working language of the regional government of Oromiya region [5]. So, since this language has many more users and different documents written by it, there must be a tool that lets the users of the language retrieve Oromo language scripts from the web. Although general purpose search engines allow users to search Oromo language scripts, they fail to provide an interface with the exception of Google and do not consider the nature (morphological richness) of the language. Taking these motivating factors, Tesfaye has developed an Affan Oromo search engine that typically considers the nature of the language [5]. During the development of the search engine Tesfaye has studied the following issues of the search engine.

- The inflectional morphology of the language.
- Short forms of different Afaan Oromo words.
- The three major components of the search engine with each sub-components
- Text and link extraction tools.
- Lucene as an indexing and searching tool.
- Java programming language for developing the system.
- JSP (java sever page) for developing the interfaces.

It is stated that Afaan Oromo is morphologically rich language. In Afaan Oromo language, most of the grammatical information is the result of affixes. As a result, almost all Afaan Oromo nouns and adjectives have person, number, gender and possession markers by attaching the affixes to the stem or singular noun form. Not only this, but also Afaan Oromo plural markers have several alternatives in comparison to English plural markers(s or es). Likewise, Afaan Oromo verbs are also highly inflected for gender, person, number, tenses, voice and transitivity. The punctuation and white space usage of Afaan Oromo search engine is exactly the same as other Latin scripts, like English. During his study, Tesfaye has designed the search

engine by devising some mechanisms to handle the natures of the language. Words are stemmed to avoid some affixes that have insignificant meaning on the word. Short forms of Afaan Oromo words are treated by putting the words in the HashMap with their corresponding expanded value. To increase the efficiency of the search engine, the researcher has developed a language specific categorizer to identify and filter Afaan Oromo documents from other documents on the web. The researcher has also developed interfaces where the user can write queries for his/her information need and the search results are displayed. The major components of the search engine and their sub-components are implemented. Although there are several open source crawlers that are developed by Java programming language, the researcher has selected JSpider to meet his requirements. From the downloaded documents in the repository, links and texts are extracted and the links are used by the downloader for further downloading as a URL. The extracted texts are identified by the categorizer (language specific) to reject non-Afaan Oromo language documents and store the languages' documents in its repository. A document in the web contains different formats (such as, PDF, HTML, MS-word, etc). As a result, Tesfaye has used open source java tools to extract a plain-text from the different file formats so that they can be used by the lucene's index and categorizer. These tools are PDFBox, NekoHTML and TextMining.org's API to extract text from pdf, html, and MS-word formats respectively. Afaan Oromo language words have their own ways of construction (cvc, cv, cvvc and cvv). So, as it is stated by the researcher, the categorizer sub-component categorizes words as Afaan Oromo language words if they have one of the above Afaan Oromo language word formation rules. This was tested by taking English and Afaan Oromo language documents to compare because two of them are Latin scripts. After categorizing the documents, the sub-components, such as the normalizer, stop word remover and stemmer were applied to store (index) the information about the Afaan Oromo documents and the indexing was done in the lucene index. The researcher has developed the query engine component using JSP (java server page). In this component, first a user feeds his/her query and some pre-processing activities (such as, normalization, stop word removal and stemming) were performed. Doing so, the indexer is consulted if there exists a match so that to retrieve the documents that contain the key word from the repository. Since all the documents that match the search query may not be equally relevant to the query, ranking techniques are used to display the documents in order of their relevance. For this to implement, lucene ranking algorithms were used.

After the completion of the design and implementation of the search engine, the researcher has used precision-recall matrix to evaluate the efficiency of the developed system. To do this, the researcher has used two seed URLs (http://www.orto.gov.et and http://www.radiofana.com) to evaluate the crawler component. The resources used for evaluating the system were a laptop computer with windows XP professional edition SP2 operating system1.5 GHz CPU, 1 GB RAM, and 40GB hard disk. As it is stated above, Afaan Oromo uses the same script with English scripts. Because the testing was done in comparison with English language documents and since there is syllable pattern similarity in constructing words, the

researcher has realized that cvv and cvcc syllable patterns are more frequent in Afaan Oromo documents than in English documents. Therefore, taking the two patterns into consideration, the researcher has used 300 document collection (with 161 Afaan Oromo and 139 English documents) to check the two syllable patterns (match the words in the document with any one of the patterns) and a performance of 78.9% in correctly identifying Afaan Oromo documents and 98.6% in correctly rejecting non-Afaan Oromo documents were found.

The researcher has also observed that the big gap (between the performances) shows that more number of non-Afaan Oromo documents match cvv syllable pattern as compared to cvcc syllable pattern. Finally, to evaluate the performance of the search engine, the researcher has used 70 documents and 10 queries and the obtained recall and precision are 57% and 93% respectively. 76% of an average precision for the 10 top results was found and this shows better user satisfaction as comparing to displaying all results.

To conclude, the precision-recall matrix was done in considering the design requirements (morphology of the language, handling short forms, etc.) of the search engine.

3. 3 Tamil search engine

As other languages, Tamil language is morphologically rich. As a result, it is a must to have a search engine that handles the unique characteristics of the language that cannot be handled by the general purpose search engines and it is called Tamil search engine which searches for Tamil language documents on the web. This search engine has 3 major components, namely the crawler, the database system and search module [38]. As in any other languages, the crawler component retrieves pages from the web and hands them to database system component. From the retrieved pages, a list of seed URLs and texts are extracted. While downloading the documents, the links are extracted and are added to the URL list if they do not exist in the list. The crawling process is periodically repeated to obtain the updated status of the web documents. Tamil search engine handles both Tamil and English language documents (contents) although the contents (words) of the languages are maintained separately. For effective retrieval, the different fonts of the words of the Tamil language are converted to the TAB font using font converters. The document in the web can be in different formats. These formats should be converted into text formats so that it can be handled by the search engine. So, to extract the root words: Stop words are removed then the root words are extracted using the morphological analyzer. Therefore, the text part is extracted using the morphological analyzer and the URL links are identified by using the HTML tags, such as <a href…> etc [38].

The database system component is responsible for storing the latest version of every page retrieved by the crawler. This component has two tables in which one table uses for storing words and the other uses for storing URLs.

The third component which is the search module is the component in which users make a query and ranking mechanism is taken place. The user interface allows the user to enter the query in English or Tamil

language. For Tamil words, morphological analysis is taken place to get the root word using the morphological analyzer. After this, using the root, the database is queried and the pages that satisfy the query are retrieved based on the frequency (word count) of the word. In this case, the documents containing all the words (corresponding to Boolean AND function of the words), would be first listed, followed by documents containing a subset of the words (corresponding to OR function).

In general, this search engine contributed a way to handle the morphological variations of the language. But, we doubt that the crawler is language specific and nothing is said about how the English and Tamil languages can be used by the search engine. The different fonts of the language's documents (Tamil) are converted into TAB font using converters. But it is not stated (specified) what font converters are used. It is not also clear why the English language is handled by the search engine.

3.4 Myanmar language search engine

Myanmar is the name of the country-"Myanmar" and the language spoken by the peoples of the country. More than 56 million population with 135 ethnic groups speaking their own vernacular languages or dialects live in the country. Because Myanmar language is spoken by more than 30 million people in the country, it is the official language used in the administrative, judicial and commercial systems throughout the nation. So, since Myanmar language has its own specific characteristics and the general purpose search engines are designed to effectively handle English queries, this language specific search engine (Myanmar language search engine) is necessary to handle these characteristics on the web [43].

For writing Myanmar, the characters are round in shape, and are written from left to right. As white space between words is common in other languages, such as Tigrigna, Amharic, English etc., no need to put white space in between words in Myanmar language. But white spaces are used after each clause in order to enhance readability in its modern writing style. This language has 33 consonant characters and 12 vowel characters [43]. As the authors have stated it, Myanmar web pages have various encodings. Although Myanmar web pages have many and different encodings, of which Zawgyi and Mixture encoding style format, Zawgyi is widely used because the developers hesitate the other encodings in which they are not familiar with.

Compound words are common in Myanmar language. It is usual to see compound words in other languages, like English, but compound words are not separated by space in this language unlike other languages. This has an effect in the retrieval system of the language.

So, the researchers have studied the:

- unique features of the language
- encodings of the Myanmar language
- compound words of the language

- Architecture of the language specific search engine.

Myanmar language search engine takes the specific features of the language into account because it has a crawler which downloads the language's documents, and a language identifier that concurrently runs with the crawler. The language identifier is the heart of the crawler because it makes it language specific (enables it to search only documents of the target language). The authors have chosen an n-gram based algorithm for language identification in the crawler because of its high accuracy in identification, its resilience to typographical error and its minimal data requirement for training corpus. The language identifier they have used is G2LI (Global Information Infrastructure Laboratory's 2nd version language identifier) [44]. This method first creates n-byte sequences of each training text. Then it checks the n-byte sequences of the collected page with those of the training text. The language having the highest matching rate is considered as the language of the web page.

It is also stated that after downloading, the downloaded web pages are passed to some pre-processing steps. These are:

- HTML parsing (HTML tags are removed to get only the text data)
- Natural language processing tasks which include converting of non-Unicode encodings to standard Unicode and tokenization (the process of segmenting words of the language and is done using long algorithms, called greedy).

Finally, the authors have got that the crawler is able to collect 100 pages by starting from three seed URLs

- http://www.linhtet.com,
- http://gtchmawbi.ning.com and
- http://burmese.bnionline.net

These pages were in three different encodings. These were:

- 14-Unicode
- 79-Zawgyi
- 7-Win-Inwa

To check the efficiency of the system, the authors traced it using queries of some keywords, subscripted-form, and compound forms. While testing, the query words were in Unicode encoding. So, this shows that the authors have confirmed their design requirements.

To summarize, the search engines reviewed above are specific to their corresponding languages and do not handle Tigrigna language web documents. We are thus planning to design and develop a web search engine that is capable of searching Tigrigna web documents.

Chapter Four: Tigrigna Language

Tigrigna is the third most spoken language in Ethiopia and it is the member of the Ethio-Semitic languages which belong to Afro-Asiatic super family [57]. As it is discussed in the motivation section, Tigrigna is primarily spoken in Tigray region of Ethiopia and in Eritrea although it is spoken in other parts of the world by immigrants. It is also discussed that there are more than six million Tigrigna language speakers worldwide and it uses the Ge'ez script- which is also called Ethiopic script this days- though this script was developed for the Ge'ez language originally.

Since Tigrigna language is syllabic in nature, a single symbol (alphabet) represents vowel and consonant combination [58, 59]. Due to its morphological richness, Tigrigna exhibits the root and stem pattern morphological phenomenon. Because the morphological variation is the result of adding affixes to the root verbs or nouns to indicate number, gender, tense, possession, etc., it is necessary to understand the behavior of Tigrigna language stems and roots. As it is studied by [41] and cited by [58], a root is a verb that indicates a third person singular masculine, such as በልዐ ፣ ሰተየ ፣ ሰርሐ, etc. and the noun that expresses a singular noun whereas a stem is a verb in which its ending letter is 'sads' (6th order) or a noun that indicates a singular number. Therefore, a stem may or may not have a meaning if the word is a verb [41].

4.1 Morphological Variations of Tigrigna Language

As every language has its own affixes and these are placed in different positions to produce the variants of each language, Tigrigna affix is a morpheme that can be added at the end, beginning or middle (inside) of the root to create the inflectional or derivational morphology of Tigrigna words. These words may be new in meaning and structure from their respective roots. Since Tigrigna affixes can also concatenate with each other, this increases the number of affixes in the language [42]. Tigrigna affixes can be classified in to four categories [42] as prefixes that come at the beginning of the root, such as ን፣ ዝ፣ እንተ፣ ም፣ ብም፣...፣ suffixes that come at the end of the root, such as ና፣ ታት፣ ት፣ ነት፣ ን፣ ... ፣ infixes that come inside the root, such as ባ in ሰባበረ፣ላ in በላልዐ፣ ታ in ሰታሰተየ፣ ... and circumfixes that are attached before and after the base form at the same time.

As it is studied by [41], Tigrigna language has eight parts of speech. These are grouped as, nouns, adjectives, verbs, adverbs and prepositions due to the role and behavioral similarity between nouns and pronouns, conjunctions and prepositions and because Interjection (exclamation) cannot stand independently. Due to the reason that prepositions are morphologically unproductive and adverbs are few in number and less productive, the derivational and inflectional morphology of the language concentrates on the rest three parts of speech (verbs, nouns and adjectives).

4.1.1 Inflectional Morphology of Tigrigna Language

Inflection is a morphological variation that does not change the part of speech category and general meaning, but the grammatical function. Since Tigrigna language is a highly inflectional language, a given root of a Tigrigna word can be found in different forms.

Inflection of Verbs

For the purpose of investigating the inflection of Tigrigna language verbs, we have used Tigrigna grammar books by Daniel Teklu [41] and Kasa G. [42]. Tigrigna verbs are found in perfective, imperfective, gerundive, jussive and imperative by employing affixes. The morphological variations of a perfective verb is formed by adding suffixes that mark (indicate) person, gender and number to the perfect verb stem. For example, ከድኩ፣ ከድካ፣ ከድኪ፣ ከድኩም፣ ከደ፣ ከደት፣ ከድክን፣ከዱ፣ ከዳ are perfective verbs formed from the stem ከድ which means 'to go'. The other verb type is imperfective verb and is formed by affixing gender, person and number markers to the imperfective verb stem. For example, እኸድ፣ ትኸድ፣ ትኸዲ፣ ይኸድ፣ንኸድ፣ ትኸዱ፣ ትኸዳ፣ ይኸዱ and ይኸዳ are imperfective verbs that illustrate how the affixes are used to inflect the imperfective verb stem. The gerundive form is inflected by adding suffixes at the end of the gerundive verb to indicate person, gender and number. For example, ሰሪሐ፣, ሰሪሕኻ፣ ሰሪሕኺ፣ ሰሪሑ፣ ሰሪሓ፣ ሰሪሕና፣ ሰሪሕኹም፣ ሰሪሕኽን፣ ሰሪሖም፣ ሰሪሐን can show how the gerundive verb ሰሪሕ morphologically varies. Jussive and imperative verbs are sometimes called mood and jussive verbs are used to express a command for first and third persons whereas imperative verb is used to express second person in the singular and plural form [40]. In addition to indicating person, gender, number and time, Tigrigna verbs express two tenses, namely, perfect and imperfect. Perfect tense indicates completed actions whereas imperfect tense expresses uncompleted actions.

Inflection of Perfect Tense

Perfect tense expresses a past tense and consists a stem, pronoun suffix and subject marker. The subject marker shows the subject’s number, person and gender [41, 58]. This is illustrated in Table 2 to indicate the subject marker and the pronoun.

Table 4-2: Inflections of perfect tense

Verb Variations	Person	Gender	Number	Pronoun
ሰሪቑ	Third	Male	Singular	He-ንሱ
ሰሪቖም	Third	Male	Plural	They-ንሳቶም
ሰሪቖን	Third	Female	Plural	They-ንሳተን
ሰሪቕኩም	Second	Male	Plural	You-ንስኻትኩም
ሰሪቕኪ	Second	Female	Singular	You-ንስኺ
ሰሪቕካ	Second	Male	Singular	You-ንስኻ
ሰሪቕና	First	Male and Female	Plural	We-ንሕና
ሰሪቑ	First	Male and Female	Singular	I-ኣነ
ሰሪቕክን	Second	Female	Plural	You-ንስኻትክን
ሰሪቓ	Third	Female	Singular	She-ንሳ

As it can be seen from the above table, the suffixes attached are ኡ/u/, ኦም/om/, ኣን/en/, ኩም/kum/, ኪ/ki/, ካ/ka/, ና/na/, ኣ/ae/, ክን/kn/, ኣ/a/ and indicate person, gender and number of the subject and the pronoun that indicates the person.

Inflection of Imperfect Tense

This tense includes indicative, subjective, jussive and imperative forms and are inflected by adding affixes to indicate gender, person and number to the imperfective verb stem. We can consider the following table to see the affixation on the root verb "ሰርሐ".

Table 4-3: Inflection of imperfect tense

Person	Gender	Singular	Plural
Third	Female	ትሰርሕ	ይሰርሓ
Third	Male	ይሰርሕ	ይሰርሑ
First	Male, Female	እሰርሕ	ንሰርሕ
Second	Male	ትሰርሕ	ትሰርሑ
Second	Female	ትሰርሒ	ትሰርሓ

In the above table, ት(t), ይ(y), እ(i), and ን(n) are prefixes and ኢ(e), ኡ(u), and ኣ(a) are suffixes. It is also possible to find the negative form of the above mentioned root by using ኣይ/ay/, ኣይት/ayt/, ኣይን/ayn/ as prefixes and ን/n/, ኣን/an/, ኡን/un/ as suffixes.

Inflection of Nouns

Like Tigrigna verbs, Tigrigna nouns inflect to show gender, person and number by adding affixes to the noun stem. Grammatically, Tigrigna language specifies two types of genders: Feminine and masculine. So, Tigrigna nouns are either male or female by nature [41]. Therefore, in order nouns to express possession, pluralism, tribe and gender, affixes such as -ኣ/a/, ታት/-tat/, ኣት/-at/, ኣን/an/, ውቲ/-wti/, ቲ/-ti/, ዊ/wi/, ና/na/, etc. are used. The table stated below shows how these affixes are used to inflect Tigrigna nouns.

Table 4-4: Inflections of Nouns

Noun Stem	Affix used	Noun after affixation	Remark
ላሕሚ	ኣ	ኣላሕም	Pluralism
ፊደል	ኣት	ፊደላት	Pluralism
ቐፅሊ	ታት	ቐፅልታት	Pluralism
መምህር	ኣን	መምህራን	Pluralism
ገዛ	ውቲ	ገዛውቲ	Pluralism
ኣድጊ	ና	ኣድጊና	Possession
ስራሕ	ቲ	ስራሕቲ	Pluralism
ኤርትራ	ዊ	ኤርትራዊ	Tribe
ሓሙ	ኣት	ሓማት	Gender

Inflection of Adjectives

Tigrigna adjectives agree with their noun in number and gender. They inflect to indicate number and gender. This is to mean that Tigrigna adjectives can have singular masculine, singular feminine and the same adjective to express both masculine and feminine genders in the plural form [41]. ት/t/, ቲ/ti/, ኣዊት /awit/, ኣዊ/awi/, ኣት/at/, ኣዊያን/awyan/, etc. are used to inflect adjectives and are illustrated how they are used in the table below.

