Using a Named Entity Tagger and a Syntactic Parser to improve Web-based Answer Extraction

Yasser Kamel

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

April 2004

Your file Votre référence
ISBN: 0-612-91055-5
Our file Notre référence
ISBN: 0-612-91055-5

ABSTRACT

Yasser Kamel

The amount and the quality of the available information on the web make it an interesting resource for seeking quick answers to simple questions. Question answering (QA) systems have proven to be helpful to users because they can provide accurate answers that do not require users to go through a large number of documents for an answer. However, despite the recent advances in QA research, the accuracy of the extracted answer is still an open domain that needs more investigation from the researchers to achieve a high accuracy. In this project, we implemented *Named-entity tagging (Gate-NE)* and *Grammar Parsing (Link parser*) as two different approaches to improve the extracted answer accuracy of an existing web-QA system. *TREC-8 (*200 questions) was used as a training set and a total of 1693 questions *of TREC-9, 10, and 2002* were used for the validation process. Our approach shows a 11% MRR increase, from 0.101 to 0.113, compared to the original system and a 9% increase in the number of correct answers extracted, from 276 to 300. Although the increase is not as high as we had hoped, we believe that our results are encouraging and this work should be considered a base and starting point for future work to achieve more performance enhancements.

Acknowledgments

There are many people to whom I owe this project, and I would like to take this opportunity to thank them.

First and foremost I would like to thank my supervisor, Prof. Leila Kosseim for her continuous support, encouragement throughout the many phases of this project. The various challenges during this work have been overcome with her constant support, She is acknowledged in the NLP community, and I am grateful for her insight and wisdom during our work.

I would like to express my sincere recognition to Concordia University, and to the Computer Science Department Staff, where I found all kind of support during my work.

Finally I am grateful to my family, my wife *Nehal* and my two little boys *Faddy* and *Ryan*, who encouraged me and gave me their emotional support during my work.

Table of Contents

List of figures

List of tables

Chapter 1

Introduction

While web search engines have made important strides in recent years, the problem of efficiently locating information on the web is far from solved. Question answering (QA) is one of many techniques that could be used to locate such information, and provide direct answer to user questions. However the difficulty of Natural Language Processing (NLP) had been one of the major QA challenges in the past few decades. Transforming a natural language question into an efficient query, is a challenge by itself, but in QA systems, this question should be transformed into a query and then an efficient and a satisfactory answer should be found to this question. Recently, the continuous improvement in the field of information technology and the tremendous growth in on-line information have increased the demand for more automated, robust, and efficient QA systems.

A typical QA system consists of the following four major components:

1- Question Analysis

2- Documents retrieval

3- Passage retrieval

4- Answer extraction

The goal of our project is to improve the accuracy of the answer extraction process of an already existing question-answer system called QUANTUM [Plamondon, Kosseim, 2002]. The original work used a rule-based approach to extract an answer. Although this approach yields reasonable results, it is still suffering from some drawbacks. The following two examples present some of the problems that limit the performance of the original system:

1) Wrong semantic type association

Given the following "***Where is Basque country located?***" to the original system to find an answer, the system will formulate a query and try to find a document containing one of the following reformulation:

```
1- Basque country located, LOCATION
2- Basque country is located LOCATION
```

One of the possible matches that could contain the correct answer is the following: "*Basque country is located in the north coast of Spain, close to the French border*". The original system is using a rule-based approach, and identifies a <LOCATION> expression as the preposition IN followed by a City or a Country name. In the above sentence, the LOCATION entity *"in the north coast of Spain"* will not be considered as a valid location expression by the current system, and accordingly the system will not be able to extract an answer.

2) Head of the noun phrase

Another problem with the original system was extracting an obviously wrong answer. Given the following "***Who is the current U.S. President?***" to the original system to find an answer, the system will formulate a query and try to find a document containing one of the following reformulations:

```
1- the current U.S. President is PERSON
2- PERSON is the current U.S. President
```

One of the possible matches that could contain an answer is the following:

"*the welcome letter from the current U.S. President is Boring*". The original system is using a rule-based approach, and uses latter capitalization to identify a <PERSON> expression, and accordingly the system will extract "*Boring*" as the answer, regardless of the fact that the subject of this sentence is the letter not the U.S. President.

