ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
SCHOOL OF INFORMATION SCIENCE

Amharic Question Answering for list questions: A case of Ethiopian tourism

Brook Eshetu Bete

JUNE, 2013
Addis Abeba, Ethiopia

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
SCHOOL OF INFORMATION SCIENCE

Amharic Question Answering for list questions: A case of Ethiopian tourism

Brook Eshetu Bete

A thesis submitted to the School of Information Science of Addis Ababa University in partial fulfillment of the requirement for the Degree of Master of Science in Information Science

JUNE, 2013
Addis Abeba, Ethiopia

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

SCHOOL OF INFORMATION SCIENCE

Amharic Question Answering for list questions: A case of Ethiopian tourism

Brook Eshetu Bete

Advisor: Solomon Teferra (Phd)

Name and signature of members of the examining board

	Name	Title	Signature	Date
1.	_____	Chairperson	_____	_____
2.	_____	Advisor	_____	_____
3.	_____	Examiner	_____	_____

DEDICATED TO:

MY FATHER; ESHETU BETE

ACKNOWLEDGMENT

Primarily, my eternal gratitude goes to God, who makes everything possible with his love and care. My gratitude is then to my advisor Dr. Solomon Teferra for his beneficial, positive and practical advice and encouragement. He also helped to overcome challenges and to realize this research at the right time.

I am also very grateful to thank iman imran, a tour guide in national tour operators (NTO), who helped in collecting documents for the courpus and answers for questions used for evaluation of the system by teaming up with her co-workers.

Special thanks also go to my families mainly for my mom Meseret Bekele, my brother Ermiyas Eshetu and my aunt Abeba Bekele, who have been behind me in supporting and encouraging me through difficult times.

Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

ACE	Automated Content Extraction
AQA	Amharic Question Answering
IE	Information Extraction
IR	Information Retrieval
LSI	Latent Semantic Indexing
MUC	Message Understanding Conference
NE	Named Entity
NIST	National Institute of Standards and Technologies
NLP	Natural Language Processing
PRP	Probabilistic Ranking Principle
QA	Question Answering
SUD	Singular Value Decomposition
TREC	Text Retrieval conference
VSM	Vector space model
WMI	Weighted Mutual Information
MRR	Mean Reciprocal Rank
HMM	Hidden Markov Model

ABSTRACT

A Question answering (QA) system searches a large text collection and finds a short phrase or sentence that precisely answers a user's question. To solve a QA problem, we might first turn to traditional IR techniques, which have been applied successfully to large scale text search problems. Alternatively, the Natural Language Processing (NLP) and Information Extraction (IE) communities have developed techniques for extracting very precise answers from text.

QA research attempts to deal with a wide range of question types including: fact, list, definition, How, Why, hypothetical, semantically constrained, and cross-lingual questions.

This research work focuses on list questions in closed domain Amharic questions answering (AQA). It applies the hypothesis, which states that answers to a list questions have same semantic entity class, answers that co-occur within the sentences of the documents are related to the target and the question and sentences containing the answers share similar context. In this research work, list questions are answered using five major modules. The function of these modules are (1) determining the answer type, (2) document retrieval, (3) Extract answer candidates from the document, (4) computing similarity value for each pair of candidate answers based on their co-occurrence within the sentence, (5) selecting final answers.

The QA system is evaluated and the experimental results show that the system registered an average of 57.5% F-score. Nevertheless, the performance of the system is greatly affected by the number for documents in the document database and techniques applied.

As a result, we managed to develop a prototype for Amharic listing QA system in the area of Ethiopian tourism.

KEYWORDS: Amharic Question Answering, List Questions, Answer Type, Candidate Answers, Co-occurance, Question Answering Evaluation.

CHAPTER ONE

1. INTRODUCTION

1.1. BACKGROUND

Today when users have questions, one of their likely tactics for finding answers is to use an online information retrieval (IR) system. In 2005, an estimated 60 million U.S. adults used a web search engine on a typical day [2].

Existing search engines are not perfect. Nevertheless, they are extremely useful. Without access to a search engine, many of us would be paralyzed. It is evident that existing search engines, with Google at the top but faced with growing competition, have many truly remarkable capabilities. But there is a very important capability which they do not have, deduction capability, the capability to synthesize an answer to a question by drawing on bodies of information which reside in various parts of the knowledge base [3]. What should be noted, however, is that there are many widely used special purpose Question/Answer (Q/A) systems which have limited deduction capability [3].

In current information retrieval systems, the general method is to specify some keywords to obtain necessary information on the Internet. Given a query, an IR system returns a list of potentially relevant documents, which the user must then scan to search for pertinent information. This method could not satisfy the user's needs to extract the adequate information efficiently from a huge set of electronic documents, even though the construction of the retrieval service is easy. QA is a technology that aims at retrieving the answer to a question written in natural language in large collections of documents. QA systems are presented with natural language questions and the expected output is either the exact answer identified in a text or small text fragments containing the answer [4].

In QA systems, users are encouraged to enter their questions as questions. The QA system handles all searching.

QA systems can be designed for factoid and/or non-factoid questions. The obvious limitation of developing any factoid QA system is that people want answers for many questions that are not factoid. It is also the case that answers cannot be found readily for non-factoid questions by simply using a good search engine [5]. The research also includes one of the famous non-factoid questions called listing questions.

1.2. STATEMENT OF THE PROBLEM

In our country, Ethiopia, the numbers of electronically generated Amharic documents is increasing on an increasing rate as researches, historical documents, fictions, magazines and many newspaper publishers started providing their works electronically.

On the other hand, asking questions is the very nature of human beings, so peoples need answer for their question that is found in the documents. This gap will raise need to have some system which can understand questions and then look for an answer in the knowledge base and give the direct answer for the question.

Having only an IR system for Amharic documents will not satisfy user needs because as users struggle to navigate the information available, they need systems that allow them to ask a question in everyday language and receive an answer quickly and briefly, with concise context to validate the answer [6]. This will lead to the need for automated QA systems more urgently.

QA research emerged as an attempt to tackle this information-overload problem [7]. Question which deals about nearly everything or specific domain (music, weather forecasting etc.) must be processed and answered accordingly.

Current IR search engines can return ranked lists of documents, but they do not deliver answers to the user. The user must look in to the documents to find the content they need. To address this problem we need systems that allow a user to ask a question in everyday language and receive an answer quickly and succinctly, with a context to validate the answer [8].

As users aspire to use their time efficiently and effectively, they will not prefer to read a whole document while they need specific or a couple of information in that document. Such problems need to be solved by designing question-answering systems for Amharic language.

As a local work, there is only one work done by Seid Muhie Yimam, 2009, Amharic question answering (AQA) for factoid questions. This work is done using Java programming language and tries to cover some factoid questions. Moreover, there is an ongoing work on definition questions for Amharic questions, which is a non-factoid question type. The combination of these three researches will come up with a complete AQA system that can answer any kind of questions.

Since there are questions that need answers in a listing format are many in real life, this research will develop a prototype of Amharic question answering system for list questions. This work shows one of the non-factoid question type, which is an approach that has never been done as a research for any local languages. The prototype used an Ethiopian tourism dataset as a corpus.

To this end, the following research questions are addressed.

- ✓ What are the characteristic features of listing questions?
- ✓ How can one detect relations between a question and answer type?
- ✓ How to retrieve candidates, extract lists and answer questions from different documents?
- ✓ What is the general architecture of AQA system for list questions?

1.3. OBJECTIVE OF THE STUDY

- General objective

The general objective of this study is to design AQA system, which can accept list questions and answer users question as effective as possible.

- Specific objectives

To achieve the main goal, the study has the following specific objectives:

- ✓ To develop Amharic answers for Amharic listing questions.
- ✓ To identify features of listing questions that are needed for AQA development.
- ✓ To detect question type and answer type relations.
- ✓ To retrieve candidates, extract list questions from different documents.
- ✓ To develop a general architecture of AQA system for list questions.
- ✓ To develop the prototype and evaluate its performance.

1.4. SCOPE AND LIMITATION OF THE STUDY

As QA systems can be designed for different languages, this study is delimited to Amharic language since it is familiar for the researcher and it is the national language of the country.

The types of questions that are addressed in this work are listing questions, from non-factoid questions in tourism sector.

This QA system will be implemented on Ethiopian tourism documents that are electronically available.

Lack of large size document corpus that are prepared using Amharic language is the major limitation of this research. The tourism offices in the country have almost no Amharic document that contain tourism related information and most tour and travel agencies use english documents from the internet which is prepared by foreign professionals as a main source. Even if the researcher face this problem, changing paper documents in to electronic documents was done to minimize the challenge and adding more documents increased the performance of the system. The other limitation is that there is no perfect or even high performance document retrival system, which can retrieve documents from a small corpus. In this case, the best document retrieval system available is used.

Collecting answers for list questions was also another problem because it is not easy for the experts to come up with a conclusive list of answers for all questions that was delivered to them, to alleviate this the researcher was forced to select questions that are answered fully.

1.5. METHODOLOGY

The design of the research, data set, literature review, system development, and testing and measurement strategy used for this thesis work are discussed in the following subsections.

1.5.1. STUDY DESIGN

The research design of our research is an experimental research. It is necessary to get first hand facts and their source, and to actively do certain things to stimulate the production of desired information. This is because of the fact that this study comes up with a prototype AQA system.

1.5.2. LITERATURE REVIEW

To understand problem, gap and state-of-the-art techniques for developing QA systems, the researcher reviewed books, journal article and Internet publications. The review helped to understand QA approaches and techniques.

As the research focuses on Amharic language, different language specific features and properties have been studied in light with QA systems for listing questions.

1.5.3. DATA SET

In this study, 29 documents that consisted of Ethiopian tourism information are collected. The documents are collected from different sources and are used to build a system that is capable of answering list questions from documents that are not purposely selected or pre-processed.

On the other hand, most frequently used tourism questions in the case of Ethiopia are collected. The questions are collected from experts and users.

1.5.4. SYSTEM DEVELOPMENT

For the development of this AQA system, python programming language is used. This programming tool is used because it is an interoperable and easily available tool. The researcher is also familiar with it.

1.5.5. TESTING

For testing purpose, domain experts were provided with different questions that are collected from different web sites. After collecting experts answer, it was compared with that of the systems answer. Having expected answers, system answers and total number of answers, the performance of the system is measured using IR effectiveness measures mainly recall, precision and F-score.

1.6. SIGNIFICANCE OF THE STUDY

This system has a great contribution for the development of a full-fledged AQA system. The system also shows direction for researchers to develop a closed or open domain system that can be used in a specific or varieties of sectors. This approach can also be applied to any sector where there is a need to retrieve any documents written in local Semitic languages (such as, Guragegna, Tigrigna, Siltegna etc) from large collection. Moreover, it is an academic exercise to fulfill the requirement of masters program that the researcher is enrolled in.

1.7. ORGANIZATION OF THE RESEARCH

This thesis consists of four chapters. The first chapter deals with the general overview of the study including background, statement of the problem, objectives, scope and limitation, methodology and significance of the research. The second chapter is devoted to literature review of question answering, and deals with theoretical aspects of QA systems.

Chapter 3 reports the systems architecture and design of the research. It also comprises the evaluation of the system with every related detail. The results of the experiment are also analyzed and interpreted in this section. The last chapter of the thesis, chapter 4, presents conclusions and recommendations.

CHAPTER TWO

2. LITERATURE REVIEW

This chapter concentrates on addressing closed domain QA system development strategies and issues.

It also presents details on QA components, particularly on approaches in Question Analysis, general QA architectures, Document Retrieval, and Answer Extraction components of a QA system.

This chapter reviews the previous works done on answering list and factoid questions and a work on AQA.

2.1. THE QUESTION ANSWERING THEORY

The development of systems that interact with human users in natural language has long been a goal of the artificial intelligence research community. Since the 1960s, when the field was in its infancy, a variety of natural language database front-ends, dialog systems, and language understanding systems have been created. Simmons at the earliest 1965 reviewed no fewer than fifteen implemented English language QA systems built over the preceding five years in his paper.

But the notion of a question answering system was born in 1950. Turing proposed a task he called an “Imitation Game” which has eventually become known as the famous “Turing Test” in which a human communicates with a machine via a teletype interface and asks questions of it. Turing would have deemed the machine “intelligent” if the human interrogator could not tell the difference between the responses of the machine and the responses of another human, also communicating via teletype.

In the early 1960s, there was interest in developing natural language front-ends for database query systems, such as BASEBALL [9] and ELIZA [10] , which relied on identifying word patterns in the user input, drastically restricting the domain of discourse, or matching simple syntactic structural templates. However, BASEBALL system as the best-known early Question Answering system, is one of the most successful systems of its kind. BASEBALL system is a program for answering questions about baseball games played in the American league over one season. The system was able to answer narrow-domain questions about statistics compiled over a season of American League play by using shallow parsing techniques on the natural language query to identify the teams and statistics in question. It was also able to handle some more comprehensive queries that involved collating data found in different records of the baseball database, and return the appropriate answer.

The well remembered early work in this tradition is the LUNAR system, which provided access to two databases containing information about moon rock samples. LUNAR was designed "to enable a lunar geologist to conveniently access, compare and evaluate the data of chemical analysis on lunar rock and soil composition that was accumulating as a result of the Apollo moon mission" [11]. The system worked by translating natural language questions into one or more queries in the database engine's query language. While these systems were excellent at responding to specific classes of questions within their domain of expertise, the systems are incapable of responding to any natural language questions that might suggest themselves during the dialog with the user, but which happen to be outside of the set of questions that the system was specifically engineered to be able to process.

As mentioned above, natural language front-ends to databases is an application area for question answering that attracted researchers early on, in the 1970s, was question answering in human-machine dialogue. Early dialogue systems such as SHRDLU and GUS were built as research systems to help researchers understand the issues involved in modelling human dialogue. SHRDLU was built for a toy domain of a simulated robot moving objects in a blocks world. The later system,

GUS, simulated a travel advisor and had access to a restricted database of information about airline flights. Both these systems, demonstrated remarkable capacity to understand natural language, especially where anaphora resolution or inference was required to carry out the user's instructions.

Attempts to build systems that were capable of basic reading comprehension arose since the middle of 1970s. The aim was to be able to evaluate a machine's ability to understand language in the same way as is done for that of a human. The most notable early work is a system called QUALM, which has been developed by [12]. The system viewed the process of question answering as a function in which both understanding and answering a question relies on the context of the story and pragmatic notions of the appropriateness of the answer. Further work has gone on in story comprehension, but much of it have been within the psychology community [13] and work on developing computational models of story understanding dwindled through the 1980s and 1990s. However, there has been a recent revival of interest in the area, following the creation of a reading comprehension evaluation task [14].

However, early explorations into question answering systems were concerned with producing natural language query front-ends for databases, dialog systems, reading comprehension programs and the like. In fact, the interest in the QA task in its current form really takes root among the research community since the end of 1990s. Especially, there has been a remarkable increase in interest in natural language question answering since the introduction of the Question Answering track in the Text Retrieval Conferences, beginning with TREC-8 in 1999. [4]

2.2. QUESTION ANSWERING SYSTEMS

As users struggle to navigate the wealth of on-line information now available, the need for automated question answering systems becomes more urgent. We need systems that allow a user to ask a question in everyday language and receive an answer quickly and succinctly, with scent context to validate the answer. Current

search engines can return ranked lists of documents, but they do not deliver answers to the user.

QA systems address this problem. Recent successes have been reported in a series of QA evaluations that started in 1999 as part of the Text Retrieval Conference (TREC). The best systems are now able to answer more than two third of factual questions in this evaluation. [8]

The combination of user demand and promising results has stimulated international interest and activity in QA. This special issue arises from an invitation to the research community to discuss the performance, requirements, uses, and challenges of QA systems. The papers in this special issue cover a small part of the emerging research in QA.

Question answering is in a phase of active system building and creative experimentation, and not so much of reflective, comparative or theoretical analysis. Thus, while it might be desirable for an issue such as this to offer a consolidating, synthetic overview of progress to date and issues for the future, in reality all it can offer is a limited view from the ground level of an exciting, dynamic research area. [8]

As an information retrieval task, QA is automatically answering a question posed in natural language. A QA system may use either a pre-structured database or a collection of natural language documents (such as the World Wide Web documents or some local collection of text) and use a pre-structured one: an ontology based knowledge base. [8]

QA often requires more complex natural language processing techniques than other types of IR such as a search engine or document retrieval, thus natural language search engines are sometimes regarded as the next step beyond current search engines.

A QA system may be closed-domain or open-domain. Closed-domain (also called restricted domain) ones deal with questions under a specific domain (for example,

tourism or hardware gadgets). Open-domain question answering deals with questions about nearly everything, and can only rely on general ontologies and world knowledge. On the other hand, these systems usually have much more data available from which to extract the answer. [15] The system to be designed in this thesis is a closed-domain one (Ethiopian tourism).