Table 4-5: Inflection of Adjectives

Masculine	Feminine	Pluralisation
ፅቡቕ	ፅብቕቲ	ፅቡቓት
ሰራሒ	ሰራሒት	ሰራሕቲ
መዐረይ	መዐረይት	መዐረይቲ
ቀታሊ	ቀታሊት	ቀተልቲ
ሰፋይ	ሰፋይት	ሰፊይቲ

4.1.2 Derivational Morphology of Tigrigna Language

As it is discussed in the introductory part of this work, Tigrigna language is morphologically rich both inflectionally and derivationally. Derivational morphology deals with adding affixes to words to bring a change in meaning and category whereas inflectional morphology deals with adding affixes to words so that the general meaning and category of the original word (stem) will not be changed. Therefore, although derivational morphology is the characteristics of Tigrigna language, we do not need it to discuss here because derivational morphology is not the issue of search engines (it should not be considered in the analysis part of the search engine). But Tigrigna language nouns, adjectives and verbs inflect derivationally.

4.2 Tigrigna Information Retrieval

Natural language processing is a field which studies the computer-human (natural language) interaction. It is the process by which a computer extracts meaningful information from natural language input and/or producing natural language output. Natural language processing deals with automatic summarization, text mining, information retrieval, search engines and speech synthesis [39]. Since the web is a huge repository and is natural language independent, users need to have the information on the web in some language. The languages of the web documents are in different forms of the target language. In order for the web to provide the required information to the user, the search terms should be in some form so that the required information will be displayed. So, one of the pre-processing tasks to improve the effectiveness of the IR system is stemming the search terms. Till now, there is no other work done on Tigrigna search engine. But Girma Berhe [40] and Yonas Fisseha [58] have studied Tigrigna language and developed Tigrigna language stemmers. Girma Berhe [40] has developed a five step iterative stemmer (stemming algorithm) for the language using a C++ programming language. This algorithm considers both inflectional and derivational morphemes of the language. In addition to this, the stemmer does not identify the cases where an affix cannot be a real affix. For example: if we take the affix ት-"t", it is not an affix in the following terms and it should not be conflated (removed).

- ዕብየት- "development"
- ስርዓት-"rule"
- ስልጠት-"power"
- ጥምየት-" hunger"
- ዓይነት-"type"
- ሕፅረት-"deficiency" etc.

But it is an affix in the following terms and should be removed.

- ፊደላት-"letters"

- ሓረስቶት-"farmers"
- ሰባት-"people" etc.

The above mentioned condition occurs in many other Tigrigna term affixes also.

The other stemmer was developed by Yonas Fisseha [58]. In his research, he has studied:

- Different stemming algorithms that can be used for stemming Ethiopic and other foreign languages.
- The morphology (inflectional and derivational) of Tigrigna language words and the causes of the morphology on the language.
- Prefixes and suffixes of the language and collected them from different sources.
- Tigrigna stop words by collecting them from different sources.
- The tokenization and normalization aspects of the language's words.

After this, Yonas has set a 10 step rule for prefixes removal and a 7 step rule for suffixes removal to apply the affixes removal procedure in his stemmer. During the implementation of the algorithm, he has applied the following tasks in the specified order i.e.

Tokenize ⟶ Normalize ⟶ Stop words removed ⟶ Transliteration ⟶ Stemming

To accomplish the above steps, he has developed his algorithm using python programming language and used a corpus to see the effects of the stemmer. So, Yonas [58] has implemented a rule-based stemmer that handles only prefixes and suffixes. But this stemmer cannot be used for our purpose due to the below mentioned reasons.

- It is aggressive because it stems both derivational and inflectional morphemes.
- It does not specify the conditions where an affix is not treated as an affix.
- Although affixes include suffixes, infixes, prefixes and circumfixes, this stemmer considers only suffixes and prefixes.
- The researcher has transliterated the corpus before stemming the words in the corpus by using SERA transliteration tool to convert the Ge'ez to Latin characters (ASCII characters). Transliteration is very important because it makes the stems easily identifiable by making vowels and consonants visible. But web documents are not found in a transliterated form.
- The normalization analysis he has made does not imply Tigrigna language because it does not have redundant alphabets.

Therefore, although both of them cannot be used as they are, Girma's algorithm can better be adopted by making modifications to handle the irregularities discussed in the algorithm. Since it over-stems the Tigrigna words (terms), it should not be iterative. As it is tried to mention above, Girma's stemmer was a five step stemmer, but we have made it to consider four affixes (single letter reduplication, prefix, suffix,

and double letter reduplication). The prefix-suffix pair removal step is left out because it brings derivational changes or changes the term in terms of meaning and category. All in all, our stemmer considers only the inflectional morphology of the language in order not to change the meaning and category of the language's words. Besides, Girma's stemming algorithm for Tigrigna language was useful for transliterated Tigrigna documents i.e. the algorithm considers the language's alphabetic nature. But Tigrigna writing style is not alphabetic, rather syllabic [58, 59]. The Unicode consortium has given an Ethiopic Unicode table and this Unicode is a representation of the syllabic pattern. So, since the programming language we have used is Java and Java understands (represents) characters in a Unicode representation, a modification is made on the algorithm.

Chapter Five: Design of Tigrigna Search Engine

As it is already discussed above, the web is a huge repository of multi-lingual contents. It is also difficult to fetch the whole web and downloading the web documents is resource intensive. To minimize these problems, language specific search engines are very important because they index only documents of interest and save computational resources. So, Tigrigna search engine is designed to be a focused search engine that considers the specific features of the language. In order to have an efficient Tigrigna retrieval system, the design of the search engine depends on the language's specific characteristics, such as morphological variations of words, short forms of words and identification of Tigrigna documents on the web. The morphological variations of Tigrigna words will be handled by employing Tigrigna stemmer and Tigrigna language documents will be identified using a Tigrigna language identifier. The short forms of Tigrigna language words will also be handled using a normalizer component so that the short and expanded forms of the word will be treated equally to enhance the efficiency of the Tigrigna retrieval system. In addition to this, the design incorporates a component that a user can enter a query and see the results of the query search.

This chapter discusses the general architecture of Tigrigna search engine. This search engine is designed to index and search only Tigrigna documents crawled from the web. As any other search engines, our search engine has three major components, namely:

- The crawling component
- The indexing component and
- The searching (query engine) component

These components have their own sub-components and are depicted in the following figure.

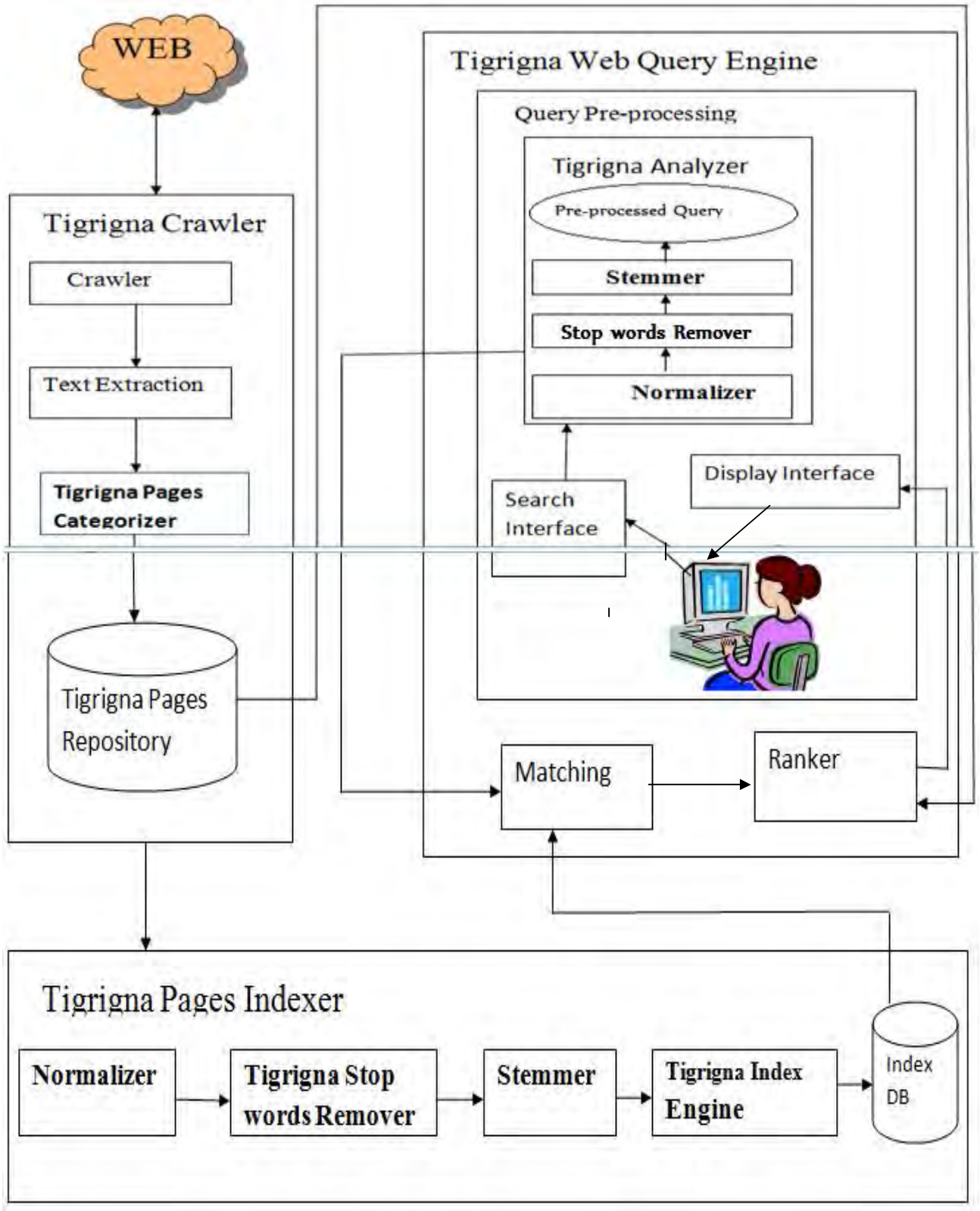


Figure 5-5: Architecture of Tigrigna search engine.

5.1 The Tigrigna Page Crawler

As it is discussed in section 2.2, all search engines have this component since it is the program that visits Web sites and reads their pages and other information in order to create entries for a search engine index. Crawlers visit sites that have been submitted by their owners as new or updated or pages that are linked in the websites. Entire sites or specific pages of a site can be selectively visited and indexed. As a result, web crawlers have got this name because they can crawl a site or page at a time following the links to other pages on the site until all pages have been visited or the specified depth is met [45].

Although web crawlers can be used by anyone who seeks to collect information from the web, they are mostly used to fetch information available on the Internet. For this reason, search engines frequently use web crawlers to collect information about what is available on the public web pages. So, web crawlers primarily collect the data on the web and search engines provide the surfer with relevant websites when he/she enters a term query on their sites. Besides, web crawlers can be also used by linguists as well as market researchers. Linguists use a web crawler to perform a textual analysis; that is, they may surf the Internet to determine what words are commonly used today and market researchers may use a web crawler to determine and assess trends in a given market. When the crawler visits a page, it reads the visible data, the hyper-links and content of the various tags used in the site. Using the gathered information by the crawler, the search engine indexes the information and the website is then included in the search engine's database and its page ranking process.

Therefore, Tigrigna language documents' web crawler consists of multi-threaded gatherer (downloader), text extraction tools, Tigrigna page identifier and Tigrigna pages repository module. So, this crawler, generally, consists:

- A sub-component that stores URLs to be visited (unvisited URLs).
- A sub-component that stores visited URLs so that the extracted link or new URL will be cross-checked with the visited URLs to discard from or save to the unvisited URLs storage.
- A sub-component that retrieves the page at the given URL
- The language identifier that categorizes Tigrigna pages and puts them in a Tigrigna pages' repository for further processing.
- A parser that extracts links from the downloaded pages to be used as URLs for further crawling.
- Text extraction tools that are used for extracting texts from different file formats.

Although web crawling can be conducted using two approaches (using HTTP request header fields and simply downloading the page corresponding to the given URL and categorizing the pages using our language identifier), due to the reason that most servers don't have a language attribute and HTTP protocol doesn't support Unicode block range [4], we used the second approach. As Tigrigna language is the Semitic

languages' family and is related with other Ethiopian languages, especially with Amharic language, in terms of script (using similar alphabets), Tigrigna web pages are efficiently identified from other languages' pages using a stop word based Tigrigna language identifier and a tool called LIM (Language Identification Module). Therefore, the procedure in which the crawling task is performed is shown below.

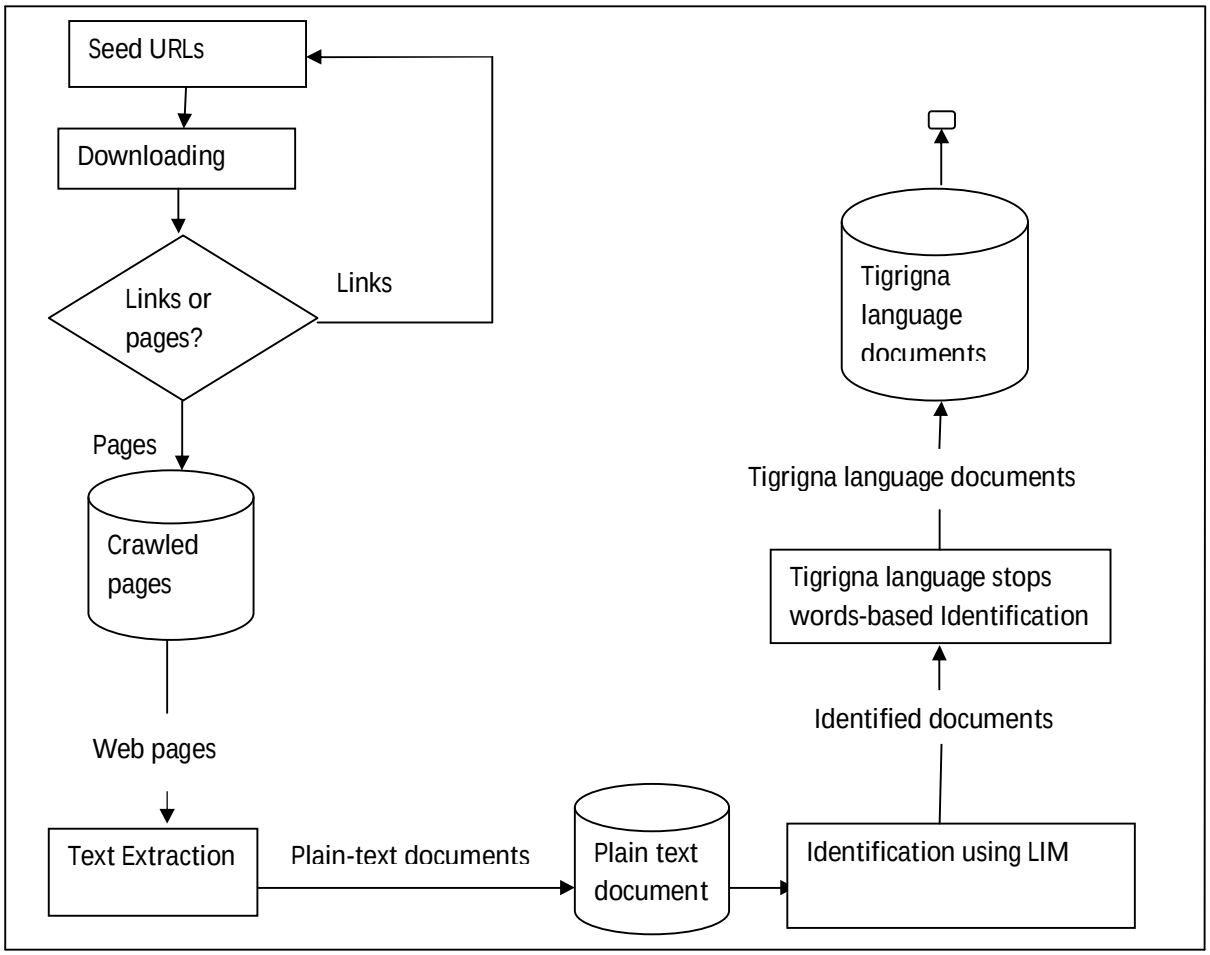


Figure 5-6: Tigrigna Pages Crawling Component.

The algorithm for the stop words based language identifier is:

1. open the folder that contains the downloaded documents from the repository
2. open the file (document) in the folder
3. read a stop word from its file
4. compare its existence in the document
5. if the stop word exists in the document
   copy the document and put it in the Tigrigna document's repository
   remove the Tigrigna document from the original repository
6. else
   go to step 2
   go to step 4
   repeat steps 2 and 4 until all the documents are checked by the stop word
7. end if
8. go to step 3
9. go to step 2 and then to step 4
10. repeat steps 2 and 4 until all the stop words are checked against the documents
11. if one of the stop words do not exist in the document
    discard the document (leave it in its original storage)
12. End document identification

Listing 5-1: Algorithm for Stop words-based language identifier

5.2 The Tigrigna Page Indexer

This is the indexing component that takes results from the Tigrigna pages' repository for further processing. Since the pages (documents) on the repository cannot be used directly for indexing and searching, this component has sub-components which have typical responsibilities. The detailed architecture of this component and its indexing process is depicted in figure 7 and each sub-component is described below.

