ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
FACULTY OF INFORMATICS
DEPARTMENT OF INFORMATION SCIENCE

AUTOMATIC THESAURUS CONSTRUCTION FOR AMHARIC TEXT RETRIEVAL

A THESIS SUBMITTED TO THE SCHOOL OF GRADUATE STUDIES OF ADDIS ABABA UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN INFORMATION SCIENCE

By

ANDARGACHEW MEKONNEN GEZMU

July 2009
ADDIS ABABA

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF INFORMATICS

DEPARTMENT OF INFORMATION SCIENCE

AUTOMATIC THESAURUS CONSTRUCTION FOR AMHARIC TEXT RETRIEVAL

By

ANDARGACHEW MEKONNEN GEZMU

Adviser

Million Meshesha (PhD)

Signature of the Board of Examiners for Approval

_____	_____
_____	_____
_____	_____
_____	_____

Acknowledgment

I would like to thank my advisor Dr. Million Meshesha for his valuable advice and for his concern especially at tough times. Without his support this research work can not achieve its goal. Thanks to Dr. Nega Alemayehu, Dr. Daniel Yacob, and Mr. Daniel Abera for their pieces of advice through e-mail. I would like to offer special thanks to Ato Tessema Mindaye for providing me the Amharic stemmer source code.

My heartfelt thanks also goes to my mother: Addisalem Legesse; my brothers: Jegnaw Fentahun, Yohannes Haileyesus, Alemahyhu Tabor, Tedla Terefe, Beaman Kidus, Fedhesa Taddese, Dr. Alemayehu Mekonnen, and Ato Bekele Nega; and my sisters: W/o Tayech Legesse, Ribka Mekonnen, Sinafikish Mekonnen, and Zeyneba Abdela for their concern and financial support.

Dedication

To my Savoir and my Lord Jesus Christ, whom I love very much.

CONTENTS

List of Tables

List of Figures

Acronyms and Abbreviations

API – Application Programming Interface

IR – Information Retrieval

LSA – Latent Semantic Analysis

NLP – Natural Language Processing

SVD – Singular Value Decomposition

Abstract

Thesauri have been used for literary composition since their inception in 1852, but nowadays their primary use is for information retrieval. Even they are among the crucial components of retrieval systems which are typically used for enhancing indexing operations and query expansions during searching.

Even though Amharic language has been a written language for a couple of centuries and huge volumes of Amharic electronic documents are accumulated, not much has been done towards the development of effective and efficient Amharic retrieval systems. In this research work much effort has been exerted to generate thesaurus automatically for text retrieval in order to help the development of an effective and efficient Amharic retrieval system.

The development of the automatic thesaurus generation system is based on the WORDSPACE model. The WORDSPACE model is derived from the inverted file index by applying Random Projection algorithm for dimensionality reduction. Nearest Neighboring clustering algorithm is employed to generate thesaurus automatically from the WORDSPACE model constructed.

An encouraging result is obtained in the experimentation of the system on Amharic Bible documents. During experimentation the accuracy of the automatically generated thesaurus is evaluated. The result on a random sample of ten terms shows that the system has accuracy of 58%. To further investigate its applicability for Amharic information retrieval, the thesaurus is integrated to an IR system for query expansion. The retrieval system is tested with and without using thesaurus in order to show the improvement made in retrieval effectiveness. Performance analysis shows that the recall of the system while using thesaurus is superior to not using it. The average recall values are 73.34% and 37.29% after and before using thesaurus for query expansion, respectively.

Keywords: Amharic Thesaurus, WORDSPACE, Information Retrieval (IR)

CHAPTER ONE

INTRODUCTION

1.1. BACKGROUND

Thesauri have been used since 1852, when the first edition of the *Thesaurus of English Words and Phrases* is published by P.M. Roget (Foskett, 1997). While the Roget's thesaurus is used for literary composition, modern thesauri are basically used for information retrieval (IR).

Amharic has been a written language for roughly 600 years (Yacob, 2005). Besides, Amharic is the most studied and best understood language of Ethiopia. Researchers today, both inside and outside of Ethiopia, are increasingly interested in computational investigations of the Amharic language.

The rise of desktop publishing along with the arrival of the Internet, web publication of newspapers, and the inclusion of the Amharic writing system (i.e., the Ethiopic writing system) in the Unicode over the last decade has helped accumulate a huge volume of electronic Amharic information (*Ibid*.). As the volume of information is increasing, information explosion or information overload becomes the major challenge. Manually searching information from a huge collection of documents is a tedious, time consuming, and difficult task.

This difficulty has attracted an interest in designing automatic IR system and in fact its techniques as promising solutions (Baeza-Yates and Ribeiro-Neto, 1999). The effective retrieval of relevant information is directly affected both by the *user*

task and the *logical view of the documents* adopted by the retrieval system. By the user task we mean the translation made by the user of his/her information need into a query in the language provided by the system. Documents in a collection are represented by a set of index terms or key words. Such index terms might be extracted directly from the text of the document or might be specified by a human subject. These representative index terms provide a logical view of the document.

A document might be represented by its full text words. In this case, the retrieval system adopts a full text logical view (or representation) of the documents. The full text is obviously the most complete logical view of a document but its usage usually implies higher computational costs in terms of time and storage. With very large collection of documents we have to reduce the set of representative index terms in order to reduce the computational costs. This can be accomplished through text operations. According to Baeza-Yates and Ribeiro-Neto (1999) the text operations include five document processing procedures: lexical analysis, elimination of stopwords, stemming, selection of index terms, and term categorization through thesaurus.

Thesauri are among the valuable components of IR systems (Aitchison *et al.*, 2000; Srinivasan, 1992). A thesaurus provides a precise and controlled vocabulary that serves to enhance retrieval performance in an IR system, both in document indexing process and document searching. The thesaurus may be invoked manually, or preferably automatically, at either indexing or searching time. Terms may be added to the indexing of a record, or to a search statement, without the explicit knowledge of the indexer or searcher. This process regarding searching is called query expansion or reformulation. An example of this is

Chemical Abstracts Service SCIFINDER retrieval package, which equates, for example, 'cancer' and 'neoplasm' for retrieval (Aitchison *et al.*, 2000).

1.2. MOTIVATION

A motivation to conduct this research rose from the shortcomings of the popular commercial search engines. The result of an experiment carried out by Tessema (2007) with the popular search engines – Google, Yahoo, and MSN – for selected Amharic queries, unveiled a serious defect in the search engines' structures with regard to Amharic text retrieval. Based on the result the researcher found that these search engines did not applied the basic text processing operations such as stemming and stopword elimination, including thesaurus utilization.

Another motivation came from the alarming rate of accumulation of huge volume of electronic Amharic information mainly due to the advent of the Internet. Moreover, Worku (1997), Million (2000), Yaregal (2002) and Wondwossen (2004) among other researchers worked in the recognition of Amharic characters with encouraging performance. All of these endeavors will contribute in digitization of huge Amharic text collections in the future.

1.3. STATEMENT OF THE PROBLEM

Thesauri are significant and valuable components of IR systems for enhanced term classification during indexing and query expansions during searching (Aitchison *et al.*, 2000; Srinivasan, 1992). So, the utilization of thesauri is indispensable to come up with effective retrieval system.

Even though a thesaurus can be constructed manually, automatic thesaurus construction is preferred. The reason is that the former requires highly skilled

experts in a subject domain and also is a highly conceptual and knowledge intensive task; and therefore it is extremely labor intensive and time consuming; and also suffers low coverage (You and Chen, 2006). Moreover, thesauri are domain dependent (Srinivasan, 1992), and hence there is always a need to reinvent the wheel for each domain when the manual thesaurus construction method is carried out.

In addition to the general purpose search engines (such as Google, Yahoo, and MSN) a lot of efforts were made to develop Amharic IR systems by different researchers. Saba (2001), Bizuneh (2003), Tewodros (2003), Samuel (2005), Yalemsew (2005), and Tessema (2007) have tried to develop Amharic text retrieval systems. But neither of them has used thesaurus though there are attempts by Bekele (1992) and Yoseph (2005) to develop thesaurus for Amharic language. As far as the thesaurus generation process is concerned both of them were used a manual (intellectual) approach, based on facet analysis which is very labor intensive and raises a need of highly skilled domain experts. Such manually built thesauri are tough to integrate to an IR system unless they are designed specifically for such systems. Apart from these initial endeavors as to the researchers' knowledge no automatic Amharic monolingual thesaurus that can be integrated to an IR system is constructed, yet. But, as long as a representative body of text is available in a specific domain, it is possible to generate automatically the thesaurus without much human intervention.

Thus, the purpose of this research work is to construct automatic thesaurus for Amharic text retrieval to assist in the development of an effective Amharic IR system.

1.4. OBJECTIVES OF THE STUDY

The general and specific objectives of this research are stated as follows.

1.4.1. General Objective

The general objective of the research is to construct automatic thesaurus based on Amharic document collections using WORDSPACE model to enhance text retrieval.

1.4.2. Specific Objectives

To achieve the above mentioned general objective the research has the following specific objectives:

- A thorough review of literature related to existing automatic thesaurus generation techniques;
- Preprocessing a corpus for the next process;
- Building an index file for the corpus using an inverted file index structure;
- Building a WORDSPACE model based on the inverted file index;
- Developing an automatic thesaurus generation system based on the WORDSPACE model;
- Evaluate the performance of the thesaurus using test datasets from Amharic Bible documents; and test its applicability for the Amharic IR system.
- Recommend further research works in the area.

1.5. METHODS

The following methods have been followed to achieve the general and specific objectives of this research work.

1.5.1. Literature Review

Literature reviews have been done on different areas relevant to this research work. Features of thesauri and approaches to automatic thesaurus construction have been studied to unveil the current state of the art in the area.

Based on literature, Amharic language has been studied with regard to its features that affect the automatic thesaurus generation process. This includes the Amharic scripts and the Ethiopic writing system with its Unicode representation in computer; and the ambiguities and inconsistencies in the writing system.

The WORDSPACE model and tools available for building the model (i.e., the Semantic Vectors and Apache Lucene Application Programming Interface (API) libraries) are also assessed from literature.

1.5.2. Development Tools

The Java programming language is used for the programming task. Java is selected for its suitability for processing Unicode encoded text. Moreover, the API libraries used in this research work are entirely implemented in Java.

The Semantic Vectors API is used to develop WORDSPACE model for document collections. Its dependency is only on Apache Lucene's APIs. Both types of APIs are created in Java.

1.5.3. Performance Evaluation

The evaluation of automatically generated thesaurus is done in two steps. First we evaluate the automatically generated thesaurus, and then the thesaurus is integrated to an IR system to prove its applicability. For the evaluation of the IR system, the two most frequent and basic measures for information retrieval effectiveness (i.e., precision and recall) are used before and after using the thesaurus for query expansion.

1.6. SCOPE AND LIMITATION OF THE STUDY

The scope of this research is the construction of automatic thesaurus for Amharic documents in a specific domain by following corpora based approach. In the present research Amharic Bible documents are used for thesaurus construction and testing its performance. The proposed automatic thesaurus generation system basically has text operation (i.e., lexical analysis, stemming, and stopword elimination), indexing, and WORDSPACE system components.

The proposed automatic thesaurus generation system has some limitations. The research is done on documents which have a Unicode encoding; and hence it does not include documents which have other encodings. The limitation concerning term coordination is that thesaurus terms are composed of single word terms. Other limitations of the study are concerned with the Ethiopic writing system. Some ambiguities of the writing system are not addressed in this research. Ambiguities arise from various forms spelling for the same word as in '**ጡዋት**' vs '**ጥዋት**', '**ጠዋት**', and '**ጧት**'; gemination as in '**ዋና**'; Amharic spelling variations due to visual redundancy as in '**ፓስታ**' vs '**ፖስታ**'; and inconsistent

manner of writing term phrases referring to terms such as '**ደቀ መዝሙር**' vs ''**ደቀመዝሙር**' are not addressed in this research work.

1.7. APPLICATION OF THE STUDY

The research can be used as an integral part for development of an effective and powerful Amharic IR system. It is believed that using thesaurus improves the retrieval performance. Since thesaurus can be used in query expansion or reformulation to retrieve related concepts, it is inevitable to integrate it in recall oriented IR systems.

The research can also be applied in the development of machine readable dictionaries and lexical databases; machine translation, and even for the development of spelling correction.

1.8. THESIS ORGANIZATION

This paper starts by presenting background, statement of the problem, objective and methodology of the study, and other introductory parts.

In chapter two a review of related literatures on Amharic language automatic thesaurus construction, APIs used for automatic thesaurus generation, and related works are presented. Chapter three details the development of the automatic thesaurus generation system. That is, the algorithms and techniques used in the development of the system are discussed thoroughly. Chapter four explains the experimentation carried out in the research to evaluate the performance of the system. Chapter five explains the development of an IR system that integrates thesaurus in its structure. Chapter six discusses the conclusion and suggests some recommendation for future study.

CHAPTER TWO

LITERATURE REVIEW

The literature review section deals with features and uses of thesauri, the WORDSPACE model, the Amharic language, and related works. The subsequent subsections that deals with thesaurus details the general features of both automatic and manual thesauri, applications of thesauri, and approaches to automatic construction of thesauri. The literature review on WORDSPACE model discusses on the techniques and tools that can be used to construct the model efficiently. The Amharic language features and its writing system are also studied. The final subsection is on related works that are directly related to this research work.

2.1. OVERVIEW OF THESAURI

According to Aitchison *et al*. (2000) a formal definition of thesaurus is a vocabulary of controlled indexing language, formally organized so that a priori relationships between concepts are made explicit, to be used in IR systems, ranging from the card catalog to the Internet.

Thesauri that are associated with IR systems are very different from the traditional thesauri such as Roget's thesaurus (Srinivasan, 1992). The traditional thesauri are designed to assist the writer in creatively selecting vocabulary in literary composition. In IR systems, however, they are used in retrieval of potentially relevant documents from large collections.

The thesaurus serves to coordinate the basic processes of indexing and document retrieval. In indexing, a concise representation of the document is

derived, while in retrieval it helps to search for relevant items. The IR thesaurus typically contains a list of terms, where a term is either a single word or a phrase, along with the relationships between them. It provides a common, precise, and controlled vocabulary that assists in coordinating indexing and retrieval.

2.2. PURPOSES AND USES OF THESAURI

The primary purpose of a thesaurus is for IR (Aitchison *et al*., 2000; Foskett, 1997). Secondary purposes include aiding in the general understanding of a subject area, providing 'semantic maps' by showing interrelations of concepts, and helping to provide definitions of terms. More unusual applications include the generation of keyword lists and the support of computer assisted abstracting. The primary use, for IR, may be achieved by using the thesaurus in the indexing (either manual or automatic) of a machine readable document databases or in searching for relevant documents that satisfy information need of users.

Basically, the same thesaurus is used for indexing and for searching a document database. This approach permits a very close mapping of the indexing and searching process, particularly when indexing and searching is carried out by the same individuals, as a document database is maintained in-house and accessed primarily by the information specialists who maintain it. It is also possible to use the thesaurus either for indexing, but not for searching; or for searching, but not for indexing. In the former case it provides a rich set of terms, including especially synonyms and broader terms, to increase the chance of successful retrieval. In the latter case, the role of the thesaurus is usually to assist in searching of a free-text database by suggesting additional search terms. The suggestion may be done explicitly, by offering the terms to the searcher for their choice, or

automatically; this process is generally referred to as ‘query-expansion’ or ‘query reformulation’. The automatic suggestion process requires rather complex algorithms since the system has to know not only to reformulate the query but also when to do so.

2.3. FEATURES OF THESAURI

According to Aitchison *et al.* (2000), Foskett (1997), and Srinivasan (1992), the most important features of both manual and automatic thesauri are vocabulary control, compound terms, vocabulary specificity and generality, term relationships, number of entries for each term, and term frequency of class members. A detail discussion of these features follows in the following subsections.

2.3.1. Vocabulary Control

Vocabulary control refers to the mapping of variant forms of terms into base expressions, thereby bringing consistency to the vocabulary. As a result, the user does not have to worry about variant forms of a term. Control of terminology in a thesaurus is achieved in various ways by applying extensive rules (Aitchison *et al.*, 2000). A simple rule is that terms should be in noun form. A second rule is that noun phrases should avoid prepositions unless they are commonly known. Also, a limited number of adjectives should be used. There are other rules to direct issues such as the singularity of terms, the ordering of terms within phrases, spelling, capitalization, transliteration, abbreviations, initials, acronyms, and punctuation.

Vocabulary control is a crucial process for Amharic language. In Amharic there are different spelling inconsistencies (e.g., ጸሀይ, ጸሓይ, ጸኀይ, ጸፅይ, ጸኻይ, and ፀሀይ meaning sun), a number of abbreviations, and a number of morphologically varied

terms having the same stem like '**የጤና**', '**ለጤና**', '**ከጤና**', '**ጤናማ**', '**ጤንነት**', '**ጤናሞች**', '**ጤናችን**', etc for stem '**ጤና**'. Different examples of these inconsistencies and ambiguities in the language are given at section 2.8.2.

2.3.2. Compound Terms

Compound terms are generated by construction of phrases from individual terms. A compound term may be either an adjectival phrase, such as 'dried vegetables' or a prepositional phrase, such as 'philosophy of education'. Two distinct coordination options for compound terms are recognized in thesauri: pre-coordination and post-coordination. A pre-coordinated thesaurus is one that can contain phrases. Consequently, phrases are available for indexing and retrieval. A post-coordinated thesaurus does not allow phrases. Instead, phrases are constructed while searching.

The advantage in pre-coordination is that the vocabulary is very precise, thus reducing ambiguity in indexing and in searching. Also, commonly accepted phrases become part of the vocabulary. However, the disadvantage is that the searcher has to be aware of the phrase construction rules employed. Thesauri can adopt an intermediate level of coordination by allowing both phrases and single words. This is typical of manually constructed thesauri. However, even within this group there is significant variability in terms of coordination level. Some thesauri may emphasize two or three word phrases, while others may emphasize even larger sized phrases. Therefore, it is insufficient to state that two thesauri are similar simply because they follow pre-coordination. The level of coordination is important as well. It should be recognized that the higher the level of coordination, the greater the precision of the vocabulary but the larger the vocabulary size. It

also implies an increase in the number of relationships to be encoded. Therefore, the thesaurus becomes more complex.

The advantage in post-coordination is that the user need not worry about the exact ordering of the words in a phrase. Phrase combinations can be created as and when appropriate during searching. The disadvantage is that search precision may fall. A more likely example given by Srinivasan (1992) is ‘library school’ and ‘school library’. The problem is that unless search strategies are designed carefully, irrelevant items may also be retrieved.

Pre-coordination is more common in manually constructed thesauri. Automatic phrase construction is still quite difficult and therefore automatic thesaurus construction usually implies post-coordination (Salton and McGill, 1983).

2.3.3. Vocabulary Specificity and Generality

While specificity of the thesaurus vocabulary is a function of the precision associated with the component terms, generality is a function of recall. A highly specific vocabulary is able to express the subject in great depth and detail. For example, ‘brown bread’ is more specific than ‘bread’. This promotes precision in retrieval. The related disadvantage is that the size of the vocabulary grows since a large number of terms are required to cover the concepts in the domain. Also, specific terms tend to change (i.e., evolve) more rapidly than general terms (Aitchison *et al.*, 2000). Therefore, such vocabularies tend to require more regular maintenance. Further, high specificity implies a high coordination level which in turn implies that the user has to be more concerned with the rules for phrase construction.

2.3.4. Term Relationships

Term relationships are the most important aspect of thesauri since the vocabulary connections they provide are most valuable for retrieval. Many kinds of relationships are expressed in a manual thesaurus. These are semantic in nature and reflect the underlying conceptual interactions between terms. Aitchison *et al*. (2000) specify three categories of term relationships: equivalence relationships, hierarchical relationships, and associative relationships.

Equivalence relations include both synonymy and quasi-synonymy. Synonyms can arise because of common nouns or scientific names, and trade names (e.g. 'amodiaquinine' vs 'camoquin'); popular names (e.g. 'spider' vs 'arachnida'); obsolete terms (e.g. 'children's homes' vs 'orphanages'). Quasi-synonyms are terms which for the purpose of retrieval can be regarded as synonymous (e.g., 'genetics' vs 'heredity' and 'speech impaired people' vs 'stammerers'), which have significant overlap in meaning.

The hierarchical relationship is used in locating broader and narrower concepts in a logically progressive sequence. A typical example of a hierarchical relationship is a generic relationship, such as 'vertebrate' vs 'mammal' and 'gastrointestinal agents' vs 'bile acids'. Another example is whole-part relationship such as 'ear' vs 'external ear' and 'ear' vs 'inner ear'.

Associative relationship is found between terms that are closely related conceptually but not hierarchically and are not members of an equivalence set. Good examples are 'operating theaters' vs 'surgical equipment' and 'buildings' vs 'doors'.

These term relationships are complex and are commonly exhibited by manually constructed thesauri (Srinivasan, 1992). It should be noted that the relative value of these relationships for retrieval is not clear. Moreover, these semantic relationships are difficult to identify by automatic methods, especially by algorithms that exploit only the statistical relationships between terms as exhibited in document collections (*Ibid*.). So, automatically generated thesauri hardly take account of these kinds of term relationships. Rather, they have some structure on the vocabulary, which usually means hierarchical arrangement of term classes. This has been done by clustering algorithms. A typical algorithm accepts all pair-wise similarity values corresponding to a collection of objects. It also uses values to partition the objects into clusters or classes such that objects belonging to the same class are more similar to each other than those of a different class.

2.3.5. Number of Entries for Each Term

It is preferable to have a single entry for each thesaurus term. However, this is rarely achieved due to the presence of homographs (or polysemes). Homographs are words having the same spelling as another but differing in origin and meaning. For example, ‘cells’ may refer to biological microsystems or electrical equipment. The semantics of each instance of a homograph can only be contextually interpreted. Therefore, it is more realistic to have a unique representation or entry for each meaning of a homograph. This also allows each homograph entry to be associated with its own set of relations. The problem is that multiple term entries add a degree of complexity in using the thesaurus. Besides, it is hard to achieve this feature automatically (*Ibid*.).