QA is the task of finding answers to users' questions in (large) text collections or/and pre-structured database. [16] Most QA systems use a technique where questions are analyzed to determine the expected answer type; passage retrieval is used to retrieve the most relevant text fragments from the text collection, and various NLP techniques are used to identify and rank phrases within the retrieved text that potentially answer the question and that match the expected answer type. An alternative method searches the corpus beforehand for answers to frequently asked question types. It requires that relation extraction patterns are constructed (manually or automatically) to extract tuples of pairs instantiating a particular relation from the corpus. For instance, if questions about locations of museums are frequent, one can design patterns that would extract the tuple hUffizi, Florence from the sentence Today the hUffizi is one of the most popular tourist attractions of Florence. For those questions for which relation extraction patterns are developed, the open-domain Dutch question answering system Joost tends to give better results than using passage retrieval. This suggests that in general it is worthwhile to invest in relation extraction for question answering. [16]

Much research on automatic learning of relation extraction patterns has concentrated on learning surface strings (i.e. sequences of words in the immediate context of the arguments of the relation) as extraction patterns. Such patterns can be found in unparsed corpora, and can be used to extract instance pairs from the web. Alternatively, one may learn dependency patterns. Dependency patterns abstract from many aspects of surface word order, and focus on those aspects of grammatical structure that seem most relevant for relation extraction. For a language such as

Dutch, which exhibits more word order variation than English, the fact that dependency patterns abstract over surface word order is important. [16]

According to [16], there are two important issues relevant to relation extraction for QA. The first, is manual construction of extraction patterns. It is, however, labour intensive, especially if many question types need to be dealt with. The second is supervised relation extraction method that tries to learn extraction patterns by bootstrapping from a small set of seed pairs, representative for the relation that needs to be learned. Such methods work well given a large corpus, in which relation instance pairs (such as hUffizi, Florencei for the museum-location relation) can be found frequently and in many different contexts.

2.3. CLOSED DOMAIN QA SYSTEMS

This kind of QA has some characteristics making it different from open-domain QA, which works over a large document collection, including the WWW. In closed-domain QA, correct answers to a question may often be found in only very few documents; the system does not have a large retrieval set abundant of good candidates for selection. Moreover, if the QA system is to be used for answering questions from a company's clients, it should accept complex questions, of various forms and styles. The system should then return a complete answer, which can be long and complex, because it has to, e.g., clarify the context of the problem posed in the question, explain the options of a service, give instructions or procedures, etc. This makes techniques developed recently for open-domain QA, particularly those within TREC (Text REtrieval Conference) competitions (e.g. TREC, 2002) less helpful. These techniques aiming at finding short and precise answers are often based on the hypothesis that the questions are constituted by a single constituent, and can be categorized into a well-defined and simple semantic classification (e.g. PERSON, TIME, LOCATION, QUANTITY, etc.). [17]

Closed-domain QA has a long history, beginning with systems working over databases (e.g., BASEBALL and LUNAR). Recently, research in QA has concentrated mostly on problems of open-domain QA, in particular on how to find a very precise and short answer. Nonetheless, researchers begin to recognize the importance of long and complete answers. [17]

As an example for closed-domain QA system, [16] developed a medical closed-domain system which show that the number of questions types is limited on the domain. Most questions will be about definitions, causes, symptoms and treatments. This suggests relation extraction could be very effective for a medical QA system. A problem for domain specific relation extraction, however, is the fact that corpora tend to be smaller than those used for open-domain QA, and thus there are fewer highly frequent instance pairs. Second, whereas relation extraction for open domain QA has concentrated on learning relations between named entities, the arguments of medical relations are often complex noun phrases that are subject to more grammatical variation than named entities. This is an additional factor that reduces the frequency of easily identifiable instance pairs. Therefore, most systems for relation extraction in the medical domain have made use of two additional resources to make the task feasible. First, a thesaurus is used to identify relevant concepts in text. Second, instead of learning from seeds, extraction patterns are learned on the basis of an annotated text corpus. In an annotated corpus, examples of the relation to be learned are marked, and the relation extraction system uses these positive examples to learn which grammatical patterns are typical for the relation.

The nature of a particular restricted domain of a QA system affects the kinds of questions asked and answers that can be expected. Consequently, different restricted domains benefit from different QA techniques. Some domains are particularly appropriate for the development of QA systems.

A restricted domain must meet the following desiderata [18]:

i. It should be circumscribed.
ii. It should be complex.
iii. It should be practical.

1. Circumscription

If the QA system is not World Wide Web–based and intended for use within a corporation, users do not face the problem of wondering whether questions are appropriate or inappropriate. Rather, a more important motivation for a circumscribed domain is the need for clearly defined knowledge sources. The range of techniques used in a restricted domain should not need to use extensive knowledge from outside the chosen domain. Rather, a domain that has authoritative and comprehensive resources is to be preferred. Examples of resources include actual databases containing the required information.

It is natural to assume that the more restricted the domain is and the more circumscribed it is, the more possible it is to obtain such comprehensive databases. For more complex domains, useful resources are terminology databases and domain ontologies. An added value is the existence of well-accepted terminology and ontology standards.

2. Complexity

A domain should be complex enough to warrant the use of a QA system. This may seem an obvious statement, but it is important to bear in mind that, in a desire to find a domain that is fully circumscribed, one might attempt to develop a QA system in a domain where a simple list of facts would be sufficient to satisfy the user's need for information.

In general, the more complex a domain is, the more interesting it becomes for the researcher and the more useful it presumably is to the user.

There is a balance to be achieved between the need for a complex domain and that of a circumscribed domain, because these two desiderata are in conflict. At some point, if a domain is complex enough, it becomes difficult to manage and there is a higher probability of requiring resources belonging to other domains; in other words, the domain becomes less circumscribed.

3. Practicality

Practicality is an important desideratum to consider when developing a QA system. The domain should be of use to a relatively large group of people. Otherwise one risks wasting effort on a system that nobody would use, such as for an artificially constructed toy domain. The choice of domain affects the kind of users to target. Therefore, for each domain it is important to determine the kinds of questions asked in the specific domain (question style and terminology used are two important factors to consider), the sort of information that is most commonly requested, and the level of detail expected in the answers.

2.4. OPEN-DOMAIN VERSUS RESTRICTED-DOMAIN QUESTION ANSWERING

Open-domain QA systems capable of extracting the answer to user queries directly from unrestricted-domain documents [18]. Restricted-domain QA systems are geared more towards providing answers from knowledge bases that cover a specific domain such as student advising or computer repairs.

In over a decade, the open-domain QA research has dwarfed the workload that has been devoted to developing restricted-domain QA systems. There are several reasons for this significant difference in focus. One of the main reasons for this difference in focus is the introduction of the internet. By using the internet, billions of indexed documents have become easily accessible through search engines. These documents serve as the main knowledge base in an open domain QA system. Another strong

reason is the addressable market that becomes available with an open domain system verses a restricted domain. An open domain system can facilitate almost anyone searching for simple factoid type questions.

A large portion of the revenue sources online originates from advertisement. Enterprise companies such as Google, Yahoo!, and Microsoft want to create products that address a wide user segment. None of these companies currently have a strong open domain QA solution, but they are all working hard on different QA solutions. Yet another strong reason that open domain QA systems have received more focus is the U.S. government's QA track competition. This competition has introduced several organizations and universities to open domain QA research.

Open domain QA systems perform well in facilitating the basic need of the casual internet user. However, these systems are not sufficient to provide answers to more complex questions that would make QA become really useful.

2.5. ANSWERING LIST QUESTIONS

There are some approaches to answering List questions. The first approach applied in [1] is based on the Distributional Hypothesis, which states that words occurring in the same contexts tend to have similar meanings. Distributional Hypothesis is the basis of Statistical Semantics defined as the study of "how the statistical patterns of human word usage can be used to figure out what people mean, at least to a level sufficient for information access". Following this view, the defined hypothesis is that:

1. The instances of the answer to a List question have the same semantic entity class;

2. The instances of the answer tend to co-occur within the sentences of the documents related to the target and the question;

3. The sentences containing the answers share similar context.

In other words, the instances of an answer to a List question tend to co-occur together and also tend to co-occur with the target and the question keywords within the sentences of the relevant documents. Co-occurrence can be an indicator of semantic similarities; generally, terms that co-occur more frequently tend to be related.

A List question is answered in the following steps: First, the answer type of the question is determined. Then, two different queries for searching the Web and the corpora are generated using the target and the question text. Extract from the collection all terms that comply with the answer type; this constitutes the initial candidate list. A similarity value is then computed for each pair of candidate answers based on their co-occurrence within sentences. Having clustered the candidates and determined the most likely cluster, the final candidate answers are selected.

The first step to answering a question is question analysis and answer type recognition. This information is crucial for the next module, Candidate Answers Extraction, which uses the answer type to extract all possible terms with that answer type. Each question is associated to one of the nine semantic entity classes: Person, Country, Organization, Job, Movie, Nationality, City, State, and Other. The class comprises all answer types which do not fall into any of the other categories and hence a subtype is defined for it. The answer type recognition module consists of three tiers: Lexical Patterns, Syntactic Patterns, Type Resolving.

Given a question, each tier attempts to predict the answer type using a different method. If it fails, the question is passed to the next tier.

Document Retrieval, documents that are relevant to the target and the question are retrieved from the corpora and the Web. For this purpose, it is needed to generate appropriate queries and use a search engine tool. Document retrieval is very important for QA systems because it, along with candidate answer extraction, defines an upper bound on the recall to be achieved.

Query generation, constitutes an extremely important phase of Question Answering because only an appropriate query can get appropriate answers using both the corpora given and the Web as document collection sources. Two different search engines are exploited: Lucene to search on the corpora and Google to search on the Web. Both IR systems use different syntax for forming queries and they provide different capabilities and features, therefore two different query generation techniques are used to maximize the quality of their output.

Although the query generation techniques are different, they share the same basic methodology. The question and the target of the series are used to which the List question belongs for creating the queries. The question and the target are first processed separately. The process is the same for both the question and the target:

1. Stop words are removed from the text using a stop-words list;

2. Terms surrounded by double quotations are left unchanged;

3. Consecutive capitalized words are quoted together as a single phrase. Quote consecutive capitalized words to form a phrase because they are very likely to refer to a proper noun. Proper nouns are perhaps the most important content words in a query;

4. The text within a pair of parentheses are processed recursively and treated as Query Refinement; The queries generated using the question and the target might be too strict and result in only a few documents and thereby a low recall. At this stage, if the number of hits is below a certain threshold, the query is loosened. Of course, different thresholds and different techniques are used for the Google and Lucene query refinements.

Document Retrieval; once the queries have been created and refined, relevant documents are retrieved using the Lucene and Google search engines. Two document collection are created; source document collection and domain document collection. The source document collection is used to extract candidate answers from. It

consists of a small number of corpora documents and Web documents. On the other hand, the domain document collection is used to extract relation between the candidate answers.

Candidate Answer Extraction; in this section, extract from the source documents an initial list of candidate answers. The system will stick to this list from this step on. Therefore, the upper bound of recall is set at this phase.

Term Extraction; For term extraction, there is a simple method: All terms that conform to the answer type are extracted from the source document collection. Depending on the answer type, the candidate terms are extracted using different approaches.

2.6. APPROACHES TO QA SYSTEMS

Let us now look at the sorts of approaches which are currently being employed to address QA task according to [8].

As a framework for discussing actual systems, it is useful to have in mind a generic architecture for the QA task. Specific systems can then be seen as instantiations of the general architecture, with particular choices being made concerning representation and processing for each component of the overall model.

Figure 1 proposes such a general architecture for the QA task, conceived as that of asking natural language questions to a system that has as its knowledge source a large collection of natural language texts. Not all QA systems will implement all components in the model (in particular most current TREC QA systems do not utilise dialogue or user models); and, there may well be systems that implement functionality not in the model, or which cannot be easily mapped into it. Still, having such a general model in mind is useful, and helps to guide and structure discussion.

The work [8] briefly describe each of the processing stages in the model, then return to each stage in somewhat more detail, discussing issues to be faced when implementing that stage and exemplifying choices by reference to actual systems for the most part systems developed to participate in the TREC QA Track. However, systems developed for, for example, the reading comprehension task can also be described in these terms.

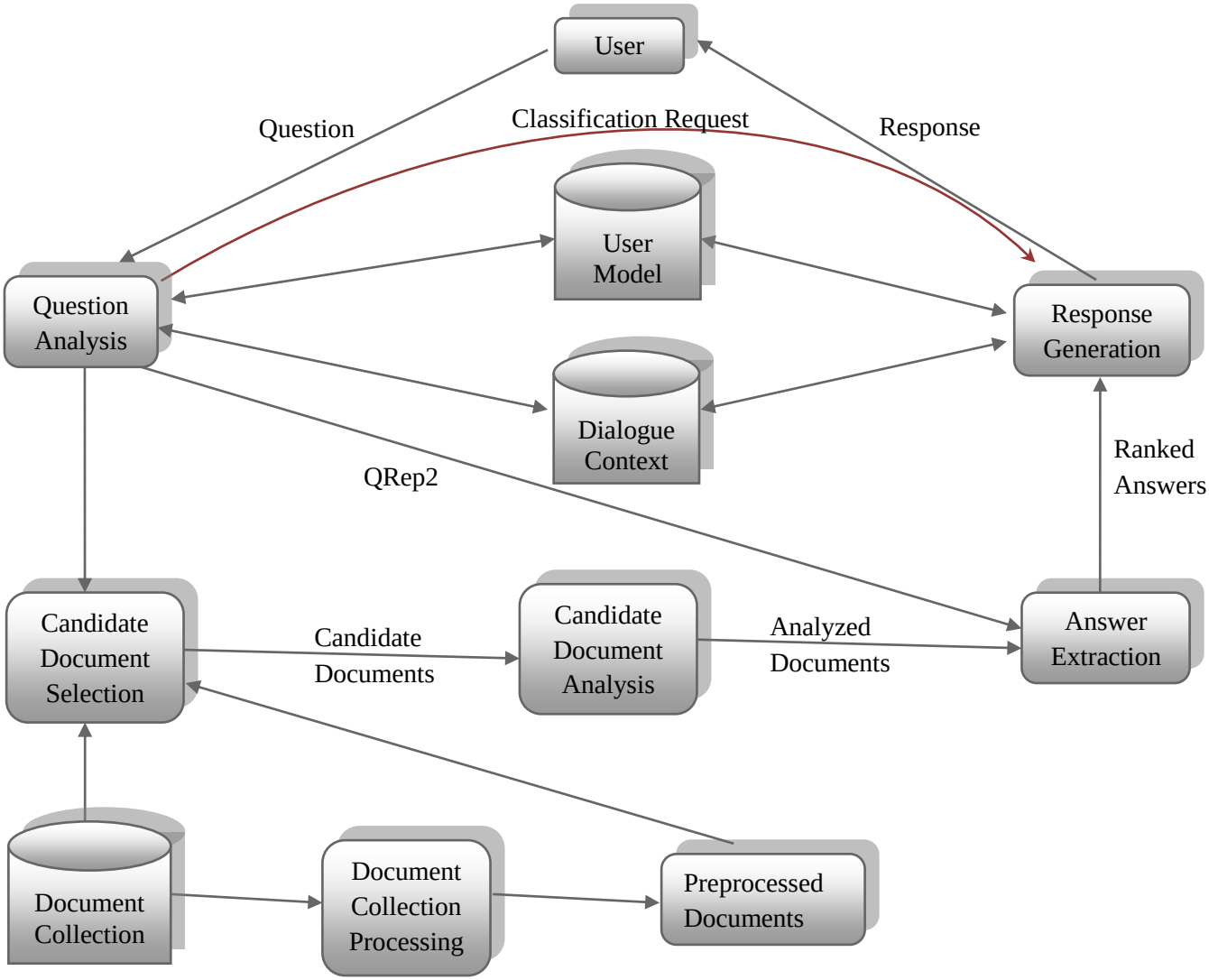


Figure 1 Generic Architecture for open and close QA system

1. Question Analysis: The natural language question input by the user needs to be analyzed into whatever form or forms are needed by subsequent parts of the system. The question may be interpreted in the context of an on-going dialogue and in the

light of a model which the system has of the user. The user could be asked to clarify his or her question before proceeding.

2. Document Collection Preprocessing: Assuming the system has access to a large document collection as a knowledge resource for answering questions, this collection may need to be processed before querying, in order to transform it into a form which is appropriate for real-time QA.

3. Candidate Document Selection: A subset of documents from the total document collection (typically several orders of magnitude smaller) is selected, comprising those documents deemed most likely to contain an answer to the question.

4. Candidate Document Analysis: If the preprocessing stage has only super-ficially analysed the documents in the document collection, then additional detailed analysis of the candidates selected at the preceding stage may be carried out.

5. Answer Extraction: Using the appropriate representation of the question and of each candidate document, candidate answers are extracted from the documents and ranked in terms of probable correctness.

6. Response Generation: A response is returned to the user. This may be affected by the dialogue context and user model, if present, and may in turn lead to their being updated.

The input to the question analysis stage is a natural language question, though this needs qualifying in several ways. First, there may be constraints on the input language. For example, the user may be required to use a subset of natural language, a ‘controlled language’, which is limited in terms of vocabulary and syntax most natural language front ends to databases are limited in this way. It may even be that the user is constrained to use a form-filling interface for expressing questions that significantly simplifies the system's task of interpreting the question, even if it limit the expressivity available to the user. Secondly, in addition to the explicit input of the question string there may be implicit input, in the form of context, if the

system supports an on-going dialogue (so, for instance, there may be ellipsis or anaphora in the question which requires access to dialogue context to be interpreted). Other implicit input could be the system's knowledge of the user and his or her goals.

Output from this stage is one or more representations of the question for use in subsequent stages. For example, if the candidate document selection mechanism to be used in the next stage is an IR system, then one question representation might be a stemmed, weighted term vector for input to the search engine. However, this representation is unlikely to be adequate to allow exact answer strings to be picked out of the documents returned by the search engine. To do this, most systems resort to more detailed analysis of the question which typically involves two steps:

1. Identifying the semantic type of the entity sought by the question (a date, a person, a company, and so on);

2. Determining additional constraints on the answer entity by, for example:

    (a) Identifying key words in the question which will be used in matching candidate answer bearing sentences; or,

    (b) Identifying relations (syntactic or semantic) that ought to hold between a candidate answer entity and other entities or events mentioned in the question.