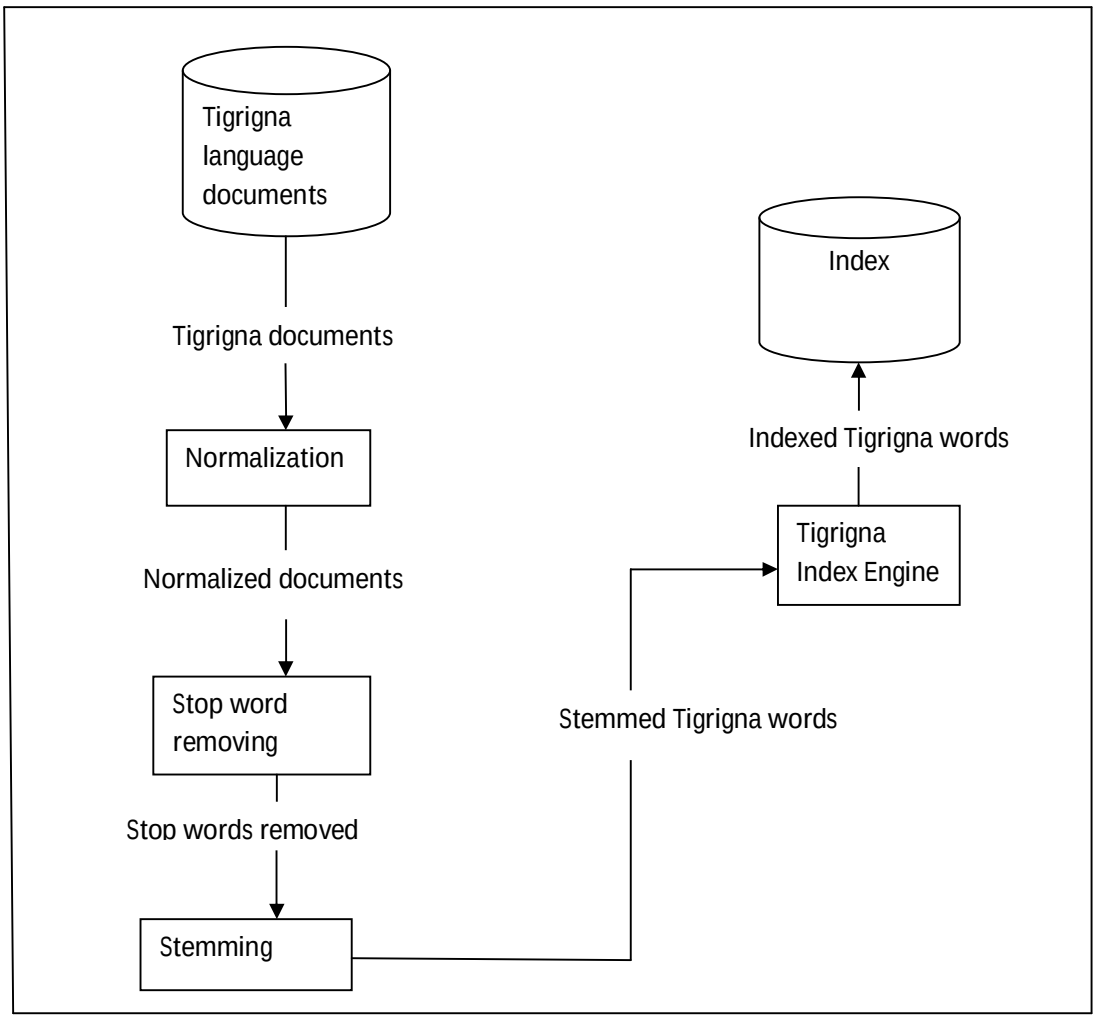


Figure 5-7: Tigrigna Pages Indexer Component.

5.2.1 Normalizer sub-component

Since the analyzer component is responsible to handle the characteristics of Tigrigna language, this sub-component deals with short forms of compound and single words separated either by “.”(Period) or “/” (slash), tokenizes Tigrigna language words based on whitespace and Tigrigna language punctuation marks.

The algorithm used for the normalization sub-component is shown in Listing 2.

While (Not end of the document)

1. Read a character
2. If the character is one of Tigrigna language punctuation marks, white space and other alphabets

   Continue
3. Else if the character is “/” or “.”

   Read the character before “/” or “.”

   If the character is found in the short words’ storage

   Replace the short form with its expanded form

   Else

   Continue
4. End If

End while

Listing 5-2: Algorithm for the Tigrigna Normalizer sub-component

Short forms can be that of compound or single words. For example ቤት ፍ/ዲ is the short form of the compound word ቤት ፍርዲ and መ/ር is the short form of the single word መምህር. So, this component handles these issues. Indexing is very important task because it minimizes memory consumption and fastens the retrieval system by putting documents in a format appropriate for a searching process. The general operation of this algorithm is described in a flowchart as follows.

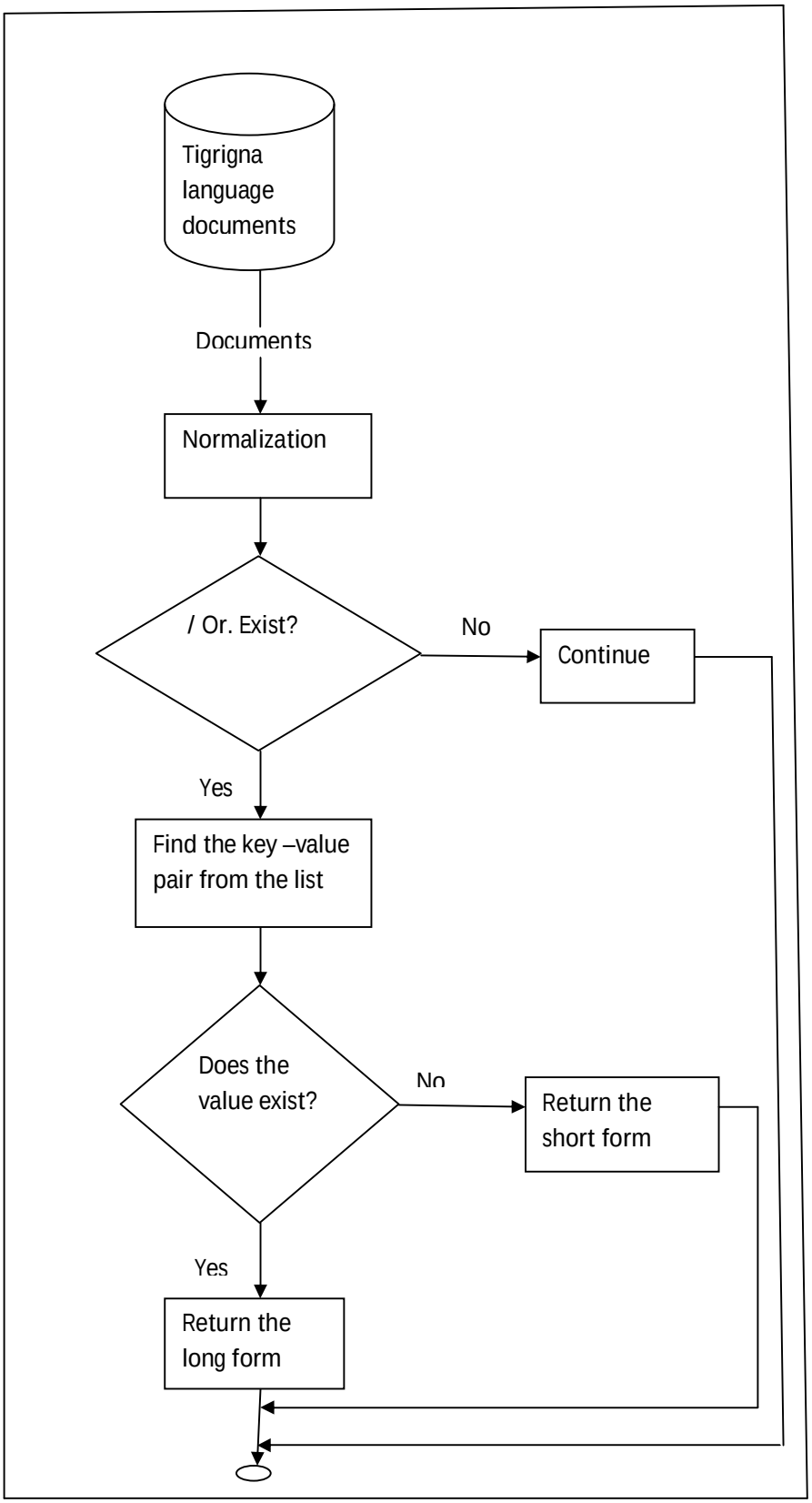


Figure 5-8: Flowchart of the Normalizer sub-component.

5.2.2 Stop words remover sub-component

Stop words are features (characteristics) of natural languages that don't have that much significant meaning in a given document. Due to this, Tigrigna stop words have to be removed so as to save disk space and speed up the searches. Therefore, Tigrigna stop words remover sub-component is needed for this purpose and its algorithm looks like the following.

1. Open the document
2. While (Not end of the file)

   Read the character

   If the character is found in the stop words list

   Remove it

   End if

   End while

3. Go to step 1 and repeat

Listing 5-3: Algorithm for the Tigrigna stop words remover sub-component

The general operation of the algorithm is given in the flowchart below.

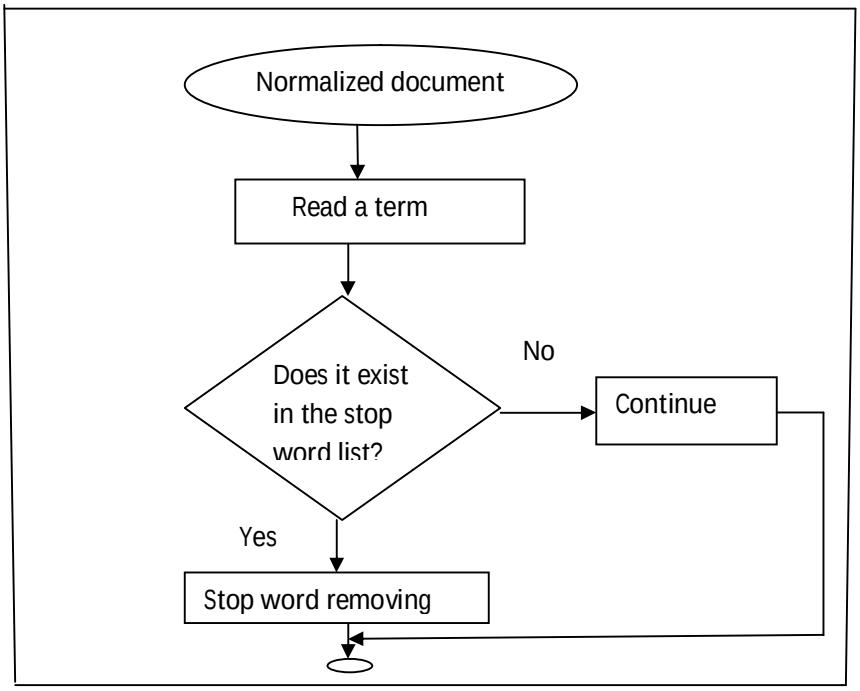


Figure 5-9: Flowchart of the Stop words remover sub-component.

5.2.3 Stemming Sub-component

As any natural language, Tigrigna is morphologically rich language. As a result, these variants must come to a common form so that the retrieval will be efficient. This is to mean that one Tigrigna word can have a lot of variations to mean the same or nearly the same thing. So, these variations must be brought to a common representation to get the same or nearly the same result while fetching from the web. In order not to make the stemmer busy, stop words must be removed first before applying the stemming task. Therefore, because stemming is the immediate task of stop words removal, we have implemented Girma's stemming algorithm by making the modifications described in the Tigrigna retrieval system section. In addition to bringing word variants to the same (common) form, stemming saves disk space and speeds up the retrieval process. The stemming algorithm works as it is described by the following flowchart.

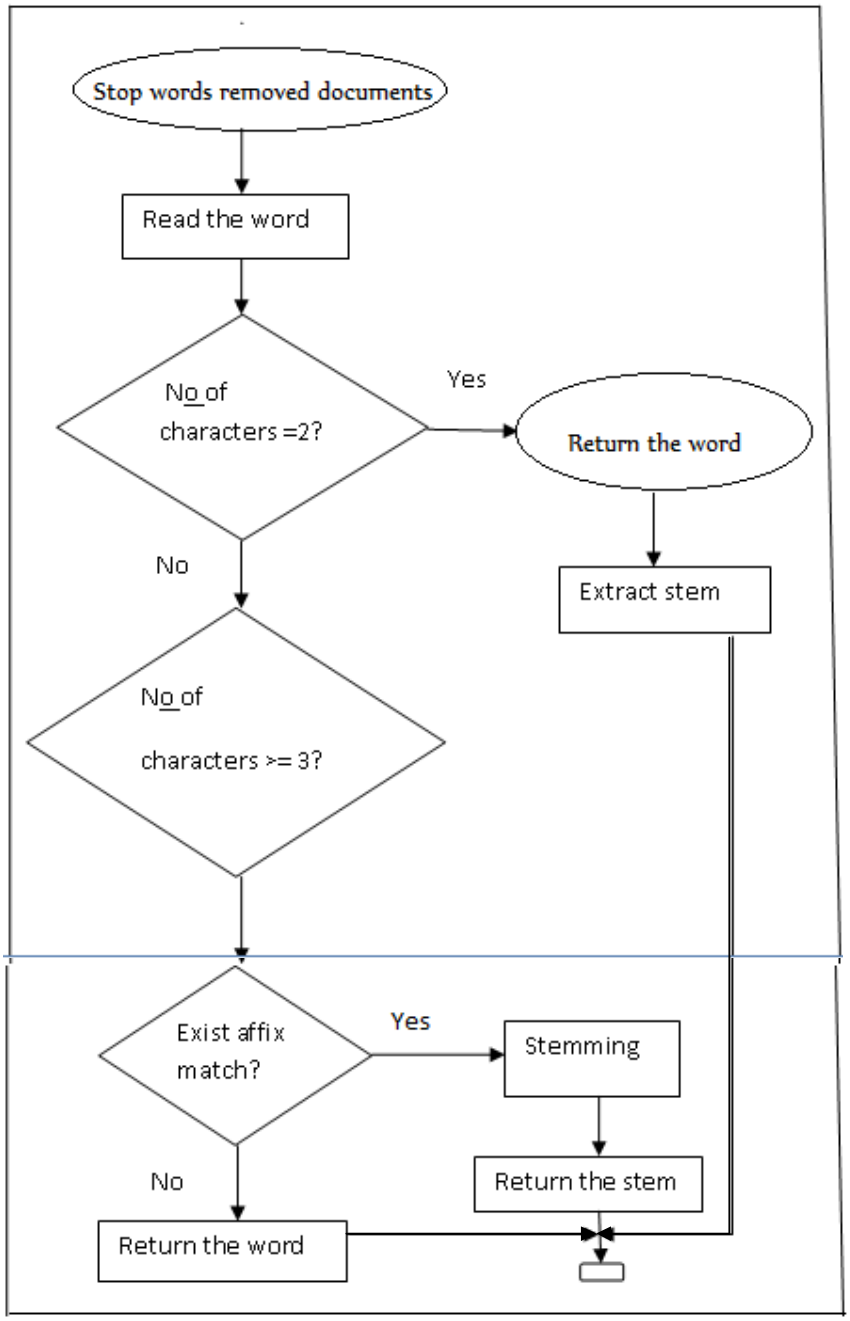


Figure 5-10: Flowchart of the Stemmer sub-component.

5.2.4 Tigrigna Index Engine sub-component

Indexer is a catalogue that represents the identified documents in a way suitable for searching. The above mentioned tasks are performed consecutively according to the order they are put and this sub-component puts the analyzed keywords in the index database.

5.3 Query Engine component

Query engine takes a description of a search request given by the user and executes the request and returns the results back to the user by displaying on a separate display interface. This component acts as an intermediate layer between clients and the underlying data sources by interpreting search requests and shielding the clients from details on how to access the data sources [47]. This component helps the user directly interact with the retrieval system (Tigrigna search engine) by letting write queries on the search interface and displaying the results of the search query on another interface. Therefore, the query pre-processing is shown in figure 11.

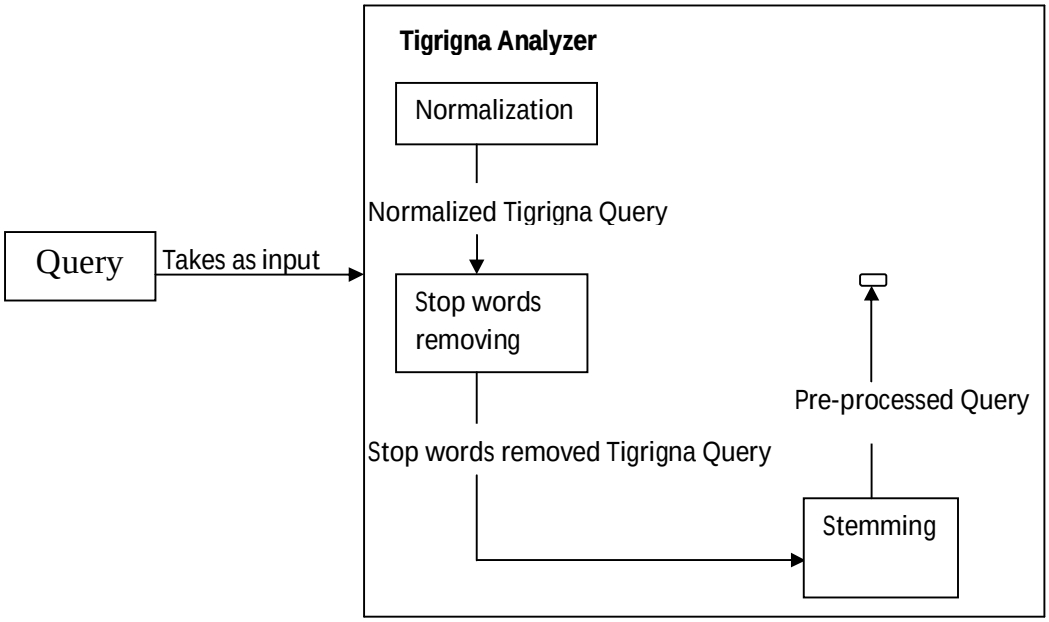


Figure 5-11: Query Pre-processing.

Because the user queries must be analyzed before searching, the Tigrigna language analyzer which consists of the normalizer, stop words remover and the stemmer are found in both the Tigrigna indexer and query engine components. So, the Tigrigna analyzer located on the query engine side expands Tigrigna language short form words (using the Normalizer), removes the language's stop words (using the Stop words remover) and stems the words (using the Stemmer) that are found in the query term so that the index will be contacted to display the query results to the user in some order of relevance using the ranker. In order to judge the effectiveness of the analyzer, we have used (checked against) ዘበናዊ ሰዋስው ቋንቋ ትግርኛ ፣ 2000 ዓ/ም by Daniel Teklu [41] and መዝገበ ቃላት ትግርኛ ፣ 2000 ዓ/ም by Dr. Kassa G/hiwot and Emaniel Gankin [60].

Chapter Six: Implementation of Tigrigna Search Engine

This chapter discusses the development environment, tools used and implementation of the prototype of the Tigrigna search engine whose design is proposed in chapter five.

6.1 Development Environment and Development Tools

The Tigrigna search engine prototype is developed in a single personal computer of Intel Pentium IV Processor with 1.80 GHz speed, Microsoft Windows XP Professional operating system, 80 GB Hard disk capacity, and 256 MB of RAM.

For the development of the search engine, we have utilized the following tools.

JDK1.5_0.6: is an open source software development kit (SDK) used to develop java programs. Since all components of our search engine are developed using java programming language, JDK (java development kit) version 1.5_0.6 is utilized and installed on windows environment for implementing the various components of the search engine.

JSpider: It is a java based open source tool used for crawling documents on the web. We have used version 0.5.0 for our purpose and it is well discussed in the overview of JSpider section.