2.3.6. Term Frequency of Class Members

This has relevance mainly for statistical thesaurus construction methods which work by partitioning the vocabulary into a set of classes where each class contains a collection of equivalent terms. Salton and McGill (1983) suggest that in order to maintain a good match between document terms and queries, it is necessary to ensure that terms included in the same thesaurus class have roughly equal frequencies. Furthermore, the total frequency of occurrence should be as close to equal for each class as possible, thus ensuring that probability of producing a match between queries and documents is approximately equal for all thesaurus classes. If these frequency characteristics are totally violated – for example, if a higher frequency term such as 'computer' is entered into the same class as a more specific term such as 'minicomputer' – queries about specific topics will produce general responses, thereby depressing the precision of the search.

2.4. APPROACHES TO AUTOMATIC THESAURUS CONSTRUCTION

Automatic thesaurus construction is one component of the wider field of automatic IR, which also includes indexing, automatic abstraction, and automatic document classification (Aitchison *et al*., 2000). In all these operations, statistical techniques or computational linguistics replace human intellectual processes. Since manual thesaurus construction is a highly conceptual, knowledge intensive and labor-intensive process, an alternative automatic method is definitely of value.

According to Manning et al. (2008), Lassi (2002), and Srinivasan (1992) three approaches are suggested for automatic construction of new thesauri. The first automatic approach is designing thesauri from document collections (corpora),

which is a standard one. The second automatic approach is user-generated thesauri based on tools from expert systems. In this approach, thesauri are built using information obtained from users. The third approach is thesaurus construction based on Bayesian network.

2.4.1. Corpora Based Thesauri

In this approach there are two main techniques. One is simply to build WORDSPACE models. The core idea behind WORDSPACE models is that words and concepts are represented by points in a mathematical space, and this representation is learned from text in such a way that concepts with similar or related meanings are near to one another in that space (Sahlgren, 2005). The other technique is to use a shallow grammatical analysis of the text and to exploit grammatical relations or grammatical dependencies. For example, entities that are grown, cooked, eaten, and digested, are more likely to be food items (Manning *et al*., 2008).

2.4.2. User-Generated Thesauri

Another approach for automatic thesaurus construction is based on tools from expert systems. Thesauri are built using information obtained from users. The procedure involves examining the types of Boolean operators used between search terms, the type of query modification performed by the user, and so on. User feedback is included to resolve any ambiguities and uncertainties. Feedback is also required to select the specific type of relationship between two terms once it has been decided that the terms are indeed related. Therefore, this approach requires considerable interaction with the user population.

2.4.3. Bayesian Networks

The third approach is to create a Bayesian network to model the thesaurus. A Bayesian network built from local term dependencies can give a probabilistic similarity distribution among the terms. The distribution is different from that of the most frequent probabilities, but will serve the purpose of classifying terms (Lassi, 2002).

2.5. AUTOMATIC THESAURUS CONSTRUCTION BASED ON WORDSPACE OR SEMANTIC VECTOR MODEL

2.5.1. The Vector Space Model

In Vector Space model a document collection can be represented by a matrix of document vectors (or a term-document matrix) as shown in figure 2.1.

	T_1	T_2	$\cdots$	T_t
D_1	d_{11}	d_{12}	$\cdots$	d_{1t}
D_2	d_{21}	d_{22}	$\cdots$	d_{2t}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
D_n	d_{n1}	d_{n2}	$\cdots$	d_{nt}

Figure 2.1. Matrix of document vectors.

Where the columns represent each unique term and the rows represent each document in the document collection with *n* number of documents and *t* unique terms. The matrix entries are weights assigned to each unique term. According to Manning *et al.* (2008) and Salton and McGill (1983), the weights can be the frequency of each unique term in that document, the inverse document frequency weight, or the composite weight. In the first case it is the frequency of term *k* in document *i*, or $FREQ_{ik}$ as shown in equation 2.1.

$$d_{ik} = FREQ_{ik} \quad (2.1)$$

The inverse document frequency weight is given in equation 2.2; and the composite weight is given in equation 2.3.

$$d_{ik} = \left(\log_2^n - \log_2^{DOCFREQ_k} + 1\right) \quad (2.2)$$

$$d_{ik} = FREQ_{ik}\left(\log_2^n - \log_2^{DOCFREQ_k} + 1\right) \quad (2.3)$$

Where document frequency, or $DOCFREQ_k$, is the number of documents in the document collection where term *k* appears at least once and *n* is the number of documents in the collection.

The weights are also calculated based on signal-noise ratio and on the term discrimination values. But these methods are not widely used since the signal-noise ratio does not give optimal performance in a retrieval environment and the term discrimination values are computationally expensive (Salton and McGill, 1983).

While the rows of the matrix represent the individual document vectors, the columns identify the term assignments to the documents. That is, a column, *j*, of the document vector matrix reflects the assignment of $TERM_j$ to the documents of the collection. A similarity measure between pairs of columns $SIMILAR(TERM_k, TERM_h)$, reflects the similarities between $TERM_k$ and $TERM_h$; and the similarity measure is used in the automatic thesaurus construction (*Ibid*.). Given term vectors of the form $TERM_k = (t_{1k}, t_{2k}, \ldots, t_{nk})$, where t_{ik} indicates the weight or value of $TERM_k$ in document *i* and assuming n documents in the collection, a typical similarity measure may then be defined as:

$$SIMILAR_1(TERM_k, TERM_h) = \sum_{i=1}^{n} t_{ik} * t_{ih} \quad (2.4)$$

Or, using a normalization factor to limit the computed results to values between 0 and 1,

$$SIMILAR_2(TERM_k, TERM_h) = \frac{\sum_{i=1}^{n} t_{ik} * t_{ih}}{\sqrt{\sum_{i=1}^{n} t_{ik}^2 * t_{ih}^2}} \quad (2.5)$$

The first coefficient shown in equation 2.4, $SIMILAR_1$, is scalar product of term vectors. The second coefficient shown in equation 2.5, $SIMILAR_2$, is the cosine coefficient. The cosine measure is easy to compute and appear to be as effective in retrieval as other more complicated functions such as *overlap* and *asymmetric* measures (*Ibid*.).

WORDSPACE (or Semantic vector) models have received considerable attention from researchers in natural language processing over the past 15 years, though their invention can be traced at least to Salton's introduction of the Vector Space Model for information retrieval. The WORDSPACE is very much like a term-document matrix as described in table 2.1, with two main differences (Widdows and Ferraro, 2008):

1. Instead of recording which terms occurred in which documents, we record which terms occurred near to frequent, important content-bearing terms; and
2. To compress this information into fewer dimensions, techniques such as Latent Semantic Analysis (LSA), Random Projection, and Hyperspace Analogous to Language are used.

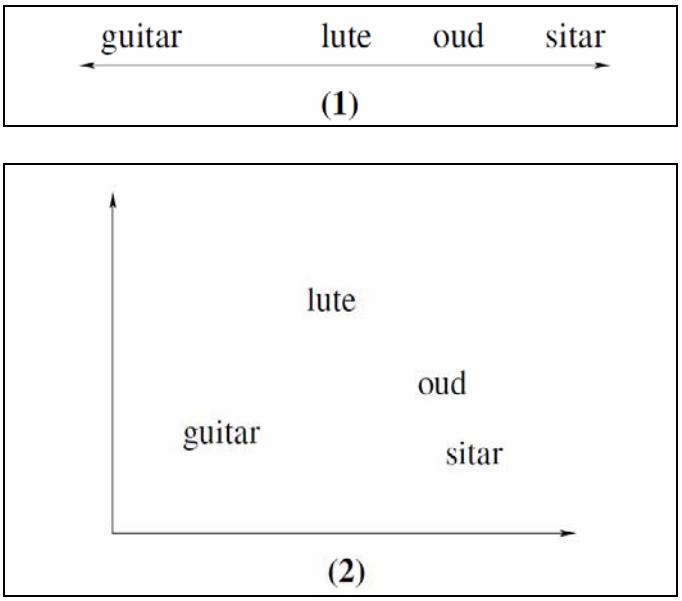
2.5.2. Wordspace or Semantic Vector Models

The general idea behind WORDSPACE or semantic vector models is to use distributional statistics to generate high-dimensional vector spaces, in which words are represented by context vectors whose relative directions are assumed to indicate semantic similarity (Sahlgren, 2005). This assumption is motivated by the distributional hypothesis, which states that words with similar meanings tend to occur in similar contexts. According to this hypothesis, if we observe two words that constantly occur with the same contexts, we are justified in assuming that they mean similar things. Note that the hypothesis does not require that the words co-occur with each other; it only requires that the words co-occur with the same other words (*Ibid*.).

In other words, the WORDSPACE model is a spatial representation of word meaning. Its central idea is that semantic similarity can be represented as proximity in n-dimensional vector space matrix, where n can be any integer ranging from 1 to some very large number which is determined by the number of columns in the matrix as discussed below. Of course, such high-dimensional spaces are impossible to visualize, but we can get an idea of what a spatial representation of semantic similarity might look like if we consider a 1-dimensional and a 2-dimensional WORDSPACE, such as those represented in figure 2.2.

In these geometric representations, spatial proximity between words indicates how similar their meanings are. As an example, both WORDSPACEs in figure 2.2 depicts '*oud'* as being closer to '*lute'* than to '*guitar'*, which should be interpreted as a representation of the meaning similarities between these words: the meaning (of) '*oud'* is more similar to the meaning (of) '*lute'* than to the meaning (of) '*guitar'*.

The core idea behind WORDSPACE models is that words and concepts are represented by points in a mathematical space, and this representation is learned from text in such a way that concepts with similar or related meanings are near to one another in that space. Such models are designed to represent words and documents in terms of underlying concepts, and as such can be used for many semantic (concept-aware) matching tasks such as automatic thesaurus generation, knowledge representation, and concept matching (Widdows, 2008).



Adapted from Sahlgren (2006)

Figure 2.2. (1) A 1-dimensional WORDSPACE, and (2) a 2-dimensional WORDSPACE.

The use of spatial proximity as a representation of semantic similarity is neither accidental nor arbitrary. On the contrary, it seems like a very intuitive and natural way for us to conceptualize similarities (Sahlgren, 2006).

In the standard WORDSPACE methodology, the high-dimensional vector space is produced by collecting the data in a co-occurrence matrix F, such that each row F_t represents a unique term t and each column F_c represents a context c, typically a multi-word segment such as a document, or another term. In the former case,

where the columns represents documents, we call the matrix a term-document matrix, and in the latter case where the columns represents terms, we call it a term-term matrix. LSA and Random Projection are examples of a WORDSPACE model that uses document-based co-occurrences, and Hyperspace Analogue to Language is an example of a model that uses term-based co-occurrences (Sahlgren, 2005).

The cells F_{tc} of the co-occurrence matrix record the frequency of co-occurrence of term t and document or term c. As an example, if we use document-based co-occurrences, and observe a given term three times in a given document in the data, we enter 3 in the corresponding cell in the co-occurrence matrix. In the same way, if we use term-based co-occurrences and observe that two given terms occur close to each other five times in the document collection, we enter 5 in the corresponding cell. The frequency counts are usually normalized and weighted in order to reduce the effects of high frequency words, and, in case document-based co-occurrences are used, to compensate for differences in document size.

The point of the co-occurrence matrix is that the rows F_t effectively constitute vectors in a high-dimensional space, such that the elements of the vectors are (normalized) frequency counts, and the dimensionality of the space is determined by the number of columns in the matrix, which is identical to the number of contexts (i.e. words or documents) in the data. We call the vectors context vectors, since they represent the contexts in which words have occurred. In effect, the context vectors are representations of the distributional profiles of words, which means that we may define distributional similarity between words in terms of vector similarity. By desirable quality of the distributional hypothesis, this makes

it very straight-forward to compute semantic similarity between terms: we simply compare their context vectors using any of a wide range of possible vector similarity measures, such as the cosine of the angles between the vectors, or the scalar product of the vectors(*Ibid*.).

2.5.3. Pros and Cons of Wordspace Models

The main attractions of WORDSPACE models, according to Widdows and Ferraro (2008) and Sahlgren (2005), are basically:

- They can be built using entirely unsupervised distributional analysis of free text.
- They make very few language-specific assumptions (e.g., it is possible to build a WORDSPACE model provided only that we have reliably tokenized text). That is, WORDSPACE models constitute a purely descriptive approach to semantic modeling; it does not require any previous linguistic or semantic knowledge, and it only detects what is actually there in the data.
- The WORDSPACE methodology makes semantics computable; it allows us to define semantic similarity in mathematical terms, and provides a manageable implementation framework.
- The geometric metaphor of meaning – the one that is shown in figure 2.2 – seems intuitively convincing, and is consistent with empirical results from psychological studies.
- Vector spaces are mathematically well defined and well understood; we know how vector spaces behave, and we have a large set of algebraic tools that we can use to manipulate them.

WORDSPACE models are theoretically attractive and experimentally successful. Over several years, the strengths of these models have been examined and evaluated in the performance of several tasks of importance to natural language processing (Widdows and Ferraro, 2008). Such applications of semantic vector models to date are listed by Widdows and Ferraro (2008) as follows:

- Information retrieval (Deerwester *et al.*, 1990). This was the original motivation for applying dimension reduction to a term-document matrix, in the hope of creating a more semantically aware search engine (e.g., a search engine that can locate documents based on synonyms and related terms as well as matching keywords).
- Lexical and ontology acquisition (Hearst and Schutze, 1993; Widdows, 2003). The core principle here is that knowledge of a few seed words and their relationships can help to infer analogous relationships for other similar words that are nearby in the semantic vector space.
- Word sense discrimination and disambiguation (Schutze, 1997; Schutze, 1998). The core principle here is that the weighted sum of vectors for words found in a particular region of text (called context vectors) can be clustered, and the centroids of these clusters can be treated as word-senses: occurrences of an ambiguous word can then be mapped to one of these word-senses, with a confidence or probability derived from the similarity between the context vector for this occurrence and the nearest centroids.
- Document segmentation (Brants et al., 2002). Given that we can compute context vectors for regions of text, one can detect document boundaries when context vectors leap from one part of the space to a completely different part of the space.

- Representing online communities and knowledge (Mcarthur and Bruza, 2003). As the amount of information shared in online communities (Usenet groups, mailing list archives, web forums, etc.) grows, it is increasingly important for new users to be able to evaluate and distinguish important information. One way of doing this is to model the utterances observed in online communities using semantic vector representations.

Although theoretically attractive and experimentally successful, WORDSPACE models have efficiency and scalability problems. This is especially true when the models are faced with real-world applications and large scale data sets or document collections. The source of these problems is the high dimensionality of the context vectors, which is a direct function of the size of the data. If we use document-based co-occurrences, the dimensionality equals the number of documents in the collection, and if we use term-based co-occurrences, the dimensionality equals the number of unique terms in the document collection, which tends to be even bigger than the number of documents. This means that the co-occurrence matrix will soon become computationally difficult when the vocabulary and the document collection grow (Widdows and Ferraro, 2008; Sahlgren, 2005).

Another problem with the co-occurrence matrix is that a majority of the cells in the matrix will be zero due to the sparse data problem. That is, only a fraction of the co-occurrence events that are possible in the co-occurrence matrix will actually occur, regardless of the size of the data. A tiny amount of the words in language are ditributionally related; the vast majority of words only occur in a very limited set of contexts. This phenomenon is well known; and in a typical co-occurrence

matrix, more than 99% of the entries are zero (Widdows and Ferraro, 2008; Sahlgren, 2005).

2.5.4. Solutions for High Dimensionality and Data Sparseness in WORDSPACE Models

In order to tackle problems with very high dimensionality and data sparseness, most well-known and successful methods, like Latent LSA and Random Projection, use statistical dimension reduction techniques. All share the same basic methodology: first sample the data in a standard term-document or term-term matrix, and then transform it into a much smaller and denser representation.

Standard LSA uses Singular Value Decomposition (SVD), which is a matrix factorization technique that can be used to decompose and approximate a matrix, so that the resulting matrix has much fewer columns and is much denser. But, dimension reduction techniques such as SVD tend to be computationally very costly, with regard to both memory consumption and execution time. For many applications, and especially for large vocabularies and large document collections, it is not practically feasible to compute an SVD (Widdows and Ferraro, 2008).

The main insight of Random Projection is that high dimensional vectors chosen at random are “nearly orthogonal”, in a way that can be formally characterized. A matrix is said to be orthogonal if its transpose is equal to its inverse. This relation makes orthogonal matrices particularly easy to compute with, since the transpose operation is much simpler than computing an inverse (Wolfram, 2009). Thus it achieves a result that is for many purposes comparable to orthogonalization methods such as Singular Value Decomposition, but spends none of the computational resources (Sahlgren, 2005). The basic procedure for creating a

random basic document vector is extremely brief and easy to implement. Consider two vectors which contain mainly zeros (i.e., they are sparse), and whose nonzero entries are comprised of an equal number of 1 and -1 entries. For example:

[0, 0, 0, 1, 0,-1, 0, 0, 0,-1, 0, 0, … , 0, 0, 1, 0, 0] and

[0,-1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, … , 0, 0,-1, 0]

It is easy to see that the expected value of the scalar product of two such randomly generated vectors is zero. When each coordinate in one vector is multiplied by the corresponding coordinate in the other vector, in most cases at least one of the multiplicands is zero. In the rare cases that both multiplicands are non-zero, it is expected half the time the signs of the multiplicands to be the same, contributing +1 to the scalar product, and the other half of the time it is expected the signs of the multiplicands to differ, contributing -1. So on average, these contributions will cancel out. To judge orthogonality the angle between the vectors is calculated, so it will be normalize by the lengths of the vectors (i.e., cosine similarity rather than the raw scalar product), leaving any non-zero results even closer to zero.

Hence, Random Projection does not rely on the use of computationally expensive matrix decomposition algorithms like SVD. This makes Random Projection a much more scalable technique in practice.

2.5.5. Semantic Analysis API libraries

Two API libraries, namely *the Semantic Vectors* and *Infomap NLP*, that are used to build WORDSPACE models are reviewed in this paper.

Semantic Vectors[1] API library can be used to easily create WORDSPACE models from a corpus of free text, and to search such models using a variety of mathematical operations including projections and algebraic product operations. The API package is created as part of a project by the University of Pittsburgh Office of Technology Management, to explore the potential for automatically matching related concepts in their technology management domain, e.g., mapping new technologies to potentially interested licensors. The project information can be found at http://real.hsls.pitt.edu.

The other API library that is used to create WORDSPACE models is *Infomap NLP*[2] It provides a variant of Latent Semantic Analysis (LSA) algorithm on free-text corpora to learn vectors representing the meanings of words in a WORDSPACE model.

While *Infomap NLP* uses SVD for dimensionality reduction on term-term matrix, *Semantic Vectors* uses Random Projection on term-document matrix. SVD is a computationally intensive algorithm which is hard to parallelize, and also hard to update incrementally. In contrary, Random Projection is much more scalable and easier to maintain incrementally.

As far as stability and ease of use are concerned, the *Infomap NLP* API library, which is all written in C, had a number of usage problems over the years with compiling the core code, with integrating with SVD package, and particularly with interfacing with the databases used to store the vectors (Widdows and Ferraro, 2008). But, the *Semantic Vectors* package, which is written entirely in Java, and its only dependency so far is *Apache Lucene*. It is easy to use, and since

[1] Available at: http://semanticvectors.googlecode.com
[2] Available at: http://infomap-nlp.sourceforge.net/

everything is running in Java, had no integration problems at all (*Ibid.*). Moreover, it can be easily implemented to handle a text with a Unicode encoding by changing only few of its library classes.

For these reasons, we used *Semantic Vectors* API package for building a WORDSPACE model for this research work.

2.6. INDEXING AND SEARCHING USING APACHE LUCENE

At the heart of all IR systems there is the concept of indexing: processing the original data into a highly efficient cross-reference lookup in order to facilitate rapid searching.

Suppose we needed to search a large number of files, and we wanted to be able to find files that contained a certain word or a phrase. A vague approach would be to sequentially scan each file for the given word or phrase. This approach has a number of flaws, the most obvious of which is that it does not scale to larger file sets or cases where files are very large. This is where indexing comes in: to search large amounts of text quickly, we must first index that text and convert it into a format that will let us search it rapidly, eliminating the slow sequential scanning process. This conversion process is called indexing, and its output is called an index.

Searching is the process of looking up words in an index to find documents where they appear. The quality of a search is typically described using precision and recall metrics. Recall measures how well the search system finds relevant documents, whereas precision measures how well the system filters out the irrelevant documents (Manning et al., 2008; Salton and McGill, 1983).

Apache Lucene is a high performance, scalable IR API library. It lets us add indexing and searching capabilities to our applications. Lucene is a mature, free, open-source package implemented in Java; it is a member of the popular Apache Jakarta family of projects. As such, Lucene is currently, and has been for a few years, the most popular free Java IR library (Hatcher, 2005).