The first step requires first looking at the key question word (when seeks a date or time; where a location; who a person). However, this is not enough, since various English question words, such as which and what do not carry much semantic typing information. The type of entity questions such as 'Which company…?' or 'What building…?' are seeking is also easy to determine. However, for questions that involve more syntactically complex constructions such as What was The Beatles' first hit single? or How many first class degrees in Computer Science were awarded at Cambridge last year? things become more dificult.

Various systems have, therefore, built hierarchies of question types based on the types of answer sought, and attempt to place the input question into the appropriate category in the hierarchy. for example, manually constructed a question type hierarchy of about 25 types from the analysis of the TREC-8 training data. Extending the analysis to look beyond the semantic type literally requested so as to classify questions like Who discovered America? as Person, while classifying questions such as Who was Christopher Columbus? A manually crafted top-level answer type hierarchy which links into parts of WordNet to extend the set of possible answer types available to the system.

Once the type of entity being sought has been identified, the remaining task of question analysis is to identify additional constraints that entities matching the type description must also meet. This process may be as simple as extracting keywords from the rest of the question to be used in matching against candidate answer bearing sentences. This set of keywords may then be expanded, using synonyms and/or morphological variants or using full-blown query expansion techniques by, for example, issuing a query based on the keywords against an encyclopedia and using top ranked retrieved passages to expand the keyword set. Or, the constraint identification process may involve parsing the question with grammars of varying sophistication. The constituent analysis of a question that it produces is transformed into a semantic representation which captures dependencies between terms in the question. Using a robust partial parser which aims to determine grammatical relations in the question where it can (e.g. main verb plus logical subjects and objects). Where these relations link to the entity identified as the sought entity, they are passed on as constraints to be taken into account during answer extraction.

2.7. Techniques Related to Question Answering

2.7.1. Information Retrieval, Information Extraction and Question Answering

For constructing an effective QA system, which allows a user to ask a question in natural language and receive an answer quickly and succinctly, the more advanced technologies are required that generally combine two related techniques of more established information access tasks known as IR and IE. [4]

IR as the retrieval of relevant documents in response to a user query, has been an active research area since the mid-1950s.

IR systems return documents, not answers, and users are left to extract answers from the documents themselves. However, IR is related to QA in the sense that users form queries because they wish to find answers to questions. It is mainly relevant to question answering for two reasons. First, IR techniques have been extended to return not just relevant documents, but relevant passages within documents. The size of these passages can be steadily reduced, at least in theory, so that in the limiting case, what is extracted is, effectively, just the answer to a question. Thus, question answering can be thought of as passage retrieval in the limit. Second, the IR community has, over the years, developed an extremely thorough methodology for evaluation, the most well-known current exemplars of which are the annual TREC, run by the US National Institute of Standards and Technology. It is from this methodology and community that the recent question answering evaluation developed, which in turn has stimulated much of the current interest in question answering.

The other strand of research that has fed into the current TREC question answering track is IE, as it was initially known, message understanding. IE can be defined as the activity of filling predefined templates from natural language texts, where the templates are designed to capture information about key role running an IE system

designed to fill this template over large volumes of text results in a structured database of information about special domain. This database can then be used for other purposes, e.g. database queries, data mining, summarization. In the current context, IE templates can be viewed as expressing a question and a filled template as containing an answer. Thus, IE may be viewed as a limited form of question answering in which the questions are static and the data from which the questions are to be answered are an arbitrarily large dynamic collection of texts.

2.7.1.1. INFORMATION RETRIEVAL

IR is concerned with selecting from a large body of text (a corpus or document collection) those portions which are in some way relevant to a given query. Typically, the body of text is divided into documents, and the job of the IR system is to find the relevant documents. IR is closely related to QA, as we mentioned in the component of document retrieval, that QA systems generally make use of IR engines in order to narrow down the number of documents to be searched and processed in order to find an exact answer to a question. The traditional IR can be considered to be similar to a web search engine, such as Google, although the original IR engines predate the existence of the web, searching locally-stored collections of documents instead. [19]

As [20] defined, IR is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

An IR engine takes as input a query expressed in the engine's query syntax, which can be as simple as a "bag of words" or as complicated as that of a system such as INQUERY, which allows the user to query on phrases, sets of synonyms and keywords in strict order over windows of text. As output, an IR engine provides a ranked list of documents drawn from the collection it has previously indexed that are relevant to the user's query, for some definition of relevance.

Most IR systems assign a numeric score to every document and rank documents by this score. Several models have been proposed for this process to adapt QA system. Some most used models in IR research are the vector space model, the probabilistic models, and the inference network model [4].

2.7.1.2. VECTOR SPACE MODEL

The vector space model (VSM) of information retrieval, first introduced by Gerard Salton, models both the documents in the collection and the query strings as vectors in a finite dimensional Euclidean vector space. The space has one dimension for each of the terms in the language, with the entry for a given term being the weighting given to that term for the document (0 if the term is absent from the document). The similarity coefficient for a given document is calculated as some function of the vectors representing the document and the query.

There are a number of weighting schemes which can be used within the vector space model. The most common term-weighting strategy for a VSM is known as the tf-idf strategy, which stands for term frequency and inverse document frequency. Term frequency refers to the number of times a term appears within a document. The inverse document frequency of a term is a measure of how rare the term is across the entire corpus. The insight is that if a term occurs frequently in a document, but not frequently in the corpus considered as a whole, then that term does a good job of semantically describing that document. In tf-idf weighting, each term is weighted by the product of its term frequency and its inverse document frequency.

Once the document and query vectors have been constructed, there are various ways to calculate the similarity coefficient. One of the best known is the cosine measure, which assumes that terms occur independently of each other, tf-idf turns out to be a fairly good term weighting strategy. Relating user queries to similar documents in the corpus is simply a matter of computing the cosine of the angle between the query vector and the projections of document vectors onto the hyperplane containing the query vector.

2.7.1.3. PROBABILITY RETRIEVAL MODELS

Probabilistic retrieval models is based on the general principle that documents in a collection should be ranked by decreasing probability of their relevance to a query, and is considered the standard model of probabilistic retrieval. This is often called the probabilistic ranking principle (PRP), which defines the degree of relevance of a document to a query in terms of the probability that the document is relevant to the query. Since true probabilities are not available to an IR system, probabilistic IR models estimate the probability of relevance of documents for a query. This estimation is the key part of the model, and this is where most probabilistic models differ from one another.

The initial idea of probabilistic retrieval was proposed by Maron and Kuhns in a paper published in 1960 [21]. Since then, many probabilistic models have been proposed, each based on a different probability estimation technique.

2.7.1.4. INFERENCE NETWORK MODEL

Most techniques used by IR systems can be implemented under this model. The main idea underlying this approach is that IR is a process of uncertain inference. These models can be seen a blend between logic and probability theory. An important aspect of this class of models is their extendibility and collection independence. In inference based models it is easy to combine different information sources: evidence is not limited to the query formulation, but can also include knowledge about the user, the domain etc. These parameters are collection independent, whereas the relevance based models contain parameters which have to be adjusted for every new collection. Inference models include two subclasses: (1) Inference networks. These are Bayesian decision networks. (2) Probabilistic inference models. These models are based on non-classical logics where the semantics of inference is modeled in probability theory.

A nice property of the inference net framework is that multiple representations (e.g., single terms, phrases, controlled terms) of the same document can be represented in the same network, and also that an information need can be modeled by a parallel evaluation of different queries. The INQUERY system [22], based on the inference network-based retrieval model, has performed well on the TREC evaluation tasks, clearly showing the feasibility of this approach. A weak point is that the conditional probabilities have to be estimated, while the model does not include a theory internal framework to estimate these probabilities. The inference network approach also provides a natural theoretical framework for combining the results of several different retrieval strategies. Combining the results from an inference network carrying out probabilistic retrieval with the results from one running hand-formulated Boolean queries for the same topics produced significantly increased performance.

2.7.2. INFORMATION EXTRACTION

IE is a new technology enabling relevant content to be extracted from textual information available electronically. IE essentially builds on natural language processing and computational linguistics, but it is also closely related to the well established area of IR and it is as a method of searching for information in some ways similar to QA. The IE community devised its own evaluation exercise, the Message Understanding Conferences (MUC), which ran between 1987 and 1998, the last MUC-7 was held in 1998. The termination of the MUC exercises, coupled with the desire to continue to push language understanding technology in novel directions via open evaluation exercises, were enabling conditions for the current TREC QA evaluation. Generally, the process of IE has two major parts. First, the system extracts individual “facts” from the text of a document through local text analysis. Second, it integrates these facts, producing larger facts or new facts (through Inference). As a final step after the facts are integrated, the pertinent facts are translated into the required output format.

2.7.2.1. TEMPLATE MATCHING

Formerly known as message understanding, the general goal of information extraction is to locate information within free text that matches prepared templates. Templates can represent events, references to objects or entities, business deals, movements of military resources, or anything else of interest to the system. Each template, like a frame, contains a number of slots that the IE system would like to fill. For example, a user requirement for information about car accidents might use a template made of fields such as “Number of injured”, “Number of cars involved”, “Names of victims”, “Location”. The IE engine would then attempt to fill these fields as if entering the information in a database. When an IE system locates some text matching one of its templates, it uses as much context as it can to fill out all of the slots in the template.

2.7.2.2. NAMED ENTITY RECOGNITION

Named Entity (NE) Recognition is a specialized form of the IE task dedicated to identifying phrases in text that refer to entities like people, organizations, date and currency amounts and facilities, and extracting their semantics. Names appear frequently in many types of texts, and identifying and classifying them simplifies further processing; names, furthermore, are important as argument values for many extraction tasks. Names are identified by a set of patterns (regular expressions) which are stated in terms of parts-of-speech, syntactic features, and orthographic features (e.g. capitalization). However, it is not enough for an NE recognizer to be able to identify that the phrase “Pope John Paul II” refers to a person; the system must be able to fill out a template of information, such that the person is male, his first name is “John Paul”, his title is “Pope” and his generation is “II”.

2.7.2.3. AUTOMATED CONTENT EXTRACTION

Automated Content Extraction (ACE) is a large-scale evaluation effort for IE systems run by the National Institute of Standards and Technologies (NIST). ACE challenges

participating systems to locate references of people, geo-political entities such as cities, states and nations, locations with physical extent, organizations and facilities within newswire text and broadcast news transcripts. Additional goals of the ACE program are to be able to track mentions of entities throughout larger bodies of text, and to recognize relationships among entities. Prior to ACE, standardized IE evaluation opportunities were provided by the various Message Understanding Conferences (MUCs) held between 1987 and 1998, the proceedings of which are available from NIST.

2.7.2.4. SYNTACTIC STRUCTURE

Identifying some aspects of syntactic structure simplifies the subsequent phase of fact extraction. After all, the arguments to be extracted often correspond to noun phrases in the text, and the relationships to be extracted often correspond to grammatical functional relations. On the other hand, the identification of the complete syntactic structure of a sentence is a difficult task. As a result, there is a great variation in the amount of syntactic structure which is explicitly identified. Some systems don't have any separate phase of syntactic analysis. Others attempt to build a complete parse of a sentence. Most systems fall in between, and build a series of parse fragments. In general, they only build structures about which they can be quite certain, either from syntactic or semantic evidence. The NYU Proteus system, following the lead of the SRI FASTUS system [23] builds structures for noun groups (a noun plus its left modifiers) and for verb groups (a verb with its auxiliaries) both of these can be built in most cases using just local syntactic information. In addition, it builds certain larger noun phrase structures (conjoined noun groups, noun groups with appositional modifiers) if it has semantic information to confirm the correctness of the structure.

All of this is done using the same regular expression pattern matcher, unlike some other systems, no special procedures are used for parsing. Although, limitations of IE systems include the fact that the templates have to be hand-edited by humans,

which can take significant effort that is usually not transferable across domains, but, a database can be compiled about the various types of events or entity references extracted information from a large body of text, and can be combined with modern natural language database query front-ends to make a kind of narrow-domain QA system. As are natural language database front-ends, IE systems are constrained in the kinds of questions they can answer by the structure of their database templates. Just as with IR engines, however, a good IE system can be an enormously useful resource for a high-quality QA system to have. IE can assist with question analysis, helping the system understand what type of entity it is looking for, and also with answer extraction, identifying entity references of the desired type among passages retrieved by upstream passage analysis and document retrieval modules [4].

2.8. AMHARIC QUESTIONS STRUCTURE

As discussed in [24] Amharic questions have a common order which is used in spoken and in written Amharic. There are two types of questions: The first is called Yes/No questions and the second, Wh-questions.

In the Wh-question section, [24] says that a word or a phrase that is to be emphasized should be placed before a verb. An interrogative, word or phrase, since the whole interest of the question lies on it, is generally used before a verb.

So, here, the discussion about Wh-questions will be limited only to question words which are placed before a verb. In Amharic, these interrogative words are either interrogative pronouns or adverbs.

A sentence, in every language, is a group(s) of word(s) that comply with the grammatical arrangement of the language and capable of conveying meaningful message to the audience. A sentence in Amharic can be a statement that is used to declare, explain, or discuss an issue; an interrogative sentence, which can be used for questioning; exclamatory and imperative. The concern of this thesis work is only on interrogative sentence.

The statement (አረፍተ-ነገር) will have the noun phrase and verb phrase combinations. The noun phrase and the verb phrase further will be divided to different particles such as other sub noun phrase and verb phrase, noun, adjectives, specifier and so on. Similarly the interrogative sentence will have the same structure with little rearrangements and introduction of question particles. Questions will be raised for different purposes such as to know something unknown, or to assure something that is known [6].

In every language, questions are constructed with the help of question particles (also known as interrogative words) and a question mark (?), which is placed at the end of the question. The question mark, by itself kept at the end of the statement, indicates that the sentence is a question. In English, the interrogative words (WH words) who, what, where, when, why, how … are used to construct a question. In Amharic, there are a number of interrogative words that will help in constructing a question. [6] Identified Amharic question particles are shown below.

ማን, ለማን, ማነው, እነማን, ማንማን, ማንን, ማናማን, እነማንን, የማን, ከማንኛው,
ማንኛው, ማንኛይቱ, ማንኛዋ, ማንኛቸው, በማን,
ተናገር, ጥቀስ, ግለፅ, ዘርዝር, ጥራ,
የት, የቱ, በየት, የቷ, የቲቱ, የቶቹ, ወዴት, ከየትኛው, የየት, የትኛው, የትኛዋ, የትኞቹ,
ከየት, እስከየት, ወደ የት, የየትኛው, በየትኞቹ
ምን, ምንድን, የምን, ምንህን, ምንሽን, ምናችሁን, ምኔ, ምኑን, ስለምን, እንደምን, ለምን,
በምን, እስከምን, ከምን, ወደምን, ምንጊዜ, ምንያህል,
መቸ, መቼ, በመቸ, እስከመቸ, ለመቸ
ስንት, ስንቱ, ከስንት,
ወይስ, ምረጥ, ወይ, ይሆን, እንደ, እንዴት

From the particles listed above, some of them are used for list questions and all Amharic listing questions need at least one of the following question particles, which is used to construct a question that need a list of items as an answer. These are:

እነማን, እነማን, ማንማን, ማናማን, እነማንን,
ጥቀስ, ዘርዝር, የቶቹ, የትኞቹ, በየትኞቹ,

A List question is answered in the following steps: First, the answer type of the question is determined. Then, a query for searching the corpora is generated using the target and the question text. We use the query to create a collection of documents in which we look for the answers to the question. We extract from the collection all terms that comply with the answer type; this constitutes the initial candidate list. A similarity value is then computed for each pair of candidate answers based on their co-occurrence within sentences. Having clustered the candidates and determined the most likely cluster, the final candidate answers are selected. This step is also used in this work.

2.9. RELATED WORKS

2.9.1. ANSWERING LIST AND OTHER QUESTIONS

The work by [1], explained an approach to answering List questions for English language. The approach attempts to select a subset of the initial candidate answers which are extracted from a number of relevant documents. To do so, the information regarding the co-occurrences of the candidates in a larger number of related documents are extracted and a method attempts to group candidates that co-occur more often. The most likely cluster to contain the answers is returned as the final candidates using the notion of spies which are the target and the question keywords.

Using empirical results based on TREC questions and also showing TREC official results showed that the hypothesis which states that instances of the answer to a List question tend to co-occur within sentences and tend to co-occur with the question or the target keywords, seems to be correct. The results demonstrate the effectiveness of the approach. While this approach has been shown promising, there is much room for improvement.

The component evaluation carried out can be used to identify weak parts of the system. The table below shows the performance of each component.

Module	Performance
Answer Type Recognition	0.98
Document Retrieval	0.79
Candidate Answer Extraction	0.62
Clustering	0.32
Pinpointing	0.95

Table 1. Razmara's [1], performance per module

As table 1 shows, the Candidate Answer Extraction and Clustering modules have relatively low performance and possibly need to be improved.

Candidate Answer Extraction: the quality of the extracted entities for each question type was evaluated. The NE taggers for Person, Organization and Job are to be enhanced; especially, for Person which constitutes about 32% and 39% of the training and test sets and have a significant impact on the final results. Movie and Other are also question types that the system does not work well with. A term-type validation module needs to be developed to filter out terms that do not comply with the answer type from the initial list.