Apache-tomcat-5.5.17: is an open source java based web container software implementation of the java servlet and JSP technologies. It handles HTTP requests and returns response from different components. So, apache tomcat version **5.5.17** is used for our purpose.

Lucene-3.5: Lucene is an open source java based library. It is a high performance text search engine library that is suitable for full text search. Lucene has its own distinguishing features such as [52]:

- Scalability
- High performance
- Fast indexing
- Ranked searching
- Small RAM requirements
- Support phrase, wildcard and range queries
- Date-range searching sorting by any field
- Multiple-index searching.

In addition to the above mentioned features, lucene takes very few (milliseconds) search duration and is used for indexing more than 4 million documents in less than 100 milliseconds [39]. It is also characterized by data structures, named, inverted index (for indexing), vector space model for considering partial

matching and supporting Boolean operators in queries. Therefore, we have used lucene-3.5 for indexing and searching of Tigrigna language documents.

Eclipse: is a multilingual software development environment that is used for writing, compiling and running programs [61]. It can be used to develop applications in java, C, C^{++}, PHP, Python, Perl, etc although it is written in java. It consists of java development tools and Eclipse compiler for java. Therefore, we have used it for writing, running and compiling the java and JSP codes for our search engine.

LIM: stands for Language Identification Module and is an N-gram based tool written in java to identify the language of the web documents. It is developed by the Language Observatory Project of University of Nagaokaut (Technology University in Japan) to identify language, script and character encoding of a given language's documents on the web [50].

6.2 The Tigrigna Page Crawler

A crawler, as a major component of any search engine, is a program that retrieves web pages for use by a search engine [46]. The crawler starts fetching the web from seed URLs obtained from the queue and repeats the process until the queue becomes empty or the crawling process is interrupted (forced to stop) by the programmer. After crawling process is finished, the downloaded documents (pages) are saved in some repository so that they will be used for further tasks of the search engine. Today, since the web contents are dramatically growing and it is difficult to scan the whole web, different open and closed source crawlers are available on the web. These crawlers enable users to fetch pages of their interest on the web by making some modifications (customizations) on them. To fulfill our interest, we have reviewed the below mentioned open source crawlers.

The Amharic crawler is a focused crawler intended to fetch Amharic web documents available on the web. It was proved that this crawler was able to download documents specified by the seed URLs. As it is tested and discussed by Hassen Redwan [39], the Amharic crawler is slow and single threaded.

Heritrix is the other reviewed crawler. It is an open source Java based crawler which is published under the GNU Lesser General Public License (LGPL). Heritrix is designed in a highly objected-oriented manner. Heritrix is designed in such a way that it allows the developer to change or add source code in an easy way such that various crawlers can be realized based on the intention of the user. This means, it is possible to exchange standard features in a crawl by non-standard or even self written modules which make Heritrix really adaptable to any goal that a user wants to reach. Although this crawler is an open source tool, it is still actively being developed [48].

The researcher has also reviewed a crawler, called JSpider. JSpider is a highly configurable web crawler that lets us prevent pages on our web site not to be spidered if we like. It is pure 100% Java open source tool developed under LGPL open source license. JSpider enables us to:

- Check our site for errors (internal server errors)
- Check outgoing and/or internal link
- Download complete web site, single page, some specific pages, etc.

Therefore, JSpider is easily configurable, scalable, multi-threaded, Java based and open source tool. Besides, it easily checks web site errors within minutes (if the URL is wrong or resources are missing), lets us add our own configurations and rules, and its spidering speed is fast and configurable [49]. Taking the above discussed issues of the web crawlers, we have chosen JSpider for our purpose. This is because:

- The Amharic crawler is slow and single threaded
- Heritrix is not stable (still being developed)
- JSpider is the stable release; easily customizable, multi-threaded Java based and allows users to add any configuration or additional module they like.

Overview of JSpider.

As it is diagrammatically described below, JSpider has three main components.

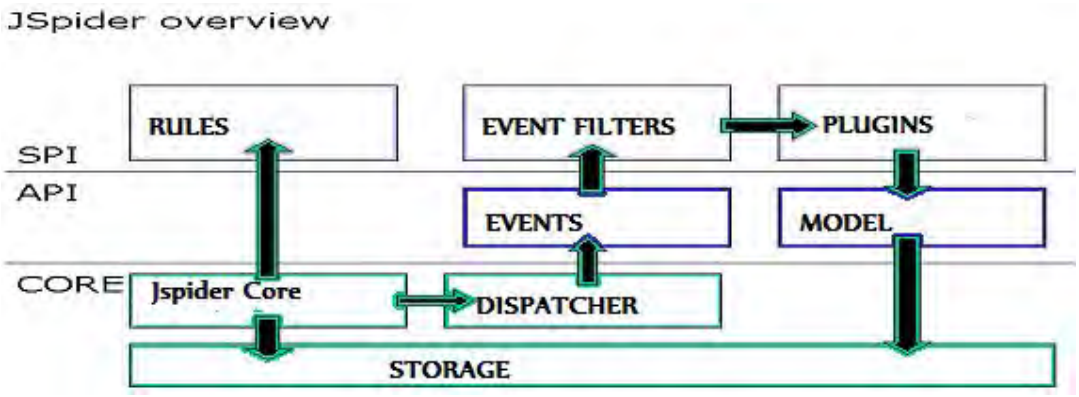


Figure 6-12: JSpider (Source: [49]).

The Engine Core: is the main part of JSpider that implements the basic functionality delivered by JSpider.

Service Provider Interface Components (SPI): These components implement the real application functionality that adheres to certain SPI interfaces. JSpider comes with a number of implementations of these components, and anybody can also implement his/her own. These are actually used to extend and add functionality to JSpider. Some of the most commonly used SPIs by JSpider are Rules, Plugins and Event Filters.

Rules: are used to decide which resources to be fetched and/or processed by a JSpider. We can define JSpider's behavior and scope by constructing a set of rules on the global or per-website basis. So, there are a lot of rule implementations that come with JSpider and we can also develop our own rules.

Plugins: are components that are notified of certain events happening and can then take appropriate actions. The actions are:

- Writing a report file
- Displaying a message on the console
- Writing a fetched resource to disk, etc.

We can also implement our Plugin and add the functionality of the JSpider. For example, we can construct a configuration in which JSpider tests a certain site for 404 errors (link errors), and send an e-mail with all error links to the webmaster and mirror a website on our local disk by enabling a plugin that writes every fetched resource into a file or hard disk.

Event Filters: select the events that have to be handled by the system as a whole or a particular plugin.

API components: the JSpider API consists an object model and event system.

An object model represents anything like sites, resources, content, cookies, and references between resources (URLs) that JSpider encounters during spidering.

An event system includes Engine events, spidering and monitoring event classes that will be used to notify all interested plugins of certain events happening during the spidering progress.

- Engine events include: spidering started, stopped, configuration chosen etc.
- Spidering events include: sites discovered, resource spidered, fetch error, etc.
- Monitoring events give information about the spidering progress and the thread pool occupation

Therefore, because JSpider is found to be best of all we reviewed, it is used as our crawler by making the necessary configurations. The configurations made on the tool are found in Appendix VII.

6.3 Text Extraction

Pages on the web can contain text, audio, video, image, etc. The page can also be in different formats, such as text, Microsoft word, excel, HTML, PDF, RTF and XML. Because we are using lucene for indexing the crawled documents and the lucene indexer does not index documents that are not plain-texts [52], it needs helping tools that extract plain-texts from different document formats. The reason that lucene index needs third-party tools is that because it does not have its own that does this task. Therefore, because most of the web documents are found in HTML, PDF, Microsoft word and text formats, we have reviewed open source

text extraction tools for the above mentioned document formats only. For this purpose, NekoHTML, PDFBOX, and wordextract are used. These are briefly discussed below.

6.3.1 PDFBOX

PDFBOX is free open source library that extracts text from PDF format web documents. PDFBOX version 0.7.3 is used for this purpose because it is Java based library that includes classes that work with lucene particularly [52].

6.3.2 NekoHTML

HTML is the most popular file format on the web. As a result, because the document may hold plain-text, image, audio, video, etc. a plain-text must be extracted so as to be useful for indexing (it can be indexed). To choose a tool that can be used for extracting a plain-text from HTML format documents, we have reviewed **Tidy** and **NekoHTML** and chosen NekoHTML version 0.9.2 because NekoHTML is java based open source tool that has classes that work with lucene particularly and **Tidy** is implemented in C [52].

6.3.3 wordextract

Microsoft file format documents are also common on the web. Such documents may contain other data in addition to the plain-text, but in our case the web document we need is a plain-text. We have reviewed wordextract and POI third-party tools that can extract text from Microsoft word file format documents. Although both are used for extracting text from Microsoft word file documents we have chosen wordextract version 0.4 due to the reason that [52]:

- wordextract is simple and optimized for extracting
- wordextract supports extracting text from word 6/95 whereas POI does not.
- POI can extract excel and MS word documents, but what we need is MS word format documents.

6.4 Tigrigna Web documents Identifier (Categorizer)

The web contains different types of data in different languages. Because the web is a huge repository and it is difficult to browse the whole web, language specific search engines play a great role. The heart of such search engines is the language's documents categorizer. A given language's documents categorizer, also called language identifier, is a component of the crawler that assigns a given document to a given category. For example, in our case, Tigrigna language documents identifier collects documents that belong to this language. To achieve this goal, we have studied open source language identifier modules, such as G2LI and LIM. We have also proposed a method of supporting Tigrigna language identifier modules with stop-word based language identifier.

The first two are obtained from the web and the last one is developed by the researcher. So, each of them are discussed below.

G2LI: is an N-gram based language identifier that identifies html and text file format documents only [51]. G2LI stands for Global Information Infrastructure laboratory language identifier developed by LOP (Language Observatory Project) at the University of Nagaokaut. This tool is used for identifying web documents of a language in which it is trained for. It can be trained by a string, file and group of files to identify Language, Script and Encoding [51]. Finally, we could not use it due to the reason that:

- It can identify html or text file formats only
- It is not usable i.e. although G2LI can be found in the web, there is no any way to integrate it with the crawler since the source code is not accessible.

LIM: is the other open source tool studied in our study. It is a program which stands for Language Identification Module. Like the above mentioned tool, it is released by the Language Observatory project to identify a language's documents. It is fully developed by Java programming language and it can identify Language, Script and Encoding of the target language [39]. The identification process is an N-gram based identification. We have first trained LIM with the UDHR translation Tigrigna document. When LIM is trained, it stores the shift-codons of the Unicode Encoded Ethiopic Script Tigrigna language documents because the training datum is a Unicode Encoded document. After we have trained it, we have tested it with Unicode and non-Unicode Tigrigna documents and it correctly identifies the Unicode Encoded documents. It also identifies Tigrigna language documents from other documents correctly. In a multi-lingual situation, LIM identifies the language's documents if the document's content for the language is 60% and above as it is stated in the programmer's documentation. At last, the problem we have encountered was, LIM cannot differentiate Tigrigna and Amharic language documents if they are in the same Encoding. This can be justified by the following screen shot which shows Amharic language Unicode encoded documents classified as Tigrigna language documents.

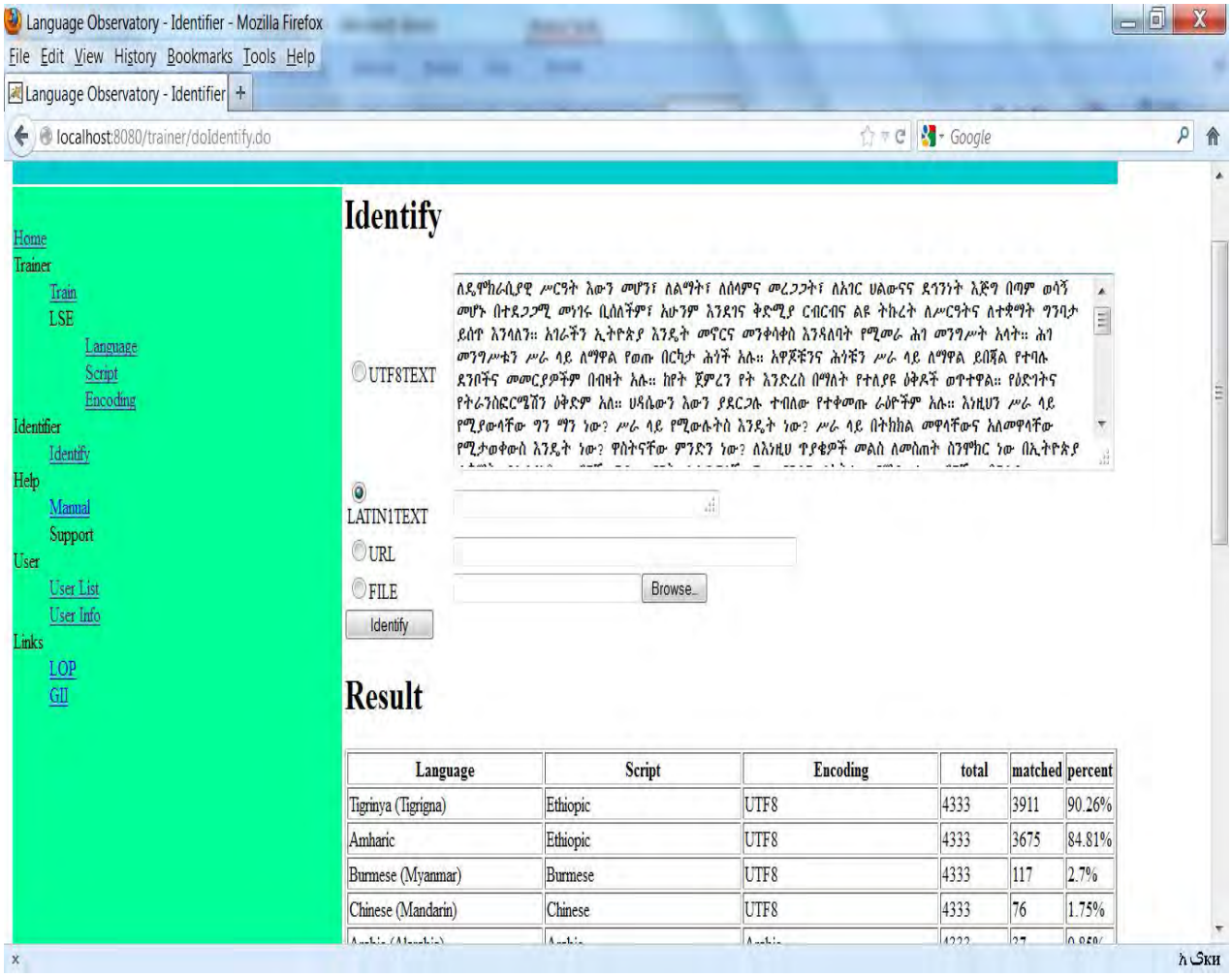


Figure 6-13: Screen shot of the result of LIM using Amharic text.

Tigrigna stop words based language identifier

As it is stated above, the problem with LIM was to identify Tigrigna language documents from Amharic language documents of the same Encoding. As a result, we have been initiated to find another mechanism. The solution we have found was collecting Tigrigna language stop words which are not found in Amharic language documents and developing a language identifier that identifies (categorizes) documents that contain at-least one of the stop words. So, after implementing this categorizer, we have tested it by using Tigrigna and Amharic language documents and correctly identifies Tigrigna language documents from the collection. Although it is very rare, documents in the web can be multi-lingual and the proportion of the target language's documents can be very small (example 10%). This percentage invalidates the language specificness of the search engine. Therefore, in a multi-lingual document the proportion (60% and above) must be kept. Due to this, we have decided to use both LIM and the Tigrigna language stop words based language identifier to categorize Tigrigna language documents so that these can be used for further processing.

6.5 Indexing

Once Tigrigna web documents are collected from what are downloaded, they must be put in a way suitable for searching to ensure the efficiency of searching. So, this component plays a pivotal role in putting the documents in a manner appropriate for searching. We may need to search large documents from a given repository. But processing the whole document at the time of searching is time and space intensive. In order to minimize this problem, large documents must be logically represented. To search large amounts of texts quickly, the documents must be first indexed and converted into a format that will let searchers search it rapidly, eliminating the slow sequential scanning process [52]. This is what indexing is and the output of this process is an index. Before indexing is conducted, there are other text analysis tasks to be done in any IR system. Because Tigrigna indexing component is one major component of Tigrigna search engine, normalization, stop words removal and stemming tasks are first performed before indexing. These tasks are done based on the features of Tigrigna language. For indexing Tigrigna language documents, we have used lucene index. In this component, the main task is to integrate the languages' features with that of lucene. As it is stated above, before indexing the crawled pages directly, analysis tasks must be first performed. These tasks are discussed below.

6.5.1 Normalization

Normally, normalization handles short forms of words and repetitive alphabets. So, this section discusses about handling single and compound short forms of Tigrigna language words abbreviated by either slash (/) or period (.). It is common to see alphabets that have the same usage and sound, but different structures in languages like Amharic. The Tigrigna language spoken in Ethiopia does not have this nature. As a result, Tigrigna language normalizer expands Tigrigna language short forms that are separated by either period like ር.ምምሕዳር, or slash like ር/ምምሕዳር to mean ርእሰ ምምሕዳር 'head of state' so that the result of searching using ር/ምምሕዳር (ር.ምምሕዳር) and ርእሰ ምምሕዳር will be the same in number and type. To realize this goal, we have collected 56 Tigrigna short form words from Tigrigna books by consulting (showing) Tigrigna language experts. These short forms are represented by Unicode values taken from the Ethiopic Unicode table. Unlike the Amharic language short form words, some compound Tigrigna language short form words, like ቤት ት/ቲ, have a space between the two words. Others like ዕ/ሓሙስ do not have space in between, but they need a space after they are expanded. Therefore, we have implemented Tigrigna language short form words expander by classifying the words into three classes.

The first class is the class that stores an array value of the short words that have the same beginning alphabet but different values. This means short words that have the same beginning alphabet are put in one list and others in another list.

The second class contains and implements the short words that do not have a space before or after "/" or ".", but that needs space after expansion. For example, the short form ሓ/ዓሰርተ does not have space and it becomes ሓለቓ ዓሰርተ when it is expanded.

The other class handles short forms that have a space between their words (when it is compound), but does not need a space after expansion. For example, ቤት ት/ቲ is a compound word that has a space before expansion and it becomes ቤት ትምህርቲ after expansion. It does not need space after expansion means there is no space between the characters ት and ቲ (between the characters before and after "/" or ".").