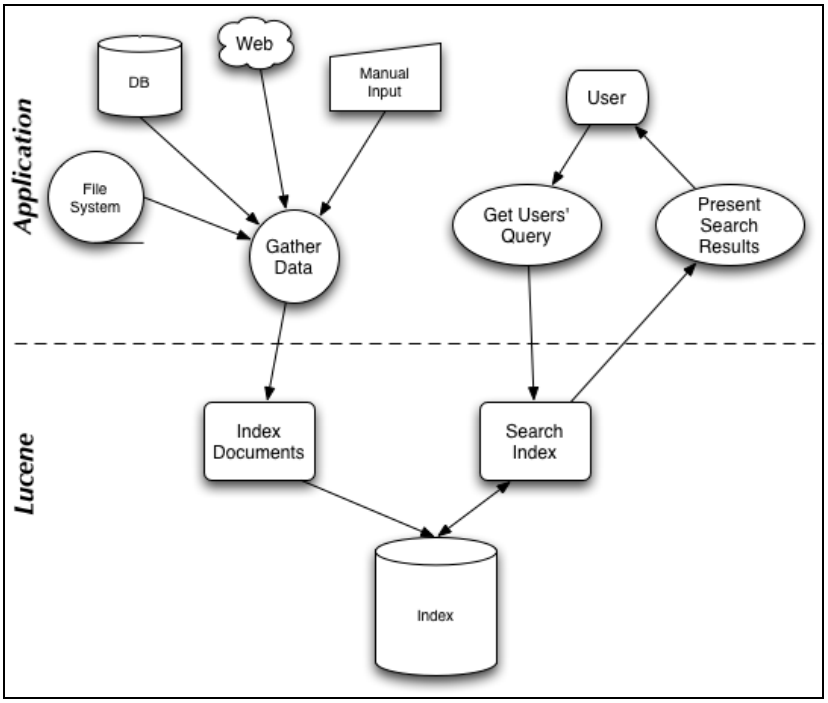
Lucene is an API library, not a full-featured search application. It is concerned with text indexing and searching. Lucene lets our application deal with business rules specific to its problem domain while hiding the complexity of indexing and searching implementation behind a simple-to-use API. We can think of Lucene as a layer that applications sit on top of, as depicted in figure 2.3.

Nowadays a number of desktop applications, web applications, and websites are using Lucene. Keller (2009) lists a total of 147 desktop applications and web applications, and 134 websites that are using Lucene APIs for the development of their retrieval systems. Just to mention from the list ActiveMath, AOL, DBSight, Disney, ICEcore, and jLibrary are among other systems.

2.6.1. Indexing Using Lucene

Lucene can handle data from any source, any format, or even any language, as long as it is converted to text. This means we can use Lucene to index and search data stored in files: web pages on remote web servers, documents stored in local file systems, simple text files, Microsoft Word documents, HTML or PDF files, or any other format from which we can extract textual information. Similarly, with Lucene's help we can index data stored in our databases, giving our users full-text search capabilities that many databases do not provide. Once we integrate Lucene, users of our applications can make searches.

A few methods of Lucene's public API need to be called in order to index a document. However, behind the simple API lies an interesting and relatively complex set of operations; Hatcher (2005) breaks them down into three major and functionally distinct groups: converting data to text, analyzing it, and saving it to the index. The operations are described in the following subsections.



Adapted from Hatcher (2005)

Figure 2.3. A typical application integration with Lucene.

2.6.1.1. Conversion to Text

To index data with Lucene, we must first convert it to a stream of plain-text tokens, the format that Lucene can process. Suppose we need to index a set of manuals in PDF format. To prepare these manuals for indexing, we must first find a way to extract the textual information from the PDF documents and use that extracted data to create Lucene index. We face the same situation if we want to index Microsoft Word documents or any document format other than plain text. Even when we are dealing with XML or HTML documents, which use plain-text

characters, we still need to prepare the data for indexing, to avoid indexing things like XML elements or HTML tags, and index the real data in those documents.

2.6.1.2. Analysis

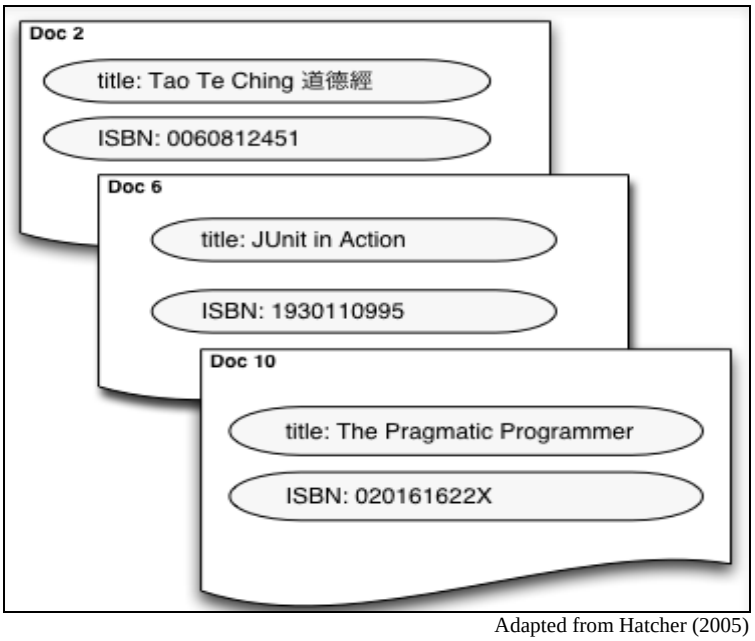
Lucene splits the textual data into chunks, or tokens, and performs a number of optional operations on them. For instance, the tokens could be lowercased before indexing, to make searches case-insensitive. Typically it is also desirable to remove all frequent meaningless tokens from the input, such as stopwords (a, an, the, in, on, and so on) in English text. Similarly, it is common to analyze input tokens and reduce them to their roots or stems. This very important step is called analysis in Lucene's terminology. This process performs the first three text operations: lexical analysis, elimination of stopwords, and stemming (Baeza-Yates and Ribeiro-Neto, 1999).

2.6.1.3. Index Writing

After the input has been analyzed, it is ready to be added to the index. Lucene stores the input in a data structure known as an inverted file index. This data structure makes efficient use of disk space while allowing quick keyword lookups (Hatcher, 2005). What makes this data structure inverted is that it uses tokens extracted from input documents as lookup keys instead of treating documents as the central entities. In other words, instead of trying to answer the question "what words are contained in a given document?" this structure is optimized for providing quick answers to "which documents contain a given word X?"

A logical view of a Lucene index can be considered as a black-box represented by group of documents that are populated with fields that consist of name and value

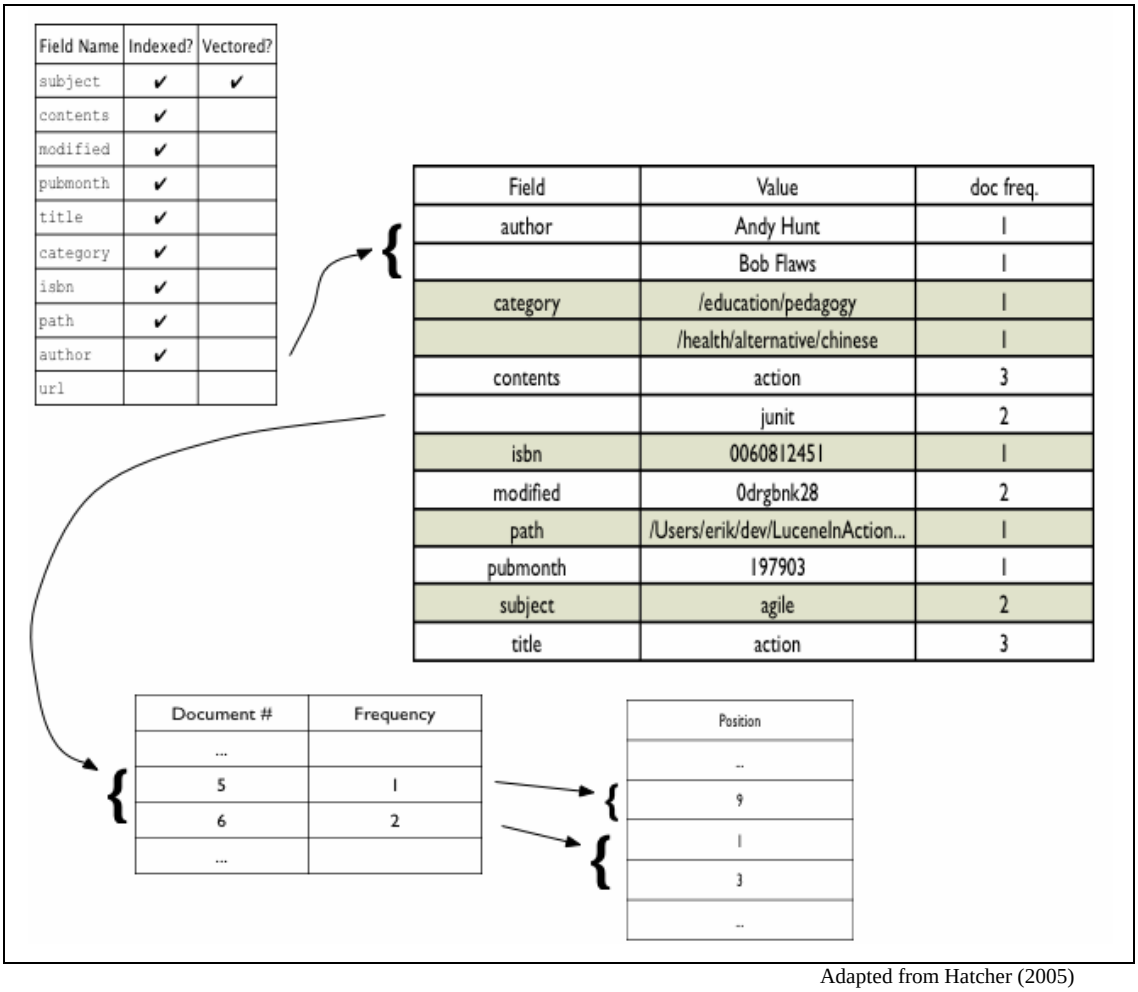
pairs. The field indicates the part of a document which this term came from (e.g., title or content of the document).



Adapted from Hatcher (2005)

Figure 2.4. The logical view of a Lucene index

The logical view of a Lucene inverted file index for book document collections is given in figure 2.4. The corresponding inverted file index of Lucene for book document collections is shown on figure 2.5. In figure 2.5, as an example, two terms (‘*Andy Hunt’* and ‘*Bob Flaws’*) are indexed from author field and other two terms (‘*action’* and ‘*junit’*) are indexed from contents field of the documents. The more outstanding feature of the inverted index created by Lucene than the traditional method is that the Lucene index has rooms for the data source or field (Hatcher, 2005; Harman *et al*., 1992). The corresponding statistics about the term such as document frequencies and position of the term within the documents are also stored in the file structure.



Adapted from Hatcher (2005)

Figure 2.5. Inverted file index format of Lucene.

2.6.2. Searching Using Lucene

When we are querying a Lucene inverted file index, an ordered collection of *hits* is returned. The *hits* collection is ordered by score. Lucene computes a score (a numeric value of relevance) for each document, given a query. The hits themselves are not the actual matching documents, but rather are references to the documents matched. Lucene uses equation 2.3 to determine a document score based on a query (Hatcher, 2005).

$$\sum_{k\ in Q} FREQ_{ik} \cdot IDF_k \cdot boost(k.field\ in\ DOC) \cdot lengthNorm(k.field\ in\ DOC) \qquad (2.4)$$

Where $FREQ_{ik}$ is term frequency factor for the term k in the document i,

IDF_k is inverse document frequency of the term k,

boost(k.field in DOC) is field boost which is set during indexing, and

lengthNorm(k.field in DOC) is normalization value of a field.

The equation is based on the composite weight, which is the product of term frequency and inverse document frequency as given in equation 2.3.

In most applications that display search results, users access only the first few documents, so it is not necessary to retrieve the actual documents for all results; we need to retrieve only the documents that will be presented to the user (*Ibid*.). For large indexes, it would not even be possible to collect all matching documents into available physical computer memory.

2.7. INFORMATION RETRIEVAL EFFECTIVENESS EVALUATION

The two most frequent and basic measures for information retrieval effectiveness are precision and recall (Manning et al., 2008).

If $RETREL_i$ is defined as the number of items retrieved and relevant, $RETNREL_i$ is the number of items retrieved but not relevant, and $NRETREL_i$ is the number of items relevant but not retrieved for query i, then the recall and the precision for query i is defined as:

$$RECALL_i = \frac{RETREL_i}{RETREL_i + NRETREL_i} \tag{2.5}$$

$$PRECISION_i = \frac{RETREL_i}{RETREL_i + RETNREL_i} \tag{2.6}$$

A user-oriented recall-average, reflecting the performance an average user can expect to obtain from the system, may than be defined by taking the arithmetic mean, over NUM sample queries, of expressions 2.7 and 2.8 (Salton, 1983).

$$AVERAGE\,RECALL_{RL} = \frac{1}{NUM}\sum_{i=1}^{NUM}\frac{RETREL_i}{RETREL_i + NRETREL_i} \tag{2.7}$$

$$AVERAGE\,PRECISION_{RL} = \frac{1}{NUM}\sum_{i=1}^{NUM}\frac{RETREL_i}{RETREL_i + RETNREL_i} \tag{2.8}$$

2.8. THE AMHARIC LANGUAGE

Amharic language is the official and working language of Ethiopia. It is the most widely spoken language throughout the country. It is spoken principally in the central highlands of the country. It is also spoken and written as a second language in many parts of the country and spoken by significant number of Ethiopians living in the Middle East, Asia, Western Europe, and North America. According to a 2006 estimate, there were approximately 17.4 million speakers of this language (Ask, 2009). Amharic is an Afro-Asiatic language of the Southwest Semitic group and is related to Ge'ez, which nowadays becomes the liturgical language of the Ethiopian Orthodox church; it also has similarity with Tigré, Tigrinya, and the South Arabic dialects (Encyclopedia Britannica, 2008; Alemayehu and Willet, 2002).

Amharic uses the Ethiopic writing system. Ethiopic originally developed for writing the Semitic language Ge'ez. Ge'ez itself now limited to liturgical usage, but its script has been adopted for modern use in writing several languages of Ethiopia and Eritrea, including Amharic, Tigrigna and Tigré (Unicode, 2006).

2.8.1. The Amharic Writing System

Amharic has been a written language for more than 600 years. It uses the Ethiopic writing system which is originated from the Semitic scripts like Hebrew and Arabic. Although its root can be traced back to the original Semitic writing systems, it would not be classified as Middle Eastern script (Unicode, 2006). The writing system is syllabic (*Ibid*.). That means, unlike alphabetic writing systems, vowels and consonants are not written separately.

Unlike Arabic and Hebrew, Ethiopic is written from left to right. The Ethiopic system makes no distinction between upper and lower case letters and has no conventional cursive form. Word boundaries are indicated by two vertically placed dots like a colon (‘**:**’); though, these days spaces are often used for word delimiting. A sentence boundary is indicated by four dots (‘::’). The old form of the question mark, three vertically placed dots (‘**፧**’), has been replaced by the question mark (‘?’). Quotes are usually in the French style (‘<<’ and ‘>>’) and parentheses and exclamation marks are as in the Roman system (‘(’, ‘)’ and ‘!’) (Bloor, 1995).

Ethiopic digit glyphs are derived from the Greek alphabet (Unicode, 2006). But in modern use, European digits are often used. The Ethiopic number system does not use a zero, nor is it based on digital-positional notation. For example, 23 is written as **፳፫** and 3545 is written as **፫ሺ፭፻፵፭**.

2.8.2. Ambiguities in Amharic Writing System

Ambiguities in Amharic writing are mostly due to symbol redundancy (Yacob, 2005). For example, four distinct syllables can represent the sound /ha/: **ሀ ሐ ኀ ኻ**, two sets represent /s/: **ሰ ሠ** and two /ts/: **ጸ ፀ**. The 44 labialized consonant

symbols are also redundant, wholly or partially, and there is considerable variation in the spelling of many words that may involve them. A similar problem is observed in usage of some letters such as 'ቈ' vs 'ቆ', 'ኰ' vs 'ኮ', and 'ጎ' vs 'ጐ'. Table 2.1 indicates these variations with few Amharic words.

In addition to the symbolic redundancy of characters, Amharic suffers slightly from visual redundancy in a few cases. Most prominently the vowel markers of; 'ው' and 'ዉ', 'ፕ' and 'ኘ', 'ፓ' and 'ፘ', 'ዱ' and 'ዩ', 'ጕ' and 'ጉ' are similar enough that the former characters are often interchanged in words with the latter. Additionally, the letter 'ቁ' is often used in place of 'ቍ' (e.g. 'ቍጥር' vs 'ቁጥር').

Table 2.1. Symbolic redundancy of Amharic words.

Canonical Amharic	Common Amharic	Possible but Improbable Amharic
ዓለም	አለም	ዐለም, ኣለም
ፀሐይ	ጸሃይ, ፀሃይ, ጸሐይ	ጸሀይ, ጸሓይ, ጸኀይ, ጸኃይ,ጸኻይ, ፀሀይ, ፀሓይ, ፀኀይ, ፀኃይ, ፀኻይ
ኃይለ ሥላሴ	ኀይለ ሥላሴ, ኃይለ ስላሴ, ኀይለ ስላሴ,ኀይለ ስላሤ, ኃይለ ስላሤ, ሀይለ ስላሴ,ሀይለ ስላሤ, ሀይለ ሥላሴ, ሃይለ ስላሴ,ሃይለ ሥላሴ, ሃይለ ስላሤ, ኀይለ ሥላሤ,ኃይለ ሥላሤ, ሀይለ ሥላሤ, ሃይለ ሥላሤ,ሐይለ ስላሴ, ሐይለ ስላሤ, ሐይለ ሥላሴ	ሐይለ ሥላሤ, ሓይለ ሥላሤ,ሓይለ ሥላሴ, ሓይለ ስላሤ, ሓይለ ስላሴ, ኻይለ ሥላሤ, ኻይለ ሥላሴ,ኻይለ ስላሤ, ኻይለ ስላሴ

Adapted from Yacob (2004)

Another ambiguity arises from various forms of spelling for the same word as in '**ማርያም**' vs '**ማሪያም**', and '**ጡዋት**' vs '**ጥዋት**', '**ጠዋት**', '**ጧት**'. The different forms of spelling ambiguities are shown in *Appendix I*.

Another problem of the Amharic writing system is that the failure of the system to indicate gemination. There are numerous cases of minimal pairs of words in Amharic where the only difference in pronunciation is the presence and absence of gemination (e.g., /alə/ 'said' and /allə/ 'is present'), but this difference is not reflected in the orthography since both are written as **አለ**.

Still other types of ambiguities, which are highly associated with term tokenization, are inconsistencies in writing some words and abbreviations. The inconsistent manner of writing refers to words such as '**ደቀመዝሙር**' vs '**ደቀ መዝሙር**' that can be written as one word or phrase.

2.8.3. Ethiopic and Unicode

Although there are many local endeavors in computerizing the Ethiopic Alphabet, the works done so far lacked standards. Nowadays, there are over 40 Ethiopic word processing software products available, each with its own character set, encoding system, typeface names, and keyboard layout (Kitaw, 2004). An encouraging development is the inclusion of the Ethiopic character set in the Unicode 3.0 standard released in the year 2000, for the first time. Unicode refers to the family of standards and technologies associated with the Unicode Consortium that can be utilized for working with a written language in a computer environment (Yacob, 2006).

The syllables of the Ethiopic script are traditionally presented as a two-dimensional matrix of consonant-vowel combinations. In the Unicode's character code table for Ethiopic, the code space range U+1200 to U+1357 is interpreted as a matrix of 43 consonants crossed with 8 vowels, making 344 conceptual syllables (*Appendix II*).

The character code table for Ethiopic also has a representation for Amharic punctuations and digits. For example, U+1361 represents the traditional word separator ('**:**').

2.8.4. Stemming for Amharic

Amharic stemming algorithms have been developed by different parties of researchers. Alemayehu and Willet (2002) developed an iterative and context-sensitive stemmer that removes both prefixes and suffixes; Alemu and Asker (2007) developed a rule-based stemmer that uses machine-readable dictionary to reduce terms to their root forms.

The stemmer developed by Alemayehu and Willet (2002), which is reported to have an accuracy of 95% to a test file of 1221 words, removes affixes iteratively. That is, it is a kind of affix removal stemmer. The original stemmer uses SERA (System for Ethiopic Representation in ASCII) to represent Ethiopic scripts. According to SERA, first an Ethiopic syllable should be transliterated to Latin based characters with consonants and vowels. For example, '**ጊ**', '**ጆ**', and '**ማ**' are represented as 'gi', 'jo', and 'ma', respectively.

The other stemmer developed by Alemu and Asker (2007) is reported to have an accuracy of 65% for an old-fashioned fiction text (a book entitled '**ፍቅር እስከ**

መቃብር) and 75% for news articles from two Ethiopian news agencies. The different accuracy levels for varied corpora are an indication that the algorithm is domain dependent. Like the previous stemming algorithm it also uses SERA to represent Ethiopic scripts.

2.9. RELATED WORKS

Related works to this research are studied in the following subsections. While the first subsection deals with Amharic thesaurus, the second one deals with non-Amharic thesaurus.

2.9.1. Amharic Thesaurus

Bekele (1992) and Yoseph (2005) have tried to develop thesaurus related to Amharic language. Their endeavor was to develop bilingual thesaurus in water technology and commercial law domains, respectively.

2.9.1.1. English-Amharic Bilingual Thesaurus for Commercial Law

Yoseph (2005) with domain experts intellectually generated thesaurus terms for the commercial law domain and evaluated its performance using an indexing and searching system that is built in-house.

In the process of generating the bilingual thesaurus, a semi-automatic approach was utilized. Firstly, the researcher constructed an English monolingual thesaurus. This was done using facet analysis technique in consultation with domain experts. After the identification of facet structure, candidate terms were identified automatically from machine-readable text database prepared from the *Commercial Code of Ethiopia*, which is already available in PDF format. From the candidate terms stopwords (non-content bearing words) were eliminated.

Secondly, the terms extracted through the deductive approach by the domain experts and those terms generated through machine assistance were matched; and those terms which were assumed relevant for the thesaurus by the domain experts were added to the term bank. All of these processes are performed with extensive manual intervention. This process requires highly skilled experts in a subject domain and also is a highly conceptual and knowledge intensive task; and therefore it is extremely labor intensive and time consuming; and also suffers low coverage (You and Chen, 2006).