Clustering: Clustering is the essence of the approach. It is, however, the most tricky and the weakest module in the system. Although many clustering methods which need absolute data points cannot be applied to this module (since only have the similarities or distances between terms), there are still other clustering method that could be tried. In addition, using fuzzy clustering, in which data elements can belong to more than one cluster with a membership degree for each association. Another important issue in clustering is the similarity measure. They also experimented the three similarity measures among which Weighted Mutual Information (WMI) seems the most promising. However, other similarity measures (e.g. F-test, Phi-squared, etc.) could also be tried.

As [1] also presented a keyword-based approach to extracting interesting sentences to answer other questions. The method is based on the identification of target-specific and universal interest markers. Target-specific markers are identified by named entities found in the Wikipedia online encyclopedia. The frequency of these named entities in the relevant documents in the corpus are then used as a measure of how interesting they really are. Target-independent interest markers are defined as the most frequent terms in the TREC-2004 vital and okay nuggets and include superlatives, numerals and specific keywords. Using these markers, extract and rank sentences from the corpus and return the top-scoring ones as the answer. When using the 2004 TREC data for development, the approach achieved an F-score of 0.265 with the 2005 TREC questions, placing it above the best scoring TREC-2005 system. In addition, the results of the system in TREC-2006 and TREC-2007 showed the approach is promising.

Currently, the system is highly dependent on the Wikipedia articles; other robust alternatives should be used in case the proper Wikipedia article is not found or it contains a large amount of data about the target as the results showed using top N corpus documents are not as appropriate. In addition, perform proper coreference resolution on the Wikipedia terms; this would allow to better rank and identify the interesting terms.

To represent interesting facts, consider only individual terms. A more precise method would ultimately be to expand the approach to extracting entire predicate structures; with roles and arguments.

2.9.2. AMHARIC QUESTION ANSWERING SYSTEM FOR FACTOID QUESTIONS

[6] has developed a QA system for Amharic with different components. The question processing module classify the question to appropriate question types, determine the expected answer type, and generate proper IR query. The document retrieval

component will retrieve relevant documents that will be further processed by the later modules. The sentence/paragraph ranker will re-rank sentences and paragraphs based on the answer particle in them. The final module, the answer selection module, will select the best answers to the user with an additional source for the user in case detailed information is needed.

This research work attempted to identify the basic language specific issues in question answering. The first task tackled is normalizing the document so that a standard document will be indexed and matching relevant documents during searching will be maximized. Also identified proper question particles as well as question focuses that will help in classifying the question. Gazetteer based and pattern based answer selection algorithms have been developed to maximize correct answer selection. The algorithm first identifies all possible answer particles in a document. Once the answer particles are identified, the distance of every question particle with the question terms will be calculated. The one with the minimum distance from the query terms will be considered the best candidate answer of that document. Once candidate answers are selected from every document, candidate answers which have been repeated more than once (i.e., appeared in more than one document) will be given higher rank. Candidate answers with maximum number of query terms matched in a document will be given higher priority in case a similar rank is given for two or more candidate answers.

The evaluation of the system, being the first AQA system, shows promising performance. The rule based question classification module classifies about 89% of the question correctly. The document retrieval component shows greater coverage of relevant document retrieval (97%) while the sentence based retrieval has the least (93%) which contributes to the better recall of their system. The gazetteer based answer selection using a paragraph answer selection technique answers 72% of the questions correctly which can be considered as promising. The file based answer selection technique exhibits better recall (0.909) which indicates that most relevant documents which are thought to have the correct answer are returned. The pattern

based answer selection technique has better accuracy for person names using paragraph based answer selection technique while the sentence based answer selection technique has outperformed in numeric and date question types. In general, their algorithms and tools have shown good performance compared with high-resourced language QA systems such as English.

The study has adopted the efforts made towards English QA systems techniques to Amharic, The study has paved the way to identify language dependent components specific to AQA, The study identified key components of Amharic QA systems which can be considered a framework for factoid questions, The study also showed the strategy, algorithms, and techniques in developing Amharic QA system.

Question answering is a very complex task, which consumes time and needs a number of different NLP tools. There are a number of rooms for improvement and modification for Amharic question answering. Some of them indicated in this research are enhancing the Amharic stemmer, statistical classifications, extending to other question types such as list and implementing QA for specific applications.

2.9.3. A STUDY ON QUESTION ANSWERING SYSTEM USING INTEGRATED RETRIEVAL METHOD

In the thesis [4], the aim was to improve accuracy of retrieving answers to the user questions in a restricted domain for English language. However, the importance of the information of special domain database for the retrieval in restricted domain is regarded. Thus, the proposed QA system integrated retrieval of answer from a Question & Answer database and the document database based on the VSM model and the shallow language analysis. The system is first developed for a medium-sized domain of sightseeing information. The result of experiments showed the system has a means of retrieving answers by the proposed method and achieved 0.76 in the MRR up to the top three answers in the restricted domain. And the Mean Reciprocal Rank (MRR) of question type correlating named entity was higher than others, such as

TIME&DATE, LOCATION, PERSON, QUANTITY&AMOUNT, the average MRR of them achieved about 0.84 up to the top three answers. As the result, the usability of the restricted domain to retrieve answers is proved. Also, discussed the limitation of the restricted domain information by analyzing the error replies.

Moreover, implemented the speech interface using an acoustic model Hidden Markov Model (HMM), a pronunciation lexicon, and a language model based on the feature of Chinese sentence patterns. An experiment to evaluate speech recognition was performed, and about 70% questions could be recognized correctly according to limited grammars. The result showed it is possible to design a speech-driven QA system for the sightseeing domain.

Considering the usability of the special domain database and its limitation, and also proposed a web-based QA system for restricted domain that integrated three processes of answer retrieval utilizing keywords expansion of semantic information.

The integrated retrieval technique which combines the OkapiBM25 of probabilistic retrieval model and semantic analysis based on the semantic database HowNet and the special domain HowNet was described. The MRR measure of over 0.8 was achieved using the proposed method in sentence level for all question types. Moreover, the processing of Named Entity Recognition was implemented to answer the questions of factoid types and the MRR of 0.78 was obtained. Also, in this thesis, they described a new approach of question expansion for restricted domain, with particular emphasis on the comparison of three stages of keyword expansion utilizing the measure of semantic similarity. The result showed that the Expansion (keyword expansion using synonym and keyword expansion using special domain QA database and document database) are feasible and efficient to find more similar documents and sentences, as well as retrieving the answers from web resources to the user questions.

In this thesis, the questions were classified by heuristic rule based on the interrogatives and the qualifiers of interrogative, but it is deficient since the end-user

commonly asks some questions in atactic spoken language. Thus, a more effective technique of question classification is necessary.

To fully support QA, the effectiveness of the system was verified using a larger-scale special domain database and transfer to other domains. This paper also attempt to develop a method to automatically create the special domain HowNet knowledge base, considering the cost of construction manually. In order to avoid the excess expansion of keyword, more deep discussion and examination about the technique of keywords expansion are necessary in the future. In addition, the techniques of asking-back and paraphrasing natural language should be reinforced to deal with the problematic questions.

CHAPTER THREE

3. DESIGN OF THE SYSTEM

This chapter gives an architectural and design overview of AQA system that attempts to answer list questions. The QA distributional hypothesis was part of the inspiration when designing the structure of the AQA system. This section also describes the many tasks and techniques that need to be addressed to develop such systems.

3.1. ARCHITECTURE OF LIST QA SYSTEM

For designing the architecture, the architecture designed in [1] for a list question is used as a benchmark. From the architectural components in [1] the webpage and HAC clustering modules are removed. The webpage is not used in this work since there are almost no Amharic web pages on tourism area and the HAC clustering module is omitted since the classification module have a better performance in the previous experiments on list question answering. This architecture also made modifications on the techniques used within modules. Lexical patterns are used in the answer type recognition module. Moreover, the architecture is constructed in a cyclical manner given that question answering is a two-way communication.

The architecture of list QA system consists of several components. Figure 2 shows the overall structure of the system. The role and responsibility of every component is shown below:

1. Query interface: is used to retrieve the question posted by the user.

2. Answer type recognition: use question classification to identify the type of the question that is used to determine the answer type.

3. Document retrieval module: is used to retrieve the document based on important keywords present in the question.

4. Candidate extraction: this component focuses on the pattern of the question to retrieve candidates from the relevant document.

5. Co-occurrence extraction and candidate selection: based on co-occurrence of candidate answers and answer type, the system filters candidate answers collection.

6. Classification: based on the Weighted Mutual Information, answers that are related with the asked question will be classified as relevant.

7. Answers: The answer with highest priority is shown to a user.

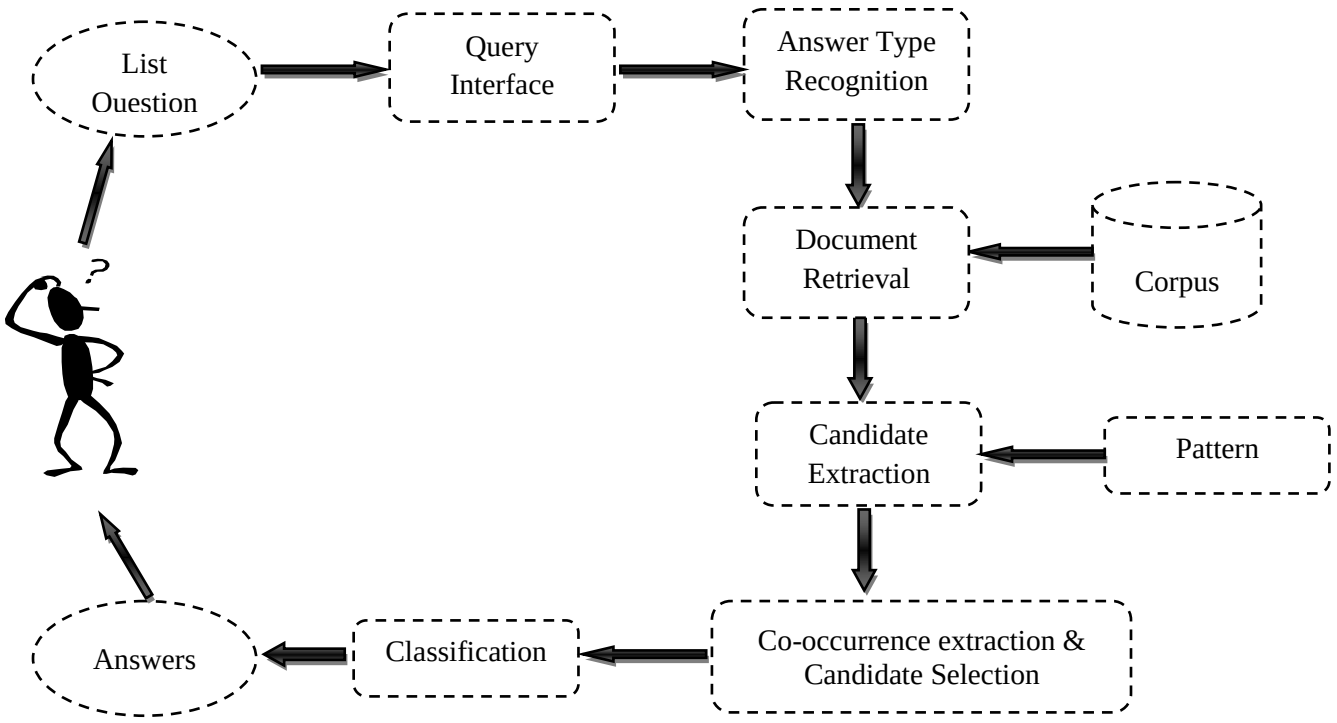


Figure 2 Architecture of the system

The AQA system is designed based on the distributional hypothesis, which states that words occurring in the same contexts tend to have the same meanings" [25]. Distributional Hypothesis is the basis of Statistical Semantics defined as the study of "how the statistical patterns of human word usage can be used to figure out what people mean, at least to a level sufficient for information access".

This hypothesis is used in developing list QA systems and showed a promising performance as demonstrated in [1].

The general idea behind the distributional hypothesis seems clear enough: there is a correlation between distributional similarity and meaning similarity, which allows utilizing the former in order to estimate the latter.

As [1] hypothesized:

- Answers to a list questions have same semantic entity class.
- Answers co-occur within the sentences of the documents are related to the target and the question.
- Sentences containing the answers share similar context.

In this research, list questions are answered based on the steps defined in [1]. These steps are:

1. Interface design
2. Determining the answer type of the question
3. Query generation for document retrieval
4. Extract all terms that comply with the answer type (initial candidate list) from the document.
5. A similarity value is computed for each pair of candidate answers based on their co-occurrence within the sentence.
6. Having classified the candidates and determined the most relevant candidates, the final answers are selected.

1. Interface design

The User Interface of a system was found to be important because users are critical about the way they are asked to input their questions. They did not always want to phrase their question as a question but sometimes preferred to use keywords, e.g. “a keyword search would be more useful”. Users also care about the way in which the

results are presented to them and whether the system desires any additional responses from them. They did not like being prompted for feedback. Considering the needs, Our AQA system also has an interface that can accept question from a user and also provide an answer in the box provided below it. It also enables a user to ask the system another or even the same question again by clicking a "እንደገና" button provided for such a task. In general, the interface has a menu, a description about the system, a single line text box to insert the question, a "መልስ" button that will execute the answering process, a multiple line text box for displaying the answer, and two other buttons, "እንደገና" and "መዝጊያ", for asking again and for exiting the program respectively. The screen shot of the system is given in figure 3.

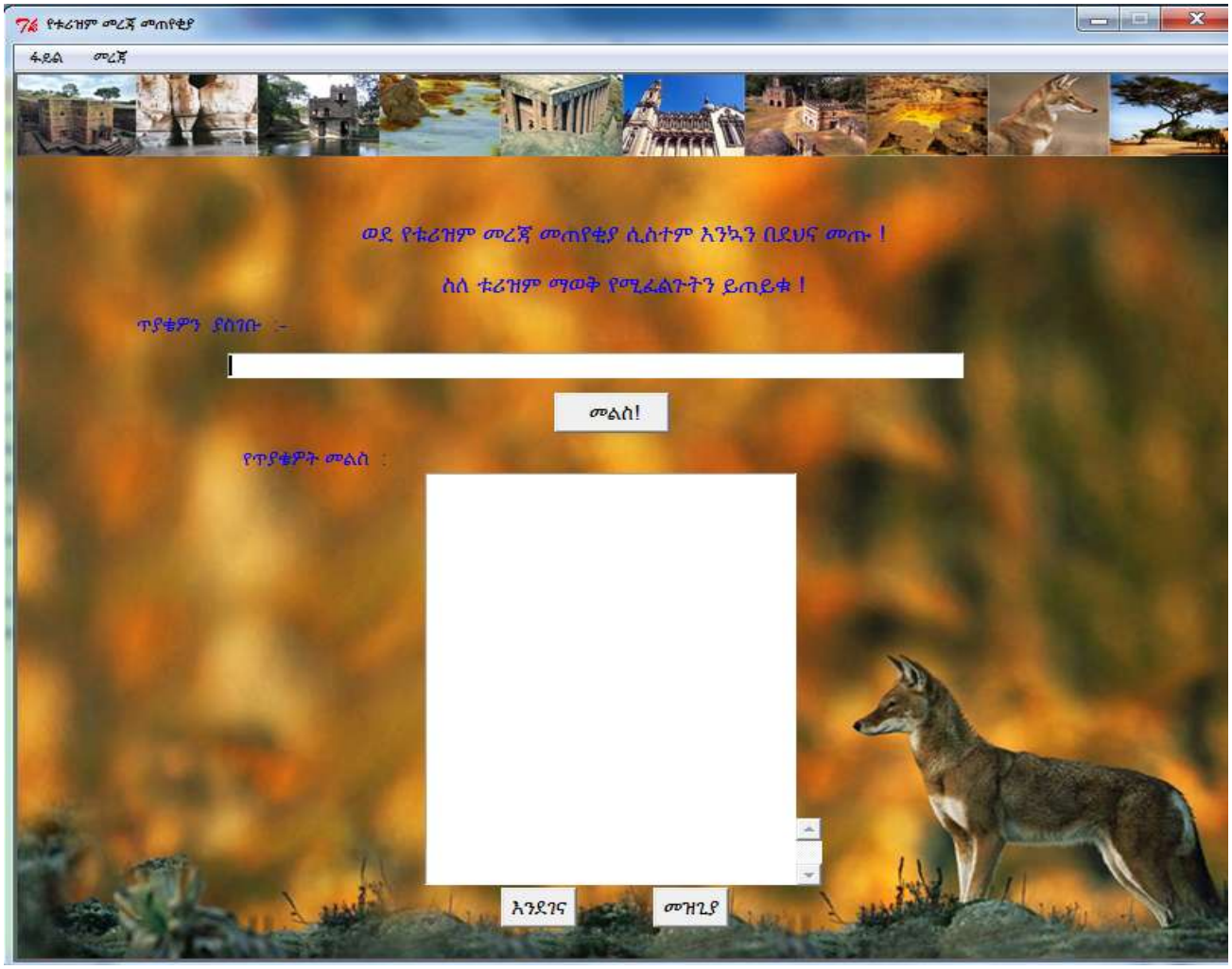


Figure 3 Interface of the system

2. Answer type recognition

In list questions, the answer to the question includes one or more entities (or things) of a given type. The type is given as part of the question and used by the system to enhance the ability of addressing the question. Since the type of the concepts of interests that answer a given question are often obvious from the question itself, this form of information need representation using an answer type.