In all the above cases, we have a data structure that stores the beginning alphabet (ፊደል) as a key and its expanded form as a value. So, what our algorithm implementation does is, it first checks the existence of "/" or "." and if it exists, it checks the characters (ፊደላት) before and after the symbols (/ or.). If the ፊደላት match with what is put in the data structure, it returns the expanded form of the word otherwise the short form will be returned as it is. The list of the collected Tigrigna language short form words is found in Appendix I.

6.5.2 Stop words removal

Stop words are common in every natural language. These are common words that carry less significant meaning than the keywords in a document. Since these stop words consume memory space and decrease the efficiency of the IR system by slowing the searching speed, search engines usually remove these words from a keyword phrase to return the most relevant result [53]. Therefore, removing the stop words saves space and speeds up the searches. As soon as a textual data is extracted from the crawled documents and identified as Tigrigna language documents, normalization activity is taken place. After this step, stop words are removed by employing the stop words remover sub-component because they hold insignificant meaning and removing these words saves space and increases the searching speed. Like any other natural languages, Tigrigna has several and frequently appearing stop words. For our search engine implementation, we have used Tigrigna language stop words (some of them collected by Girma [40] and the rest are from Tigrigna books and Tigrigna language documents). Tigrigna stop words remover first reads the stop words in a list and removes them from the document they exist. For this thesis work, we have collected 257 stop words as it is shown in Appendix II. This step again hands over its output to the stemming part (section).

6.5.3 Stemming

As it is discussed in the above sections, natural languages vary inflectionally or derivationally. This variation must be handled in some way so that the IR system will be efficient. This is achieved by employing a stemming technique. Therefore, stemming is the process for reducing inflected words to their stem [53]. This means the different morphological variants of a word are brought to a common form of the word to obtain the same or nearly the same result. For example, as Tigrigna language is one of the natural

languages that highly inflect, the word ገዛ which means house can inflect as ገዛና፣ ገዛውቲ፣ ገዛኺ፣ ገዛኹም፣ ገዛኽን፣ etc. The common representation for all these variants is ገዛ. As a result, these variations must be brought to a common representation 'ገዛ' so that the result of all the varying queries will be the same. To do this, language specific stemming algorithms (in this case Tigrigna stemming algorithm) are needed. Stemming algorithms reduce words into their stem by removing affixes (prefixes, suffixes, infixes and prefix-suffix pair). For our system, we have adopted Girma's [40] stemming algorithm by making some modifications to make it suitable for our purpose. Girma's [40] stemming algorithm was iterative five step algorithm and it did not consider Unicode encoded characters, i.e. it only considered the transliteration of the Unicode characters. Since most of the web documents are Unicode encoded and Java programming language represents characters in a Unicode, we could not use this as it is. The other reason is that, the stemmer did not identify the condition where a given affix is not an affix in a given word. For example, ት-'t' is a suffix in the word ፊደላት but not in ልምዓት, ም is a prefix in ምስራሕ but not in ምግቢ, ምሩፅ, etc. In addition to these, the original stemmer considers derivational morphology of the language like ገዛ is a 'house' and ገዛእቲ is 'buyers' or 'colonials' which has a totally different meaning and category from 'ገዛ'. Because Girma's [40] stemmer is iterative it over stems the given word and the first step (prefix-suffix pair removal) brings a change in meaning and category. Therefore, our stemmer considers all the above mentioned problems. In implementing the Tigrigna language stemming algorithm, we have represented the Tigrigna affixes by Unicode values taken from the Ethiopic Unicode table given by Unicode consortium. The Ethiopic Unicode table is in Appendix III.

6.5.4 Tigrigna Index Engine

Documents are indexed after accomplishing the analysis operations (normalization, stop words removing and stemming). Although lucene has its own analyzer (standard analyzer) that can be used for many languages (such as, German, English, Russian, etc.), it cannot consider Tigrigna language features. As a result, we have developed our own analyzer and this analyzer is given to the lucene's indexer in order the language's features be considered. The Tigrigna language analyzer expands short forms of compound and single Tigrigna words, removes Tigrigna stop words, stems the different morphemes of Tigrigna words into their stem and removes Tigrigna punctuation marks. After conducting these operations, indexing is taken place and an index is created. To perform indexing, we need the following classes [52].

IndexWriter: is a class that either creates or maintains an index. Its constructor accepts a Boolean that determines whether a new index is created or whether an existing index is opened. It provides methods to add, delete, or update documents in the index. IndexWriter provides write access, but it does not provide read and search rights.

Directory: is an abstract class that represents the location where index files are stored. There are two subclasses, namely, FSDirectory and RAMDirectory. The first one is an implementation of Directory that stores indexes in the actual file system for large indices and the second one is an implementation that stores all the indices in the memory and it is suitable for smaller indices that can be fully loaded in memory and destroyed when the application terminates. So, because we index large amount of documents and need to store the index in the file system directory, we have used FSDirectory.

Analyzer: is responsible for pre-processing (normalization, stop words removal and stemming) the text data and converting it into tokens stored in the index.

Document: Lucene document class is a collection of fields that allow us to add the fields that are needed to be indexed. It is a container for a set of indexed fields.

Field: Field is a class that contains classified information about the document. It has a name and a text value. A given document may have fields like Title, Author, date modified etc. In our case, this Indexer contains Title, URL and content as fields.

Lucene stores the input data in a data structure called an inverted index which is stored on the file system or memory as a set of index files. In our case we have used this data structure because [52]:

- It is used by most web search engines
- It lets users perform fast keyword look-ups and
- Finds the documents that match a given query.

To see the result of the indexing, we have used an index viewer tool called Luke and its snapshot looks like the following figure.

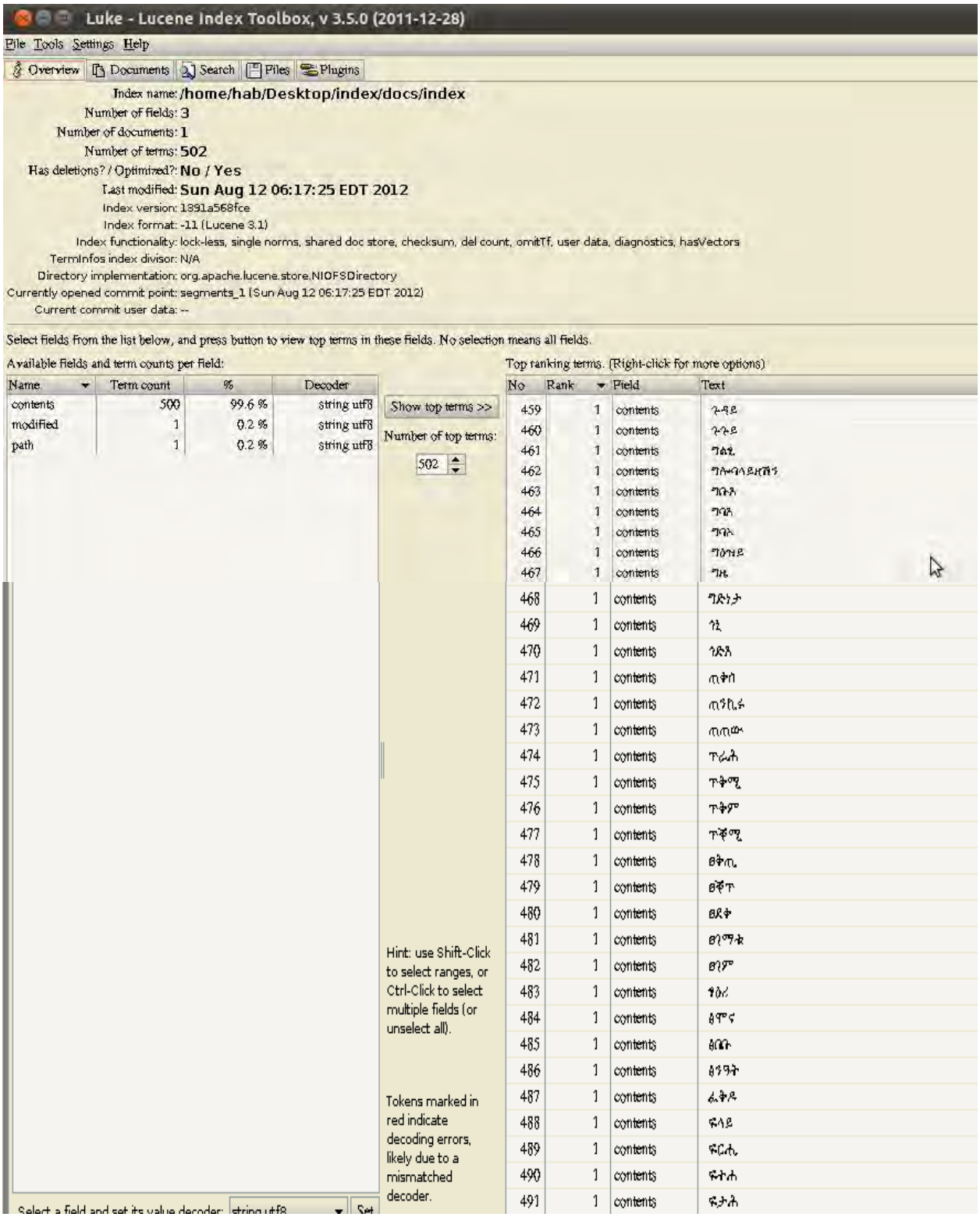


Figure 6-14: Screen shot of the Sample Indexing Result.

6.6 Query Engine Component

Query engine is one of the three major components of a search engine that contains query interface, Tigrigna language Analyzer, Ranker and result interface. Query interface lets users enter their queries (typically by using keywords). Because the analyzer used in the indexer part is used in this side also, the query is pre-processed. When matches to the keyword are found in the index, the documents that contain this keyword are displayed on the result display (interface) in some order so as to be viewed by the user. The documents are displayed according to their relevancy to the query. Therefore, our query engine is developed by using JSP (Java server page) and it is hosted on Apache server (apache-tomcat-**5.5.17**). It enables users to enter Tigrigna language queries and has a '**ድለ**' (submit) button that users submit their query to the system to obtain their information need. As soon as the Tigrigna language query is submitted, the query is pre-processed and the index is contacted. If the query has got a match in the index, the document that the Tigrigna language keyword is found will be displayed on the display interface. Lucene again plays a great role in this component. After the user entered query is analyzed, the lucene's IndexSearcher class searches for matches in the index [4]. The result of this searching is displayed in the result displaying component.

IndexSearcher is a lucene class that is used to search keyword matches in the index. Figure 12 shows the snapshot of Tigrigna search engine query engine component.

Figure 6-15: Screens shot of the Query Engine Component.

Ranking

Always, people need only to look at the fist displayed (at the beginning of the display interface) pages by a web search engine. To satisfy the users' interest, the retrieved documents must be returned in some order that brings documents that are important for the query term at the beginning of the display interface. IR systems like search engines, use the combination of text-based and link-based ranking to rank the retrieved relevant documents i.e. the sum of text-based and link-based ranking.

Text-based Ranking is a function that relies on the documents' contents other than on the internal relations among the documents. Text-based ranking is explained in terms of TF-IDF based ranking [54].

In the TF-IDF based ranking, documents and queries are expressed as vectors and their correlation is calculated using cosine similarity [54]. This method is used to find documents that are similar to the query by calculating the cosine of the angle between them. In our case, the text based ranking is calculated using the lucene scoring which is calculated by vector space model (term vector model) of lucene library. Vector space model (VSM) is an algebraic model that is used to represent text documents as vectors. In this case, documents and queries are represented as vectors. As a result, lucene calculates TF-IDF by using the cosine similarity by considering the angle between the document and query vectors. So,

$$TF - IDF = \mathrm{tf}_{t,d} \,.\, \mathrm{Idf}_t$$

$$= \mathrm{tf}_{t,d} \,.\log \frac{|D|}{|\{d \in D | t \in d\}|}$$

$$= \frac{freq_{t,d}}{max_{\{freq(w,d)\,|\,w \in d\}}} \,.\log \frac{|D|}{|\{d \in D | t \in d\}|}$$

where - $\mathrm{tf}_{t,d}$ is the term frequency of the term t in document d.

- $freq_{t,d}$ is the term count of t in document d.

- $max_{\{freq(w,d)\,|\,w \in d\}}$ is the maximum term count of d

- $\log \frac{|D|}{|\{d \in D | t \in d\}|}$ is the inverse document frequency

- |D| is the total number of documents in the document set

- $|\{d \in D | t \in d\}|$ is the number of documents containing the term t.

Therefore, lucene calculates the TF- term frequency- which is the frequency of the occurrence of the query term in the document and IDF- inverse document frequency- which is the number of documents in which the query term is found. Finally, the product of TF and IDF is performed by the lucene Score and the documents are ranked and displayed in a descending order of their relevance.

Link- based ranking is the ranking of documents based on the hyperlinks that connect them with each other [55]. Link-based ranking is calculated using the PageRank algorithm depending on the number of the incoming and outgoing links to/from the page as it is described in section 2.1.8.

Usually, the relevance of documents in a collection is measured by adding the text-based and the link based ranking methods (values). However, since lucene indexes and searches plain-texts only [52] and the downloaded documents cannot be directly used by the language identifier or the documents to be categorized by LIM should be plain-texts [4], text extraction is applied before the language identification. As a result, we could not calculate the link-based rank of the documents. Because the text extraction tools that we have used in our work remove the hyperlinks, images, audio, video and the tags, the hyperlinks in which the link-based rank is to be calculated disappear. As a result, we have used only text-based ranking to display the search results in the correct order.

Chapter Seven: Experimental Results

This chapter discuses the experiment conducted to test the effectiveness of Tigrigna Search Engine. The test environment used for was: a single personal computer of Intel Pentium IV Processor with 1.80 GHz speed, Microsoft Windows XP Professional operating system, 80 GB Hard disk capacity, and 256 MB of RAM. We have tested our system using 6Mb/Second bandwidth of the Addis Ababa University [56]. The experiment of the components is taken place in the order described below.

7.1 Test Result of the Crawler

To run the crawler, we have first made some configurations on the tool (JSpider). The thread and depth are the most important. As a result, we have made the thread to be 10 and the depth to be 10. Because as it is stated in the tool's developers' site, these are the recommended values. Having these and other configurations, the crawling process was conducted in Addis Ababa University, computer science postgraduate lab on the day time of August 25, 2012 for six hours by taking six Tigrigna sites as seed URLs. The URLs are:

- www.dmtsiweyane.com/ (website of Tigray regional state radio program)
- www.bahlitigrai.com/ (website of Tigray Culture Association)
- www.tigrayyouth.org/ (website of Tigray Youth Association)
- www.tdaint.org/ (website of Tigray Development Association)
- www.mekelecity.gov.et/ (website of Mekelle city fm radio or 104.4)
- www.ethiopia.gov.et/ (website of the FDRE government)

These sites are selected because they contain large number of Tigrigna language documents. The crawling process continued until all the URLs in the frontier are visited, i.e. until the frontier becomes empty. Finally, the crawling process shows that the crawler has processed 23,788 Pages and 858 of them are pages with Tigrigna language content. Although the contents in these sites are mostly Tigrigna language, majority of them are audio files. As a result, the identified Tigrigna language plain-text documents are 858. The sample report of the crawling process of the crawler conducted on August 25, 2012 is shown in Appendix VI.

7.2 Precision and Recall evaluation Results

As it is discussed in section 2.1.7, precision and recall are the effectiveness measures of an IR system. This is because all documents in the collection may not be retrieved and may not be relevant for the query. Both are defined in terms of the list of documents produced by a web search engine for a query and the list of all documents on the Internet that are relevant for a certain topic. Generally:

- Precision is the ratio of the number of relevant documents to the total number of retrieved documents i.e.

$$\text{Precision} = \frac{\textit{Retrieved relevant documents}}{\textit{Total retrieved documents}}$$

- Recall is the fraction of documents that are relevant to the query i.e.

$$\text{Recall} = \frac{\textit{Retrieved relevant documents}}{\textit{Total relevant documents in the collection}}$$

So, in this section, we discussed the effectiveness of our search engine by taking 96 documents and 11 randomly selected queries in terms of precision and recall as it is summarized in the following table.

Table 7-6: Precision and Recall evaluation Results

Query	Relevant Documents in the collection	Retrieved Documents		Relevant Retrieved Documents		Precision		Recall	
		AND	OR	AND	OR	AND	OR	AND	OR
ዛንታ ኪሮስ	2	2	3	1	2	0.5	0.67	0.5	1
ማሕበር ባህሊ ትግራይ	48	44	96	43	47	0.98	0.49	0.90	0.98
ምስላታት ትግረኛ	1	1	8	1	1	1	0.13	1	1
ቐዳማይ ሚኒስትር መለስ ዜናዊ	41	37	86	37	41	1	0.48	0.90	1
መግለፂ ሓዘን	38	36	57	36	36	1	0.63	0.95	0.95
ከተማ ሸራሮ	23	3	86	3	23	1	0.27	0.13	1
ፕሬዚዳንት ግርማ	4	1	4	1	4	1	1	0.25	1
ህንፀት ዓበይ ግድብ ኢትዮጵያ	24	24	71	23	24	0.96	0.34	0.96	1
ቃለ መሕትት መፅሄት ባህልና	33	25	61	25	32	1	0.53	0.76	0.97
ወረዳ እምባላጀ	21	21	41	20	21	0.95	0.50	0.95	1
ፅጉማት ተምሃሮ	3	2	30	2	3	1	0.10	0.67	1
Average						0.95	0.47	0.73	0.99

As it can be seen from the above table, the average precision for the operators AND and OR is 95% and 47% respectively and the average recall for the operators is 73% and 99% respectively. From this, it can be understood that AND operator gives better precision and less recall than the OR operator.

7.3 Testing of the Tigrigna pages categorizer

Due to the reasons mentioned in the implementation section, we have used LIM and Tigrigna stop words based identifier as Tigrigna language documents categorizers. So, this section discusses the testing approach used and the results obtained.

7.3.1 Testing of the stop words based identifier

The target of this page identifier is to find one of the Tigrigna language stop words in a document and if this is found to be true, it categorizes the page as Tigrigna language document. For this purpose, we have put 320 documents, of which 304 Amharic language documents, 5 Tigrigna language documents and 11 English language documents in one folder. We have also prepared an empty folder that receives the identified (Tigrigna language content) documents automatically when the program is run. Therefore, during testing this module, we have run our program and found only the five (5) Tigrigna language documents on the target folder. We can conclude that as far as the criterion is the existence of the stated Tigrigna language stop words, it is 100% efficient.