For the final construction of Amharic thesaurus, the researcher took advantage of the parallel nature of the *Commercial Code of Ethiopia* corpus. Hence, direct translation was made to construct the English-Amharic bilingual thesaurus, due to the perfectly parallel nature of the corpus.

The evaluation was carried out by measuring the retrieval performance of the indexing and searching system, which is built by the researcher, with respect to the monolingual versus the bilingual thesaurus. The well-known retrieval performance measures (i.e., precision and recall) were used during the experiment. But the indexing and searching system was built on Microsoft Access database management system, which is not proper as far as IR is concerned. A database management system deals with an organized and well-structured data items, while IR deals with huge collections of unstructured and unorganized information items such as multimedia, pictures, and documents (Manning *et al*., 2008; Baeza-Yates and Ribeiro-Neto, 1999).

2.9.1.2. English-Amharic Bilingual Thesaurus for Water Technology

Bekele (1992) manually developed a thesaurus for water technology terms and stored a sample database of the index terms using CDS/ISIS database package.

Instead of constructing the thesaurus based on some manual or automatic thesaurus techniques, an already existing English thesaurus known as *Interwater Thesaurus for Community Water Supply and Sanitation* was used as a source material. The remaining task was to translate the thesaurus to Amharic. So as to facilitate the translation process, the researcher along with domain experts compiled an *Amharic Water Technology* dictionary based on Amharic dictionaries, Amharic documents, and audio-visual materials dealing with water.

Finally, it was reported that a sample database of the index was formed using CDS/ISIS program to demonstrate how the thesaurus works. But, from the research paper, it was not clear how the thesaurus was integrated to the index. Moreover, no evaluation was made on the system.

2.9.2. Non-Amharic Thesaurus

In the subsequent subsections research works on the construction of English, Arabic, and Chinese thesauri are discussed.

2.9.2.1. English Monolingual Thesaurus

Chen *et al*. (1995) automatically generated a domain-specific, English monolingual thesaurus by analyzing stored documents using externally acquired controlled term (key-word) lists, automatic indexing techniques, and statistics-based cluster analysis algorithm, asymmetric co-occurrence analysis. The domain

for the thesaurus was the *Worm Community System*. The data existing in the *Worm Community System* include: archival data such as gene descriptions, a physical map, DNA sequences, a cell list, and cell lineage; formal and informal literature such as the *Worm Reference Book*, journal abstracts, *Worm Breeder's Gazette* articles, conference proceedings, lab directory, and community teachings; and analysis software (e.g., programs for displaying maps and sequences). But, the researchers used four main sources of textual documents in the *Worm Community System* for thesaurus generation. The four sources were: the *Worm Reference Book*, Journal abstracts, *Worm Breeder's Gazette*, and conference proceedings. Their reason for using these sources was that these knowledge sources provided different forums for community members to discuss topics ranging from finished project and ongoing research, to laboratory observations and personal communications; and the vocabularies used and concepts discussed tended to vary significantly. It seems that the researchers were considering these text corpora representatives for the *Worm Community System* domain.

In the term selection and filtering stage of thesaurus construction, a stemming algorithm that takes into account the English language morphology and a stopword elimination process were carried out.

Using the stemmed terms, the researchers created two versions of the worm thesaurus: *KB1* and *KB2*. The first version, called *KB1*, treated the four *Worm Community System* corpora as one. Co-occurrence analysis was performed only once for the entire collection. Conversely, *KB2* analyzed each corpus individually. The researchers first ran co-occurrence analysis against all documents in the

Worm Reference Book. Then, they ran the same procedure for journal articles, *Worm Breeder's Gazette*, and conference proceedings, respectively. Results from the four separate thesaurus generation processes were then combined to form one thesaurus.

A prototype system was developed in *ANSI C*. It took 9.2 and 4 hours of CPU time to generate *KB1* and *KB2*, respectively. The resulting sizes for *KB1* and *KB2* were 12.3 and 12.6 Mega Bytes, respectively. The researchers also developed an *X-Windows* interface to use the thesaurus.

Both versions of thesauri, KB1 and KB2, were tested by objective and subjective measures. The objective testing measure was based on precision and recall retrieval performance measures. On average the recall level without using thesaurus was at 28.60%; whereas using KB1 and KB2 were at 61.89% and 59.56%, respectively On the other hand, the thesauri produced low average precision levels; without using thesaurus it was 77.08%; and using KB1 and KB2 it was 24.17% and 23.66%, respectively.

The subjective measure was carried out with six subjects. These subjects included two experts, two novices, and two outsiders. Different subjective type questions were asked while using both versions of thesauri and the feedbacks of these subjects was analyzed. The evaluation results indicated that both versions of thesauri performed equally well, did produce relevant terms, did improve recall, and would be useful to the *Worm Community System*.

Furthermore, an outstanding feature of the resulting thesauri was that their ability of automatic and periodic update of their vocabularies and relationships.

2.9.2.2. Arabic Monolingual Thesaurus

Ghwanmeh *et al*. (2005) automatically generated an Arabic monolingual thesaurus. The corpus they used for the thesaurus generation was composed of documents that deal with computer science, education, Arabic language, information system, and engineering. They took abstracts of 242 of such documents. The documents contain on average size of 300 words.

The Arabic language has one basic difference with English: Arabic is written without vowels that leads to different meanings for the same letter sequence. But this ambiguity was not dealt with in their research.

In the term selection and filtering stage of thesaurus construction, though the researchers eliminated stopwords based on term frequency, they did not use stemming. They discarded terms either their total frequency is less than 3 or more than 7.

At the next stage of thesaurus construction, after constructing an inverted file, they used the indexed terms to build the thesaurus. Basically the thesaurus was build by computing term-term similarity using cosine similarity function. The inverted file was made on Microsoft Access database using Visual Basic 6.0 programming language. This process has a serious drawback. A database management system deals with an organized and well-structured data items, while IR deals with huge collections of unstructured and unorganized information items such as multimedia, pictures, and documents (Manning *et al*., 2008; Baeza-Yates and Ribeiro-Neto, 1999).

In order to evaluate the performance of the thesaurus generated, the researchers used the precision and recall retrieval performance measures with and without using the thesaurus for query expansion. In their experiment they have used 50 queries selected from the document collections. In the experiment they got an improvement in the retrieval performance with respect to recall. An average recall of 85% and 57% achieved in using and without using thesaurus. But a reduction from 39.2% to 27.5% in average precision level was observed while using thesaurus.

2.9.2.3. Chinese Monolingual Thesaurus

Foo *et al*. (2000) automatically generated a Chinese monolingual thesaurus in the Economics domain. For the actual generation of thesaurus, asymmetric term co-occurrence analysis, as the previous research for English thesaurus construction, was used. The corpus used for this research was comprised of three months economics news items (documents) downloaded from the *People's Daily* newspaper website. The original 900 documents were filtered down to 811 documents after eliminating those documents that were either too short or containing little information worth indexing.

Unlike English, in Chinese there is no word delimiter. Moreover, it is inappropriate to carry out word stemming and stopword elimination. The researchers utilized a Chinese dictionary called *Economics Terminology Dictionary* for term selection and filtering. In addition to the dictionary terms, for co-occurrence analysis, an extra word extraction step known as word segmentation from the collected corpus was done.

Instead of developing a new Chinese IR system, the generated thesaurus was integrated to an English IR system called *Managing Gigabytes*, after adapting the system to Chinese.

Finally, to evaluate the generated thesaurus, a set of queries was derived after manually reading through the complete corpus. Likewise, the documents relevant to the queries were also identified. A total of 30 queries and the relevant documents for each query were derived and identified. This formed the basis for the retrieval effectiveness experiments. Average recalls at different percentiles for three word segmentation approaches, with and without using the thesaurus for query expansion, were reported. In all cases, the use of the thesaurus has significantly improved the document recall. On the other hand, the experimentation showed a drop in precision.

2.10. LESSONS LEARNED

The basic lessons drawn from the literature review are the necessary steps that should be followed in automatic construction of thesauri based on corpora and techniques on the evaluation of the automatically generated thesaurus. The fundamental steps that are followed in the automatic construction of thesauri are outlined as follows:

1. Construction of vocabulary. This stage includes normalization and selection of terms. The normalizations are achieved through stabilizing spelling variations, discarding digits, eliminating stopwords, and stemming.

2. Similarity computation for identification of statistical associations between terms. This can be done in different techniques such as Latent Semantic Analysis and Random Projection.

3. Organization of vocabulary generally into a hierarchy structure based on the association computed in the above stage. This can be achieved by utilizing clustering algorithms.

Huge collections of documents are needed in a specific domain for corpora based automatic thesaurus construction. The rule of thumb for the determination of the corpus size is that it should be representative of the domain.

CHAPTER THREE

DEVELOPMENT OF THE THESAURUS

Automatic development of the thesaurus is gone through the process of creation of an inverted file index for a specific document collection, construction of

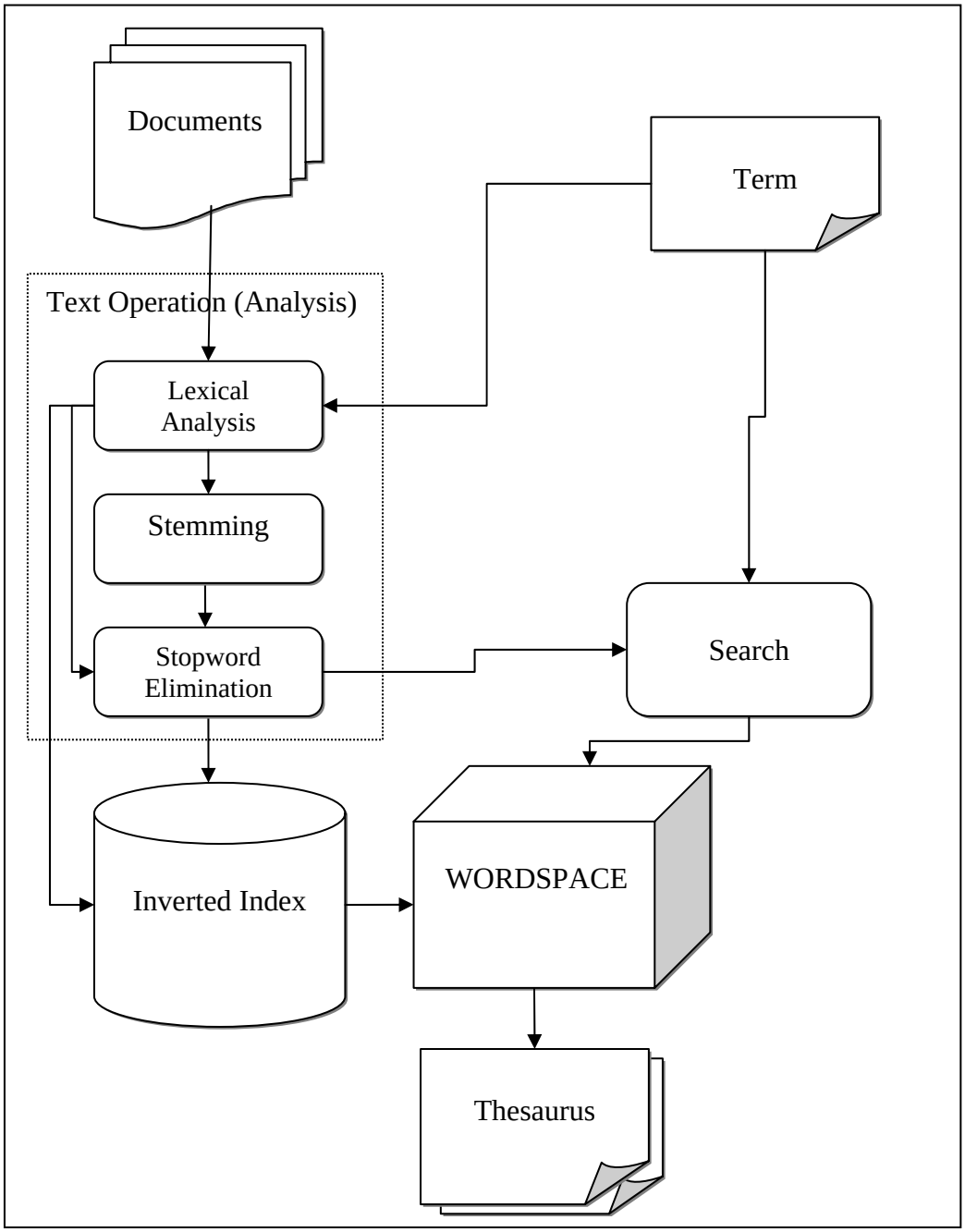


Figure 3.1. The system architecture of the thesaurus generation system.

WORDSPACE model based on the inverted file index, and searching the WORDSPACE to generate thesaurus terms.

The system architecture for thesaurus generation system is depicted in figure 3.1. Basically the system architecture has text operation (i.e., lexical analysis, stemming, and stopword elimination), indexing, and WORDSPACE components. The system accepts document collections, applies text operation on the documents, and adds the documents to the inverted file index. Based on the inverted file index the WORDSPACE model will be built. The resulting WORDSPACE model is then searched with a term from the WORDSPACE model to generate the nearest neighbors of the term; these nearest neighbors are automatically generated thesaurus terms for that particular term. The system components are discussed in the later subsections.

3.1. TEXT OPERATION AND INDEXING SYSTEM COMPONENTS

The algorithm for developing the inverted file index is given in figure 3.2. The parent directory, DOC_DIR, is the directory where the Amharic documents reside. Since document files are stored in subdirectories within the parent directory, the algorithm has to recursively search through the subdirectories. FILE_ITEM can be a document or a file directory. If a FILE_ITEM encountered is a document, then the algorithm applies text operations or analysis (i.e., lexical analysis, stopword elimination, and stemming) and adds the document to the inverted file index.

During text operations (or analysis), for lexical analysis, Amharic and English digits, Amharic and English punctuation marks, leading and trailing spaces, and other non-letter characters are discarded. Amharic word syllables are normalized for spelling inconsistency (*see section 2.8.2*) which is not uncommon in written

Amharic. Basically the normalization is to map different spelling variations to common one.

```
1. Open parent directory, DOC_DIR
2. Open sub DOC_DIR
3. For each FILE_ITEM in the sub DOC_DIR:
    If the FILE_ITEM is Directory
        GOTO 2
    Else
        Apply text operation on FILE_ITEM
        Add the FILE_ITEM to the inverted file index, INDEX_DIR
```

Figure 3.2. Algorithm for creating inverted file index.

In the second step of analysis, stopwords are eliminated from word tokens. A list of stopwords (or a negative dictionary) is compiled from Alemayehu and Willet (2002) and Tessema (2007); and some other list of stopwords is also added to the list by consulting with domain experts. Some of the stopwords can occur in many different forms (e.g., **እነዚህ በእነዚህ ከእነዚህ የእነዚህ ለእነዚህ** etc. for stem **ዚህ**). If all the different forms of the stopwords were included, the list would have been very long. Therefore, such variants are stemmed to their root form and stopword elimination is accompanied with stemming. The list of stopwords is shown in *appendix III*.

In the third step of analysis, to stem word tokens to their root term, the stemmer algorithm developed by Alemayehu and Willet (2002) is used. The algorithm is implemented in Java by Tessema (2007) after modifying it to suite to Unicode encoded text files. The stemmer developed by Alemayehu and Willet (2002)

removes affixes iteratively by employing a minimum stem length and context sensitive rules; with prefixes are removed before suffixes. For the minimum stem length condition, the shortest form of Amharic word is considered to be composed of two syllables. Therefore, the stemming is applied only on those words whose length is more than two syllables. In addition the stemming process deals with the removal of the doubling consonants. For example, '**ፈላለገ**' is stemmed to '**ፈለገ**' by removing '**ላ**'. Regarding the context sensitive rules two actions (i.e., to perform or not to perform affix removal) would be performed based on a predefined list of words. If a word is found in the list, affix removal will not be applied to it. This operation is done mainly to avoid over-stemming.

The original stemmer, which uses SERA (Alemayehu and Willet, 2002), is based on the idea that Amharic writing system is alphabetic, that is vowels and consonants are written separately. The suffix and prefix removing routines assume that vowels are represented in writings. But, the common written Amharic is syllabic, which is vowels are not represented in writings. Moreover, Unicode consortium gives codes to Ethiopic alphabets by recognizing it as syllabic (Unicode, 2006). So, the implementation of the algorithm in Java for Unicode encoded text files has considered this discrepancy and made some adjustment. In the original algorithm when a suffix with a vowel is removed, its last character is changed to *sades* (the sixth order of Amharic character or syllable). Based on this the adjustment is to ignore the vowels and take only the consonant as suffix. For example, the suffix -**ኣችን** will become -**ችን**. In light with this, the modified algorithm works as follows: if a word contains a suffix, then the suffix will be removed and the last character of the remaining word is changed to *sades*.

After the text operation is performed the term is added to the inverted file index along with its statistics. The corresponding statistics about the term is document frequencies and position of the term within the documents.

3.2. WORDSPACE MODEL SYSTEM COMPONENT

After the inverted file index for a document collection is created, the index can be mapped to a basic term-document matrix. A WORDSPACE model can be created from this term-document matrix, using Random Projection algorithm to perform dimensionality reduction.

1. Open the inverted index for reading
2. For each document:
    Create basic random vector, RAND_VEC
    For each term in the document:
        Screen out the term based on field filters
        For each document that contain the term:
            FREQ = frequency of the term within the document
            *TERM_VEC = TERM_VEC + (FREQ * RAND_VEC)*
        Normalize TERM_VEC

Figure 3.3. Algorithm for creating WORDSPACE model.

The algorithm for developing the WORDSPACE model is given in figure 3.3. First the inverted file index is opened for reading to access the term-document matrix. Then for each document basic random vectors are created using Random Projection algorithm for dimensional reduction. At the later stage terms are screened out. The screening out of the terms is based on the field filter and

frequency filter. Since the index terms are from *title* (i.e., absolute path in the file system), *modified* (i.e., the file's modification date), and *content* (i.e., string of file content) fields of the documents, there is a need to discard terms related to the title and modified fields information. The field indicates the part of a document which this term came from. So the field filter algorithm, filters out terms related to the *title* and *modified* fields.

At the final stage, the term vectors are computed and normalized based on the term frequency within each document and the random vector produced as an output from the Random Projection algorithm. The WORDSPACE model is composed of the normalized term vectors in the document collection.

3.3. GENERATION OF THESAURUS TERMS

In order to generate thesaurus terms, the WORDSPACE model is searched with a term for which thesaurus matches are required. The algorithm for thesaurus generation operation is given in figure 3.4.

In order to generate thesaurus for a term, the term should be normalized and stemmed. Normalization of spelling inconsistency of the term is done in the same manner as done in section 3.1 for index terms; and to stem the term the Amharic stemmer, which is implemented by Tessema (2007) in Java, is used. Since the term is normalized and stemmed, it is possible to proceed to the next operation.

1. Accept a term for which to generate thesaurus
2. Normalize and stem the term
3. Open the WORDSPACE model
4. Select a term vector that correspond to the term in the model
5. For each term vector in the WORDSPACE model:

$$COS\_SIM = \frac{\sum_{i=1}^{n} TERM\_VEC * TERM\_VEC_i}{\sqrt{\sum_{i=1}^{n} TERM\_VEC^2 * TERM\_VEC_i^2}}$$

6. Compute nearest neighbors for the term vector based on COS_SIM

Figure 3.4. Algorithm for thesaurus generation operation.

The next operation is to open the WORDSPACE model and to select a term vector that corresponds to the term for which thesaurus is required. Cosine similarity, COS_SIM, is computed based on the term vector and all the other term vectors in the model. The result provides a candidate list of vectors from which the nearest neighbor is to be chosen. The criterion for choosing nearest neighbors for the term is the highest cosine similarity value. That is, a term which has higher cosine similarity value with the term, for which thesaurus is required, is more related to the term than another term with lower cosine similarity value.

CHAPTER FOUR

EXPERIMENTATION

Automatic Amharic thesaurus generation system is implemented for Amharic Bible documents using Apache Lucene and Semantic Vectors APIs and its accuracy is evaluated.

Since the stemming and stopword elimination components of the system play an important role in the thesaurus generation process, these components are also evaluated. The evaluation proceeds as follows: simple random samples of 30 terms are selected from the WORDSPACE model and are examined carefully whether these terms are properly stemmed or are not stopwords.

4.1. SYSTEM IMPLEMENTATION

Thesaurus generation system is developed in such a way that to build up a WORDSPACE model based on inverted file index. The resulting WORDSPACE model is then searched with a term from the WORDSPACE model to generate the nearest neighbors of the term; these nearest neighbors are automatically generated thesaurus terms for that particular term. The inverted file index is created for Amharic Bible corpus in order to train the WORDSPACE model.

4.1.1. Corpus Collection

For corpus establishment Amharic Bible documents are downloaded from the *Lapsley/Brooks Foundation's*[3] website. The first complete Amharic Bible was produced in 1840, and went through several revisions thereafter. The version of

[3] Available at: http://www.bible.org/foreign/amharic/ (Date Accessed 10/08/2008)

the Amharic Bible used here is the one which is published in 1962. All of the documents are saved as text files with Unicode's UTF-8 encoding. This is mainly due to the fact that Lucene APIs can process only a stream of plain-text tokens. At this stage 1,189 documents are downloaded; and the documents are classified into training dataset and testing dataset.

The basic assumptions in allotting training dataset and testing dataset is that both of them should be representative samples of the underlying domain, and the testing dataset must be independent instance that have played no part in generation of the thesaurus (Witten and Frank, 1999).

In this research the training dataset is composed of the entire Old Testament books of the Amharic Bible; and the testing dataset is composed of the entire New Testament books of the Amharic Bible. This allocation is based on the Bible scholars' justification of the fact that both parts of the Bible –New Testament and Old Testament– have conveyed equivalent concept (Lee, 1997). For every concept, symbol, and allegory mentioned in the Old Testament we get full description in the New Testament. Hence, both parts can be representatives of the Amharic Bible corpus. The training and testing dataset statistics is shown in table 4.1. There are 929 documents in the Old Testament and 260 documents in the New Testament. That means 78% and 22% of the corpus is used for training and testing, respectively.