Initially, by considering words included in the question, the kind of answer can be determined. Since this work is focused on list questions, this module only checks if a question is of list type or not.

Next, Different instances of an answer to a list question have a special relation to one another. Beside the fact that they are all the same entity class (e.g. parks, museums, hotels, …), they either co-occur within sentences, or occur in different sentences having lexical similarities or occur in different sentences that are partially or totally semantically equivalent.

Question analysis will be done to be familiar with answer type. Questions and documents are reviewed to come up with a semantic entity classes that any question may be included in it.

Roderiguez [26] defined three basic components for the representation of entity classes in an ontology:

1. A set of synonym words (synset) that denotes an entity class,
2. A set of semantic interrelations among these entity classes, and
3. A set of distinguishing features that characterize entity classes.

Each question in the Ethiopian tourism domain is associated to one of the following eleven semantic entity classes collected from experts in the tourism sectors.

Birds (ወፎች), Celebrations (በአላት), Hotel (ሆቴል), Languages (ቋንቋ), Museums (ሙዚየሞች), Culture (ባህል), Nationalities (ብሔር-ብሔረሰብ), Park (ፓርኮች), Tourtravels (አስጎብኚዎች),lakes (ሃይቅ), Touristsites (መዳረሻዎች), Wildanimals (የዱር እንስሳት) and monuments (ሃውልቶች).

Based on the above set of synset lexical patterns are developed for answer type recognition module.

Table 2 shows the lexical patterns written in regular expretions. (/) separates alternatives and INIT means if a word or set of words are 'in it'.

Lexical pattern	Type
INIT = (ዘርዝር / ጥቀስ / እነማን / የትኞቹ / ማንማን / ማናማን / እነማንን / የቶቹ / በየትኞቹ)	
INIT = (ወፍ / ወፎች / አእዋፍ / አእዋፋት / አእዋፋቶች)	ወፎች
INIT = (በአላት / በአል / ክብረበአላት / ክብረበአል / በአሎች / ክብረበአሎችን)	በአላት
INIT = (ሆቴሎች / ማደሪያዎች / መዝናኛዎች / መመገቢያዎች / መብያዎች)	ሆቴል
INIT =(ቋንቋ / ቋንቋዎች / መግባቢያ / 'መግባቢያዎች)	ቋንቋ
INIT = (ሙዚየም / ሙዚየሞች / ሙዝየም / ሙዝየሞች)	ሙዚየሞች
INIT = (ሕዝቦች / ብሔርብሔረሰቦች / ብሔሮች / ብሔረሰቦች)	ብሔር-ብሔረሰብ
INIT = (ፓርኮች / ፓርኮች)	ፓርኮች
INIT = (አስጎብኚዎች / የጉዞ-ወኪሎች)	አስጎብኚዎች
INIT = (መዳረሻዎች / መስሕቦች / ሃብቶች / ቅርስ / ሜዳ / ቦታዎች / ቅርሶች / ከተሞች / ሜዳዎች / ገዳም / ገዳማት)	መዳረሻዎች
INIT = (እንስሳት / አራዊት / የዱርእንስሳት / የዱርአራዊት)	የዱር እንስሳት
INIT = (አልባሳት / ልብሶች / መጠጦች / ምግብ / ምግቦች / ሙዚቃ / ጭፈራ / ሙዚቃዎች / ጭፈራዎች)	ባህል
INIT = (ሀይቅ / ወንዝ / ወንዞች / ሀይቆች)	ሀይቅ
INIT = (ሀውልቶችን / ሀውልቶች / ሀውልት)	ሀውልት

Table 2. lexical patern of the system

In the table 2, INIT = (ዘርዝር / ጥቀስ / እነማን / የትኞቹ / ማንማን / ማናማን / እነማንን / የቶቹ / በየትኞቹ); is used to identify if a question is a list question or not. Any task in this system will proceed if the question contains one of the question particles mentioned above. This will delimit the scope of the research in to a manageable range.

In view of the fact that Ethiopia is a country where there are more than 80 nations and nationalities, they can be an answer named "ብሔር-ብሔረሰብ", if a question contains (ሕዝቦች / ብሔርብሔረሰቦች / ብሔሮች / ብሔረሰቦች) in it. And as shown in table 2, the other types will be determined by the words they contain. For example, If a question is " በኢትዮጵያ የሚገኙ ብሔራዊ ፓርኮችን ዘርዝር ", It will be recognized as a list question because it contains the word "ዘርዝር" and the answer type is Park because of the word "ፓርኮችን".

3. Document retrieval

In this module, documents that are containing the answers will be retrieved from the large corpus. For this task, a probabilistic IR system for Amharic language by Amanuel [19] has been applied with some modifications on customizing steaming and query generation modules.

As a local work, a number of works have been done to retrieve Amharic text documents. For instance, Tewodros [27] developed Amharic text retrieval using latent semantic indexing (LSI) strategy with Singular Value Decomposition (SVD). LSI indexing technique were used which partially handle the problems of exact term matching by organizing terms and documents into a semantic structure. The result of the experiment shows that, the recall-precision graph of the LSI method was above standard vector method. The average precision registered by the system was 71%.

Moreover, others also developed Amharic IR system by integrating query expansion techniques to enhance the performance of the system to solve the problem of polysemous and synonymous which exists in Amharic text. Several query expansion techniques were tested, such as, global analysis, local analysis, bi-gram analysis, bi-gram based thesaurus and statistical co-occurrence method.

Even if several IR systems have been developed in the last decade, most of the works are attempted to design an Amharic IR system using vector space model, which may not control uncertainty nature of IR system.

The most recent one by Amanuel [19] is aimed to develop probabilistic based Amharic IR system to enhance the performance of the Amharic IR system. The obtained result indicates that probabilistic IR system for Amharic language registered encouraging performance that increases in 10% F-measure as compared to the recent IR system developed for Amharic language [19].

The probabilistic model is based on the following fundamental assumption as stated in [28]. Given a user query q and a document d_j in the collection, the probabilistic model tries to estimate the probability that the user will find the document d_j interesting or relevant. The model assumes that this probability of relevance depends on the query and the document representations only. Further, the model assumes that there is a subset of all documents, which the user prefers as the answer set for the query q. Such an ideal answer set is labeled R and should maximize the overall probability of relevance to the user. Documents in the set R are predicted to be relevant to the query. Documents not in this set are predicted to be non-relevant.

In IR, searching is possible or efficient when the text is small. However, in large databases searching will take much more time and space unless indexing structure is used to organize documents [19]. And of course, when indexing a document for our question answering system, a data of size 291kb took over 24 hrs.

In [19] a binary independent model is used to design the system. This is because; the first step in most probabilistic methods is to make some simplifying assumptions. Thus, binary independent model is the model that has been used with the probabilistic ranking principle by introducing some simple assumptions which makes estimating the probability function practical.

Once the relevant documents are retrieved using the probabilistic IR, two document collections are created; source document collection and domain document collection.

Source document is a set of documents where candidate answers are extracted from. Domain document is a collection used to extract relationship between each and every candidate answers. The source document collection is, indeed, a subset of domain document collection, which contains the most relevant documents from the corpus.

4. Candidate answer extraction

The next section is candidate answer extraction. This candidate answers are extracted from source documents. The answer candidate identification ability of the system is highly related to the answer finding strategies applied by the question answering system.

Candidate answer identification involves three strategy based on question focus. This question focus strategy include search for a specific entity category in the case of list question types.

All terms that conform to the answer type are extracted from the source document collection. These types will direct to a gazetteer that contains full documentation of the tourism sector in different entity classes.

Answer candidates are to be processed farther in the next stages and selected as the final answers for each question.

5. Co-occurrence information extraction

As [1] stated, the answer instances to a list question co-occur within the sentences of the documents related to the target and the question.

The list of initial candidate answer will be used to compute the similarity between themselves how frequently they co-occur in the domain documents; this will be used

by the candidate answer selection module to select terms that tend to co-occur more often.

The similarity between two candidate terms is a normalized value denoting how often they co-occur within documents related to the target. For this purpose, using the query generated in the previous section, a list of relevant documents from the IR is retrieved. This constitutes the domain collection from which sentences will be extracted to compute co-occurrence information. Once all the data regarding term co-occurrences is collected, the similarity between each pair of terms is computed. The classification module uses this information later on using terms that tend to co-occur more often. This is used as an indicator of their semantic similarity and 1st order co occurrence (if two terms co-occur with in a sentence) is used as a sign for their co-occurrence behavior.

Co-occurrence matrix is where sentences are checked as to whether they include one or more candidate terms. The information regarding the occurrence and co-occurrence of the candidate terms are stored in a 3D co-occurrence matrix [sentence – candidate – candidate] (3D) matrix, contain information regarding the occurrence and co-occurrence of candidate terms and each sentence is represented using 2D [candidate – candidate] matrix.

```
# candidate term to term occurance
    coo=[]
    for i in range(0,len(cands)):
        for j in range(0,len(cands)):
            if cands[i] != cands[j]:
                if (cands[j] + ':' + cands[i]) not in coo:
                    for k in range(0,len(senten)):
                        if senten[k] != '':
                            if cands[i] in senten[k] and cands[j] in senten[k]:
                                coo.append({cands[i] + ':' + cands[j]:'1'})
                            elif cands[i] in senten[k] and cands[j] not in senten[k]:
                                coo.append({cands[i] + ':' + cands[j]:'2'})
                            elif cands[i] not in senten[k] and cands[j] in senten[k]:
                                coo.append({cands[i] + ':' + cands[j]:'3'})
```

Figure 4 Term to term co-occurrence in a sentence

6. Candidate answer selection

To come up with a candidate list out of the extracted answers in the later stage, it is possible to use two approaches namely classification and clustering.

As [1] stated in its evaluation section Cumulative Classification performs pretty well comparing to the clustering method. Therefore, in this research classification is used since it is much more effective in classifying candidates in to class as Relevant and Irrelevant, then, the candidates in the class Relevant are returned.

We measure the similarity between two candidate terms as a normalized value denoting how often they co-occur within sentences of the domain documents. This value is normalized by the frequencies of the terms. Once all the data regarding term appearances and co-occurrences is collected, the similarity between each pair of terms is computed.

To compute the similarity between two terms, $term_i$ and $term_j$, a 2 X 2 contingency table is used.

	$Term_i$	**$Term_i$**	**Total**
$Term_j$	O_{11}	O_{21}	O_{11+} O_{12}
$Term_j$	O_{12}	O_{22}	O_{12+} O_{22}
Total	O_{11+} O_{12+} O_{21+} O_{22}		N

Table 3. (2 X 2) Contingency Table

Where:

O_{11} : Number of sentences in which $term_i$ and $term_j$ co-occurred;

$S_{11} = \{ S_k \mid S_k [i; j] = 1 \}$; $O_{11} = |S_{11}|$

O_{12} : Number of sentences in which $term_i$ appeared but $term_j$ did not appear;

$S_{12} = \{ S_k \mid S_k [i; j] = 2 \}$; $O_{12} = |S_{12}|$

O_{21} : Number of sentences in which $term_i$ did not appear but $term_j$ appeared;

$S_{21} = \{ S_k \mid S_k [i; j] = 3 \}$; $O_{21} = |S_{21}|$

O_{22} : Number of sentences containing neither $term_i$ nor $term_j$;

$S_{22} = \{ S_k \mid S_k [i; j] = 0 \}$; $O_{22} = |S_{22}|$

N : Total number of sentences in the domain collection.

$N = |S|$ or $N = O_{11} + O_{12} + O_{21} + O_{22}$

The above contingency table is coded as shown below.

```
# contenjency table; term(i,j)= 1(i,j), 2(i,^j),  3(^i,j), 0(^i,^j)

    y={}
    for i in range(0,len(coo)):
        yy=list(coo[i].keys())
        xx=list(coo[i].values())
        if yy[0] not in y:
            y[yy[0]]=[0, 0, 0]
        if  xx[0] == '1':
            y[yy[0]][0]+=1
        elif  xx == ['2']:
            y[yy[0]][1]+=1
        elif  xx == ['3']:
            y[yy[0]][2]+=1
```

Figure 5 2X2 contingency table representation

An experiment [1] showed that weighted Mutual Information (WMI) have a better performance than chi-square and Yates' Chi-Square Test. WMI tells us about the amount of information one point contains about the other, or in other words, the reduction in uncertainty of one point due to knowing about the other.

WMI is zero when the number of co-occurrences is exactly the same as expected, O_{11} = (F1 * F2) / N. Although the function is not defined at O_{11} = 0, the WMI value gets close to 0 as O_{11} approaches 0. As the number of co-occurrences (O_{11}) grows from 0, the WMI value decreases until the turning point, at which the slope equals zero. The behavior of the function is undesirable between these two points since the similarity value of two terms should increase, as there are more co-occurrences between them. From the turning point onward, the value of WMI increases, as we would expect from a suitable similarity measure.

To measure the WMI of $Term_i$ and $Term_j$;

$$WMI\ (Term_i; Term_j) = O_{11} \times \log_2 \frac{N \times O_{11}}{(O_{11} + O_{12})\ (O_{11} + O_{21})}$$

The WMI calculation is performed as shown in figure 6 using python code.

```
# wehighted mutual information

    wmui={}
    logg=0
    yy= list(y.keys())
    for i in range (0,len(y)):
        for j in range(0,len(yy)):
            n = len(senten)
            c = ((y[yy[j]][0]+y[yy[j]][1])*(y[yy[j]][0]+y[yy[j]][2]))
            if c != 0:
                logg= float((n*y[yy[j]][0])/c)
            if logg!=0:
                x = math.log(logg,2)
            wmi= y[yy[j]][0] * x
            if wmi != []:
                if wmi >= 0:
                    wmui[yy[j]]=wmi
```

Figure 6 calculating weighted mutual information

After knowing the similarity of each word with another cumulative classification is computed using the cumulative similarity of a candidate term. In this method, candidate terms are simply classified into two classes: Relevant and Irrelevant. The classification is based on the same co-occurrence information calculated.

The idea is that candidate terms that generally co-occur more often with other candidate answers, hence with a relatively higher similarity value, are more likely to be the answer. Therefore, we use the sum of the similarities of each term to other terms as an indicator of how often each term co-occurs with all other terms.

Candidates with a cumulative similarity greater than a zero are labeled as Relevant and the rest are Irrelevant. The steps to classify the candidate answers into Relevant and Irrelevant are as follows:

1) For each term in the candidate list, compute the sum of its similarities to all other candidate terms;
2) Sort the candidate terms based on their cumulative similarities;
3) Return the candidate terms as the final candidate terms.

```
# Cumulative similarity

    comfr={}
    zz=list(wmui.keys())
    w=[]
    for i in zz:
        x=i.split(':')
        for j in x:
            w.append(j)
    for i in range(0,len(w)):
        comfr[w[i]]=0
        for j in zz:
            if w[i] in j:
                comfr[w[i]] += wmui[j]

# Ranking based on cumulative frequency

    xc=list(comfr.keys())
    xv=list(comfr.values())
    xv.sort()
    xv.reverse()
    sortd=[]
    if comfr:
        pass
    else:
        for i in range(0,len(cands)):
                sortd.append(cands[i])
    for i in range(0,len(xv)):
        if xv[i] > 0:
            for k in range(0,len(xc)):
                if comfr[xc[k]]==xv[i] and xc[k] not in sortd :
                        sortd.append(xc[k])
```

Figure 7 cumulative similarity and ranking python code

The variable 'sorted' shown in figure 7, contains the final ranked answers and it will be delivered for the user of the system through the multiline text box displayed in the interface of the system.

3.2. PERFORMANCE EVALUATION AND FINDINGS

In this section, the performance of the system in answering list questions will be illustrated for each answer type and finally, the results will be discussed.

3.2.1. EVALUATION MEASURE

An answer to a List question is evaluated using precision and recall based on the complete list of known distinct correct answers provided by the tour operators. For each question, instance precision (IPrecision) is the percentages of instances returned that are correct and instance recall (IRecall) is the percentage of the expected correct instances that are returned.

$$\text{IPrecision} = \frac{\text{number of correct answers}}{\text{Total number of returned answers}}$$

$$\text{IRecall} = \frac{\text{number of correct answerrs}}{\text{Total number of expected answers}}$$

F-score (also known as F-measure) which is a weighted harmonic mean of precision and recall is used to evaluate the quality of an answer.

$$\text{IF-score} = \frac{(\beta^2 + 1) \times \text{IPrecision} \times \text{IRecall}}{\beta^2 \times \text{IPrecision} + \text{IRecall}}$$

Here, β is the relative weight of recall versus precision. The instance F-score used to evaluate list questions gives equal weight to precision and recall [1]. So β = 1 is used:

$$\text{IF-score}_{\beta=1} = \frac{2 \times \text{IPrecision} \times \text{IRecall}}{\text{IPrecision} + \text{IRecall}}$$

The F-score of an entire run (a set of question series) is the average instance F-score over all the questions in the run, which is used to evaluate a run. Therefore:

$$\text{F-score} = \frac{\sum \text{IF-score}}{N}$$

Where N refers to the number of questions in the run. Similarly, the precision and recall of a run is computed as:

$$\text{Precision} = \frac{\sum \text{IPrecision}}{N} \qquad \text{Recall} = \frac{\sum \text{IRecall}}{N}$$

As an Experimental setup, the data from tourism experts is used to evaluate our approach as well as the whole system. 50 list questions that have been collected from different tour and travel web pages are distributed for experts and 30 of them are selected for having a complete answer and used for testing our approach. These questions are on the focus of the name entities shown in figure 8.