To avoid these two main problems, we decided to use a combination of the following approaches:

- Named-Entity Tagging to solve the wrong semantic type problem
- Grammar Parsing to parse and ensure that the named entity answers are syntactically correct and the sentence's subject is the searched answer.

Named-entity tagging mainly associates a phrase or a word with its semantics. For example, "Canada" is associated with "LOCATION", "Peter" with "PERSON" and "April 3, 2004" with "DATE". Most of the named-entity taggers are trained on a tagged corpus using statistical language processing techniques. A first attempt at Named Entity tagging consists of using lists that include proper names, people, places, or any other names. However, this approach cannot work well. First of all, the lists will be very huge, million and millions of entries. Secondly, any list could be out of date as soon as it is ready to be used, especially in the case of organizations and person names. Moreover, there is still a problem of lists overlapping, such as "Washington", or "Paris" which can refer to a Person, or Location or even an organization. To avoid these problems and other issues, and to have an automated QA system, we decided to use one of the available named-entity tagging tools, which comes with GATE (General Architecture for Text Engineering). GATE is an architecture, development environment, and framework for building systems that process human language. It has been in development at the

University of Sheffield since 1995, and has been used for many research projects, including information extraction in multiple languages and media [Cunningham et al, 2002]. It is an open source system available in C++/TCL (version 1, 1995-1997) and in Java (version 2, 1999-2002). The GATE-NE module will extract all the possible answers that match a given semantic entity. This process does not and could not take into consideration the syntactical structure of the sentence, and will limit our overall result improvement. Although, we believe a NE tagger is a very useful tool for QA systems, we also believe that a QA system should not rely too heavily on NE tagging. As the number of new terms changes rapidly and the tagging process fails in many cases to associate proper names with the PERSON entity, specially in the case of non-English names. Therefore, we also have decided to use a syntactic parser, in addition to a named entity tagging tool, to filter and improve the GATE-NE system answer extraction process. The Parser takes the GATE-NE answer as input and evaluates and checks if this answer represents the head of the given phrase or not. We have selected the Link Parser tool [Temperley, Sleator, and Lafferty 1991] to be our system Parser based on Stratica's experiments and evaluation [Stratica, 2002]. The following example shows how a parser can improve our results. Given the following question ***"Who is Author of the Lord of the Rings?"***, the following sentences are considered both valid and potential candidates for the extraction process:

```
1- The author of the Lord of the Rings is <PERSON>.
2- The wife of the author of Lord of the Rings is <PERSON>.
3- <PERSON> is the author of the Lord of the Rings.
```

All person expressions will be extracted by the GATE-NE system as potential answers. However, the syntactic parser will only extract the persons in sentences 1 and 3 because the *author* is the head of the noun phrase in the subject position only in sentences 1 and 3. Therefore, the PERSON in sentence 2, *the author's wife,* will be filtered out by our system. Throughout this research we will present more examples to show how the parser's approach improves our system.

After this brief introduction, chapter two will present a literature review in the Question-Answering and Natural Language processing fields. Based on this review, we describe how GATE and the Link Parser can fulfill our project's requirements. In chapter three, we will present in detail our system architecture and approach and will give a brief overview of the TREC competition and questions that will be used as training and validating set of our experiments. In chapter four, we will explain the named entity design, implementation, and limitations of the GATE-NE system; section 4.2 will present some examples of how the system extracts the searched entities and how these extractions represent the correct answers. In chapter five, we will describe the Link Parser approach and its implementations and limitations. Section 5.3 will present how the system will be or will not be able to filter the GATE-NE system answers of the same examples that were presented in section 4.2. Chapter six will present our experimental results, system evaluation, and performance and will present some of our system limitations. Finally, in chapter seven, we will conclude our work and discuss future work
to be done.