Table 4.1. Statistics of the training and testing dataset of Amharic Bible documents.

Training Dataset		Testing Dataset	
Book Name	No. of Documents	Book Name	No. of Documents
ኦሪት ዘፍጥረት	50	የማቴዎስ ወንጌል	28
ኦሪት ዘጸአት	40	የማርቆስ ወንጌል	16
ኦሪት ዘሌዋውያን	27	የሉቃስ ወንጌል	24
ኦሪት ዘኍልቁ	36	የዮሐንስ ወንጌል	21
ኦሪት ዘዳግም	34	የሐዋርያት ሥራ	28
መጽሐፈ ኢያሱ ወልደ ነዌ	24	ወደ ሮሜ ሰዎች	16
መጽሐፈ መሣፍንት	21	1ኛ ወደ ቆሮንቶስ ሰዎች	16
መጽሐፈ ሩት	4	2ኛ ወደ ቆሮንቶስ ሰዎች	13
መጽሐፈ ሳሙኤል ቀዳማዊ	31	ወደ ገላትያ ሰዎች	6
መጽሐፈ ሳሙኤል ካል	24	ወደ ኤፌሶን ሰዎች	6
መጽሐፈ ነገሥት ቀዳማዊ	22	ወደ ፊልጵስዩስ ሰዎች	4
መጽሐፈ ነገሥት ካልዕ	25	ወደ ቆላስይስ ሰዎች	4
መጽሐፈ ዜና መዋዕል ቀዳማዊ	29	1ኛ ወደ ተሰሎንቄ ሰዎች	5
መጽሐፈ ዜና መዋዕል ካልዕ	36	2ኛ ወደ ተሰሎንቄ ሰዎች	3
መጽሐፈ ዕዝራ	10	1ኛ ወደ ጢሞቴዎስ	6
መጽሐፈ ነህምያ	13	2ኛ ወደ ጢሞቴዎስ	4
መጽሐፈ አስቴር	10	ወደ ቲቶ	3
መጽሐፈ ኢዮብ	42	ወደ ፊልሞና	1
መዝሙረ ዳዊት	150	ወደ ዕብራውያን	13
መጽሐፈ ምሳሌ	31	የያዕቆብ መልእክት	5
መጽሐፈ መክብብ	12	1ኛ የጴጥሮስ መልእክት	5
መኃልየ መኃልይ ዘሰሎሞን	8	2ኛ የጴጥሮስ መልእክት	3
ትንቢተ ኢሳይያስ	66	1ኛ የዮሐንስ መልእክት	5
ትንቢተ ኤርምያስ	52	2ኛ የዮሐንስ መልእክት	1
ሰቆቃው ኤርምያስ	5	3ኛ የዮሐንስ መልእክት	1
ትንቢተ ሕዝቅኤል	48	የይሁዳ መልእክት	1
ትንቢተ ዳንኤል	12	የዮሐንስ ራእይ	22
ትንቢተ ሆሴዕ	14		
ትንቢተ ኢዮኤል	3		
ትንቢተ አሞጽ	9		
ትንቢተ አብድዩ	1		
ትንቢተ ዮናስ	4		
ትንቢተ ሚክያስ	7		
ትንቢተ ናሆም	3		
ትንቢተ ዕንባቆም	3		
ትንቢተ ሶፎንያስ	3		
ትንቢተ ሐጌ	2		
ትንቢተ ዘካርያስ	14		
ትንቢተ ሚልክያስ	4		
Total	929		260
Percentage	78%		22%

4.1.2. Implementation of Inverted File Index

To create an inverted file index, Lucene API package of version 2.4.0 is used. Lucene 2.4.0[4] includes index format changes that are different from older versions of Lucene. The Semantics Vector package is compatible only to this new index format (Widdows and Ferraro, 2008).

The code snippet for developing the inverted file index is given in *Appendix IV*. The IndexWriter class of Lucene API library contains a method, addDocument, to add each analyzed document to the inverted file index. For the addDocument method to add a document to the index, it requires the Document.add method to provide field information (such as the document content, absolute path of the file document, and the file modification date). While reading a file content, the Document.add method expects the file to be in the system's default encoding. If that is not the case reading for special characters like Ethiopic characters will fail. Therefore, to address this problem the code snippet shown in figure 4.1 is written to read from UTF-8 encoded Amharic text files and give the content as an argument to the method. Full descriptions of IndexWriter.addDocument and Document.add methods from Lucene API documentation is given in *Appendix V*.

The text operation is performed by utilizing the AmharicAnalyzer class which is a subclass of Lucene's Analyzer class. Basically the AmharicAnalyzer class discards non-letter characters, normalizes Amharic characters for spelling inconsistency, eliminates stopwords, and stems terms. Code snippets for discarding non-letter characters and normalizing Amharic characters for spelling inconsistency are given in *Appendix IV*. The source code of the Amharic Stemmer

[4] Available at: http://lucene.apache.org/java/ (Date Accessed 04/05/2008)

written by Tessema (2007) is used to stem terms. The stopwords are eliminated after stemming is carried out by using the negative dictionary (stoplist). The reason for applying stopword elimination after stemming is that some of the stopwords occur in different forms as discussed in section 3.1.

```
FileInputStream fileInput = new FileInputStream(filePath);
InputStreamReader inputStream = new InputStreamReader(fileInput, "utf-8");
BufferedReader bufferedReader = new BufferedReader(inputStream);
StringBuffer stringBuffer = new StringBuffer();
String lineContent = null;
while ((lineContent = bufferedReader.readLine()) != null) {
        stringBuffer.append(lineContent);
}
String content = stringBuffer.toString();
```

Figure 4.1. Code snippet to read from UTF-8 encoded Amharic text files.

Finally, the algorithm is run to index all of the documents in the training dataset. The inverted file index is saved in binary file format on the disk for efficiency reasons. To display the index in a human readable manner, an open source tool, which is created using Java, called Luke[5] is used. Figure 4.2 displays screenshot of the index using Luke. The inverted file index is composed of 21,312 index terms (or key words) extracted from 929 documents.

4.1.3. Development of WORDSPACE Model

Semantic Vectors API package version 1.16[6] is used to create a WORDSPACE model. The source code for developing the WORDSPACE model is given in *Appendix IV*. To open the inverted file index for reading Lucene's API class

[5] Available at: http://www.getopt.org/luke/ (Date Accessed 22/11/2008)
[6] Available at: http://code.google.com/p/semanticvectors/ Date Accessed 14/03/2008

IndexReader.open method is used. To create basic random vector, in order to apply Random Projection for dimensionality reduction for the term-document matrix read from the inverted file index, Semantic Vectors' API class *VectorUtils*' *generateRandomVector* method is used. The dimension is reduced to 200 using the default value of the API. Lucene's *Term.field* method is used to filter out the *modified* and *title* fields as they are not used for thesaurus generation. The WORDSPACE model is composed of the normalized term vectors in the document collection.

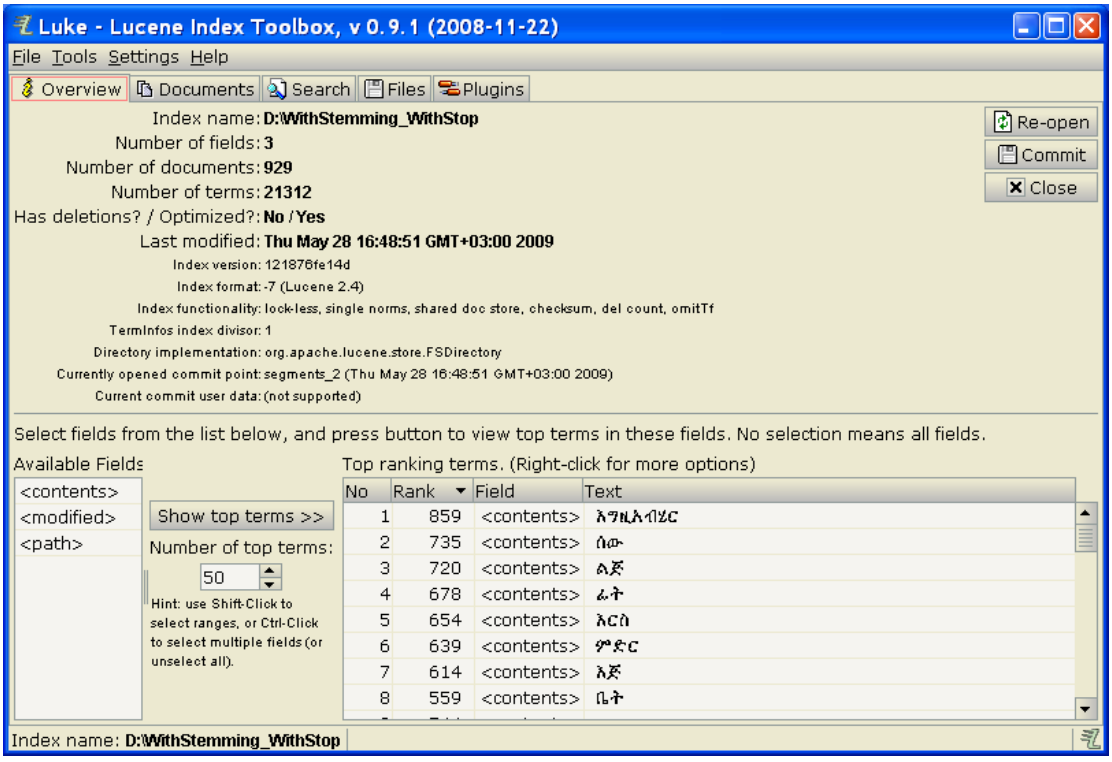


Figure 4.2. Screenshot of the inverted file index using Luke.

Finally, two methods, *WriteVectors and WriteVectorsAsText,* in Semantic Vectors' API class called *VectorStoreWriter,* provides methods for writing the learned vectors to disk, using one of the implemented file formats (i.e., either an optimized binary format or a simple pipe-delimited text format), respectively. The binary format indexes are quite smaller than the plain text format. In our case, the binary

index size for the Bible term vectors is 2.67 Mega Bytes, where as the text format is 6.40 Mega Bytes. The text format is noticeably slower for searching. But it is valuable for interoperability with other software, for example, for exporting a model to be manipulated by another analysis tool such as Matlab (Widdows, 2008). Full descriptions of the methods from Lucene and Semantic Vectors API documentations are given in *appendix V*.

A pipe-delimited text format for the WORDSPACE model is shown in figure 4.3. In the figure the index is in the form $Name|a_1|a_2| \ldots |a_n$; where $Name$ is a string identifier (i.e., a word) and the following numbers are the coordinates of the corresponding vector. The original class uses Lucene's input and output package, which is proved to be much faster than the Java's native *java.io.DataOutputStream* (*Ibid*.). In the WORDSPACE model 20,315 term vectors are created. The difference in number of terms in the WORDSPACE model and the inverted file index is due to the fact that the index terms in the inverted file are from *title* (i.e., absolute path in the file system), *modified* (i.e., the file's modification date), and *content* (i.e., string of file content) fields of the documents. In the WORDSPACE model, however, terms related to the *title* and *modified* fields must be discarded as they are not needed for thesaurus generation as discussed in section 3.2. Total time taken to develop the inverted file and WORDSPACE model is 12 minutes and 8 seconds on a machine running Windows XP Professional version 2002 that has Intel dual Core CPU with 2.40 GHz (Giga Hertz) clock speed and 2 GB (Giga Bytes) of RAM.

ደፋር|0.0|0.0|0.0|0.0|0.15811388|0.15811388|0.0|0.0|0.0|0.0|0.15811388|0.0|0
.0|0.0|-
0.15811388|0.0|0.15811388|0.15811388|0.15811388|0.15811388|0.0|0.0|0.0|-
0.15811388|0.0|0.0|-0.15811388|0.15811388|0.0|-
0.15811388|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|
0.0|0.0|0.15811388|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.15811388|0
.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|-
0.31622776|0.0|0.0|0.0|0.0|-
0.15811388|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.15811388|0.0|0.0|0
.0|0.15811388|0.0|0.0|0.0|0.0|0.0|0.0|0.0|-0.15811388|0.0|0.0|0.0|-
0.15811388|0.15811388|0.0|0.0|0.0|0.0|0.0|-
0.15811388|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.15811388|0.0|0.0|0.0|0.0|0.0|-
0.15811388|-
0.15811388|0.0|0.0|0.0|0.0|0.0|0.0|0.15811388|0.0|0.0|0.0|0.15811388|0.0|0.
0|0.0|0.0|0.0|-
0.15811388|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.15811388|0.0|0
.0|0.0|0.0|-0.15811388|-
0.15811388|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.15811388|0.0|0.0|0.0|0
.0|0.15811388|0.0|0.0|0.0|0.0|0.0|-0.15811388|0.0|0.0|0.0|0.0|0.0|0.0|-
0.15811388|-0.15811388|0.0|-0.15811388|0.0|0.0|0.0
መክሊት|0.09647549|-0.07718039|0.019295098|0.15436079|-
0.019295098|0.07718039|-0.057885297|-0.057885297|0.038590197|-0.019295098|-
0.038590197|0.057885297|-
0.057885297|0.019295098|0.0|0.019295098|0.07718039|-0.038590197|0.0|-
0.038590197|-0.17365588|-0.09647549|-0.038590197|-
0.09647549|0.019295098|0.057885297|-0.038590197|0.0|-0.15436079|-
0.057885297|-0.13506569|0.0|-0.019295098|0.13506569|0.0|0.0|0.019295098|-
0.057885297|0.019295098|0.038590197|0.019295098|-
0.038590197|0.0|0.057885297|-0.07718039|-0.038590197|-
0.038590197|0.13506569|0.019295098|0.019295098|0.019295098|-0.17365588|-
0.038590197|0.07718039|-0.038590197|-0.28942648|-
0.038590197|0.038590197|0.0|-0.019295098|-0.038590197|0.019295098|-
0.019295098|0.11577059|-0.15436079|0.019295098|-0.13506569|-
0.09647549|0.11577059|-0.019295098|0.09647549|0.0|0.0|0.038590197|-
0.019295098|0.09647549|-0.038590197|0.11577059|-0.019295098|0.057885297|-
0.057885297|0.057885297|0.0|-0.019295098|0.11577059|-
0.09647549|0.019295098|0.13506569|0.07718039|-0.019295098|0.0|-
0.019295098|0.019295098|-0.07718039|0.07718039|-
0.019295098|0.057885297|0.11577059|0.0|-0.038590197|0.09647549|0.19295098|-
0.07718039|0.0|0.0|0.057885297|0.0|0.11577059|-0.07718039|-0.07718039|-
0.019295098|-0.057885297|0.0|0.0|0.019295098|0.038590197|-0.019295098|-
0.09647549|0.11577059|0.13506569|0.07718039|0.0|-0.019295098|-0.057885297|-
0.07718039|0.07718039|0.07718039|-0.057885297|0.07718039|-
0.019295098|0.038590197|0.0|0.057885297|0.07718039|0.07718039|0.038590197|-
0.09647549|-0.057885297|0.019295098|-
0.019295098|0.0|0.057885297|0.038590197|-0.13506569|-
0.038590197|0.038590197|-0.038590197|-0.019295098|0.038590197|-
0.019295098|0.07718039|0.0|-0.07718039|0.038590197|-
0.038590197|0.21224609|-0.07718039|-0.019295098|-
0.019295098|0.038590197|0.019295098|0.0|-0.07718039|-
0.13506569|0.019295098|0.057885297|0.0|-0.057885297|0.0|0.038590197|0.0|-
0.09647549|-0.11577059|0.038590197|-0.019295098|0.019295098|0.038590197
ጉልበት|0.0|0.0|0.0|0.0|0.15430336|0.0|-0.15430336|-
0.15430336|0.0|0.0|0.0|0.15430336|
0.15430336|0.0|0.15430336|0.0|0.0|0.0|0.0|0.15430336|0.0|0.0|0.0|0.0|0.0|0.

Figure 4.3. A sample of pipe-delimited text format for the WORDSPACE model.

4.1.4. Generation of Thesaurus Terms

In order to generate thesaurus terms, the WORDSPACE model is searched with a term for which thesaurus matches are required. The source code for generation of thesaurus terms is given in *Appendix IV*.

In order to generate thesaurus for a term, the term should be normalized for spelling inconsistency and stemmed. Normalization of spelling inconsistency of the term is done in the same manner as done in section 4.1.2 for index terms; and to stem the term the Amharic stemmer, which is implemented by Tessema (2007) in Java, is used. Since the term is normalized and stemmed, it is possible to proceed to the next operation.

The next operation is to open the WORDSPACE model and to select a term vector that corresponds to the term for which thesaurus is required. To open the WORDSPACE model *MMapDirectory.openInput* method of Lucene API is used.

To create and normalize a term vector for the given term *getAdditiveQueryVector* method of *CompoundVectorBuilder* Semantic Vectors' API is used. The method returns a normalized term vector created by adding together vectors retrieved from WORDSPACE model. Nearest neighbors of the term vector are computed by utilizing *VectorSearcher.getNearestNeighbors* method of Semantic Vectors API. The *getNearestNeighbors* method works as follows: Cosine similarity is computed based on the term vector and all the other term vectors in the model. The result provides a candidate list of vectors from which the nearest neighbor is to be chosen. The criterion for choosing nearest neighbors for the term is the highest cosine similarity value. That is, a term which has higher cosine similarity value

with the term, for which thesaurus is required, is more related to the term than another term with lower cosine similarity value.

To accept a query term and to display its thesaurus terms a graphical user interface is developed. The user interface is created using *Swing GUI Components,* which are parts of the Java programming platform. The screenshot of the graphical user interface is shown in figure 4.4.

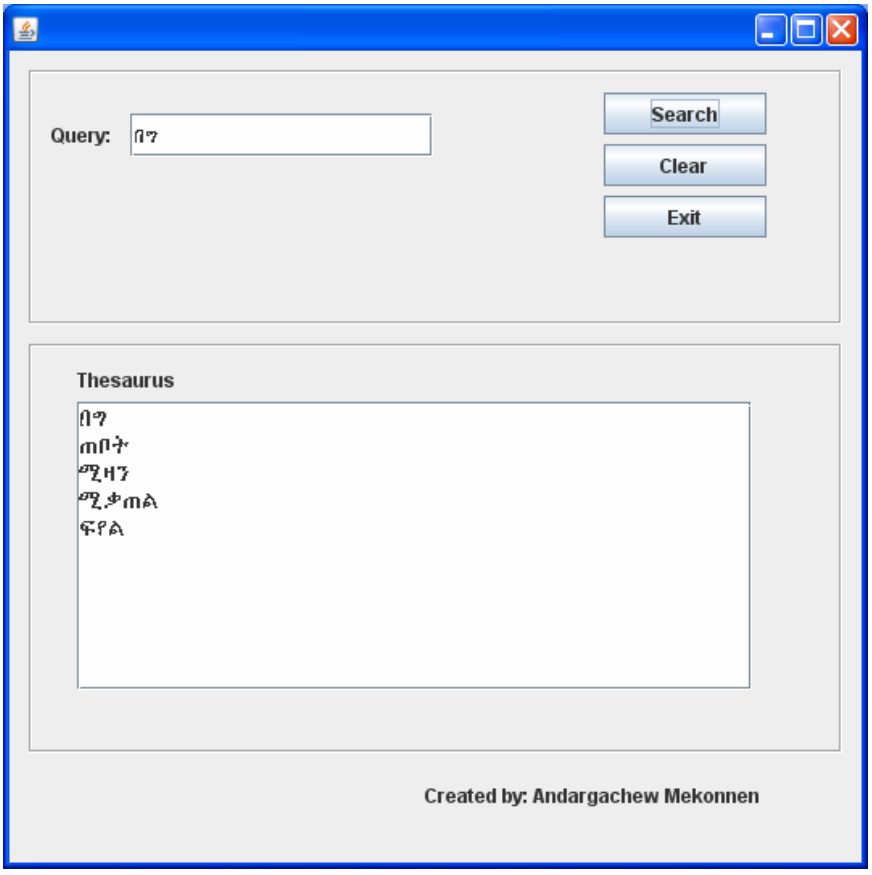


Figure 4.4. GUI of thesaurus generation system.

4.2. EVALUATION OF STEMMER AND STOPWORD

The stemming and stopword elimination components of the system are very crucial in the thesaurus generation process. The evaluation of these components are proceed as follows: random sample of thirty terms are selected from the WORDSPACE model and are examined whether these terms and their morphological

Table 4.2. Sample terms from the WORDSPACE model.