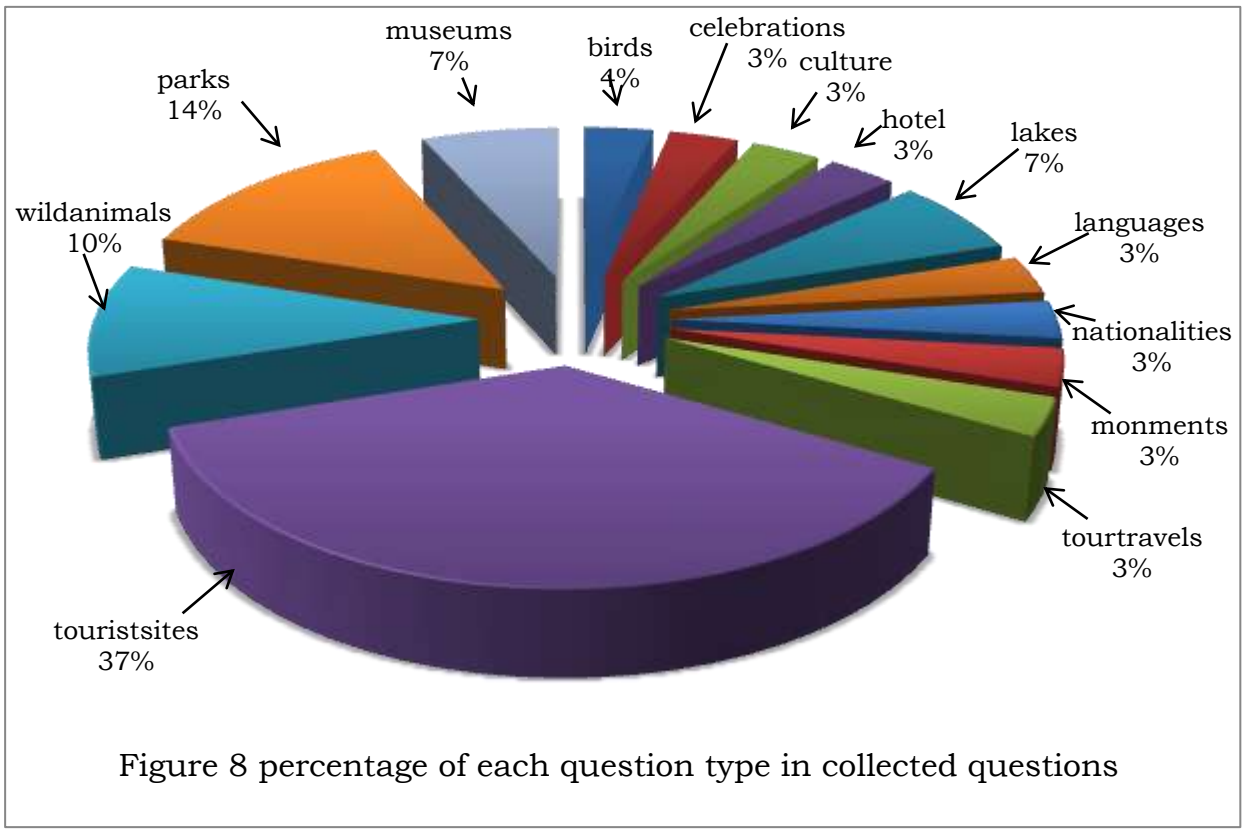


Figure 8 percentage of each question type in collected questions

3.2.2. COMPONENT EVALUATION

In this section, the performance of each component, namely Answer Type Recognition, Document Retrieval, Candidate Answer Extraction, Evaluation of Co-occurrence information extraction and Candidate answer selection module are evaluated. For this to succeed, different measures have been used to evaluate each these components.

3.2.2.1. EVALUATION OF ANSWER TYPE RECOGNITION

Table 4 shows the number of questions that are correctly classified using the Answer Type Recognition techniques. The accuracy of the module is defined as the number of correctly classified questions over the total number of questions.

Questions	Incorrect	Correct	Accuracy
30	0	30	1.0

Table 4. Answer type recognition module evaluation result

This indicates that the Answer Type Recognition module works well in classifying the questions into the defined respective answer types.

3.2.2.2. EVALUATION OF DOCUMENT RETRIEVAL

This section evaluates the document retrieval module. Among the retrieved documents, eight of them are found to contain relevant answers through experiment in this work. Therefore, these eight documents are used to extract co-occurrence information and evaluating the documents based on co-occurrence information. Using the list of correct answers to the questions, the retrieved documents from the corpora are evaluated.

Figure 9 illustrates the number of distinct answer instances found in different documents retrieved from the corpus.

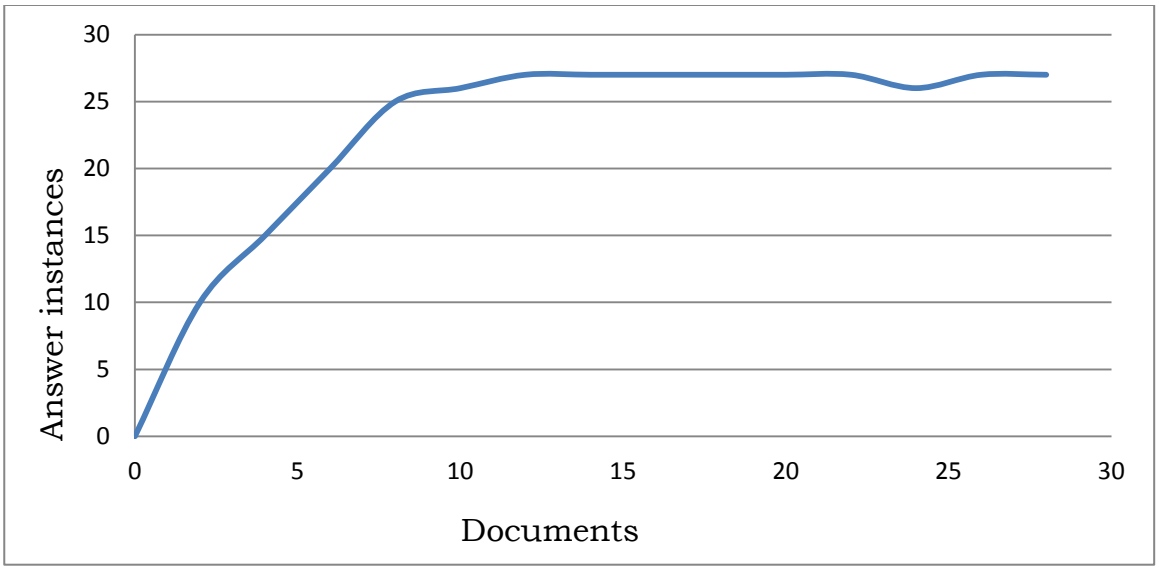


Figure 9 Number of answer instances in Documents

In figure 9 above, at document size eight, the corpus text contains more instances of answers and performs better than documents at any larger size.

As a definition, the answers of closed domain questions are found in few documents of related subject. Therefore, if at list half of the correct set of answers is contained in the retrieved document, then the document is said to be relevant. The base for the evaluation of this module is the retrieved eight documents and the answers extracted for each question. The system provides answers that are found among the retrieved eight documents. If half of the candidate answers from the retrived documents are in the expected answers list for each question set, the documents that are retrieved are said to be relevant.

As shown in table 5 below, first the questions are grouped based on the answer types since all of the questions are classified on their correct answer type in the former module. Second, the number of questions for each answer type and the total number of documents retrieved is counted given that the number of documents retrieved varies when the number of question for different answer types vary. Third, the correct and wrong answer candidates extracted from the documents and the expected answers will be count and compared to each other. Finally, if the number of

correct answers is more than half of the number of expected answers, then the set of documents is Relevant. Otherwise, it is relevant.

Document set per answer type	No of questions	Documents retrieved	Correct answers	Expected answers	Relevant
Tourist sites	11	88	51	133	No
Museums	2	16	20	21	Yes
Parks	4	32	17	36	No
Lakes	2	16	6	25	No
Celebrations	1	8	11	11	Yes
Birds	1	8	16	16	Yes
Wild animals	3	24	32	40	Yes
Culture	1	8	5	12	No
Hotel	1	8	15	15	Yes
Tour & travels	1	8	10	10	Yes
Monuments	1	8	13	13	Yes
Nationalities	1	8	16	60	No
Languages	1	8	11	80	No
Total	30	240	224	194	

Table 5. Document retrieval module evaluation result based on question type

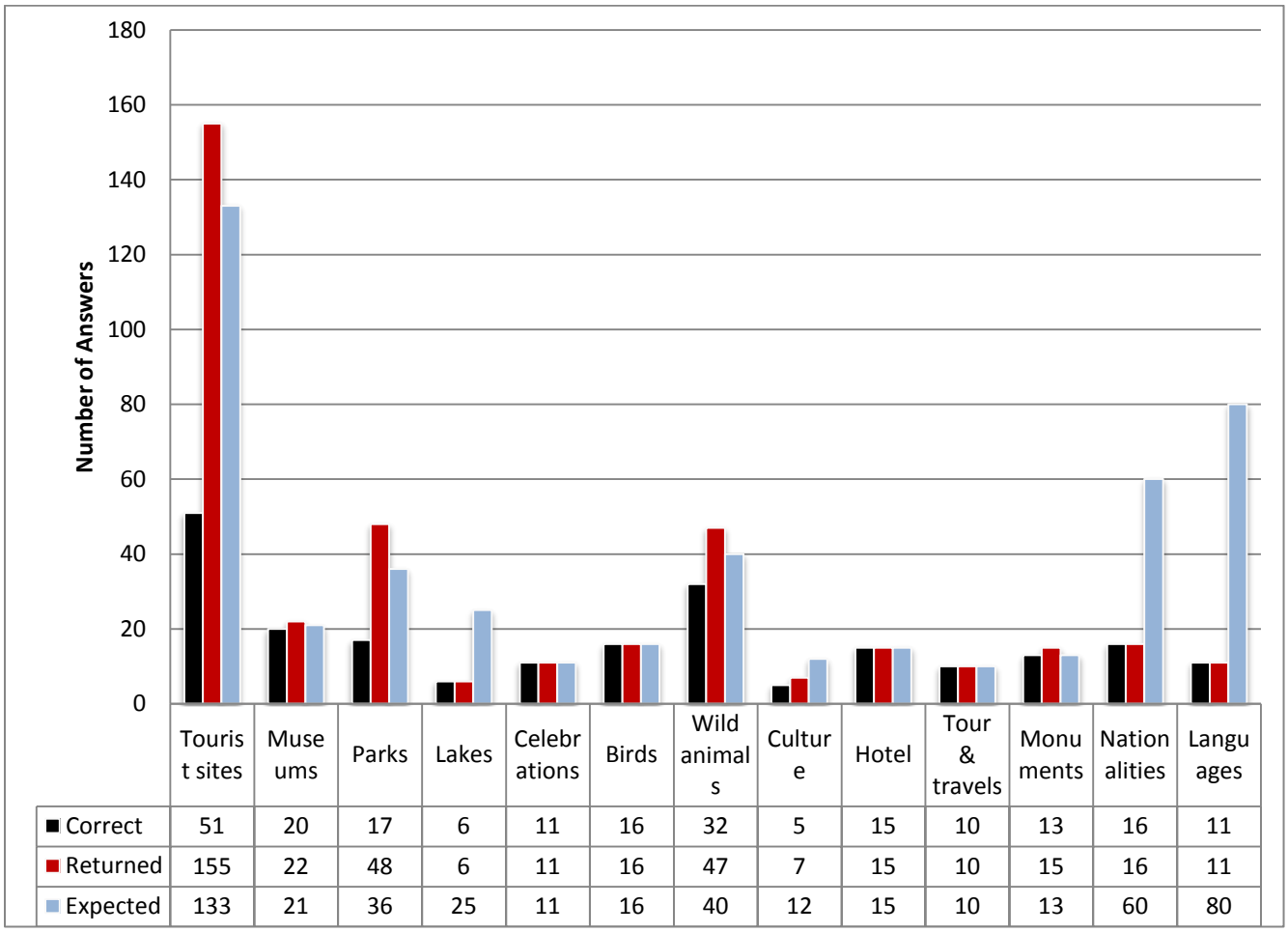
Having results from table 5, tourist sites, parks, lakes, culture, nationalities and languages related documents (46%) are not relevant for the answer types identified.

3.2.2.3. EVALUATION OF CANDIDATE ANSWER EXTRACTION

In this section, the Candidate Answer Extraction module will be assessed. The evaluation used the estimated threshold, eight documents, where the candidate answers are extracted. The extracted candidate answers are compared with the expected answers.

Figure 10 shows the number of correct instances, returned instances and expected instances for all the questions collected and for each question types.

Using the information in Table 6, the overall recall of the candidate terms for this module can be defined as the recall of a list consisting of all the initial candidate answers for all questions. The overall recall of this module, which is computed as the number of correct instances over the number of expected answers is 0.47 (223/472).



	Tourist sites	Museums	Parks	Lakes	Celebrations	Birds	Wild animals	Culture	Hotel	Tour & travels	Monuments	Nationalities	Languages
Correct	51	20	17	6	11	16	32	5	15	10	13	16	11
Returned	155	22	48	6	11	16	47	7	15	10	15	16	11
Expected	133	21	36	25	11	16	40	12	15	10	13	60	80

Figure 10 Evaluation of candidate answer extraction module

Having the results from figure 10 the total number of returned answers, correct answers and expected answers of the thirty questions provided for the module is shown below in table 6.

	Correct	Returned	Expected
Total	223	339	472

Table 6. Total candidate answers extracted by the system

As discussed above, the module returned 47% of the expected answers. The precision of the module, which is computed as number of correct instances over number of returned (extracted) answers is 0.66 (223/339). The precision show that 66% of the extracted answers are correct. Having 47% recall and 66% precision, the f-score of the module is 0.55. This shows that the module has 55% accuracy calculated as an average of the precision and recall.

3.2.2.4. EVALUATION OF CO-OCCURRENCE INFORMATION EXTRACTION AND CANDIDATE ANSWER SELECTION MODULE

As discussed in 3.1, co-occurrence information extraction module computes co-occurrence information for all possible term-to-term pairs in each sentence that are found in the documents retrieved as relevant. The information about two the terms is calculated, first by fragmenting a document in to sentence using '::' as a sentence end indicator. Next, the module will go through the sentence to count the number of sentences that contain the two candidate terms ($term_i$, $term_j$) together, contain only one of the terms or contain none of the terms in the sentence. In doing so, the module managed go through all the sentences and collected all information accurately, since the algorism is easy and efficient.

Using the co-occurrence information about all the extracted terms, candidate answers are selected based on classification technique applied.

Candidate Answer Selection module classified documents Relevant and Irrelevant based on their WMI. In evaluating this module, answers found in the relevant class are compared with expected answers. For each class, Relevant and Irrelevant,

number of answers that are classified correctly and wrongly are counted in table 7 below.

Answer types	Relevant class		Irrelevant class		Expected answers
	Correct	Wrong	Wrong	Correct	
Tourist sites	51	104	9	62	133
Museums	20	2	0	0	21
Parks	17	31	7	17	36
Lakes	6	0	0	0	25
Celebrations	11	0	0	0	11
Birds	16	0	0	0	16
Wild animals	32	15	4	0	40
Culture	5	2	0	0	12
Hotel	15	0	7	0	15
Tour & travels	10	0	0	0	10
Monuments	13	2	0	0	13
Nationalities	16	0	6	0	60
Languages	11	0	0	0	80
Total	223	156	33	79	194

Table 7. Candidate answers selection module result

As seen in table 7, the total number of candidate answers classified as relevant are 389 (233+156). Among these 60% of them are classified correctly and the rest 40% was supposed to be classified in the irrelevant class. The class or irrelevant contains 112 (33+79) terms and 71% of the terms are correctly classified as irrelevant. The rest 29% are correct answers, which were supposed to be in the relevant class.

This module classified 61% or 302 (223+79) terms in their correct class that they belong to, and 39% or 189 (156+33) terms in the wrong class that they should not be contained.

Generally, table 8 summarizes the performance of the module as follows:

Module	Performance
Answer type recognition	1.0
Document retrieval	0.54
Candidate answer extraction	0.55
Co-occurrence information extraction	1.0
Candidate answer selection	0.61

Table 8. System performance per module

As table 8 shows, the document retrieval and candidate answer extraction modules have relatively low performance and possibly need to be improved. In this evaluation, the researcher observed that the reason for the low performance of the Candidate Answer Extraction module is the document retrieval module. If the relevant documents are not retrieved, the candidate extraction module will not be able to extract candidate answers since the documents do not contain the right answers. The candidate extraction module is evaluated based on the expected answers and if the retrieved documents is not relevant, this module will have a problem in extracting the correct candidates answers.

Having the above case, the document retrieval module is the lowest performing module and it is the cause for other modules to perform low.

3.2.3. TEST RESULT AND ANALYSIS

Based on the measurement schema and the set, the collected questions are asked to types is shown in the following figure.