Chapter 2

Literature Review

Question answering (QA) has become an important and widely researched area for information access because of its ability to provide succinct answers to user's questions without having him/her search through a large number of documents to find a simple answer. Since the early days of artificial intelligence in the 60's, researchers have been occupied with answering natural language questions issues [Light et al. 2001]. However, the difficulty of natural language processing (NLP) has limited the scope of QA to domain-specific expert systems. In recent years, improvements in information retrieval (IR) and NLP techniques have attracted researchers in a special class of QA systems that answers natural language questions by consulting a repository of documents, many of their systems are answering questions over the web. A QA system utilizing this resource has the potential to answer questions of a wide variety of topics, and will constantly be kept up-to-date with the web with minimum efforts. Most of the current question answering systems can be decomposed into four components: Question analysis, Document retrieval, Passage retrieval, and Answer extraction. Figure 2.1 shows a general overview of a QA's system architecture. The question analysis component classifies natural language questions by the expected answer entity and creates a "bag of word" query from it. For example in *"Where is the highest Dam in the world*?"; the entity searched for is a *<LOCATION>*. Typically, queries generated by this analysis are used by the document retriever to find a set of potential documents from the document collection (web, database, etc.).

Passage retrieval then searches in these potential documents and selects paragraph size texts that are likely to contain the answer. These paragraphs are then passed to the following Answer Extraction phase. Finally, the answer extraction component searches the potential passages using one or a combination of several NLP techniques such as: Part of Speech tagging (POS), word similarity, Named-entity tagging, and others to extract the final and correct answer.

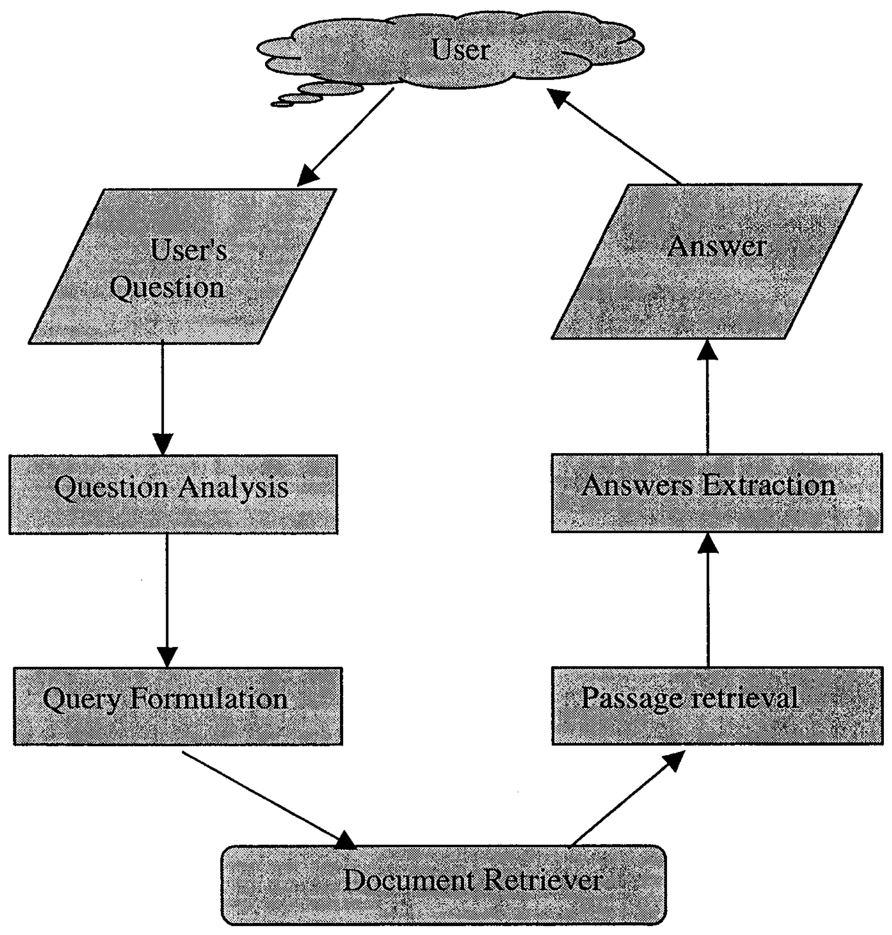

Figure 2.1 Typical QA system Architecture

These processes show that Natural Language Processing (NLP) represents one of the major QA system components, and any improvements in the field of NLP will reflect on the improvement on QA systems.