7.3.2 Testing of LIM

Before using LIM, we have trained the module using the UDHR (Universal Declaration of Human Rights) translation document. Because this text is Unicode (UTF-8) encoded, Ethiopic script document. Doing so, we subject this tool to categorize documents that are English, Tigrigna and Amharic documents. It is 100% efficient to identify that English document is not a Tigrigna language document. But it categorizes Amharic language documents as Tigrigna language documents if the Amharic language documents are Unicode encoded documents. Because we have trained LIM by Unicode encoded Ethiopic script, it does not categorize an Amharic language document as Tigrigna language document if its character encoding is out of UTF-8. So, this module is 100% efficient to identify the character encoding of the document in which it is trained for.

For multi-lingual web documents, we put some threshold in which LIM can categorize the document as the target language's document. Therefore, documents in which their Tigrigna content is above 60% (as stated in the categorizer section) are categorized as Tigrigna language documents.

When using these identifiers, we first apply LIM on the documents and then we check the existence of Amharic language documents using the Tigrigna language stop words based identifier.

7.3.3 Test Result of Tigrigna Analyzer

As it is indicated in the implementation section, Tigrigna analyzer handles only the inflectional variants of a word in the language although it is both inflectionally and derivationally rich. The inflectional variants considered in this testing section are the normalizer which handles short forms of Tigrigna words and the stemmer which stems Tigrigna words to obtain the key words in a document which is obtained from the crawler.

7.3.3.1 Test Results of the Stemmer

In the motivation section, it is stated that Tigrigna words that belong to the same category or saying something about the same thing give different results when they are used as a query in the general purpose search engines. This is because the general purpose search engines do not have a component that handles these features of the language. Therefore, in our search engine, this component treats this morphological variation of a word to get the same or nearly the same result of the variation. This is demonstrated below by using the queries “ባህሊ” and ”ባህልና” along with their respective screen shots, and the full contents are obtained from the web by clicking on the hyperlinks.

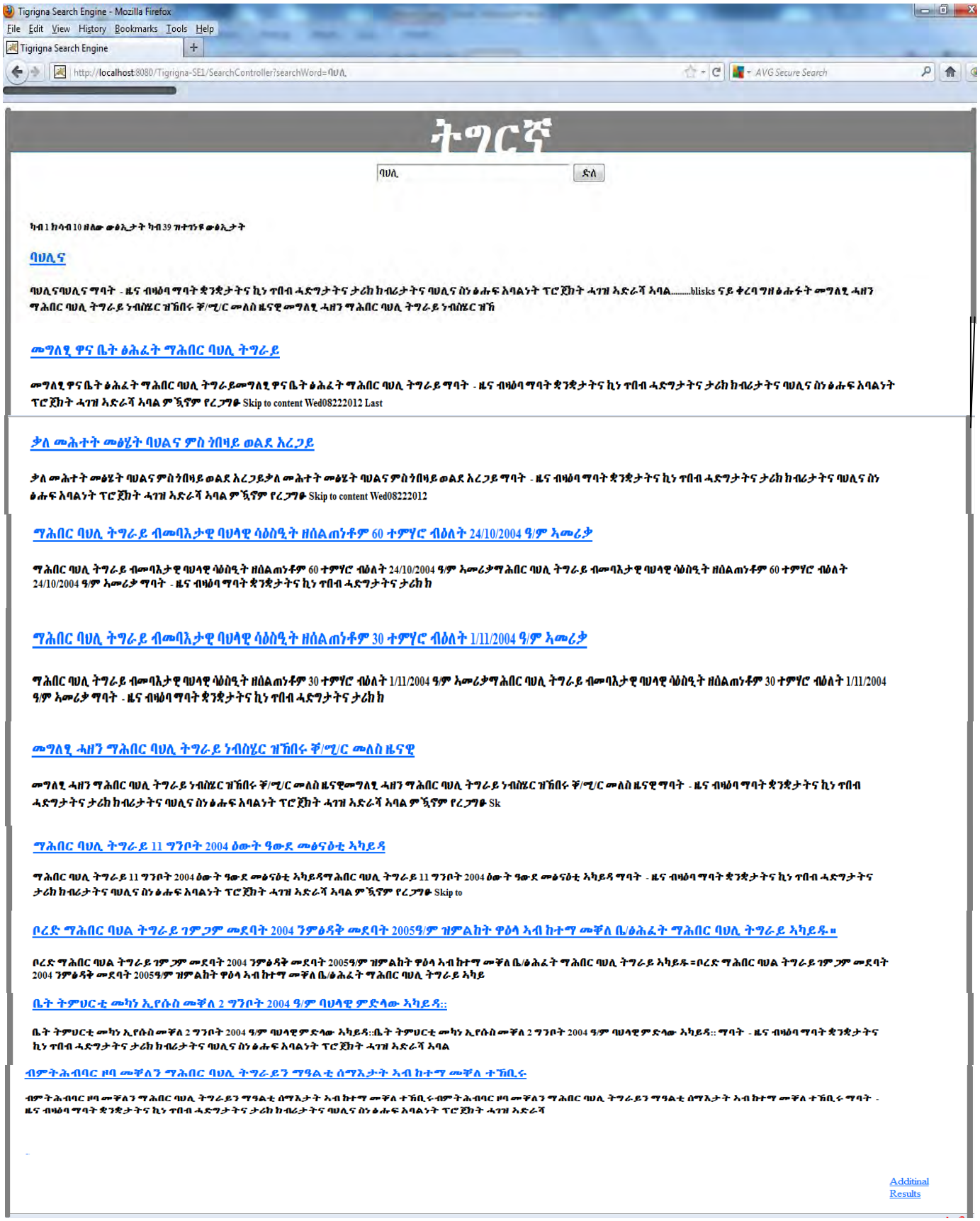


Figure 7-16. Screen shot of a result of the query "ባህሊ".

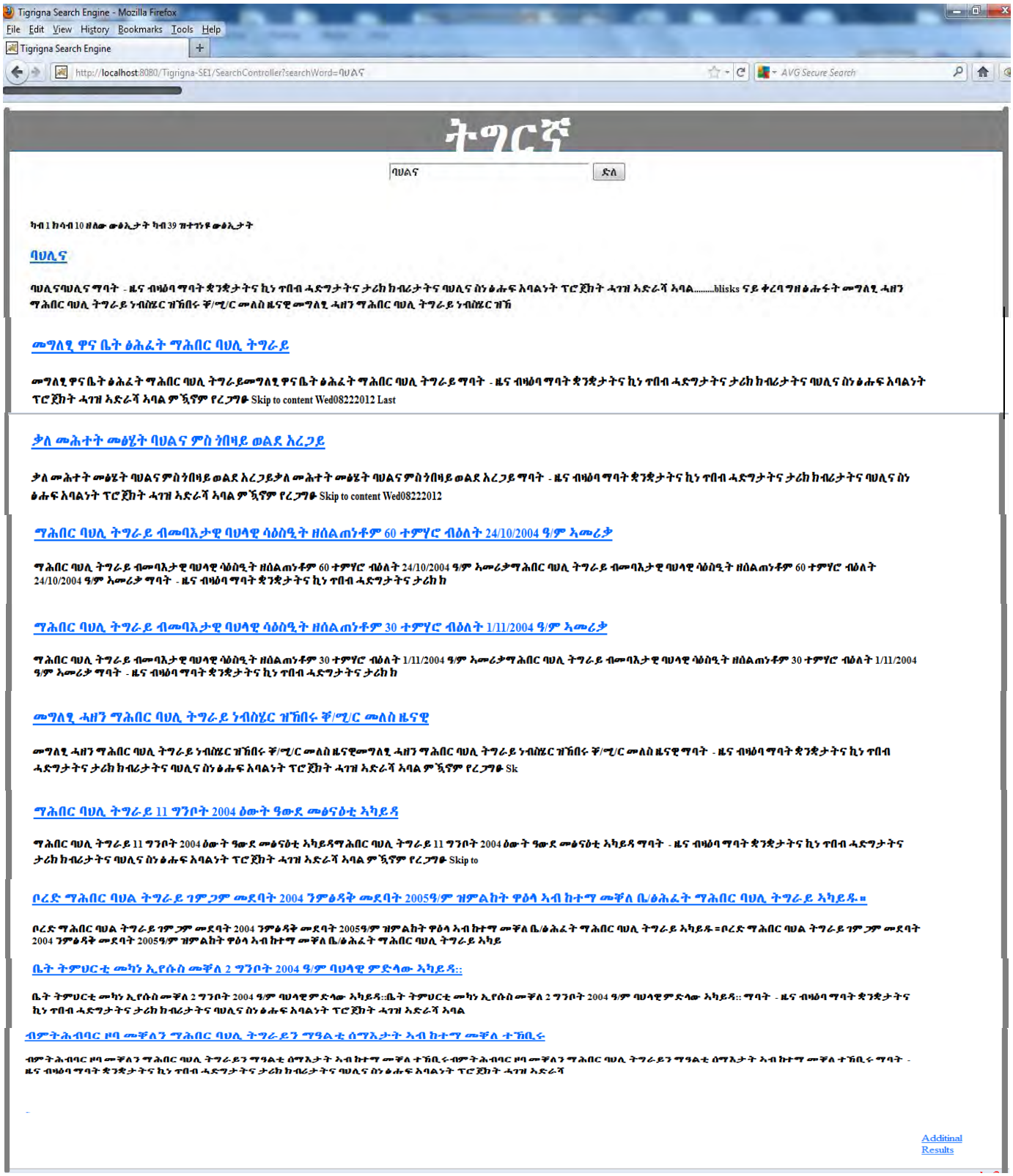


Figure 7-17. Screen Shot of a result of the query "ባ ህ ል ና ".

So, the above screen shots show that our search engine handles the inflectional richness of Tigrigna language.

7.3.3.2 Test Result of the Normalizer

General purpose search engines give different results for the short and long forms of the same Tigrigna word. This is because these engines do not have a feature that considers this morphological variation of Tigrigna words. Therefore, this component enables us to obtain the same query result of the long and short form of Tigrigna words by expanding the query into its corresponding long form if it is a short form. This is illustrated in the following screen shots by taking the queries ”ቤት ትምህርቲ ” , “ቤት ት/ቲ” and “ቤት ት.ቲ” and the full contents are obtained from the web by clicking on the hyperlinks.

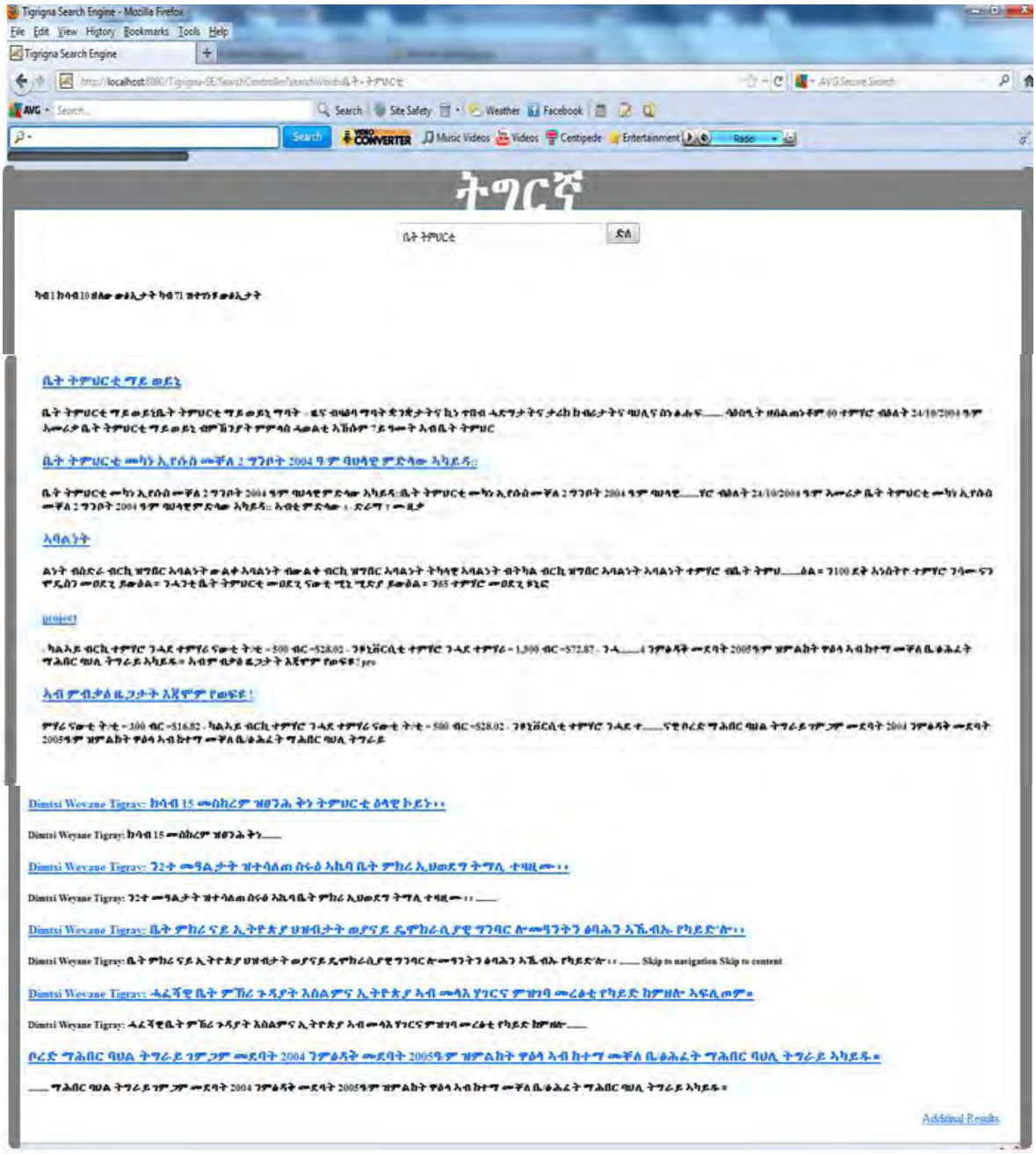


Figure 7-18. Screen Shot of a result of the query “ቤት ትምህርቲ”.

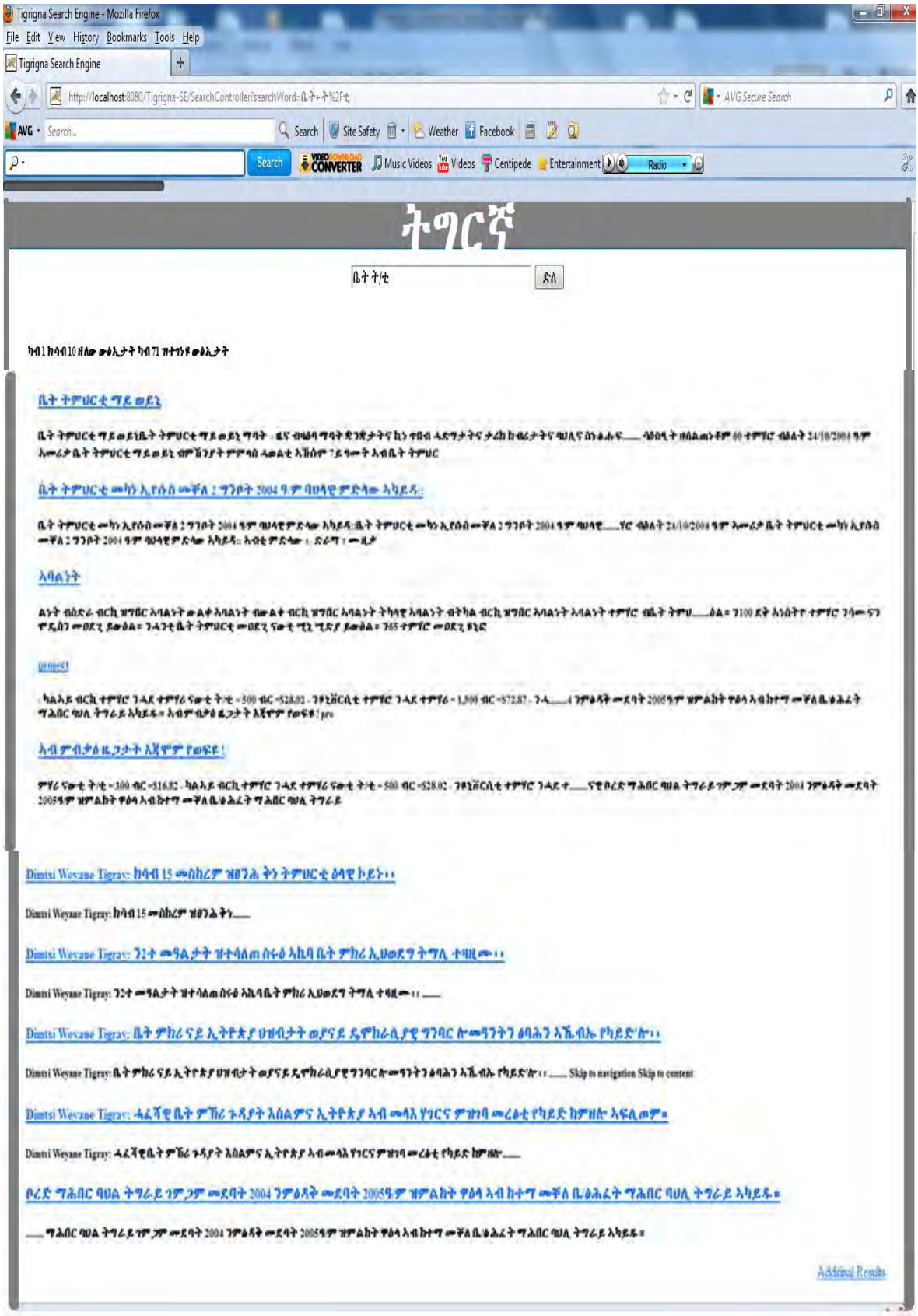


Figure 7-19. Screen Shot of a result of the query "ቤት ት/ቲ".