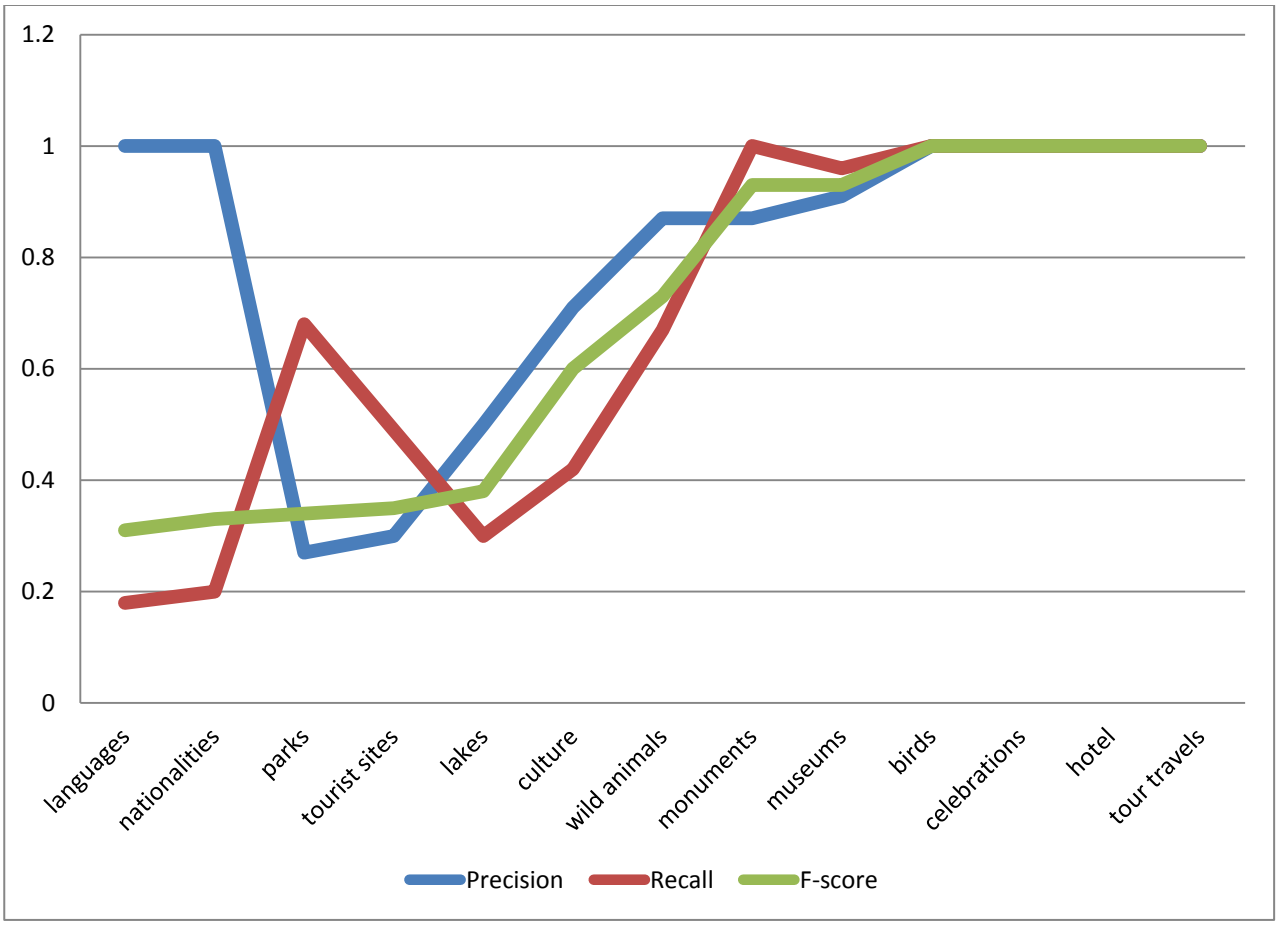


Figure 11 Precision, Recall and F-score of Each Question Type

Figure 11 shows that in bird, celebration, hotel, museums, monuments and tour travels questions' precision, recall and F-score increased at the same time. In the case of languages and nationalities questions, the precision is very high but the recall is very low because the number of expected answers for this questions is very large in the case of Ethiopian tourism.

Tourist sites, lakes and culture related questions contain the average evaluation results and Tourist sites questions dominated (37%) the overall evaluation since they contain the higher portion of the questions collected.

If we look at a one of the tourist site related questions " በባህር ዳርና አካባቢው የሚገኙ መስህቦችን ጥቀስ " it has an F-score of 40%. The answers returned are the following,

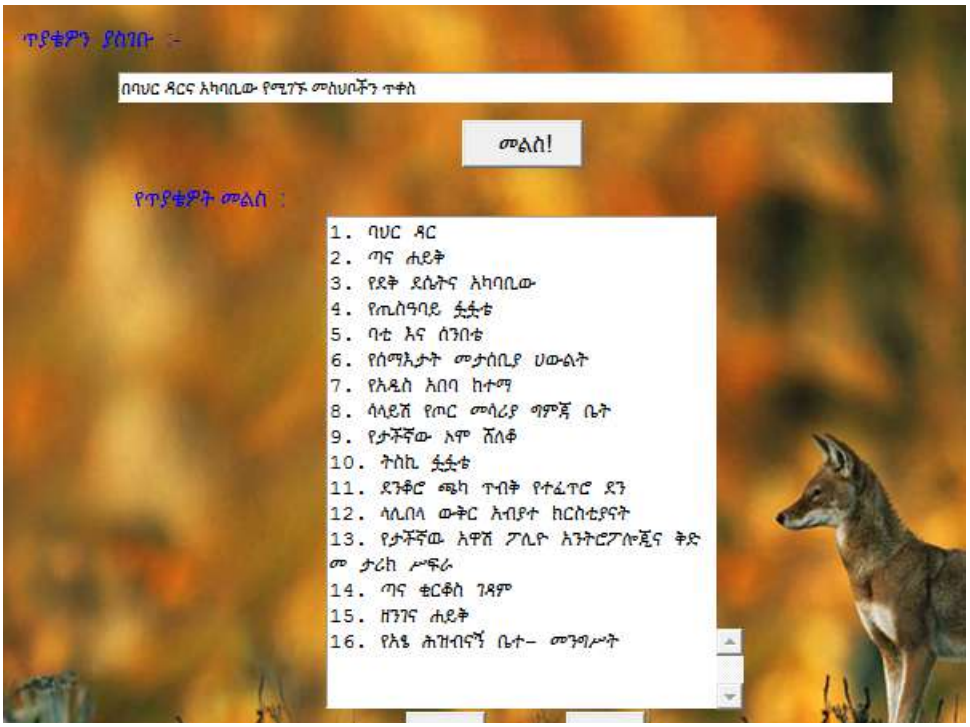


Figure 12 Answer result for Q1

In the answers above only four of them are correct, and according to the tourist guides information there are about 4 tourist attraction site in Baher Dar. This lead to an individual precision of 25% and individual recall of 100% since all attraction sites are listed in the listed answer.

From this question answer set, we can see that unwanted answers are in the list even if all the correct answers also exist in it. The reason for this problem is the co-occurrence module. This module is developed by using sentences because answers co-occur within the sentences of the documents related to the answer type and the question, but the researcher experiment 'what if it was in a paragraph' scenario for this question and results found help in having a higher weighted mutual information which will eliminate other unnecessary answers found in the list. The problem here is that this scenario works only for this question.

Museum related questions have an average F-Score of 93%. The list of answers provided for the question may not be correct all the time. For instance if we see some

examples, for the question “በአዲስ አበባ የሚገኙ ሙዝየሞችን ዘርዝር ” the system provides an answer shown in figure 13.

Figure 13 Answer result for Q2

When analyzing the above list answers, only answer number 5 is wrong, but there are 13 museums in Addis Ababa and this will decrease the recall of the system. The reason behind the incompleteness of the answer is that numerous documents on the area of museums are unavailable. In the case of Addis Ababa museums, there is only one document talking about museums.

Another question that have faced a problem is “ በኢትዮጵያ የሚገኙ እፅዋትን ጥቀስ ” and it returns a result shown below.

Figure 14 Answer result for Q3

This question returned a message showing that there is no answer for the question, and this is because the corpus does not contain a document that talk about plants.

The reason for the researcher to use overall precision and overall recall rather than precision and recall is that in overall precision and overall recall, all instances have equal weight while in precision and recall, each question is given equal weight and the weight is further divided into the answer instances of the question. Therefore, instances in questions with fewer answers have more impact than those in questions with more answers.

Overall precision can be computed as the number of correct instances over the number of returned answers. The overall precision of the overall system is 0.61. Knowing the answer-based precision can be useful to compare the different techniques used for the modules in the system.

Having the expected number of answers for a question from experts, it is possible to define the overall recall of the system. It can be computed as the number of correct instances over the number of expected answers. Therefore the overall recall of the system is 0.636.

Since the F-score is a harmonic mean of precision and recall and the harmonic mean tends strongly toward the lowest element. F-measure is used to measure the effectiveness of the question answering system designed. For this system, the integrated overall individual precision (IPrecision), Recall (IRecall) comes up with an F-score of 0.575.

3.2.4. FINDINGS AND CHALLENGES

The results indicate that building QA can be developed with a good accuracy. It is also indicated that tourism questions can be answered from Amharic document collection using QA. The study showed that we could come up with architecture for developing list QA systems.

However, there are several challenges which limits to get the optimum performance expected.

The first challenge comes from the document retrieval system. The retrieval was not retrieving the needed documents whenever the question is asked and it was almost impossible to limit the retrieved documents using a threshold. However, from the experiment it is found that mostly the needed documents are among the first eight documents retrieved. The other challenge comes from the unavailability of electronic tourism documents as discussed in the introduction section.

CHAPTER FOUR

4. CONCLUSION AND RECOMMENDATION

This chapter summarizes the overall conclusion incorporated to answering list questions. It also presents our future directions in the area of QA systems.

4.1. CONCLUSION

The QA system attempts to select a subset of the initial candidate answers which are extracted from a number of relevant documents. To do so, the information regarding the co-occurrences of the candidates in a larger number of related documents is extracted and uses a method that attempts to group candidates that co-occur more often.

Since the closed domain QA system development is in an infant stage, this research showed that domain documents can be used to give specific answer for a list question in a specific area, tourism.

The research identified features of Amharic questions, which are a very good means to know if a question is an Amharic list question or not. The research recognized a way on how to detect answer types from a question, stapes on how to retrieve candidates, how to extract list questions from different documents, and finally build up system architecture.

This research comes up with a prototype that can provide Amharic answers for Amharic listing questions in the area of tourism. Furthermore, the research showed that instances of the answer to a list question tend to co-occur within sentences and tend to co-occur with the query terms.

Generally, our research realized the development of list QA system for Amharic language in a closed domain.

4.2. RECOMMENDATION

Even if the research is promising, there is much room for improvement. This can be seen in the answers that have problems. QA needs large memory space, processor speed and time for better performance and design. The recommendation listed below will help to improve the performance of AQA.

- Developing an IR system that can work for a small collection of documents is very essential because it will provide the documents that contain the candidate answers and this in turn will decrease the number of documents to be processed for answer extraction.
- Name entity tagger that is able to identify other name entities than only language, nationalities and locations would help the system better identify answers to more question types.
- The corpus of this research contains very low information about birds, culture and languages. Integrating these documents will also improve the systems performance.
- Another suggestion point is answer verification. Verifying the list of answers using an already collected question answer match can enhance the performance of the system. This question answer match can be updated using users relevance feedback about the answers provided.
- Question answering systems can also be developed for other domain sectors. For example, in the area of medicine, pharmacy and business can have their own question answering systems.
- Having the previous work on factoid question, this work and the ongoing work on definition question, one can integrate the three with modifications and come up with a fully integrated multi-type QA system.

REFERENCE

[1] Razmara, Majid, "Answering list and other questions," The department of computer science and software enginering, Montreal,Canada, August 2008.

[2] Mark D. Smucker, James Allan, and Blagovest Dachev, "Human Question Answering Performance using an Interactive Information Retrieval System," Center for Intelligent Information Retrieval, Massachusetts, USA, 2008.

[3] Lotfi A. Zadeh, "From search engines to question answering systems - The problems of world knowledge, relevance, deduction and precisiation," Department of EECS, University of California, Berkeley, CA, USA.

[4] Hu, Haiqing, "a study on question answering system, using integrated retrieval method," Information science and systems enginering, The university of Tokushima,, Tokushima, Japan, 2006.

[5] Radu Soricut, Eric Brill, "Automatic question answering: Beyond the Factoid," Information sciences institute, University of southern california, CA, USA.

[6] Seid Muhie, "Amharic question answering(AQA)," Department of computed science, Addis Ababa University, Addis Ababa, 2009.

[7] Muthukrishnan Ramprasath, Shanmugasundaram Hariharan, "A survey on question answering system," *International journal of research and reviews in information sciences (IJRRIS),* vol. 2 No.1, 2012.

[8] L. Hirschman, r. Gaizauskas, "Natural language question answering: the view from here, natural lauguage engineering," cambridge university press, 275-300, 2001.

[9] Green, W.,Chomsky, Laugherty, C.K., "BASKETBALL: An authomatic question answerer,," *Proceedings of the western joint computer conference,* pp. 219-224, 1961.

[10] Weizenbaum, J. Eliza, "A computer program for the study of natural language communication between man and machine," *commun, ACM,* vol. 9(1), pp. 36-45, 1966.

[11] Woods, W.A., "Progress in natural language understanding: An application to lunar geology," *AFIPS Conference proceedings,* vol. 42, pp. 411-450, 1973.

[12] Lehnert, W., "A conceptual theory of question answering," *In proceedings 5th international joint conference on artificial intelligence,* pp. 158-164, 1977.

[13] Kintsch, W., "Comprehension: A paradigm for cognition," Cambridge university, Cambridge, 1998.

[14] Hirschman, L., Light, M., Berck, E. & Burger, j., "Deep Read: A reading comprehension system," *In proceedings 37th annual meeting of the association for computational linguistics,* pp. 325-332, 1999.

[15] Omid Moradian Nasab, Amin Mozhgani, Javad Nouri, "A special domain question answering system for persian," Phoenix-company NLP research lab, Narmak, Tehran, Iran.

[16] Gosse Bouma, Ismail Fahmi, Jori Mur, "Relation Extraction for open and closed Domain question answering," The Netherlands, 2010.

[17] Hai Doan-Nguyen, Leila Kosseim, "Improving the precision of a closed-domain question answering system with semantic informaton," CLaC laboratory, concordia university, Montreal, Quebec, Canada, 2004.

[18] Liljenback, Erik, "CONTEXTQA:Experiments in interactive restricteddomain question answering," Faculty if san diego state university, San diago, 2007.

[19] Hirpa, Amanuel, "Probablistic information retrieval system for Amharic Language," department of information scince, addis abeba university, Addis Ababa, Ethiopia, 2012.

[20] Christopher D. manning, Prabhakar Raghavan, Hinrich S., Introduction to Information Retrieval, New York, USA: Cambridge university press, 2008.

[21] Maron, M. E. & Kuhns, J.L., "On relevance, probabilistic indexing and information retrieval," *Journal of the ACm,* vol. 7, pp. 216-244, 1960.

[22] Broglio, J.,Callan, J.P., Croft, W.B., Nachbar, D. W., "Document retrieval and routing using the INQUERY system," *in harman Nist special publication,* Vols.

500-236, pp. 29-38, 1995.

[23] Douglas A., Jerry H., Jhon B., David I., Megumi K., Andy K., David M., Karen M., Mabry T., "SRI International FASTUS system: MUC-6 test results and analysis,," in *In processing sixth messasge understanding conferrence (MUC-6)*, Columbia, 1995.

[24] Leslau, Wolf, "An amharic reference grammer," Dept. of Near Eastern ans African Languages, california university, Los angeles.

[25] Harris, Zelling, "The structure of lunguage, chapter Distributional Structure," Prentice-Hall, 1954, pp. 33-49.

[26] Rodriguez, A.,M Egenhofer, R. Rugg, "Assessing Semantic Similarity Among Geospatial Entity Class Definitions," *Interoperating Geographic information systems,* pp. 189-202, 1999.

[27] H., Tewodros, "Amharic text retrieval: An experiment using latent semantic indexing (LSI) with singular value decomposition(SVD)," Department of information science, Addis Ababa University, Addis Abeba, Ethiopia, 2003.

[28] Ricard Baeza-Yates, Berthier Ribero-Neto, Modern Information Retrieval, New York: ACM Press, 1999.

[29] Jiangping Chen, Anne R. Diekema, Mery D. Taffet, Nancy McCracken, "Question answering: CNLP at the TREC-10 Question answering track," center for natural launguage processing, syracuse University, Syracuse, NY, 2010.

[30] kosseim, Majid Razmara and leila, "Answering list questions using co-occurrence and clustering," Department of computer science and software engineering, Concordia University, Montreal, Canada, 2008.

APPENDIXES

Appendix 1: Selected questions from the web, used for evaluation.