R.N.	Stemmed Terms	Properly stemmed	Stopword
1	ጥድ	Yes	No
2	አብርሀም	Yes	No
3	ታቦት	Yes	No
4	ሙስ	Yes	No
5	እፈትንህል እፈት እፈትናል እፈትናቸዋል እፈትናቸውማል እፈትንህል	No	No
6	ኢ.ያስ	Yes	No
7	ሽማግል	Yes	No
8	አላገኘህት አላገኘሁብ አላገኘሁት አላገኘህብ አላገኘኋት አላገኙት አላገኙእ አላገኛት አላገኝ	No	No
9	አልቀበሩት አልቀብ	No	No
10	በግ	Yes	No
11	እጥፍ	Yes	Yes
12	ኖህ	Yes	No
13	አጥብቅ	Yes	Yes
14	ዮፍታህ	Yes	No
15	አባረሩእ አባረር አባረርሁት አባር አባርረሩት አባርሩእ አባርራል አባርራቸዋል አባርራችኋቸውማል አባርር ትጠሩኛል ትጠሪኛል ትጠሪዋል	No	No
16	ትጠሪያል ትጠራል ትጠራታል ትጠራቸውም ትጠራዋል ትጠር	No	No
17	አዝሙድ	Yes	No
18	አለት	Yes	No
19	ኪዳን	Yes	No
20	ዘብዛል ዘብዛቸዋል	No	
21	ተናገረብ ተናገረኽ ተናገረውስ ተናገረውንስ ተናገሩብ ተናገሩት ተናገሩእ ተናገራት ተናገራችሁት ተናገር ተናገርሀት ተናገርሁት ተናገርኋ ተናገርኋት ተናገርሽ ተናገርን ተናገርኽ ተናጋር ተናግረሀል ተናግረናል ተናግረኽ ተናግረዋል ተናግራችኋል ተናግሬአል ተናግሬአችኋል ተናግሬያል ተናግር ተናግሮላችኋል ተናግሮባችኋል ተናግሮብሀል ተናግሮብሻል ተናግሮታል ተናግሮናል ተናግሮኛል ተናግሮአል ተናግአል	No	Yes
22	ተሉሀል ተሉታል ተሉት ተሉናል ተሉኛል ተሉአታል	No	Yes
23	ፈረስ	No	No
24	ተበላሽ ተበላሸት	No	No
25	አንበስ	Yes	No
26	ስብከት	Yes	No
27	ሽንት	Yes	No
28	እንጨት	Yes	No
29	መርመር መርምረሀል መርምር	No	No
30	መታየት	Yes	No
Total		19	4
Percentage		63%	13%

variants are stemmed to the same root; and if these sampled terms are not stopwords. Cochran (1977) suggests taking thirty or more samples for this kind of

population of terms with unknown distribution for objective evaluation. The sampled terms are shown in table 4.2. According to the sample of terms from the WORDSPACE model 37% of the terms are not properly stemmed and 13% of the terms are stopwords.

There are some reasons for the improper stemming of the terms. The composition of complete list of affixes (prefixes and suffixes) and list of words associated with context sensitive rules requires intensive manual intervention. To compile exhaustive lists of affixes and list of words associated with context sensitive rules a team of domain experts should be engaged in a project work. Due to time constraint the lists employed in the stemmer are not adequate. The original stemmer adopts SERA for representation of the Amharic writing system where as the stemmer used in this research is based on Unicode with some adjustment on the original stemmer. Still, the other reason is that the original stemming algorithm assumes the Amharic writing system is alphabetic while the latest recognition of the writing system is syllabic.

Few stopwords appear in the sample because no standard and exhaustive stoplist (negative dictionary) is used to eliminate non-content bearing terms. Also, improperly stemmed terms have also affected the detection of stopwords. Stopword elimination is done after stemming; and if an improperly stemmed stopword is encountered, the system has no means to eliminate it as it is not found in the stoplist.

4.3. THESAURUS EVALUATION

Some properly stemmed terms are taken from the sample of table 4.2 and thesaurus is automatically generated for these terms using the developed system. Table 4.3 shows the terms and their corresponding thesaurus matches.

As table 4.3 shows 58% of the output has related concept to the respective terms. The rest 42% has no related concept. The reason for lower accuracy of the system is that the thesaurus generation algorithm misses homographs (e.g., **በዓል** meaning celebration or god is taken as a single term). The other reason for this anomaly may be that of the improperly stemmed terms and the undetected stopwords. If all of the

Table 4.3. Sample terms and their thesaurus matches.

Term	Thesaurus	
	Related Concepts	Non-related Concepts
ሙሴ	ሙስ አሮን	ማህበር ደጃፍ ደቃቅ
በግ	በግ ጠቦት ፍየል	ሚዛን ሚቃጠል
ታቦት	ታቦት አቢዳር አቤንኤዘር ተሸከም	እርቃን
ኖህ	ኖህ መርከብ ተደመሰስ	አብረው፡ት አይቼሀል
አለት	አለት ድንጋይ ወነጭፍ	ቁባት ሚመዝዝ
አብርሀም	አብርሀም ሳር ይስሀቅ ያህው	ባቱኤል
አንበሳ	አንበስ ደቦል ግልገል	በል ገጠም
እንጨት	እንጨት ጥድ ግራር	ደርብ ቅድስት
ኪዳን	ኪዳን ቃል	ሞግዚት ካራው፡ያን ጎቶልያ
ፈረስ	ፈረስ ሰረገል	አላብ ዝንጀር ትፈትን
Total	29	21
Percentage	58%	42%

terms were properly stemmed and stopwords were controlled, a better output would have been expected.

CHAPTER FIVE

APPLICATION OF THESAURUS IN IR SYSTEM

For application of thesaurus, an IR system is implemented and its retrieval performance is evaluated based on precision and recall. The precision and recall measures are computed before and after using thesaurus for query expansion.

5.1. DEVELOPMENT OF RETRIEVAL SYSTEM

For application of the thesaurus, an IR system is developed. The system architecture is depicted in figure 5.1. The system accepts document collections, applies text operation on the documents, and adds the documents to the inverted file index. When a query is submitted to the system, it searches for most likely relevant documents either using the query term alone or using the expanded query based on the thesaurus. The system components are discussed in the next subsections.

5.1.1. Text Operation and Indexing System Components

The text operation and indexing system components are designed as their counterparts of the thesaurus generation system in chapter three.

5.1.2. Searching System Component

When a user enters query the system searches for most likely relevant documents either using the query term alone or using the expanded query based on the thesaurus.

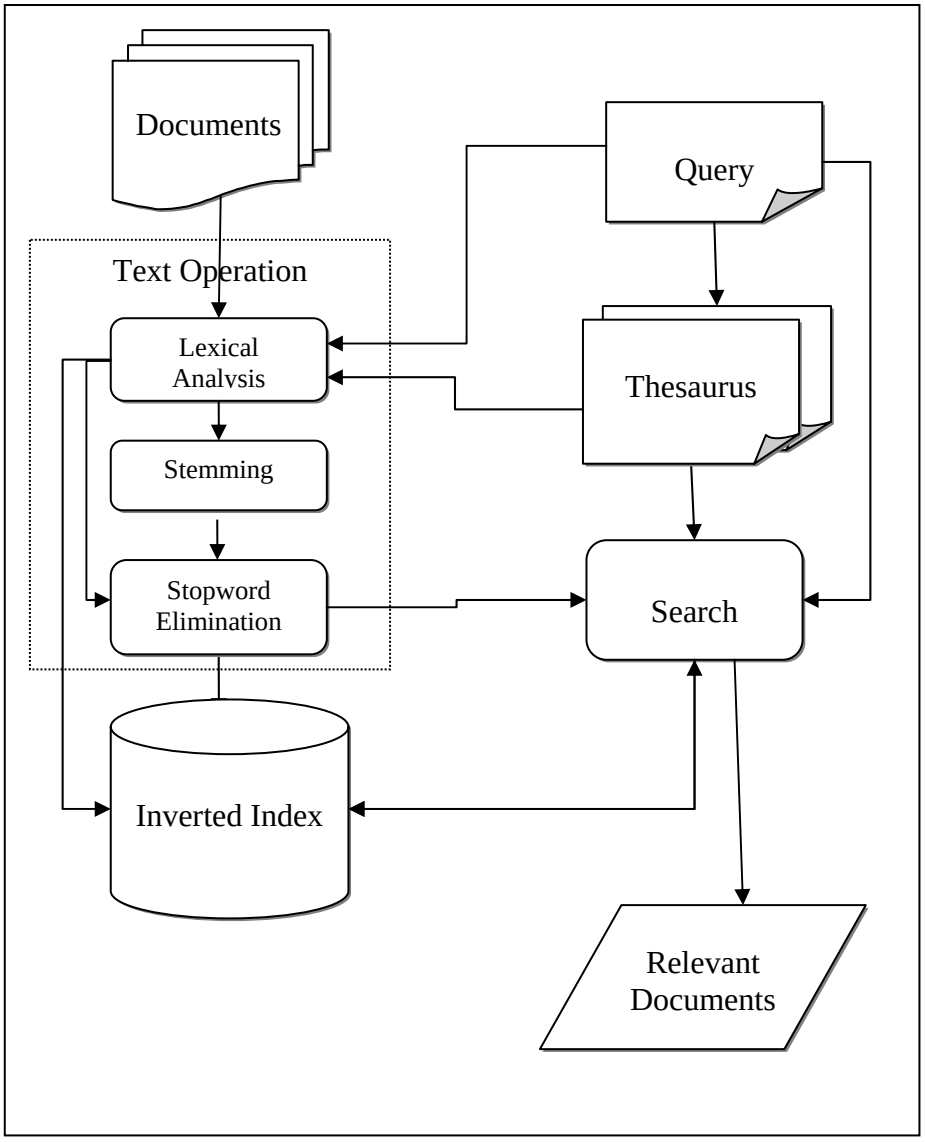


Figure 5.1. System architecture for IR system

The algorithm for the system's searching component is given in figure 5.2. The query term is first analyzed. That means, normalization of characters, lexical analysis, stemming, and stopword elimination will be done on the query term as it is fully discussed in section 3.1 for creation of inverted file index. Then the query term is parsed. Typically, a query parser is used to translate the user-specified keywords into a query with various operators that is executed against the underlying indexes. If query is expansion is selected, the system automatically generates thesaurus terms for the query term and computes a score (a numeric

value of relevance) for each document, given a query. The output results are not the actual matching documents, but rather are references to the documents matched.

1. Accept query terms
2. Analyze the query terms
3. Parse the query terms
4. If query expansion is selected
    a. Generate thesaurus for the query terms
5. Open the inverted index
6. Search the inverted index with query
7. Display most likely relevant documents for the query

Figure 5.2. Algorithm for the search component.

5.2. IMPLEMENTATION OF IR SYSTEM

An IR system is implemented by using Lucene API package. The text operation and indexing system components are implemented as their counterparts of the thesaurus generation system in section 4.1. When a user enters query the system searches for most likely relevant documents either using the query term alone or using the expanded query based on the thesaurus.

QueryParser.parse method of Lucene API is used to parse query terms; and IndexSearcher.search method is used to search the inverted file index. Full description of the APIs is given in *Appendix V*.

The graphical user interface of the system, which is implemented in Java using its Swing components, is shown in figure 5.3.

The Amharic Bible corpus is classified into training and testing datasets. Then precision and recall of the IR system developed is computed before and after using the thesaurus for query expansion.

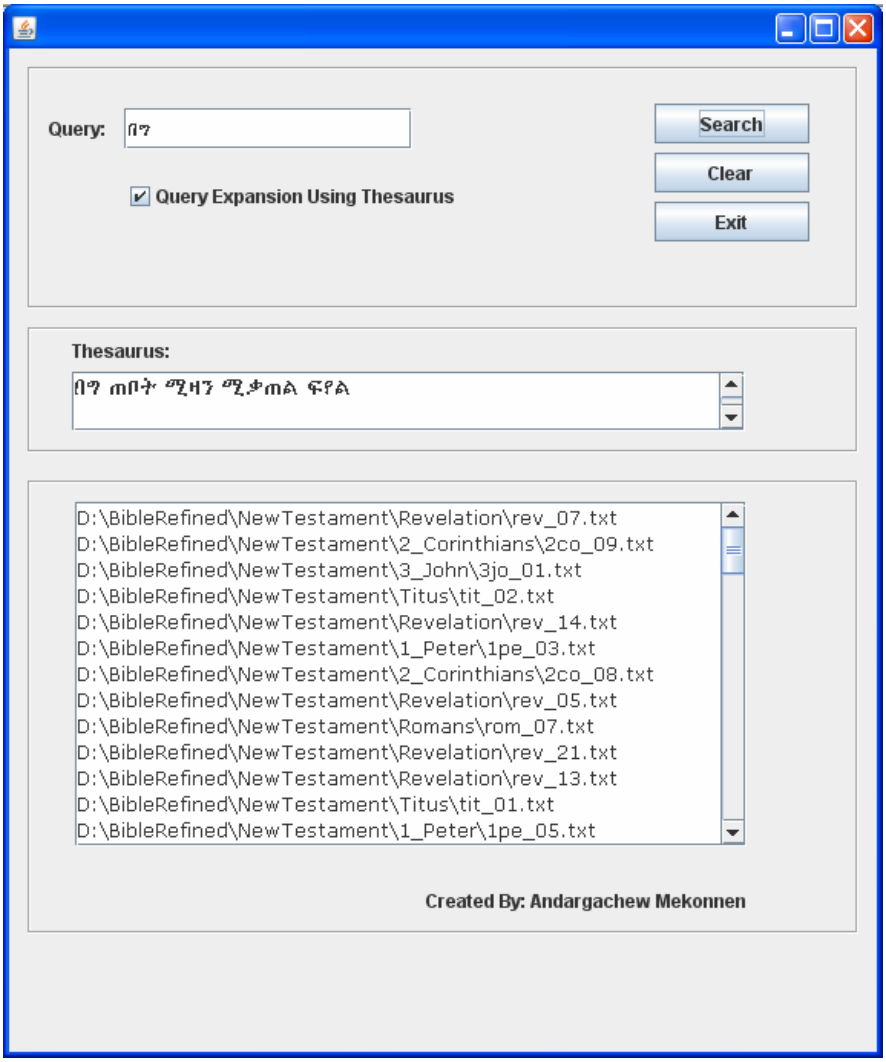


Figure 5.3. Graphical user interface of the IR system.

Already WORDSPACE model has been developed based on the training dataset (i.e., Old Testament documents) for the generation of the automatic thesaurus. When query expansion is required, automatic thesaurus is generated based on the WORDSPACE model.

An inverted file index is constructed based on the testing dataset (i.e., New Testament documents) in order to evaluate the effectiveness of the thesaurus in query expansion while searching the testing dataset. The evaluation is carried out in terms of precision and recall of the IR system with query expansion and without query expansion.

5.3. EVALUATION OF IR SYSTEM

For objective evaluation of the automatically generated thesaurus, random samples of thirty terms are selected from the WORDSPACE model, *see table 4.2*. Among the thirty terms only ten are selected as a query term, since twenty of the sampled terms are found to be either stopwords which are not controlled, not properly stemmed by the developed system, or convey only a general concept. The discussion of the sampled terms with regard to stemming and stopword elimination is found in section 4.2. Since it is difficult to compile a list of relevant items for general concept terms and stopwords from the testing dataset, these terms are not used as query term. Then the testing dataset is searched in two different scenarios: with the query term alone and with the expanded query. In order to generate thesaurus terms that are used for query expansion, the WORDSPACE model is searched with a query term that is provided. The top five most related thesaurus terms for the query are used for query expansion. If more than the top five terms are selected, the output of relevant documents for expanded query will be too much. In order to make it manageable only the top five most related terms are used for query expansion. Finally, the precision and recall of the system is computed in these scenarios. The experimentation result is shown in table 5.1. The relevant documents for each query term are composed with domain experts.

Table 5.1. Precision and recall results of the experimentation.

QUERY	REL	USING THESAURUS						WITHOUT USING THESAURUS					
		RET	RETREL	NRETREL	RETNREL	R	P	RET	RETREL	NRETREL	RETNREL	R	P
ሙሴ	47	77	46	1	31	0.9787	0.5974	24	24	23	0	0.510638	1
በግ	33	80	19	14	61	0.5758	0.2375	8	7	26	1	0.212121	0.875
ታቦት	6	2	2	4	0	0.3333	1	2	2	4	0	0.333333	1
ኖህ	7	18	5	2	13	0.7143	0.2778	3	3	4	0	0.428571	1
አለት	41	17	13	28	4	0.3171	0.7647	4	4	37	0	0.097561	1
አብርሀም	27	76	26	1	50	0.963	0.3421	11	11	16	0	0.407407	1
አንበሳ	6	16	6	0	10	1	0.375	5	4	2	1	0.666667	0.8
እንጨት	4	34	4	0	30	1	0.1176	2	1	3	1	0.25	0.5
ኪዳን	31	27	14	17	13	0.4516	0.5185	16	10	21	6	0.322581	0.625
ፊረስ	4	20	4	0	16	1	0.2	2	2	2	0	0.5	1
AVERAGE						0.7334	0.4431					0.372888	0.88

REL - the number of items that are relevant

RET - the number of items retrieved

RETREL - the number of items retrieved and relevant

NRETREL - the number of items relevant but not retrieved

P - Precision

RETNREL - the number of items retrieved but not relevant

R – Recall

The experimentation result in table 5.1 shows average precision and recall in terms of the user-oriented recall-average, which reflects the performance an average user can expect to obtain from the system. The average recall values are 37.29% and 73.34% before and after using thesaurus for query expansion, respectively. The corresponding average precisions are 88% and 44.31%, before and after using thesaurus. While there is an enhancement in recall of the system in using thesaurus, there is a decline in precision. The reason for the decline in precision is that automatically generated thesauri are used to improve retrieval performance by substituting the appropriate cluster of terms for one of its members. The heterogeneous nature of the clusters enhance recall of the retrieval system while decline the precision. For instance, if our query term is **በግ** and search documents without query expansion, then all documents that contain the term **በግ** will be retrieved. Hence, the precision will be high but the recall will be low. On the other hand, if the searching is with query expansion, not only the relevant documents to the query term **በግ** will be retrieved but also those relevant to thesaurus matches of **በግ** such as **ጠቦት** and **ፍየል**. In this case while the recall will be superior to the previous case, the precision will be lower. So, the usage of the thesaurus for recall oriented IR systems is very crucial.

CHAPTER SIX

CONCLUSION AND RECOMMENDATION

This chapter presents the conclusion and recommendations for future study based on our findings.

6.1. CONCLUSION

Amharic has been the written language for a couple of centuries and has been widely spoken in Ethiopia and other parts of the world either as a mother tongue or a second language. The desktop publishing, the advent of the Internet, and the introduction of Unicode and its inclusion of the Ethiopic writing system – the writing system which Amharic uses – have brought a new era for the digitization of the language. As a result huge volume of Amharic information is accumulated in digitized format. This in turn leads to information explosion. IR has a solution for this newly emerging problem.

Among the components of IR systems, thesauri play a crucial role for the enhancement of recall. Manual construction of thesauri is a labor intensive and time-consuming task; and also suffers low coverage. It is costly, as it requires highly skilled experts in a subject domain; and is a highly conceptual and knowledge intensive task. Therefore, there is always a need towards the development of a specialized system to generate thesaurus automatically. The unique requirement of the system is that it should be domain independent, unlike the manual thesaurus construction techniques. In addition, the system should be scalable to process huge collection of documents; and it should have fast response time especially when it is integrated to IR systems.

A corpora based automatic thesaurus generation approach is used in this research to construct thesaurus for Amharic document collections. A WORDSPACE model, which is used to generate thesaurus for a term, is developed by the thesaurus generation system. In order to construct a WORDSPACE model, first an inverted file index is constructed. Apache Lucene and Semantic Vectors APIs are used to construct the inverted file index and the WORDSPACE model, respectively. These APIs have been proved for having powerful features for the development of IR systems (Widdows and Ferraro, 2008; Hatcher, 2005).

The Amharic Bible documents are used to train and test the thesaurus generation system developed. 78% and 22% of the documents are used as the training and testing dataset, respectively.

Automatic Amharic thesaurus generation system is implemented for Amharic Bible documents using Apache Lucene and Semantic Vectors APIs. Since the stemming and stopword elimination components of the system play an important role in the thesaurus generation process, these components are evaluated. The evaluation proceeds as follows: random samples of 30 terms are selected from the WORDSPACE model and are examined carefully whether these terms are properly stemmed or are not stopwords. The objective evaluation of the stemming and stopword elimination components of the system shows that majority of the terms are properly stemmed and are not stopwords. Only 37% of the terms are not properly stemmed and only 13% of the terms are found to be stopwords. However, further research is recommended for the development of a better stemming algorithm and stopword elimination technique to enhance the automatic

generation of thesaurus. The original stemmer has been reported to have an accuracy of 95%. But its implementation in Java for Unicode encoded texts has lower accuracy. This is mainly due to the stemmer was developed with the assumption that Amharic writing system is alphabetic, not syllabic, and utilizes SERA for representation of Amharic letters. But in this research it is adapted to Unicode encoded text which assumes a syllabic writing system. Also, due to time constraint exhaustive list of affixes and list of words associated to context sensitive rules are not utilized.

The accuracy of the automatic Amharic thesaurus generation system is evaluated. A random sample of ten terms is selected from the WORDSPACE model and thesaurus is generated for each term automatically. The experimentation result shows 58% of the output has related concept to the respective terms. The rest 42% has no related concept. The reason for this anomaly may be that of the improperly stemmed terms and the undetected stopwords. If all of the terms were properly stemmed and stopwords were controlled, a better output would have been expected.

In the application of the thesaurus for query expansion, an IR system is developed and its retrieval performance is evaluated based on precision and recall. The precision and recall measures are computed before and after using thesaurus for query expansion. In order to evaluate the system ten queries are randomly selected from the WORDSPACE model for the computation of precision and recall. Then the testing dataset is searched in two different scenarios: with the query term alone and with the expanded query. In order to generate thesaurus terms that are used for query expansion, the WORDSPACE model is searched

with a query term provided by the system user. The top five most related thesaurus terms for the query are used during query expansion. The experimentation result shows an average precision and recall in terms of the user-oriented recall-average. The average recall values are 37.29%and 73.34% before and after using the thesaurus for query expansion, respectively. The average precisions are 88% and 44.31% before and after using the thesaurus for query expansion, respectively. While there is an enhancement in recall of the system in using thesaurus, there is a decline in precision. The reason for the decline in precision is that automatically generated thesauri are used to improve retrieval performance by substituting the appropriate cluster of terms for one of its members. The heterogeneous nature of the clusters enhance recall of the retrieval system while decline the precision. For instance, the results of tests done by Foo *et al*. (2000) and Chen *et al*. (1995) have shown improved in recall and decline in precision. So, the usage of the thesaurus for recall oriented IR systems is very crucial.