In this chapter will present the TREC competition, followed by a literature review of previous QA work, and finally we will describe the current system that we are trying to improve.

2.1 The TREC Competition

Over the past few years, and to support research within the information retrieval community, providing the infrastructure necessary for large-scale evaluation of text retrieval methodologies, the *Text Retrieval Conference* (TREC) - co-sponsored by the National Institute of Standards and Technology (NIST) and the Defense Advanced Research Projects Agency (DARPA)- was created. The TREC competitions [Voorhees, 2000, 2001, 2002] have brought formal and rigorous evaluation methodologies to bear on the question-answering task. The evaluation process consists of measuring and comparing QA systems MRR (Mean Reciprocal Rank) which is calculated by the following equation:

$$\mathbf{MRR} = \sum_{\mathbf{i=1}}^{n} \mathbf{(1/rank) / N}$$

Where: n = Total number of questions to be answered
rank = Ranking of the correct answer that is given by a system
N = Total number of questions to be answered

The higher MRR the system has, the better the results are.

2.2 Previous work in QA

We conducted research about the answer extraction techniques in some previous work, and found that there is a variety of approaches. In 1993, Kupiec developed the MURAX [Kupiec, 1993] using an on-line encyclopedia. His system used robust shallow parsing but suffered from the lack of basic information extraction support. In fact, the most significant answer extractions advance, namely Named Entity extraction, occurred after Kupiec, thanks to the MUC program (MUC-7, 1998). In 1999, Srihari and Li [Srihari and Li, 1999] used *Textract 1.0,* a simple Named Entity extractor approach, for the TREC-8 QA and obtained a 66.0% tagging accuracy results. High-level answer extraction technology beyond Named Entity has not been in the stage of possible application until recently. Harabagiu introduced a complex abductive inference approach [Harabagiu et al., 2000], which has limited result improvements over the simplest tagging systems.

AskJeeves launched a QA portal (www.askjeeves.com), which is equipped with a fairly sophisticated natural language question parser. When it can find the exact answer, the system gives it, but it does not provide direct answers to the asked questions. When no answer can be found, it directs the user to the relevant web pages, just as a traditional search engine does. The same concepts was used by Zheng [Zheng, 2002], who introduced the AnswerBus system which uses word occurrence frequency to determine the potential answers from different search engines on the Web.

In the following sections, we will look in more detail at some on-line and off-line QA systems, and highlight their major components and their similarities or differences with our system.

2.2.1 The START system

Boris Katz introduced the START system (SynTactic Analysis using Reversible Transformation) that uses natural language annotation [Katz, 1990; Katz, 1997]. START is a natural language question answering system that has been available on the World Wide Web since December 1993, and can answer simple user questions concerning geography, weather, movies and many other areas. The START system uses semantic template-expressions for the question semantic parsing. Based on this parsing result, the system decides where to find the answer on the Web. Although the system served millions and millions of users in the last decade, maintaining and expending its knowledge base is a time-consuming task that requires trained and experts engineers.

Our system is an automated, self-maintained system that doesn't need trained or expert engineers to adopt or improve its current status.

2.2.2 The QA-LaSIE system

The QA-LaSIE system [Scott and Gaizauskas, 2001] finds answer to questions against large collections of documents. The system uses a query to do passage retrieval from the text collection (see Figure 2.1). The answer extraction system does partial syntactic and semantic analysis on the top ranked passages from the IR system, along with the question itself to identify the potential answers. The system uses Brill's tagger [Brill, 1992] to assign part of speech tags to each token in the text that was retrieved by the passage retrieval phase.

LaSIE was build on the top of Eric Brill's tagger, but our system integrates both GATE-NE and Link Parser as a part of the QA system.

2.2.3 The QALC system

The QALC system at LIMSI [Ferret et al, 2001] is based on web searching. The system uses its own named entity tagger, and takes advantage of the WordNet semantic database in the answer extraction process, whenever the expected answer type is not a named entity. The system achieves encouraging results where about 70 % of the answers were ranked in the top five answers at the TREC-11 competition.