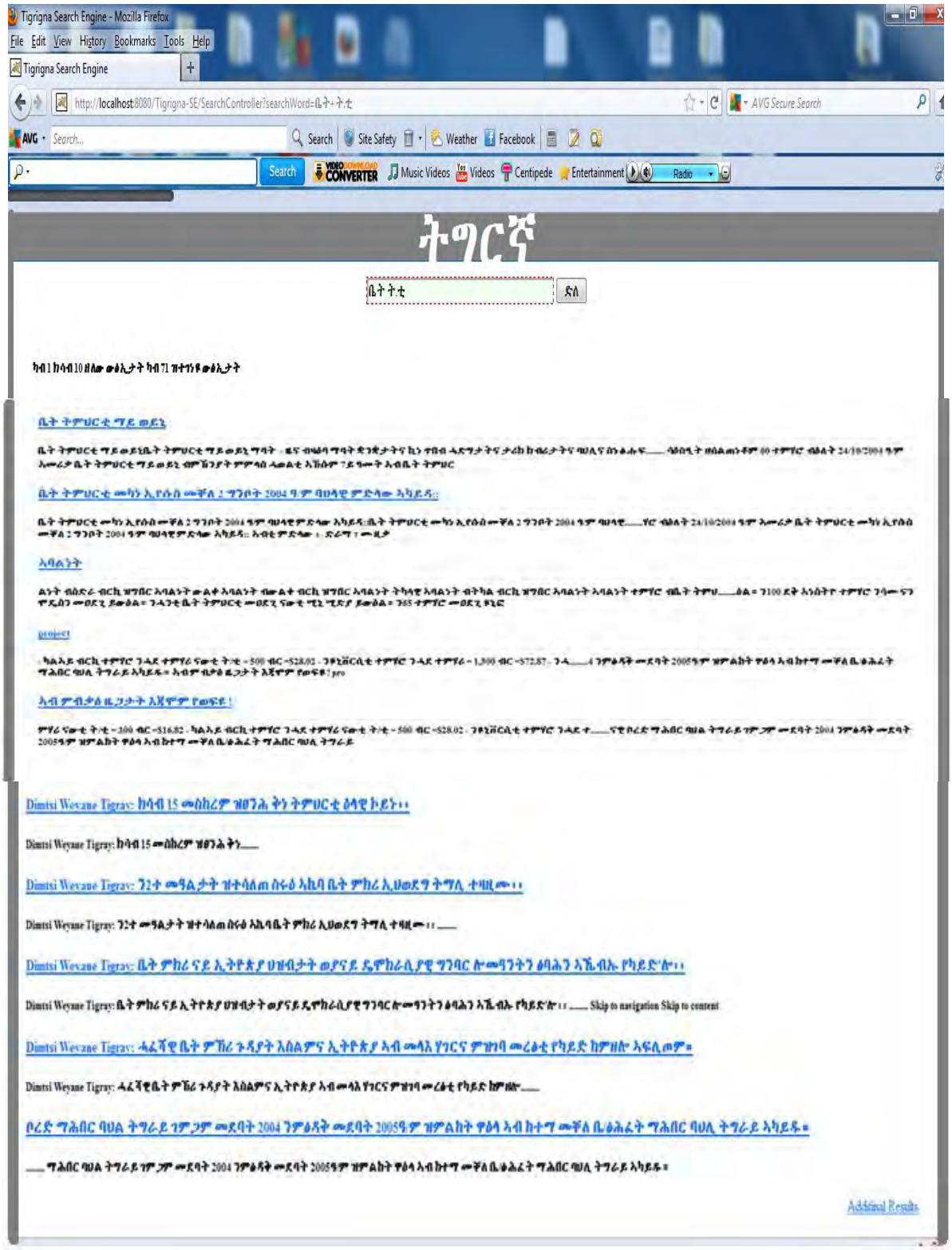


Figure 7-20. Screen shot of a result of the query "ቤት ት.ቲ".

As it is demonstrated in the above three figures, the query results are the same although Tigrigna language words can be abbreviated by either a period or slash.

Chapter Eight: Conclusion and Recommendations

8.1 Conclusion

Information is an important asset which enables people to perform different activities and decisions on their everyday life. Information can be obtained from different sources, but the web has become to be the most dominant and biggest source of information. The web is not only information repository, but also facilitates different web applications like E-commerce, tele-medicine, secured web based transaction, E-learning etc. Although the web is a huge repository for information, it is not enough by itself to provide the information it contains. This is to mean that, since the web consists of huge information content in different languages and in different type, and it is difficult to visit the whole web, there must be a program (software) that enables us fetch what we are interested in from the web. So, in this respect, search engines play a great role to make the information that the surfer needs accessible. Today, there are a lot of search engines that are capable of searching information from the web. But it is difficult to say the existing search engines are sufficient enough to handle the specific features of all languages' documents available on the web. The general purpose search engines are typically designed and developed to support the natural properties of English language. But, nowadays, different languages' documents are available on the web and users of these languages need to have the information they need in their language. Even though there is no statistically supported evidence that indicates how much Tigrigna language documents are available on the web, Tigrigna is one of the languages that are struggling to be available on the web. Therefore, because the general search engines could not entertain the morphological richness of the language, there must be a search engine that handles the morphological variations of Tigrigna language. As a result, we have designed and implemented a focused search engine that handles Tigrigna language characteristics.
As any other search engines, Tigrigna search engine has the three major components with their sub-components.

The crawler is one of the major components that consists the downloader, text extraction tools, categorizer and Tigrigna language documents repository. The crawler crawls the pages and puts the Tigrigna language documents in the repository through the help of its language identifier (categorizer). This component hands over the Tigrigna language pages to the second major component (Indexer).

The indexer component has sub-components to accomplish the analysis tasks. These tasks are performed in some order. So, the indexer component sub-components are the normalizer, stop words remover, stemmer and the Tigrigna index engine. The normalizer sub-component deals with expanding short forms of Tigrigna language into their long forms to obtain the same result when the user makes a query of one of these forms. Because stop words of any natural language carry less significant meaning, slow an IR system and consume a disk space, they must be removed. For this task, stop words remover sub-component plays a role. Stemmer

is the other component of the Tigrigna analyzer that stems different inflectional morphemes of Tigrigna language words into their stem. It removes the affixes which are attached before, after and inside the stem of the word. Tigrigna index engine is also another sub-component of the indexing component that creates the index (searchable form) of the Tigrigna language documents.

Query engine component is the third major component in which our search engine directly interacts with the user. It provides an interface that a user can write queries in Tigrigna language and the result of the query is also displayed to the user in the result interface. Since the user query must be analyzed before the index is contacted, the query engine contains the Tigrigna analyzer (the analyzer used in the indexing component is integrated with it). The analyzer integrated to the query engine contains the normalizer, stop words remover and the stemmer.

During the course of our work, we have used lucene for indexing and searching and JSpider for crawling.

During searching, all the documents on the collection might not be equally important. So, there must be a mechanism in which the query results can be prioritized. For this purpose, we have used a ranking component that displays the query results in a descending order. This means, the similarity of the query and the documents is measured by calculating the text-based similarity (ranking). Finally, the documents are displayed in the order of descending the similarity values.

Because the effectiveness of any IR system must be evaluated, we have used precision–recall matrix to evaluate the effectiveness of our search engine. Accordingly, we have obtained a precision value of 95% for AND operator and 47% for OR operator and a recall value of 73% for AND operator and 99% for OR operator.

The challenges we have faced during the thesis work were absence of standard references, dictionary, and list of Tigrigna language Stop words, Short form words, and Affixes.

8.2 Summary of our contributions

Nowadays, natural languages are becoming available on the web. The interest of users to obtain the web resources in the language they are most familiar with is also increasing. But the web technology is mostly suitable for English language resources. As a result, language specific web technology is very important to entertain the users' information need. As it is discussed in the motivation section of the study, non-English, specifically Tigrigna language, queries are treated differently in terms of type and number of the displayed results though the query is the same, but in different structures (forms), such as ቤት ትምህርቲ፣ ቤት ት/ቲ and ቤት ት.ቲ.

This is because the web technologies (search engines) are designed and implemented to handle English language queries. To overcome these problems, we have studied a Tigrigna Search Engine by contributing the following features.

- A Tigrigna language normalizer which handles Tigrigna language short form words so that the type and number of the result of the short and long forms of a given Tigrigna word, either abbreviated by period or forward slash, is the same unlike the results we have got from the general purpose search engines.
- Implementing Tigrigna language stop words remover in order to increase the speed of indexing and decrease the space consumption.
- Adopting a Tigrigna language words stemmer in a manner suitable for our purpose by making it to handle suffixes, prefixes, single and double letter reduplication so that the type and number of the results of a query in different morphologies is the same. Girma's stemming algorithm is adopted by making it to only treat inflectional morphology, Unicode encoding in addition to incorporating the languages' features (the languages' typical stem formation rules) and distinguish the conditions where some affixes are not always affixes.
- Developing an algorithm that categorizes Tigrigna language web documents in addition to LIM because LIM cannot correctly differentiate Tigrigna language web documents from Amharic language web documents if they have the same character encoding.
- In order to realize the above tasks, we have collected Tigrigna language affixes, stop words and short form words from various sources.

Finally, as it can be seen from the experimental results section, the differences in type and number of the results that we have seen in the general purpose search engines is solved and the results of the various morphologies is the same.

8.3 Recommendations

We have designed and developed a search engine that considers Tigrigna language characteristics although developing a full-fledged search engine is resource (time, money, people from different areas of study) intensive. As a result, the following recommendations can enhance the search engine.

- Due to the reason that there are Tigrigna words that irregularly vary, the Tigrigna stemmer must be researched in order to bring these irregular words into their stem
- Incorporating a component that handles non-Unicode encoded Tigrigna web documents.
- A study that enables to retrieve Tigrigna multi-media from the web.
- A research to handle regional differences such as ገረዚሃር, ገረዚቯር, ንብረእግዚኡብሄር etc.
- A study that handles Tigrigna short forms that contain more than one slash or period.

REFERENCES:

1. Peter Lyman and Hal R. Varian. How Much Information, 2000. Available at: http://www.sims.berkeley.edu/howmuch-info/; Last Accessed on September 2, 2011
2. Search Engines. Available at: http://www.webopedia.com/TERM/S/search_engine.html; Last Accessed on September 4, 2011
3. H. Moukdad, “Lost In Cyberspace: How Do Search Engines Handle ArabicQueries?” The 12th International World Wide Web Conference, Budapest, Hungary, 2003.
4. Tessema Mindaye, ”Design and Implementation of Amharic Search Engine”, Master’s thesis, Addis Ababa University, Department of Computer Science, 2007.
5. Tesfaye Guta,”Design and Implementation of Afaan Oromo Search Engine”, Master’s thesis, Addis Ababa University, Department of Computer Science, 2010.
6. Judit Bar-Ilan and Tatyana Gutman, “How do Search Engines Handle Non-English Queries? - A case study”. In Proceedings of the Alternate Papers Track of the 12th International WWW Conference, Budapest, Hungary, 2003.
7. Paul McNamee, Charles Nicholas and James Mayfield, “Addressing Morphological Variation in Alphabetic Languages”. In Proceedings of the 32nd international ACM SIGIR conference on Research and development in Information retrieval, 2009.
8. SHANNON, C.E. and WEAVER, W., The Mathematical Theory of Communication, University of Illinois Press, Urbana, 1964.
9. LANCASTER, F.W., Information Retrieval Systems: Characteristics, Testing and Evaluation, Wiley, New York, 1968.
10. BARBER, A.S., BARRACLOUGH, E.D. and GRAY, W.A. 'On-line information retrieval as a scientist's tool', Information Storage and Retrieval, 9, 429-44, 1973.
11. Ricardo Baeza-Yates and Berthier Ribeiro-Neto, Modern Information Retrieval.
12. KEENAN, E.L., 'On semantically based grammar', Linguistic Inquiry, 3, 413-461, 1972.
13. KEENAN, E.L., Formal Semantics of Natural Language, Cambridge University Press, 1975.
14. ANDREWS, K., 'The development of a fast conflation algorithm for English'. Dissertation submitted for the Diploma in Computer Science, University of Cambridge (unpublished), 1971.
15. LOVINS, B.J., 'Development of a stemming algorithm'. Mechanical Translation and Computational Linguistics, 11, 22-31, 1968.
16. LOVINS, B.J., 'Error evaluation for stemming algorithms as clustering algorithms', Journal of the American Society for Information Science, 22, 28-40, 1971.

17. FARRADANE, J., RUSSELL, J.M. and YATES-MERCER, A., 'Problems in information retrieval. Logical jumps in the expression of information', Information Storage and Retrieval, 9, 65-77, 1973.
18. KEEN, E.M. and DIGGER, J.A., Report of an Information Science Index Languages Test, Aberystwyth College of Librarianship, Wales, 1972.
19. LANCASTER, F.W., Information Retrieval Systems: Characteristics, Testing and Evaluation, Wiley, New York, 1968.
20. SPARCK JONES, K., 'A statistical interpretation of term specificity and its application in retrieval', Journal of Documentation, 28, 111-21, 1972.
21. Building and Managing a keyword cluster. Available at: http://www.prlog.org/10712649-building and managing –keyword-cluster: Last Accessed on Sunday, January 22, 2012.
22. HAYES, R.M., 'Mathematical models in information retrieval'. In Natural Language and the Computer (Edited by P.L. Garvin), McGraw-Hill, New York, 287, 1963.
23. JARDINE, N. and van RIJSBERGEN, C.J., 'The use of hierarchic clustering in information retrieval', Information Storage and Retrieval, 7, 217-240, 1971.
24. GOODMAN, L. and KRUSKAL, W., 'Measures of association for cross-classifications', Journal of the American Statistical Association, 49, 732-764, 1954.
25. GOODMAN, L. and KRUSKAL, W., 'Measures of association for cross-classifications II: Further discussions and references', Journal of the American Statistical Association, 54, 123-163, 1959.
26. Gerard Salton and M. J. McGill. Introduction to Modern Information Retrieval. McGraw Hill Book Co., New York, 1983.
27. KNUTH, D.E., the Art of Computer Programming, Vol. 3, Sorting and Searching, Addison-Wesley, Reading, Massachusetts, 1973.
28. SALTON, G., Automatic Information Organization and Retrieval, McGraw-Hill, New York, 1968.
29. Van RIJSBERGEN, C.J., 'The best-match problem in document retrieval', Communications of the ACM, 17, 648-649, 1974.
30. Van RIJSBERGEN, C.J., 'Further experiments with hierarchic clustering in document retrieval', Information Storage and Retrieval, 10, 1-14, 1974.
31. ARNAUDOV, D.D. and GOVORUN, N.N. Some Aspects of the File Organization and Retrieval Strategy in Large Databases, Joint Institute for Nuclear Research, Dubna, 1977.
32. The Anatomy of a large scale Hypertextual web search engine, Sergey Brin and Lawrence page, Computer Science Department, Stanford University, Stanford, CA 94305, USA.

33. PageRank. Available at: http://en.wikipedia.org/wiki/pageRank; Last Accessed on Sunday, January 22, 2012.
34. Castillo, cartos. Effective Web Crawling (Ph.D. thesis). University of Chile. Retrieved 2010-08-03, 2004.
35. Brin, S. And page, L. The anatomy of a large scale Hypertextual web search engine, Computer Networks and ISDN systems, 30(1-7), 107-117, 1998.
36. How Do Web Search Engines Work. Available at: http://www.webopedia.com/DidYouKnow/Internet/2003/HowWebSearchEnginesWork.asp; Last Accessed on Monday, January 2011.
37. AltaVista. Available at: http://www.altavista.com/; Last Accessed on Wednesday, January 2011
38. J. Deepa Devi, Ranjani Parthasarathi and T.V. Geetha. Tamil Search Engine Sixth Tamil Internet conference, Chennai, Tamilnadu, August 2003.
39. Hassen Redwan "ENHANCED DESIGN OF AMHARIC SEARCH ENGINE" Master's thesis, Addis Ababa University, Department of Computer Science, 2008.
40. Girma Berhe. "A Stemming algorithm development for Tigrigna language text documents", Master's thesis, Addis Ababa University, Department of Information Science, 2001
41. Daniel Teklu. "ዘበናዊ ሰዋሰው ቋንቋ ትግርኛ, መቐለ: Mega printing enterprise, 2000 ዓ/ም.
42. Kassa G. ሰዋሰው ትግርኛ, ኣዲስ ኣበባ: Mega printing enterprise, 2004.
43. Pann Yu Mon, Yoshiki Mikami, Myanmar Language Search Engine, IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 2, March 2011
44. Okell, John, "A reference grammar of colloquial Burmese" London: Oxford University Press, 1969.
45. Crawler. Available at: http://searchsoa.techtarget.com/definition/crawler; Last Accessed on April 10, 2012.
46. What is a Web Crawler? Available at: http://www.wisegeek.com/what-is-a-web-crawler.htm; Last Accessed on May 7, 2012.
47. Tim Wellhausen, Query Engine, A Pattern for Performing Dynamic Searches in Information Systems, Jan 24, 2006
48. Brian Pinkerton. Finding what people want: Experiences with the web crawler. In Proceedings of the Second World-Wide Web Conference, Chicago, Illinois, October 1994.
49. JSpider. Available at: http://j-spider.sourceforge.net/; Last Accessed on May 25, 2012.
50. The Language Observatory Project (LOP). Available at: http://gii2.nagaokaut.ac.jp:8080/g2liWebHome/index.jsp; Last Accessed on May 28, 2012.

51. Language Observatory. Available at: http://gii2.nagaokaut.ac.jp/gii/lopdiary.php; Last Accessed on June 1, 2012, at 11:30
52. Erik Hatcher and Otis Gospodnetic. Lucene in Action. Manning Publications Co, 2005.
53. Why remove stop words. Available at: http://www.fromzerotoseo.com/stopwords-remove/; Last Accessed on June 1, 2012.
54. Tf-idf. Available at: http://en.wikipedia.org/wiki/Tf*idf; Last Accessed on August 18, 2012.
55. S. Chakrabarti, B.E. Dom, P.Raghavan, S. Rajagopalan, D. Gibson, and J.Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. In Proceedings of the 7th International World Wide Web Conference (WWW7), pages 65-74, 1998.
56. Network Infrastructure (LAN/WAN) Available at: http://www.aau.edu.et/index.php/ict-office; Last Accessed on August 30, 2012.
57. Tigrinya (ትግርኛ) Available at: http://www.omniglot.com/writing/tigrinya.htm; Last Accessed on January 05, 2012.
58. Yonas Fissha, "Development of Stemming Algorithm for Tigrigna Text", Master's thesis, Addis Ababa University, Department of Information Science, 2011.
59. Michael Gasser," Semitic Morphological Analysis and Generation Using Finite State Transducers with Feature Structures" Indiana University, School of Informatics, Bloomington, Indiana, USA, gasser@indiana.edu
60. Dr. Kssa G/hiwot and Emaniel Gankin መዝገበ ቃላት ትግርኛ, አዲስ አበባ: Emay printers, 2000 ዓ/ም.
61. Eclipse (software). Available at: http://en.wikipedia.org/wiki/Eclipse_software; Last Accessed on January 21, 2013.
62. Information. Available at: http://www.businessdictionary.com/definition/information.html; Last Accessed on September 05, 2011.