The developed system for automatic thesaurus generation for Amharic text retrieval is domain independent. That means, the system can be trained and used in a different domain. Since it is domain independent, it can be applied to other domains without any further modification as long as the document collections have a UTF-8 Unicode encoding. Another outstanding feature of the system is its efficiency in terms of its response time. It takes only 12 minutes and 8 seconds to create 20,315 term vectors in the WORDSPACE. After the WORDSPACE is created it takes only few milliseconds to generate thesaurus for a given term. The WORDSPACE is created only once –and of course offline– unless new documents are added to the training dataset. So, the automatic thesaurus

generation system can be an ideal component for real time IR systems that provide fast response.

The WORDSPACE model is very sensitive to spelling errors since it considers a term wrongly spelt in different places as different terms. Besides, the model is built by assuming co-occurrence of terms within the “context window” at document level. But, the “context window” can be smaller portion of a document (e.g., a paragraph).

6.2. RECOMMENDATION

Based on the findings of the research we would like to make the following recommendations for future work:

- Automatic thesaurus generation based on WORDSPACE model with another dimensionality reduction algorithms like Hyperspace Analogous to Language.
- Automatic thesaurus generation based on "sliding context window" approach, which is a variant of WORDSPACE model.
- Automatic thesaurus generation based on some other clustering algorithms such as k means clustering and single link clustering.
- Automatic thesaurus generation for multilingual thesauri which have parallel corpora (e.g. Bible and Legal Codes) with Amharic language.
- A domain independent Amharic stemming algorithm that does not require much human intervention and can process Amharic text with a Unicode encoding.
- An efficient method to eliminate non-content bearing terms (stopwords) or a standard Amharic stoplist.
- Development of Amharic spelling and grammar correction.
- Development of standard Amharic corpus that can be used for training and testing systems developed in the area of IR and Natural Language Processing (NLP).

REFERENCES

Aitchison, J., Gilchrist, A., and Bawden, D. (2000). *Thesaurus Construction and Use: a Practical Manual*, Aslib, Fitzroy Dearborn Publishers, Chicago, U.S.A.

Alemayehu, N. and Willet, P. (2002). Stemming of Amharic Words for Information Retrieval. *Literary and Linguistics Computing* 17(1):1-17.

Alemu, A., Asker, L. (2007). An Amharic Stemmer: Reducing Words to their Citation Forms. *Proceedings of the 5th Workshop on Important Unresolved Matters*. Association for Computational Linguistics, Prague, Czech Republic, pp. 104-110

Ask (2009). *Amharic Language*. Available at: http://ask.reference.com/web?q=Amharic+Language&qsrc=2892&l=dir&o=10601 (Date Accessed 7/02/2009)

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*, ACM Press, Addison-Wesley Longman Limited, New York, U.S.A.

Bekele, H. (1992). *Construction of a Computer Based Amharic Thesaurus in Water Technology for Use with Information Storage and Retrieval Systems*, Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Bizuneh, M. (2003). *The Application of WEBSOM for Amharic Text Retrieval*, Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Bloor, T. (1995). *The Ethiopic Writing System: a Profile*. Available at: http://www.spellingsociety.org/journals/j19/ethiopic.php (date accessed 8/05/2008)

Brants, T., Chen, F., and Tsochantaridis, L. (2002). Topic-Based Document Segmentation with Probabilistic Latent Semantic Analysis. In *Conference on Information and Knowledge Management (CIKM)*, pp. 211–218.

Chen, H., Schatz, B., Yim, T., and Fye, D. (1995). Automatic Thesaurus Generation for an Electronic Community System. *Journal of the American Society for Information Science*. 46(3): 175–193.

Cochran, G. W. (1977). *Sampling Techniques*, 3rd ed., Wiley Publishers, New York, U.S.A.

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Encyclopedia Britannica (2008). *Encyclopedia Britannica Online*. Available at: http://www.britannica.com/EBchecked/topic/20500/Amharic-language

Foo, S., Hui, S. C., Lim, K. H., and Hui, L. (2000). Automatic Thesaurus for Enhanced Chinese Text Retrieval. *Library Review*, MCB University Press, 49(5):230-239.

Foskett, D. J. (1997). Thesaurus. In *Readings in Information Retrieval* K. Sparck Jones and P. Willet (Eds), Morgan Kaufmann Publishers, Inc., pp. 111-134.

Ghwanmeh S., Kanaan G., Al-Shalabi R., and Aisrehin N. O. (2005) Automatic Query Expansion for Arabic Text Retrieval System Based On an Association Clustering Thesaurus, *Information Technology Journal.* Asian Network for Scientific Information, 4 (4): 476-483.

Harman, D., Fox, E., Baeza-Yates, R., and Lee, W. (1992). Inverted Files. In Frakes, W. B. and Baeza-Yates, R. *Information Retrieval: Data Structures and Algorithms*. Englewood Cliffs, Prentice Hall.

Hatcher, E. and Gospodnetic, O. (2005). *Lucene in Action*. Manning Publications Co., Greenwich, CT, U.S.A.

Hearst, M., and Schutze, H. (1993). Customizing a Lexicon to Better Suit a Computational Task. In *ACL SIGLEX Workshop*, Columbus, Ohio.

Keller, E. (2009). *Powered by*. Available at: http://wiki.apache.org/lucene-java/PoweredBy (Date Accessed 12/03/2009)

Kitaw, Y. *Cultural Identity and Local Content Development on the World Wide Web*. available at: http://unpan1.un.org/intradoc/groups/public/documents/un-ther/unpan022040.pdf (date accessed 8/06/2008)

Lassi, M. (2002). *Automatic Thesaurus Construction*. Graduate School of Language Technology, Swedish School of Library and Information Science, University College of Borås. Available at: http://www.adm.hb.se/~mol/gslt/thesauri.pdf (date accessed 4/09/2008)

Lee, W. (1997). *Life Study of Genesis*, Living Stream Ministry, Anaheim California, U.S.A.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Available at: http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf (date accessed 2/05/2008)

Mcarthur, R., and Bruza, P.D. (2003). Dimensional Representations of Knowledge in Online Community. In *Chance Discovery*, Ohsawa, Y. and McBurney, P. (Eds), Springer-Verlag, pp. 98–114.

Million, M. (2000). *A Generalized Approach to Optical Character Recognition (OCR)*, Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Saba, A. (2001). *The Application of Information Retrieval Techniques to Amharic Documents on the Web*, Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Sahlgren, M. (2005). An Introduction to Random Indexing. In H. Witschel, *Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering*, TKE'05,Copenhagen, Denmark.

Sahlgren, M. (2006). *The Word-Space Model Using Distributional Analysis to Represent Syntagmatic and Paradigmatic Relations Between Words in High-Dimensional Vector Spaces*, Doctoral Dissertation, Department of Linguistics, Stockholm University. Universitetsservice US-AB, Sweden.

Salton, G., and McGill, M. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill, New York.

Samuel, E. (2005). *Amharic Text Retrieval Using Neural Networks (NN): A Comparative Study Using News Items*, Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Schutze, H. (1997). *Ambiguity Resolution in Language Learning*. CSLI Publications, Stanford, CA.

Schutze, H. (1998). Automatic Word Sense Discrimination. *Computational Linguistics*, 24(1):97-124.

Srinivasan, P. (1992). Thesaurus Construction. In *Information Retrieval: Data Structures and Algorithms,* Frakes, W. B. and Baeza-Yates, R., Englewood Cliffs, Prentice Hall.

Tessema, M. (2007). *Design and Implementation of Amharic Search Engine*, Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Tewodros, H. (2003). *Amharic Text Retrieval: An Experiment Using Latent Semantic Indexing (LSI) with Singular Value Decomposition (SVD)*, Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Unicode (2006). *The Unicode 5.0 Standard*, The Unicode Consortium. Pearson Education, Inc.

Widdows, D. (2006). *Comparison of Semantic Vectors package with Infomap-NLP*. Available at: http://code.google.com/p/semanticvectors/wiki/InfomapComparison (date accessed 8/06/2008)

Widdows, D. (2003). Unsupervised Methods for Developing Taxonomies by Combining Syntactic and Statistical Information. In *Proceedings of Human Language Technology/ North American Chapter of the Association for Computational Linguistics*, Edmonton, Canada. pp. 197-204.

Widdows, D. and Ferraro, K. (2008). Semantic Vectors: A Scalable Open Source Package and Online Technology Management Application. In *Proceedings of the Sixth International Language Resources and Evaluation* (LREC'08), Marrakech, Morocco, pp. 55–73.

Widdows, D. (2008). *The Semantic Vectors Package*. Available at: http://code.google.com/p/semanticvectors/ (date accessed 8/06/2008)

Witten, I. H. and Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Technique*, 1st edition, Morgan Kaufmann, San Francisco, CA.

Wolfram (2009). Orthogonal Matrix. *Wolfram Math World*. Available at: http://mathworld.wolfram.com/OrthogonalMatrix.html (date accessed 1/03/2009)

Wondwossen, M. (2004). *OCR for Special Type of Handwritten Amharic Text ("Yekum Tsifet")*. Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Worku, A. (1997). *Application of OCR Techniques to the Amharic Script*. Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Yacob, D. (2004). *Application of the Double Metaphone Algorithm to Amharic Orthography*. Available at: http://yacob.org/papers/DanielYacob-ICESXV.pdf (date accessed 2/06/2008)

Yacob, D. (2005). *Developments Towards an Electronic Amharic Corpus*. Available at: http://yacob.org/papers/DanielYacob-TALN2005.pdf (date accessed 2/06/2008)

Yacob, D. (2006). *Unicode for Under-Resourced Languages*. Available at: http://yacob.org/papers/DanielYacob-SALTMIL2006.pdf (date accessed 2/06/2008)

Yalemsew, M. (2005). *Application of Hierarchical Document Clustering Based Information Retrieval System for Amharic Documents*. Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Yaregal, A. (2002*). Development of Versatile Character Recognition System for Amharic Text*, Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

Yoseph, S. (2005). *Application of Multilingual Thesaurus for Cross Language Information Retrieval (CLIR) [Amharic - English CLIR for the Legal Environment]*, Masters Thesis, Addis Ababa University, Addis Ababa, Ethiopia.

You, J. and Chen, K. (2006). Improving Context Vector Models by Feature Clustering for Automatic Thesaurus Construction. *Fifth SIGHAN Workshop on Chinese Language Processing*, pp. 1-8.

APPENDICES

Appendix I

Ambiguities in Amharic writing. (Source: Yacob (2004))

Type 1: Syllographic Redundancy

Canonical Amharic	Common Amharic
ሆሣዕና	ሆሳእና, ሆሳና, ሆሣና
ሐምሌ	ሀምሌ, ሃምሌ, ኃምሌ
ሒሳብ	ሂሣብ
ሕማ	ህማ
ሥላሴ	ስላሴ, ስላሤ, ሥላሤ
ሥጋ	ስጋ
ታኅሣሥ	ታህሳስ
ኀምሳ	ኃምሳ, ሃምሳ
ኃይለ	ኀይለ, ሀይለ, ሃይለ, ሐይለ
ድኅና	ደህና
ዐወቀ	አወቀ
ዓለምፀሐይ	ዓለምፀሃይ, ዓለምፀሀይ,ዓለምጸሐይ, ዓለምጸሃይ, ዓለምጸሀይ, ዐለምፀሐይ, ዐለምፀሃይ, ዐለምፀሀይ, ዐለምጸሐይ, ዐለምጸሃይ, ዐለምጸሀይ, አለምፀሐይ, አለምፀሃይ, አለምፀሀይ, አለምጸሐይ, አለምጸሃይ, አለምጸሀይ
ዓሣ	አሳ, ዓሣ
ዕንቊጣጣሽ	እንቁጣጣሽ
ወሀ	ውኃ, ዉሃ

Type 2: Glypheme Misidentification

Canonical Amharic	Common Amharic
ሲንሰርሺፕ	ሲንሰርሺኘ
ቄነሰ	ቆነሰ
ቄይ	ቆይ
ቄጠረ	ቀጠረ, ቆጠረ
ቊጥራቸውም	ቁጥራቸውም
ቊረጥ	ቁረጥ
ቊርባን	ቁርባን
ቊር	ቁር
ቊርስ	ቁርስ
ብርኰት	ብርኮት
ነጉላላ	ነሁላላ
ነው	ነዉ
አውሮፕላን	አውሮኘላን
ኢትዮጵያ	ኢትዮድያ
ኮከብ	ኰከብ
ኮአፕሬሽን	ኮአኘሬሽን
ኰነነ	ኮነነ
ኰላብ	ኩላብ
ደቄስ	ደቆስ
ዲፕሎማት	ዲኘሎማት
ዲፕሎማሲ	ዲኘሎማሲ
ዲሲፕሊን	ዲሲኘሊን
ጎንዳር	ጐንዳር
ጎጃም	ጐጃም
ጕበት	ጉበት
ጕዳይ	ጉዳይ
ፕሪሚየር	ኘሪሚየር
ፕራይቬታይዜሽን	ኘራይቬታይዜሽን
ፕሬስ	ኘሬስ
ፕሬዚዳንት	ኘሬዚዳንት
ፕሮጀክት	ኘሮጀክት
ፕሮግራም	ኘሮግራም
ፕሮፌሰር	ኘሮፌሰር
ፕሮፖጋንዳ	ኘሮፖጋንዳ, ፕሮፖጋንዳ, ኘሮፖጋንዳ
ፖሊቲካ	ፖሊቲካ

Type 4: Assimilations and Alternations

Canonical Amharic	Common Amharic
ሀገር	አገር
ላንፋ	ላምፋ
ሸንብራ	ሸምብራ
ብሎአቸው	ብሎዋቸው
ቅርንፉድ	ቅርምፉድ
ተባዕት	ተባት
ጎምሳ	አምሳ
አንበሣ	አምበሳ
አንበጣ	አምበጣ
እንቢ	እምቢ
እንብርት	እምብርት
አንፋር	አምፋር
ከበጉዋይ	ከበጓይ, ከበጉይ
ውሽንፍር	ውሽምፍር
ወንበር	ወምበር
ዝንብ	ዝምብ
ግልንቢጥ	ግልምቢጥ
ግንብ	ግምብ
ግንፎ	ግምፎ
ጥንብ	ጥምብ

Type 3: False Ge'ezisms

Canonical Amharic	Common Amharic
ሀገር	ኃገር
ሁለት	ኁለት, ኹለት
ሁሉ	ኁሉ, ኹሉ
ምልክት	ምልእክት, ምልዕክት
ትዕግሥት	ትዕግዕሥት

Appendix I (Continued)

Type 5: Orthographic Abbreviations and Elisions

Canonical Amharic	Common Amharic
መልአኩ	መላኩ
ሚያዝያ	ሚያዚያ
ማርያም	ማሪያም
ምክንያት	ምክኒያት
በንደ	በእንደ
ሰማንያ	ሰማኒያ
ነጭላውያን	ነጭላዊያን
አንባብያን	አንባቢያን
አብዮታውያን	አብዮታዊያን
ኢትዮጵያውያን	ኢትዮጵያዊያን
አንባብያን	አንባቢያን
ክርስቲያን	ክርስትያን
ወንጌላውያን	ወንጌላዊያን
የአማርኛ	ያማርኛ

Type 6: Disjoint Labiovelars

Canonical Amharic	Common Amharic
ሆኗል	ሆኖዋል, ሆኖአል
ቍንጫ	ቁንጫ
ተቃውሞዋች	ተቃውሞአች
በእርስዋም	በእርሷም
ዓድዋ	ዓዷ, አድዋ
ይዟል	ይዞአል
ጀሮአቸውን	ጀሮዋቸውን
ገልጿል	ገልፀዋል
ጐረመሰ	ጎረመሳ
ጡዋት	ጥዋት, ጠዋት, ጧት

Type 7: Dialect Variations

Canonical Amharic	Common Amharic
ሂጅ	ሂጂ
አይዶለም	አይደለም
ዐመፀ	ዐመጠ
ዓዜ	ዓጤ, አዜ, ሐዜ

Type 8: Foreign Language Transcription

Canonical Amharic	Common Amharic
ቴክኖሎጂ	ቴክኒዎሎጂ
አቪኖር	አቢኖር
ኢሜይል	ኢሜል, ኤሜል, ኤሜይል
ኢንተርኔት	ኢንተርነት, ኢንቴርኔት, ኢንቴርኔት
ኮምፒዩተር	ኮምፒውተር
ፕሬዚዳንት	ፕረዚደንት

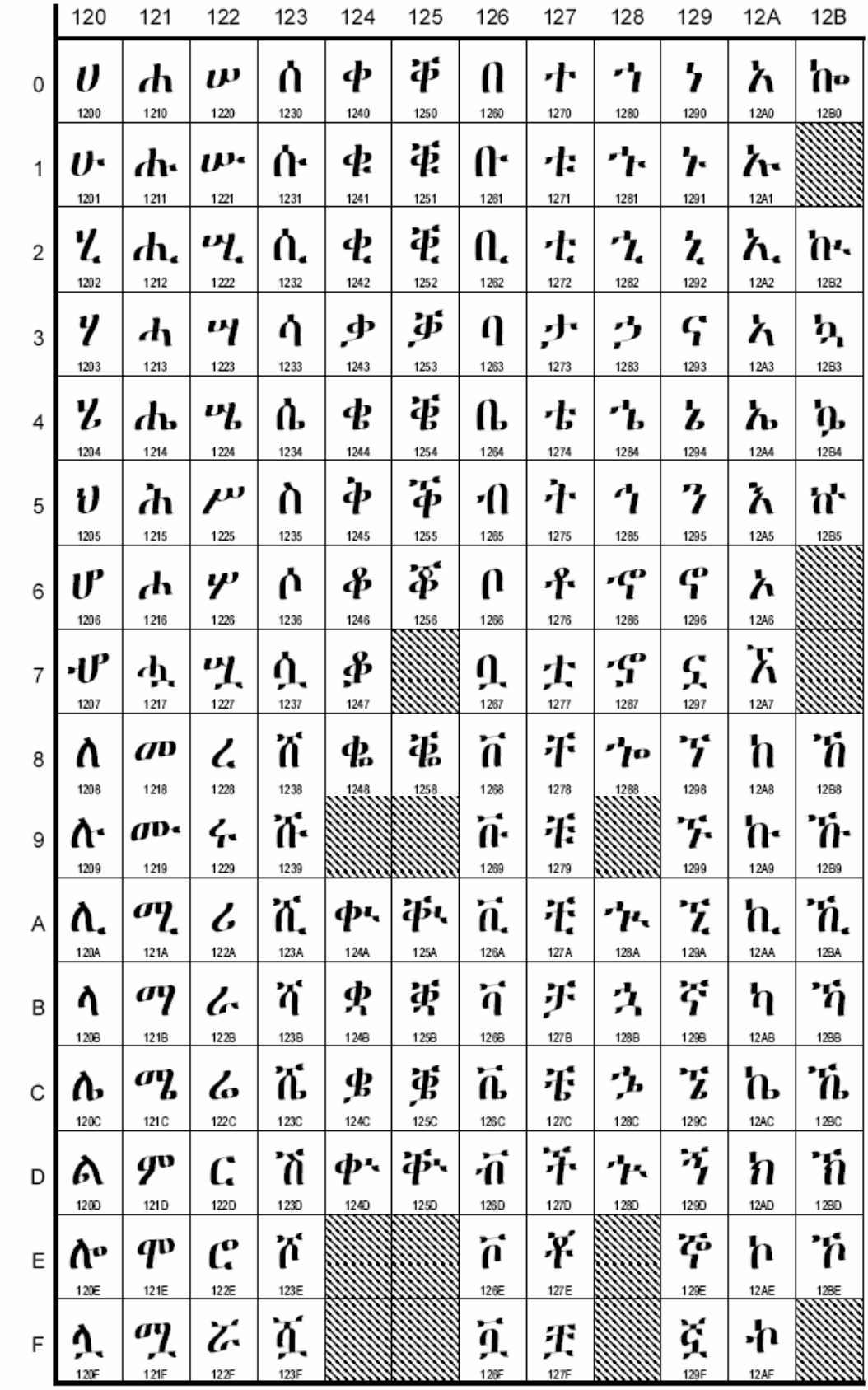
Type 9: Mistrikes

Canonical Amharic	Common Amharic
ሥርዓት	ሥርአት, ሥራት
አርአያ	አራያ
ኢትዮጵያ	ኢትዮፕያ
ጤና	ጠና, ቴና, ጤኛ
ወጤት	ወጢጥ

Appendix II

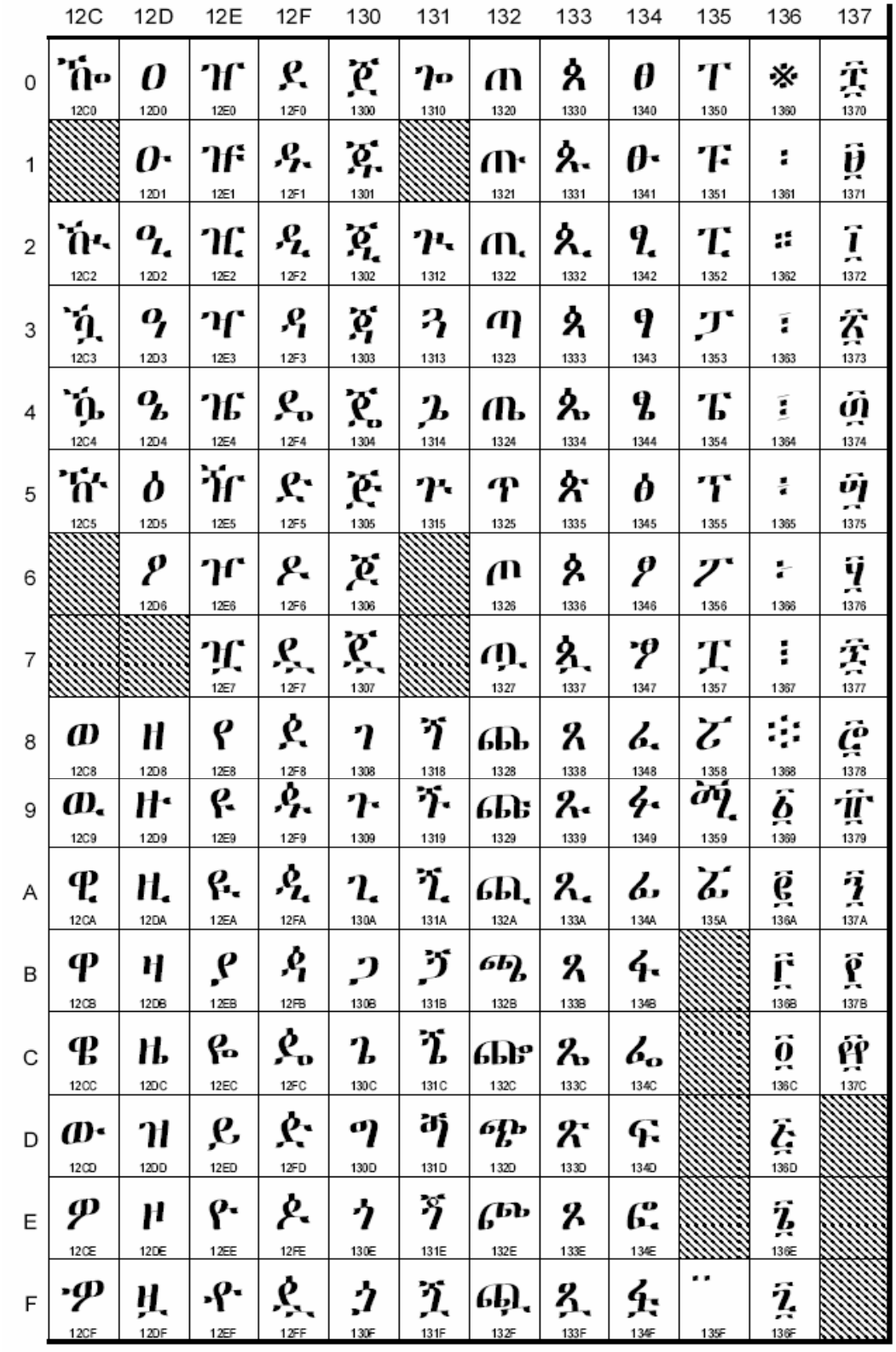
Character code table for Ethiopic.