1 በኢትዮጵያ የሚገኙ ሙዝየሞችን ጥቀስ
2 በአዲስ አበባ የሚገኙ ሙዝየሞችን ዘርዝር
3 በአማራ ክልል የሚገኙ መስህቦች የትኞቹ ናቸው
4 የኢትዮጵያ ዋና የቱሪስት መስህብ ሃብቶች እነማን ናቸው
5 በኢትዮጵያ የሚገኙ ብሔራዊ ፓርኮችን ዘርዝር
6 በጎጃም መንገድ ተጎብኚ ቦታዎችን ዘርዝር
7 ተዘውትረው የሚጎበኙ መስህቦችን ጥቀስ
8 በባህር ዳርና አካባቢው የሚገኙ መስህቦችን ጥቀስ
9 በአማራ ክልል የሚገኙ ፓርኮች የትኞቹ ናቸው
10 በአለም ቅርስነት የተመዘገቡ መስህቦችን ጥቀስ
11 በአየር መንገድ አቅራቢያ የሚገኙ መስህቦች የትኞቹ ናቸው
12 በአፋር ክልል የሚገኙ ፓርኮች የትኞቹ ናቸው
13 ዝሆን የሚገኝባቸውን ቦታዎች ዘርዝር
14 ላሊበላ የሚገኙ መስህቦችን ጥቀስ
15 በኦሮሚያና አካባቢው የሚገኙ መስህቦችን ጥቀስ?
16 በኢትዮጵያ የሚገኙ እፅዋትን ጥቀስ
17 በአዲስ አበባ የሚገኙ ፓርኮች የትኞቹ ናቸው
18 በኢትዮጵያ የሚገኙ ሃይቆችን ዘርዝር
19 በኢትዮጵያ የሚገኙ ወንዞችን ጥቀስ
20 በአዲስ አበባ የሚከበሩ በዓላት እነማን ናቸው
21 ብርቅዬ አዋፋትን ዘርዝር
22 በአማራ ክልል የሚገኙ የዱር እንስሳት የትኞቹ ናቸው
23 በኢትዮጵያ የሚገኙ ዋና ዋና የዱር እንስሳትን ዘርዝር
24 በአዲስ አበባ የሚገኙ የዱር አራዊትን ጥቀስ
25 በኢትዮጵያ የሚገኙ ባህላዊ ምግቦችን ጥቀስ
26 በአዲስ አበባ የሚገኙ ዋና ዋና ሆቴሎች እነማን ናቸው
27 ታዋቂ አስጎብኚዎች እነማን ናቸው
28 በአዲስ አበባ የሚገኙ ሐውልቶችን ዘርዝር
29 በኢትዮጵያ የሚገኙ ብሄር ብሄረሰቦችን ጥቀስ
30 በኢትዮጵያ የሚገኙ ቋንቋዎችን ጥቀስ

Appendix 2: Amharic Question Answering System python code

```
import os
import sys
import math
import codecs
import string
import tkinter
import tkinter.messagebox

#---------- lexical pattern ----------
def anstype(quer):
    query=quer.split()
    qtype=''
    xy=[]
    qlist=['ዘርዝር','ጥቀስ','እነማን','የትኞቹ','ማንማን','ማናማን','እነማንን','የቶቹ','በየትኞቹ','ጥቀስ','የት']
    birds = ['ወፍ','ወፎች','አእዋፍ','አእዋፋት','አዋፋት','አእዋፋቶች']
    celebrations = ['በአላት','በአል','ክብረበአላት','ክብረበአል','በአሎች','ክብረበአሎችን']
    culture = ['አልባሳት','ልብሶች','መጠጦች','ምግብ','ምግቦች','ሙዚቃ','ጭፈራ','ሙዚቃዎች','ጭፈራዎች']
    hotel = ['ሆቴሎች','ማደሪያዎች','መዝናኛዎች','መመገቢያዎች','መብያዎች']
    lakes=['ሀይቅ','ወንዝ','ወንዞች','ሀይቆች']
    languages = ['ቋንቋ','ቋንቋዎችን','ቋንቋዎች','መግባቢያ','መግባቢያዎች']
    museums = ['ሙዚየም','ሙዚየሞች','ሙዝየም','ሙዝየሞች']
    nationalities = ['ህዝቦች','ብሄርብሔረሰቦች','ብሄሮች','ብሄረሰቦች']
    organization = ['ድርጅቶ','ማህበራት','ማእከላት']
    park = ['ፓርኮች','ፓርክ']
    tourtravels = ['አስጎብኚዎች','የጉዞ-ወኪሎች']
    touristsites =['መዳረሻዎች','መስህቦች','መስህብ','ሀብቶች','ቅርስ','ሜዳ','ቦታዎች','ቅርሶች','ከተሞች','ሜዳዎች','ገዳም','ገዳማት']
    monments = ['ሀውልቶችን','ሀውልቶች','ሀውልት']
    wildanimals = ['እንስሳት','አራዊት','የዱርእንስሳት','የዱርአራዊት','የዱር', 'እንስሳት','የዱር', 'አራዊት']

    for i in range (0,len (query)):
        if qlist.__contains__(query[i]):
            for i in range (0,len (query)):
                if(query[i].endswith('ን')):
                    query[i]=query[i].replace('ን','')
            for j in range (0,len(query)):
                if birds.__contains__(query[j]):
                    qtype = 'birds'
                elif culture.__contains__(query[j]):
                    qtype = 'culture'
                elif celebrations.__contains__(query[j]):
                    qtype = 'celebrations'
                elif hotel.__contains__(query[j]):
                    qtype = 'hotel'
                elif lakes.__contains__(query[j]):
                    qtype = 'lakes'
                elif languages.__contains__(query[j]):
                    qtype = 'languages'
```

```
            elif museums.__contains__(query[j]):
                qtype = 'museums'
            elif nationalities.__contains__(query[j]):
                qtype = 'nationalities'
            elif organization.__contains__(query[j]):
                qtype = 'organization'
            elif park.__contains__(query[j]):
                qtype = 'park'
            elif tourtravels.__contains__(query[j]):
                qtype = 'tourtravels'
            elif touristsites.__contains__(query[j]):
                qtype = 'touristsites'
            elif monments.__contains__(query[j]):
                qtype = 'monments'
            elif wildanimals.__contains__(query[j]):
                qtype = 'wildanimals'
    return str(qtype)

def readIndexData(UserQuery):
    flag=0
    L1,L2,L01,L02=[],[],[],[]
    canddoc=[]
    N = int(len([fiLe for fiLe in os.listdir("./corpus") if os.path.isfile(os.path.join("./corpus", fiLe))]))
    a=codecs.open("vocablaryfile.txt", 'r', encoding = 'utf-8')
    b=a.readlines()
    for j in b:
        c=j.split()
        f=c[0],int(c[1])
        L01.append(f)
    for i in L01:
        L1.append((i[0],i[1]))
    a.close()
    n=codecs.open("postingFile.txt", 'r', encoding = 'utf-8')
    m=n.readlines()
    for i in m:
        s=i.split()
        f2=int(s[0]),int(s[1])
        L02.append(f2)
    for i in L02:
        L2.append((i[0],i[1]))
    n.close()
    L3,L6,vector=[],[],[]
    d,h,g,a,m,c=0,0,0,0,0,0
    while c < len(L1):
            L=[]
            L30=[]
            for h in range(L1[d][1]):
                L.append((L1[m][0], L2[a][0]))
                a = a + 1
                m = m + 1
```

```
        d = d + 1
        c=c+1
        vector = vector + L
querylist=[]
provector=[]
userI=normal(UserQuery)
userInput=userI.split()
sw=codecs.open("stopw.txt", 'r', encoding = 'utf-8')
stpw=sw.read()
for i in userInput:
    i = punctuationremove(i)
    if i=="\u1219\u12dd\u12e8\u121d":
        i=="\u1219\u12da\u12e8\u121d"
    if i not in stpw.split():
        querylist.append(sades(suffix(preffix(normal(i)))))
for n in range(len(querylist)):
    for j in range(1, N+1):
        for t in range(len(vector)):
            if j==vector[t][1]:
                if querylist[n]==vector[t][0]:
                    provector.append((vector[t][0],j,1))
                else:
                    pass
count=0
forweight=[]
for i in range(len(querylist)):
    count=0
    for j in range(len(provector)):
        if querylist[i]==provector[j][0]:
            count=count+1
    forweight.append((querylist[i],count))
weight=[]
for i in range(len(forweight)):
    sim1 = float(((N)-(forweight[i][1])) + 0.5)
    sim2=float((forweight[i][1]) + 0.5)
    sim=float(sim1/sim2)
    simm=math.log((sim),10)
    weight.append(simm)
count=1
rank=[]
count=1
score=[]
score2=[]
for i in range(len(querylist)):
    rank.append((querylist[i],count))
    count=count+1
for i in range(len(querylist)):
    for j in range(len(provector)):
        if querylist[i]==provector[j][0]:
            for m in range(1,(N+1)):
```

```
                if provector[j][1]==m:
                    score.append((m,rank[i][1]))
                    score2=sorted(score)

                else:
                    pass
add=0
scoree=[]

for k in range(len(weight)):
    for j in range(1,(N+1)):
        for i in range(len(score2)):
            if score2[i][0]==j:
                add=(add)+(weight[(score2[i][1])-1])
                scoree.append((score2[i][0],add))
        add=0
    break
if len(userInput)>0 and len(userInput)<10:
    if len(userInput)>=1 and len(userInput)<10:
        finalRes={}
        finalRes=dict(scoree)
        Ranking=[]
        Ranking=list(finalRes.keys())
        f1=open("documentsID.txt", 'r')
        f2=f1.readlines()
        DocL=[]
        for i in f2:
            record=i.split()
            DocL.append(record)
        DocN=1
        DocL2=[]
        pathList=[]
        for i in DocL:
            DocL2.append((DocN,i[0]))
            DocN=DocN+1
        docDict=dict(DocL2)
        chekpoint=0
        for i in range(0,len(Ranking)):
            if finalRes[Ranking[i]] != 0:
                chekpoint=1
        for i in range(len(Ranking)):
            for j in range(i+1,len(Ranking)):
                if finalRes[Ranking[i]]<finalRes[Ranking[j]]:
                    rank=Ranking[i]
                    Ranking[i]=Ranking[j]
                    Ranking[j]=rank
        if chekpoint==1:
            for i in range(0,len(Ranking)):
                #if finalRes[Ranking[i]]>=0:
                    pathList.append(docDict[Ranking[i]])
```

```
                    f=codecs.open(docDict[Ranking[i]],'r', encoding="utf-8")
                    f.close()
                    if i < 9:
                        canddoc.append(docDict[Ranking[i]])

    candans(canddoc,anstype(userI))

def candans(canddoc,atype):
    print (canddoc)
    print (atype)
    cands=[]
    senten=[]
    y=[]
    for i in range(0,len(canddoc)):
        showfile = codecs.open('./'+ canddoc[i],'r',encoding="utf-8")
        q=showfile.read()
        s=q.split('::')
        for h in s:
            senten.append(h)
        if len(cands) > 20:
            break
        else:
            y=q.split()
            if atype in
['birds','celebrations','culture','monments','hotel','lakes','languages','museums','nationalities','
organization','park','tourtravels','touristsites','wildanimals']:
                xx=codecs.open( './gazetteer/' + atype + '.txt','r',encoding='utf-8')
                x=xx.readlines()
                for j in range (0,len(y)):
                    for k in range (0,len(x)):
                        x[k]=x[k].rstrip('\n')
                        x[k]=x[k].rstrip('\r')
                        zz=x[k].split()
                        for kk in zz:
                            if y[j] in kk:
                                if x[k] not in cands:
                                    cands.append(x[k])

#-----------------------------------------------
# Classification                              |
#-----------------------------------------------

# candidate term to term occurance
    coo=[]
    for i in range(0,len(cands)):
        for j in range(0,len(cands)):
            if cands[i] != cands[j]:
                if (cands[j] + ':' + cands[i]) not in coo:
                    for k in range(0,len(senten)):
                        if senten[k] != '':
```

```
                    if cands[i] in senten[k] and cands[j] in senten[k]:
                        coo.append({cands[i] + ':' + cands[j]:'1'})
                    elif cands[i] in senten[k] and cands[j] not in senten[k]:
                        coo.append({cands[i] + ':' + cands[j]:'2'})
                    elif cands[i] not in senten[k] and cands[j] in senten[k]:
                        coo.append({cands[i] + ':' + cands[j]:'3'})

# contenjency table; term(i,j)= 1(i,j), 2(i,^j),  3(^i,j), 0(^i,^j)

    y={}
    for i in range(0,len(coo)):
        yy=list(coo[i].keys())
        xx=list(coo[i].values())
        if yy[0] not in y:
            y[yy[0]]=[0, 0, 0]
        if  xx[0] == '1':
            y[yy[0]][0]+=1
        elif  xx == ['2']:
            y[yy[0]][1]+=1
        elif  xx == ['3']:
            y[yy[0]][2]+=1

# wehighted mutual information

    wmui={}
    logg=0
    yy= list(y.keys())
    for i in range (0,len(y)):
        for j in range(0,len(yy)):
            n = len(senten)
            c = ((y[yy[j]][0]+y[yy[j]][1])*(y[yy[j]][0]+y[yy[j]][2]))
            #if c != 0:
            if c == 0:
                c=0.0001
            logg= float((n*y[yy[j]][0])/c)
            #if logg!=0:
            if logg == 0:
                logg=0.0001
            x = math.log(logg,2)
            wmi= y[yy[j]][0] * x
            if wmi != []:
                #if wmi >= 0:
                    wmui[yy[j]]=wmi

# Cumulative similarity

    comfr={}
    zz=list(wmui.keys())
    w=[]
    for i in zz:
```

```
        x=i.split(':')
        for j in x:
            w.append(j)
    for i in range(0,len(w)):
        comfr[w[i]]=0
        for j in zz:
            if w[i] in j:
                comfr[w[i]] += wmui[j]

# Ranking based on cumulative frequency

    xc=list(comfr.keys())
    xv=list(comfr.values())
    xv.sort()
    xv.reverse()
    sortd=[]
    if comfr:
        pass
    else:
        for i in range(0,len(cands)):
            if i < 11:
                sortd.append(cands[i])
    for i in range(0,len(xv)):
        #if xv[i] > 0:
            for k in range(0,len(xc)):
                if comfr[xc[k]]==xv[i] and xc[k] not in sortd :
                        sortd.append(xc[k])

# Print to doc

    b=codecs.open('candidates.txt','w',encoding='utf-8')
    for i in range(0,len(sortd)):
        if i <= 15:
            b.write(str(i+1)+'. '+sortd[i]+'\n')

# Interface

def Main():
    tk = tkinter.Tk()
    tk.title('የቱሪዝም መረጃ መጠየቂያ')
    tk.resizable(width='FALSE', height='FALSE')

    def quit():
        tkinter.messagebox.showinfo("መዝጊያ","እናመሰግናለን !")
        tk.destroy()
    def Inp():
        x = entert.get()
        if x=='':
            ontert.insert('end', "\n ይቅርታ! ምንም ጥያቄ አልጠየቁም ።"+'\n'+"  እንደገና የሞክሩ! ")
```

```
        else:
            readIndexData(x)
            with codecs.open("candidates.txt", 'r',encoding="utf-8") as inp_file:
                yx=inp_file.read()
                if yx=='':
                    ontert.insert('end', "\n ይቅርታ! ለጠየቁት ጥያቄ መልስ አልተገኘም።" + '\n' + "  እንደገና የሞክሩ! ")
                ontert.insert('end', yx)
    def agn():
        entert.delete(0, 'end')
        ontert.delete(1.0, 'end')

    bar = tkinter.Menu(tk)
    fileMenu = tkinter.Menu(bar,tearoff=0)
    fileMenu.add_command(label=" ክፈት ")
    fileMenu.add_command(label=" ማተም ")
    fileMenu.add_command(label=" ሴቭ ")
    fileMenu.add_command(label=" ዝጋ ",command=quit)
    bar.add_cascade(label=" ፋይል ",menu=fileMenu)
    fileMenu2 = tkinter.Menu(bar,tearoff=0)
    fileMenu2.add_command(label=" እርዳታ ")
    fileMenu2.add_command(label=" ስለ... ")
    bar.add_cascade(label=" መረጃ ",menu=fileMenu2)
    tk.config(menu=bar)

    bgimage=tkinter.PhotoImage(file="bgrd.pgm")
    w=bgimage.width()
    h=bgimage.height()
    tk.geometry("%dx%d+0+0" % (w,h))

    answ = tkinter.StringVar()

    painel = tkinter.Canvas(width=w, height=h)
    painel.create_image(0,0, image=bgimage,anchor='nw')
    painel.pack(fill='both', expand='yes')
    painel.create_text(400,150, text="ወደ የቱሪዝም መረጃ መጠየቂያ ሲስተም እንኳን በደህና መጡ !"+'\n'+'\n'+"ስለ
ቱሪዝም ማወቅ የሚፈልጉትን ይጠይቁ !"+'\n', justify='center', fill='blue',anchor='center', font=("Helvetica",
13))
    painel.create_text(80,180, text='ጥያቄዎን  ያስገቡ  :-  ',justify='center', fill='blue',anchor='nw',
font=("Helvetica", 11))
    entert = tkinter.Entry(painel, width=80, justify='left')
    entert.place(relx=1, x=-660, y=210)
    entert.focus_set()
    ansb = tkinter.Button(painel, text="    መልስ!   ", font=("Helvetica", 11), command=Inp,
justify='center')
    ansb.place(relx=1, x=-445, y=240)
    painel.create_text(150,280, text='የጥያቄዎት መልስ  :  ',justify='center', fill='blue',anchor='nw',
font=("Helvetica", 11))

    ontert = tkinter.Text(painel, width=30, height=19)
    ontert.place(relx=1, x=-530, y=300)
```

```
s = tkinter.Scrollbar(painel, orient='vertical',command=ontert.yview)
s.place(relx=1, x=-286, y=558)
ontert.config(yscrollcommand=s.set)

buton1 = tkinter.Button(painel,text="እንደገና", font=("Helvetica", 11), command=agn)
buton1.place(relx=1, x=-480, y=610)
buton = tkinter.Button(painel,text="መዝጊያ", font=("Helvetica", 11), command=quit)
buton.place(relx=1, x=-380, y=610)

tk.mainloop()
Main()
```

DECLARATION

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

_____

BROOK ESHETU BETE

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

_____

SOLOMON TEFERRA (PhD)

June 2013