APPENDIX I. List of Collected Tigrigna short words

ቤት ት/ቲ = ቤት ትምህርቲ

ቤት ፍ/ዲ = ቤት ፍርዲ

ት/ቲ = ትምህርቲ

ክፍለ ት/ቲ = ክፍለ ትምህርቲ

ሃ/ስላሴ = ሃይለስላሴ

መ/ር = መምህር

ወ/ር = ወታደር

ወ/ሮ = ወይዘሮ

ወ/ሪት = ወይዘሪት

ወ/ስላሴ = ወልደስላሴ

ፍ/ስላሴ = ፍቅረስላሴ

ቤት ፅ.ት = ቤት ፅሕፈት

ፕ/ር = ፕሮፌሰር

ቀ.ሚንስትር = ቀዳማይ ሚኒስትር

ዶ/ር = ዶክተር

ገ/ጊዮርጊስ = ገብረጊዮርጊስ

ቤ/ክርስትያን = ቤተ ክርስትያን

ም/አቦወንበር = ምክትል አቦወንበር

ቤት ም/ሪ = ቤት ምኽሪ

ተ/ሃይማኖት = ተክለሃይማኖት

ሚ/ር = ሚኒስቴር

ኮ/ል = ኮሌኔል

ሜ/ጄነራል = ሜጀር ጄነራል

ብ/ጄነራል = ብርጋዴር ጄነራል

ሌ/ኮለኔል = ሌቴናል ኮለኔል

ኣ/ኣ = ኣዲስ ኣበባ

ሓ/ማሕበር = ሓረስቶት ማሕበር

ደ.ኣንስትዮ =ደቂ ኣንስትዮ

ኢ/ያ = ኢትዮድያ

ገ/ልምዓት = ገጠር ልምዓት

ሕ.ወኪል = ሕርሻ ወኪል

ላ/ማይጨው = ላዕላይ ማይጨው

ታ.ማይጨው = ታሕታይ ማይጨው

ገ/ማርያም = ገብረማረያም

ገ/ዚሄር = ገረዚሄር

ሓ/ዓሰርተ = ሓለቓ ዓሰርተ

ሓ.ሚኢቲ = ሓለቓ ሚኢቲ

ሓ.ሽሕ = ሓለቓ ሽሕ

ሓ.ዘመን = ሓዲሽ ዘመን

ር/ምምሕዳር = ርእሰ ምምሕዳር

ዓ/ግ = ዓድግራት

ዕ.ሓሙስ = ዕዳጋ ሓሙስ

ማ/ጨው = ማይጨው

ማ/ሰብ = ማሕበረ ሰብ

ዓ.ዓ = ዓመተ ዓለም

ማ/ኮሚቴ = ማእኸላይ ኮሚቴ

ር/መምህር = ርእሰ መምህር

ፕ/ት = ፕሬዚዳንት

ሃ.ተፈጥሮ = ሃፍቲ ተፈጥሮ

ቤት ፍ/ሒ = ቤት ፍትሒ

ሚ/ሕርሻ = ሚኒስቴር ሕርሻ

ቤት ህ/ት = ቤት ህንፀት

ር/ከተማ = ርእሰ ከተማ

ዓ.ም = ዓመተ ምህረት

Appendix II: List of Tigrigna Stop Words Used

ኣብቲ	ኩሉ	በዙይ	የለዋን
ማለት	ኣነ	ናይዞም	ምስ
እዩ	ንስኻ	ኩሎም	ከማይ
ከም	ንስኺ	ኢሎም	የብሉን
ነቲ	ዘለው	ዘለክን	ከማና
ኣብ	ንሱ	ናይቲ	ኩለን
ከማኻ	ንሳ	እንተድኣ	እያ
ከምዘለኺ	ንሕና	ናይ	ናይዘን
ከምዘለዎ	ንስኻትኩም	ዘለዎም	ናይቶም
እቲ	ንስኻትክን	ከምኡውን	ዝኾነ ኮይኑ
ወይ	ስጋዕ	እዮም	ኣብዛ
ድማ	ብዛዕባ	ካብተን	ኣብዚኣ
ክሳብ	ከለዋ	ድኣ	ኣላ
መን	ነታ	ናይዞም	የብልናን
ኣበይ	እታ	ናይዘን	የብልካን
እምበር	ናብ	ናይቶም	ኣለ
በቶም	ዘለኒ	ናይተን	ከምተን
ነዚ	ግን	ኣብዚኣም	ኢሉ
ይኹን	ጥራሕ	እዙይ	ኢላ
ኣሎ	ካብቶም	ዘለዋ	ኢለን
እዚ	እዞም	የብለንን	ኣይኮኑን
ዝኾነ	እኳ	የብለይን	የብሎምን
ኸዓ	ምኳኖም	እንተዝኾና	ከምዚኣቶም
እዛ	ብኡኣም	እዚኦም	ከምዚኣተን
ኣይኮነን	በቲ	ዘለወን	ኣይኮነትን
እንታይ	ብቲ	ከምዘለኪ	ከምቶም
እታ	ክኾና	ስለዝኾነ	ዘለና
ካብ	ብዘይ	ነይርወን	ዘለካ
ነይሮም	ምእንቲ	ዘለወን	ዘሎ
ከምቶም	ስለ	ዝባሃላ	የለዉን
እቶም	እሞ	ምስቲ	ዘላ
ከምዘለኒ	ዘለኩም	ዘለኪ	ይኹን እምበር
ከምዘለኩም	ድሕሪ	ምኳና	ይኹን ደኣ'ምበር
እውን	ቅድሚ	የለን	የብላን
ውን	ክሳዕ	ኮይኑ	ወዘተ
ኣብዚ	በዚ	ስለዝኾነውን	ስለዝኾኑ
ነበሩ	ብዚ	ከምኡ	ስለዝኾና
ነበረ	በኡኣም	ምኳን	ስለዝኾነት

ስለዝኾንኩ
ስለዝኾንኩም
ማለተን
እንትኸውን
ብምኻንኩም
ኩላትና
ብምኻና
እንትኾን
ቅድሚ
ከማኽን
ብምኻኑ
እንተዝኸውን
ቅድሚት
ከማኹም
ናይዚ
ማለትክን
ከምዘለና
ከምኣቶም
ዝበሃሉ
ምስኦም
ከምዘለኻ
ከምኣተን
ናታ
ማለቶም
ከምዘለኽን
ኣብዝሓ
ናቱ
ማለትና
ናትና
ብኣብዝሓ
ናይቱ
ማለትኪ
ናታትክን
ካሊእ
መዓዝ
ማለትካ
ንሳተን
ክንዲቲ
ምስዚ
ምስኦቶም
ነቱይ
ከምታ
ምስኡ
ምስኣተን
ከምአን
ካልኦት
የብልኩምን
ብምኻኖም
እየን
ከምዘለክን
የብልክንን
ማለታ
ኩላተን
ከምዘለኹም
ከለና
ማለቱ
ብምኻንክን
ከምዙይ
ክብሉ
ንማለት
ብምኻንኪ
ከምዚኣ
ኩልኻትክን
ምስኡ
ብምኻኑ
ብቅድሚት
ከማኺ
ኩሉኻትኩም
ነቶም
ብድሕሪት
ብምኻንና
እማ
ከከም
ብቅድሚ
ከምዘለዋ
እንተዝኾኑ
ኩሉኹም
ድሕሪ
ምስኣ
እንተዝኾን
ኩልኽን
ንቶም
ምሳና
ብዝኾነ
ስለዝኾንና
ንተን
ብምኻነን
ስለዝኾንክን
ናታትኩም
ምሳኹም
ከለው
ናታቶም
ምሳኽን
ከምዘለካ
ናታተን
ማለትኩም

Appendix III. Ethiopic Unicode table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x1200	ሀ	ሁ	ሂ	ሃ	ሄ	ህ	ሆ		ሰ	ሱ	ሲ	ሳ	ሴ	ስ	ሶ	ሷ
0x1210	ሐ	ሑ	ሒ	ሓ	ሔ	ሕ	ሖ	ሗ	መ	ሙ	ሚ	ማ	ሜ	ም	ሞ	ሟ
0x1220	ሠ	ሡ	ሢ	ሣ	ሤ	ሥ	ሦ	ሧ	ረ	ሩ	ሪ	ራ	ሬ	ር	ሮ	ሯ
0x1230	ሰ	ሱ	ሲ	ሳ	ሴ	ስ	ሶ	ሷ	ሸ	ሹ	ሺ	ሻ	ሼ	ሽ	ሾ	ሿ
0x1240	ቀ	ቁ	ቂ	ቃ	ቄ	ቅ	ቆ		ቈ		ቊ	ቋ	ቌ	ቍ		
0x1250	ቐ	ቑ	ቒ	ቓ	ቔ	ቕ	ቖ		ቘ		ቚ	ቛ	ቜ	ቝ		
0x1260	በ	ቡ	ቢ	ባ	ቤ	ብ	ቦ	ቧ	ቨ	ቩ	ቪ	ቫ	ቬ	ቭ	ቮ	ቯ
0x1270	ተ	ቱ	ቲ	ታ	ቴ	ት	ቶ	ቷ	ቸ	ቹ	ቺ	ቻ	ቼ	ች	ቾ	ቿ
0x1280	ኀ	ኁ	ኂ	ኃ	ኄ	ኅ	ኆ		ኈ		ኊ	ኋ	ኌ	ኍ		
0x1290	ነ	ኑ	ኒ	ና	ኔ	ን	ኖ	ኗ	ኘ	ኙ	ኚ	ኛ	ኜ	ኝ	ኞ	ኟ
0x12A0	አ	ኡ	ኢ	ኣ	ኤ	እ	ኦ	ኧ	ከ	ኩ	ኪ	ካ	ኬ	ክ	ኮ	
0x12B0	ኰ		ኲ	ኳ	ኴ	ኵ			ኸ	ኹ	ኺ	ኻ	ኼ	ኽ	ኾ	
0x12C0	ዀ		ዂ	ዃ	ዄ	ዅ			ወ	ዉ	ዊ	ዋ	ዌ	ው	ዎ	
0x12D0	ዐ	ዑ	ዒ	ዓ	ዔ	ዕ	ዖ		ዘ	ዙ	ዚ	ዛ	ዜ	ዝ	ዞ	ዟ
0x12E0	ዠ	ዡ	ዢ	ዣ	ዤ	ዥ	ዦ	ዧ	የ	ዩ	ዪ	ያ	ዬ	ይ	ዮ	ዯ
0x12F0	ደ	ዱ	ዲ	ዳ	ዴ	ድ	ዶ	ዷ	ዸ	ዹ	ዺ	ዻ	ዼ	ዽ	ዾ	ዿ
0x1300	ጀ	ጁ	ጂ	ጃ	ጄ	ጅ	ጆ	ጇ	ገ	ጉ	ጊ	ጋ	ጌ	ግ	ጎ	
0x1310	ጐ		ጒ	ጓ	ጔ	ጕ			ጘ	ጙ	ጚ	ጛ	ጜ	ጝ	ጞ	ጟ
0x1320	ጠ	ጡ	ጢ	ጣ	ጤ	ጥ	ጦ	ጧ	ጨ	ጩ	ጪ	ጫ	ጬ	ጭ	ጮ	ጯ
0x1330	ጰ	ጱ	ጲ	ጳ	ጴ	ጵ	ጶ	ጷ	ጸ	ጹ	ጺ	ጻ	ጼ	ጽ	ጾ	ጿ
0x1340	ፀ	ፁ	ፂ	ፃ	ፄ	ፅ	ፆ		ፈ	ፉ	ፊ	ፋ	ፌ	ፍ	ፎ	ፏ
0x1350	ፐ	ፑ	ፒ	ፓ	ፔ	ፕ	ፖ	ፗ	ፘ	ፙ	ፚ					
0x1360		፡	።	፣	፤	፥	፦	፧	፨	፩	፪	፫	፬	፭	፮	፯
0x1370	፰	፱	፲	፳	፴	፵	፶	፷	፸	፹	፺	፻	፼			
0x1380	ᎀ	ᎁ	ᎂ	ᎃ	ᎄ	ᎅ	ᎆ		መ	ᎏ	᎐	᎑	᎒			
0x1390	᎓	᎔	᎕	᎖	᎗	᎘	᎙		ም	᎛	᎜	᎝	᎞			
0x13A0	Ꭰ	Ꭱ	Ꭲ	Ꭳ	Ꭴ	Ꭵ	Ꭶ		ᚠ	ᚡ	ᚢ	ᚣ	ᚤ			
0x13B0	Ꭷ	Ꭸ	Ꭹ	Ꭺ	Ꭻ	Ꭼ	Ꭽ		ᚰ	ᚱ	ᚲ	ᚳ	ᚴ			
0xFDF0	﷼	﷽	ﷺ	ﷹ	ﷸ	ﷷ	ﷶ	ﷵ	ﷴ	ﷳ	ﷲ	ﷱ	«	»	?	

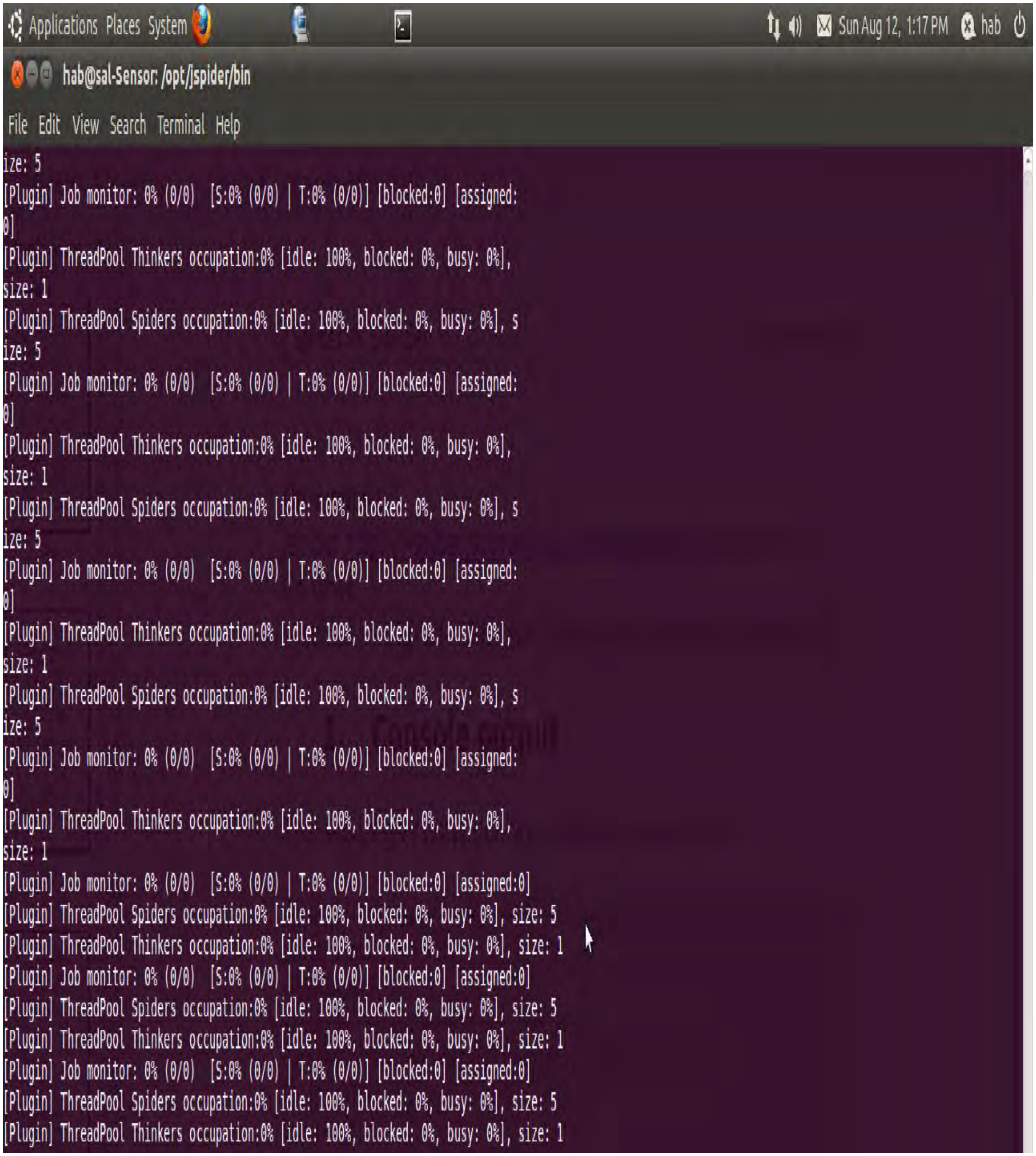
Appendix IV: List of Tigrigna Prefixes Used

ም	ይ
ብ	እንተዝ
ዝ	ኣይት
ብም	ክይተ
ነን	ክም
ን	ክምዝ
ንም	ስለ
ንዝ	እንተዘይተ
ብዝ	ስለዘይ
ዘይ	እንተዘይ
ዘይተ	ክ
እንተዘይ	እንድሕር
እንተይተ	እንድሕርዘይ
ዝተ	እንድሕርዘይተ
ክምዘይተ	እናተ
ምስ	ዝተ
ክምት	ክ
እና	እንት
ክን	እን
ተ	እንተ
ክየ	እንተይ
በቢ	ክምእን
ኣይተ	ስለዝ
ኣይ	ክት
ኣይን	ተተ
ኣይተ	
ክምዘይ	

Appendix V. List of Tigrigna Suffixes Used

ታት
ና
ዉቲ
ዉቲና
ት
ኩም
ኹም
ን
ናን
ክን
ኩምን
ነን
ዉቶም
ዉተን
ዉትኹም
ቶም
ካ
ኪ
ም
ቶ
ትአም
ዎ
አም
አን
ኣን
ላዋን
ምን
ቲ
ሎም
ሎምን
ያዊ
ያዊን
ያዊያን
ያውያን
ዊን
ናዮም
ውን
ለይ
ትለይ
ናለይ
ነት
ዊ
ነታዊ
ለን
ተ
ሉ
ተኛ
ትን
ታትን
ኡ
ሎምን
ኛ
ናለን
ውነቱ

Appendix VI: Crawling Process Report



Appendix VII: Configuration for JSpider

Name	Value
jspider.proxy.use	True
jspider.proxy.host	Cache.aau.edu.et
jspider.proxy.port	8080
jspider.proxy.authenticate	False
jspider.proxy.user	
jspider.proxy.user	
jspider.threads.thinkers.monitoring.enabled	True
jspider.threads.spiders.count	10
site.rules.spider.3.config.depth.min	0
site.rules.spider.3.config.depth.max	9

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: Hailay Beyene

Signature: _____

Date: _____

Confirmed by advisor:

Name: Solomon Atnafu (PhD)

Signature: _____

Date: _____

Place and date of Submission: Addis Ababa, March 1, 2013