(Source: http://www.unicode.org/charts/PDF/U1200.pdf)



	120	121	122	123	124	125	126	127	128	129	12A	12B
0	ሀ 1200	ሐ 1210	ሠ 1220	ሰ 1230	ቀ 1240	ቐ 1250	በ 1260	ተ 1270	ኀ 1280	ነ 1290	አ 12A0	ኰ 12B0
1	ሁ 1201	ሑ 1211	ሡ 1221	ሱ 1231	ቁ 1241	ቑ 1251	ቡ 1261	ቱ 1271	ኁ 1281	ኑ 1291	ኡ 12A1	
2	ሂ 1202	ሒ 1212	ሢ 1222	ሲ 1232	ቂ 1242	ቒ 1252	ቢ 1262	ቲ 1272	ኂ 1282	ኒ 1292	ኢ 12A2	ኲ 12B2
3	ሃ 1203	ሓ 1213	ሣ 1223	ሳ 1233	ቃ 1243	ቓ 1253	ባ 1263	ታ 1273	ኃ 1283	ና 1293	ኣ 12A3	ኳ 12B3
4	ሄ 1204	ሔ 1214	ሤ 1224	ሴ 1234	ቄ 1244	ቔ 1254	ቤ 1264	ቴ 1274	ኄ 1284	ኔ 1294	ኤ 12A4	ኴ 12B4
5	ህ 1205	ሕ 1215	ሥ 1225	ስ 1235	ቅ 1245	ቕ 1255	ብ 1265	ት 1275	ኅ 1285	ን 1295	እ 12A5	ኵ 12B5
6	ሆ 1206	ሖ 1216	ሦ 1226	ሶ 1236	ቆ 1246	ቖ 1256	ቦ 1266	ቶ 1276	ኆ 1286	ኖ 1296	ኦ 12A6	
7	ሇ 1207	ሗ 1217	ሧ 1227	ሷ 1237	ቇ 1247		ቧ 1267	ቷ 1277	ኇ 1287	ኗ 1297	ኧ 12A7	
8	ለ 1208	መ 1218	ረ 1228	ሸ 1238	ቈ 1248	ቘ 1258	ቨ 1268	ቸ 1278	ኈ 1288	ኘ 1298	ከ 12A8	ኸ 12B8
9	ሉ 1209	ሙ 1219	ሩ 1229	ሹ 1239			ቩ 1269	ቹ 1279		ኙ 1299	ኩ 12A9	ኹ 12B9
A	ሊ 120A	ሚ 121A	ሪ 122A	ሺ 123A	ቊ 124A	ቚ 125A	ቪ 126A	ቺ 127A	ኊ 128A	ኚ 129A	ኪ 12AA	ኺ 12BA
B	ላ 120B	ማ 121B	ራ 122B	ሻ 123B	ቋ 124B	ቛ 125B	ቫ 126B	ቻ 127B	ኋ 128B	ኛ 129B	ካ 12AB	ኻ 12BB
C	ሌ 120C	ሜ 121C	ሬ 122C	ሼ 123C	ቌ 124C	ቜ 125C	ቬ 126C	ቼ 127C	ኌ 128C	ኜ 129C	ኬ 12AC	ኼ 12BC
D	ል 120D	ም 121D	ር 122D	ሽ 123D	ቍ 124D	ቝ 125D	ቭ 126D	ች 127D	ኍ 128D	ኝ 129D	ክ 12AD	ኽ 12BD
E	ሎ 120E	ሞ 121E	ሮ 122E	ሾ 123E			ቮ 126E	ቾ 127E		ኞ 129E	ኮ 12AE	ኾ 12BE
F	ሏ 120F	ሟ 121F	ሯ 122F	ሿ 123F			ቯ 126F	ቿ 127F		ኟ 129F	ኯ 12AF	

Appendix II (Continued)



Appendix III

List of stopwords

ሀምሳ	ሀያ	ሁለት	ሁሉ	ሆነ	ሆን
ሆይ	ኋላ	ላይ	ሌላ	መቶ	መካከል
ማን	ምን	ምንድር	ምክንያቱም	ምክንያት	ስላሳ
ሰማንያ	ሰሞን	ሰባ	ሰባት	ሰአት	ሲሆን
ሲል	ሳለ	ስለ	ስልሳ	ስምንት	ስነ
ስንት	ስንኳ	ስድሳ	ስድስት	ሶስት	ራሱ
ሺህ	ቀጥሎ	ቁጥር	በላይ	በርስ	በርካታ
በቀር	በኩል	በውኑ	በጣም	በጣም	በፊት
ባለ	ብቻ	ብዙ	ተለያየ	ተጨማሪ	ታች
ታች	ትናንት	ነህ	ነሽ	ነበር	ነው
ነገር	ናት	ናቸው	አሁን	አለ	አምስት
አስራ	አስር	አራት	አርባ	አቤቱ	አንተ
አንቺ	አንድ	አንድ	አይደለም	እልፍ	እሱ
እስከ	እሷ	እርሱ	እርስ	እርስዋ	እርሷ
እባክህ	እነርሱ	እና	እና	እናንተ	እኔ
እንኪያስ	እንኳ	እንደ	እንደገና	እንዲሁም	እንጂ
እንግዲህ	እኛ	እያንዳንዱ	እጅግ	ከዛ	ወዘተ
ወይም	ወይስ	ወደ	ወደፊት	ወዲህ	ወዬት
ዋና	ውስጥ	ውጪ	ዘንድ	ዘጠና	ዘጠኝ
ዚህ	ዚህ	ያህል	ይሁን	ይህ	ይህን
ይሆናሉ	ይሆናል	ደግሞ	ዳሩ	ድረስ	ጊዜ
ጋር	ግን	ጥቂት	ጸረ	እነሆ	እንዲህ

Appendix IV
Source Codes

Source code for developing the inverted file index

```
static final File INDEX_DIR = new File("D:/index");

   // Index all text files under C:/AmharicBible directory.
   public static void main(String[] args) {
      // Check if an index already exists
      if (INDEX_DIR.exists()) {
         System.out.println("Cannot save index to '" + INDEX_DIR + "' directory, please
               delete it first");
         System.exit(1);
      }

      // Check the existance of readable files under D:/AmharicBible directory
      final File docDir = new File("D:/ AmharicBible ");
      if (!docDir.exists() || !docDir.canRead()) {
         System.out.println("Document directory '" + docDir.getAbsolutePath() + "' does
               not exist or is not readable, please check the path");
         System.exit(1);
      }

      // Index files under C:/BibleRefined directory
      Date start = new Date();
      try {
         IndexWriter writer = new IndexWriter(INDEX_DIR, new AmharicAnalyzer(),
               true, IndexWriter.MaxFieldLength.LIMITED);
         System.out.println("Indexing to directory '" + INDEX_DIR + "'...");
         indexDocs(writer, docDir);
         System.out.println("Optimizing...");
         writer.optimize();
         writer.close();

         Date end = new Date();
         System.out.println(end.getTime() - start.getTime() + " total milliseconds");

      } catch (IOException e) {
         System.out.println(" caught a " + e.getClass() +
               "\n with message: " + e.getMessage());
      }
   }

   static void indexDocs(IndexWriter writer, File file)
         throws IOException {
      // do not try to index files that cannot be read
      if (file.canRead()) {
         if (file.isDirectory()) {
```

```
            String[] files = file.list();
            // an IO error could occur
            if (files != null) {
               for (int i = 0; i < files.length; i++) {
                  indexDocs(writer, new File(file, files[i]));
               }
            }
         } else {
            System.out.println("adding " + file);
             writer.addDocument(FileDocument.Document(file));
         }
      }
   }
```

Source code for discarding non-letter characters

```
 /**
    * Discard Amharic and English punctuation marks, leading and trailing
    * spaces, and other non-letter characters.
    * @return true when the character is letter, otherwise return false.
    *
    * Implements: org.apache.lucene.anlaysis.CharTokenizer.isToken(char)
    */
   protected boolean isTokenChar(char c) {
      if (!Character.isLetter(c)) {
         return false;
      }
      return true;
   }
```

Source code for normalizing Amharic characters for spelling inconsistency

```
   /**
    * Normalize Amharic word syllables for spelling inconsistency by replacing
    * characters that are used interchangably with a common one
    *
    * Overrides: org.apache.lucene.analysis.CharTokenizer.normalize(char)
    */
   @Override
   protected char normalize(char c) {
      //Normalize h
      if (c == '\u1203' || c == '\u1210' || c == '\u1213' || c == '\u1280' || c == '\u1283' || c ==
                  '\u12BB') {
         return '\u1200';
      } //Normalize hu
      else if (c == '\u1211' || c == '\u1281') {
         return '\u1201';
      } // Normalize hi
```

```
else if (c == '\u1212' || c == '\u1282') {
   return '\u1202';
} // Normalize hy
else if (c == '\u1214' || c == '\u1284') {
   return '\u1204';
} // Normalize he
else if (c == '\u1215' || c == '\u1285') {
   return '\u1205';
} // Normalize ho
else if (c == '\u1216' || c == '\u1286' || c == '\u12C0') {
   return '\u1206';
} // Normalize s
else if (c == '\u1220') {
   return '\u1230';
} // Normalize su
else if (c == '\u1221') {
   return '\u1231';
} // Normalize si
else if (c == '\u1222') {
   return '\u1232';
} // Normalize sa
else if (c == '\u1223') {
   return '\u1233';
} // Normalize sy
else if (c == '\u1224') {
   return '\u1234';
} // Normalize se
else if (c == '\u1225') {
   return '\u1235';
} // Normalize so
else if (c == '\u1226') {
   return '\u1236';
} // Normalize x
else if (c == '\u12A3' || c == '\u12D0' || c == '\u12D3') {
   return '\u12A0';
} // Normalize xu
else if (c == '\u12D1') {
   return '\u12A1';
} // Normalize xi
else if (c == '\u12D2') {
   return '\u12A2';
} // Normalize xy
else if (c == '\u12D4') {
   return '\u12A4';
} // Normalize xe
else if (c == '\u12D5') {
   return '\u12A5';
} // Normalize xo
else if (c == '\u12D6') {
   return '\u12A6';
```

```
    } // Normalize ts
    else if (c == '\u1340') {
       return '\u1338';
    } // Normalize tsu
    else if (c == '\u1341') {
       return '\u1339';
    } // Normalize tsi
    else if (c == '\u1342') {
       return '\u133A';
    } // Normalize tsa
    else if (c == '\u1343') {
       return '\u133B';
    } // Normalize tsy
    else if (c == '\u1344') {
       return '\u133C';
    } // Normalize tse
    else if (c == '\u1345') {
       return '\u133D';
    } // Normalize tso
    else if (c == '\u1346') {
       return '\u133F';
    } // Normalize qo
    else if (c == '\u1248') {
       return '\u1246';
    } // Normalize qu
    else if (c == '\u124D') {
       return '\u1241';
    } // Normalize ko
    else if (c == '\u12B0') {
       return '\u12AE';
    } // Normalize hwa
    else if (c == '\u12C3') {
       return '\u128B';
    } // Normalize go
    else if (c == '\u1310') {
       return '\u130E';
    } // Normalize gwa
    else if (c == '\u1313') {
       return '\u130F';
    }

    return c;
  }
```

The source code for developing the WORDSPACE model

```
indexReader = IndexReader.open(indexDir);

//Create random document vectors
VectorStoreSparseRAM randomBasicDocVectors = new VectorStoreSparseRAM();
for (int i = 0; i < numVectors; ++i) {
```

```
            short[] sparseVector = VectorUtils.generateRandomVector(seedLength, random);
            this.sparseVectors.put(Integer.toString(i), sparseVector);
    }
    this.basicDocVectors = randomBasicDocVectors;

    // Iterate through an enumeration of terms and create termVector
    TermEnum terms = this.indexReader.terms();

    terms = indexReader.terms();
    while (terms.next()) {
            Term term = terms.term();

            // skip terms that don't pass the filter
            if (!termFilter(terms.term())) {
               continue;
            }

            // initialize new termVector
            float[] termVector = new float[ObjectVector.vecLength];
            for (int i = 0; i < ObjectVector.vecLength; ++i) {
               termVector[i] = 0;
            }

            TermDocs tDocs = indexReader.termDocs(term);
            while (tDocs.next()) {
               String docID = Integer.toString(tDocs.doc());
               float[] docVector = this.basicDocVectors.getVector(docID);
               int freq = tDocs.freq();

               for (int i = 0; i < ObjectVector.vecLength; ++i) {
                  termVector[i] += freq * docVector[i];
               }
            }
            termVector = VectorUtils.getNormalizedVector(termVector);
            termVectors.put(term.text(), new ObjectVector(term.text(), termVector));
        }

    private boolean termFilter(Term term) throws IOException {
        // Field filter.
        if (this.fieldsToIndex != null) {
            boolean desiredField = false;
            for (int i = 0; i < fieldsToIndex.length; ++i) {
               if (term.field().equals(fieldsToIndex[i])) {
                  desiredField = true;
               }
            }
            if (desiredField == false) {
               return false;
            }
        }
```

```
        return true;
    }
```

The source code for generation of thesaurus terms

```
int defaultNumResults = 5;
String queryContent = queryTextField.getText();
String queryContentNorm = normalize(queryContent);
String qTerms[] = queryContentNorm.split(" ");
String qStemmedTerms[] = stem(qTerms);
LinkedList<SearchResult> results = RunSearch(defaultNumResults, qStemmedTerms);

//Nearest Neighbours output results.
String output = "";
if (results.size() > 0) {
    for (SearchResult result : results) {
        output += ((ObjectVector) result.getObject()).getObject().toString() + " ";
    }
} else {
    output = "I'm sorry, there were no semantics matches.";
}
txtAreaOutput.setText(output);


public static LinkedList<SearchResult> RunSearch(int numResults, String[] queryTerms)
                    {
            VectorSearcher vecSearcher;
            LinkedList<SearchResult> results = new LinkedList();
            // Create VectorSearcher and search for nearest neighbors.
                    vecSearcher = new
                                        VectorSearcher.VectorSearcherCosine(queryVecReader,
                                         searchVecReader, lUtils, queryTerms);
                    results = vecSearcher.getNearestNeighbors(numResults);
            return results;
    }
```

Appendix V

Lucene and Semantic Vectors APIs Used in This Research

Source:
Lucene API - http://lucene.apache.org/java/2_4_0/api/index.html
Semantic Vectors API - http://semanticvectors.googlecode.com/svn/trunk/doc/index.html

---

addDocument

```
public void addDocument(Document doc)
                 throws CorruptIndexException,
                        IOException
```

In class:
IndexWriter

Adds a document to this index. If the document contains more than `setMaxFieldLength(int)` terms for a given field, the remainder are discarded.
Note that if an Exception is hit (for example disk full) then the index will be consistent, but this document may not have been added. Furthermore, it's possible the index will have one segment in non-compound format even when using compound files (when a merge has partially succeeded).
This method periodically flushes pending documents to the Directory (see above), and also periodically triggers segment merges in the index according to the `MergePolicy` in use.
Merges temporarily consume space in the directory. The amount of space required is up to 1X the size of all segments being merged, when no readers/searchers are open against the index, and up to 2X the size of all segments being merged when readers/searchers are open against the index (see `optimize()` for details). The sequence of primitive merge operations performed is governed by the merge policy.
Note that each term in the document can be no longer than 16383 characters, otherwise an IllegalArgumentException will be thrown.
Note that it's possible to create an invalid Unicode string in java if a UTF16 surrogate pair is malformed. In this case, the invalid characters are silently replaced with the Unicode replacement character U+FFFD.
Throws:
`CorruptIndexException` - if the index is corrupt
`IOException` - if there is a low-level IO error

---

add

```
public final void add(Fieldable field)
```

In Class:
Document

Adds a field to a document. Several fields may be added with the same name. In this case, if the fields are indexed, their text is treated as though appended for the purposes of search.
Note that add like the removeField(s) methods only makes sense prior to adding a document to an index. These methods cannot be used to change the content of an

existing index! In order to achieve this, a document has to be deleted from an index and a new changed version of that document has to be added.

---

open

```
public static IndexReader open(File path)
                        throws CorruptIndexException,
                               IOException
```

In class:
IndexReader

Returns a read/write IndexReader reading the index in an FSDirectory in the named path. **NOTE**: starting in 3.0 this will return a readOnly IndexReader.
Parameters:
`path` - the path to the index directory
Throws:
`CorruptIndexException` - if the index is corrupt
`IOException` - if there is a low-level IO error

---

generateRandomVector

```
public static short[] generateRandomVector(int seedLength,
                                           java.util.Random random)
```

In Class:
VectorUtils

Generates a basic sparse vector (dimension = Flags.dimension) with mainly zeros and some 1 and -1 entries (seedLength/2 of each) each vector is an array of length seedLength containing 1+ the index of a non-zero value, signed according to whether this is a + or -1.
e.g. +20 would indicate a +1 in position 19, +1 would indicate a +1 in position 0. -20 would indicate a -1 in position 19, -1 would indicate a -1 in position 0.
The extra offset of +1 is because position 0 would be unsigned, and would therefore be wasted. Consequently we've chosen to make the code slightly more complicated to make the implementation slightly more space efficient.
Returns:
Sparse representation of basic ternary vector. Array of short signed integers, indices to the array locations where a +/-1 entry is located.

---

freq

```
int freq()
```

In class:
TermDocs

Returns the frequency of the term within the current document. This is invalid until `next()` is called for the first time.

---

field
`public final String field()`

In Class:
Term

Returns the field of this term, an interned string. The field indicates the part of a document which this term came from.

---

getNormalizedVector
`public static float[] getNormalizedVector(float[] vec)`

In class:
VectorUtils

Returns the normalized version of a vector, i.e. same direction, unit length.
Parameters:
`vec` - Vector whose normalized version is requested.

---

openInput
`public IndexInput openInput(String name)`
`throws IOException`

In class:
MMapDirectory

Returns a stream reading an existing file.
Overrides:
`openInput` in class `FSDirectory`
Throws:
`IOException`

---

getAdditiveQueryVector
`protected float[] getAdditiveQueryVector(java.lang.String[] queryTerms)`

In class:
CompoundVectorBuilder

Returns a (possibly weighted) normalized query vector created by adding together vectors retrieved from vector store.
Parameters:
`queryTerms` - String array of query terms to look up

---

getNearestNeighbors

```
public java.util.LinkedList getNearestNeighbors(int numResults)
```

In class:
VectorSearcher

This nearest neighbor search is implemented in the abstract VectorSearcher class itself: this enables all subclasses to reuse the search whatever scoring method they implement. Since query expressions are built into the VectorSearcher, getNearestNeighbors no longer takes a query vector as an argument.

Parameters:

`numResults` - the number of results / length of the result list.

---

parse

```
public Query parse(String query)
            throws ParseException
```

In class:
QueryParser

Parses a query string, returning a `Query`.

Parameters:

`query` - the query string to be parsed.

Throws:

`ParseException` - if the parsing fails

---

search

```
public TopDocs search(Weight weight,
                      Filter filter,
                      int nDocs)
               throws IOException
```

In class:
IndexSearcher

Description copied from interface: `Searchable`

Expert: Low-level search implementation. Finds the top `n` hits for `query`, applying `filter` if non-null.

Called by `Hits`.

Applications should usually call `Searcher.search(Query)` or `Searcher.search(Query,Filter)` instead.

Specified by:

`search` in interface `Searchable`

Specified by:

`search` in class `Searcher`

Throws:

`IOException`

DECLARATIONS

This thesis is my original work, has not been presented for a degree in any other university and all sources of material used for the thesis have been duly acknowledged.

_____

Andargachew Mekonnen Gezmu

July, 2009

The thesis has been submitted for examination with my approval as university advisor

_____

Million Meshesha (PhD)

July, 2009