2.2.4 Selective Relation

Katz and Lin introduced and showed how syntactic processing can improve precision in question answering [Katz and Lin, 2002]. They demonstrated that syntactic analysis enables a question-answering system to successfully handle semantic symmetry and ambiguous modification issues that all current question-answering systems are not able to handle.

As per Katz's approach, we are using a syntactical tool (Link Parser) in our system.

2.2.5 The Microsoft system

Eric Brill and his colleagues [Brill et al, 2002] utilize the Google search engine to find answers on the Web. Given a question, he formulates multiple queries to send to the search engine. His system uses manual re-write rules with possible verb movement for query reformulation. The system retrieves the best 100 matching pages for each question, and then harvests the returned summaries for further processing. A set of potential answers is extracted from the summary text, with each potential answer string weighted by a number of factors, including how well it matches the expected answer type and how often it occurred in the retrieved page summaries. Then, with the given set of possible answers, the system performs answer tiling, which merges similar answers. For example, if an answer includes 3 words "X Y Z" and another includes "W X Y", then they will be merged into a single answer "W X Y Z". At the TREC-11 competition the system obtained an MRR of 0.437 but 39.6 % of the questions were not answered.

As we will see in section 2.7, this MS approach is the most similar system to ours; it uses manual re-write rules with possible verb movements for query reformulation.

2.2.6 The PiQASso QA system

PiQASso [Giuseppe, 2001] is based on a series of semantic filters for selecting texts containing a justifiable answer. These filters are based on several NLP tools: a POS tagger, a named-entity tagger, and a lexical database. The system performs question keyword extraction and query formulation as a first step in the question answering process. Then, a keyword search is performed for selecting candidate answer sentences from the document collection. The sentences returned by the query are analyzed and checked for the presence of entities of the proper answer type, as determined by question analysis. Sentences are parsed and recognized entities are tagged and passed to the relation matching filter, which performs a more semantic analysis, verifying that the answer sentence contains words that have the same type and relation as the corresponding words in the question. PiQASso only achieved a MRR of 0.271 in the TREC-2001 and half of the questions (49%) had no answer found. The main reason of this poor performance is due to both keyword extraction and the named-entity tagger failure to identify the correct text and answer, they either do not return any results, or return too many non relevant texts.

After this brief QA systems review, the following section presents an overall view of the current system.

2.3 Overview of the Current System

As mentioned above, our system is an extension of an already existing system [Guillemette, 2001] that uses a query reformulation approach similar to Microsoft filtering approach (section 2.2.5). In this work we are trying to increase the answer extraction accuracy by using named-entity tagging and syntactic parsing. The following sections present the main parts of the original system.

2.3.1 Question Set

Since the TREC test data collections are available to researchers to evaluate their own retrieval systems, the original system used TREC-8 and TREC-9 as training sets and TREC-10, and TREC-11 for the validation.

2.3.2 Query Reformulation

Given the following question, "*Where is Basque country located?*", the system will follow the typical architecture outlined in Figure 2.1. First, the question is classified into one of several categories depending on the semantic type of the answer (ex, who, what, where, etc.), each of which is mapped to a particular set of rewrite rules [Guillemette, 2001]. The rewrites generated by the system were simple string-based manipulations. For instance, some question types involve query rewrites with possible verb movement; the verb "is" in the above question should be moved in formulating the desired rewrite query. A simple approach was taken in the verb's possible movements, by moving the verb to all possible positions in the query. While such an approach results in many nonsensical rewrites (e.g. "*Basque country located is*", "*Basque is country located*"), this rarely retrieves any document from the web. To be able to formulate a declarative sentence, the system uses WordNet to transform verb simple tense to the main verb.

2.3.3 Search Engine

We know that the answer formulation could be in several forms, one of these forms will be "*Basque country is located <LOCATION>*". Therefore, the system tries to perform an on-line search for a string that matches this formulation and all other possible formulations. The system used Yahoo and Google as the search engines, by passing the formulation to the engine as a query and getting the question relevant documents as the engine's output.

2.3.4 Answer Extraction

After retrieving the question relevant documents from the Web, the system identifies the answer candidates by unification, and performs a simple semantic check, which is mainly checking for letter capitalization, word's length, etc.

2.3.5 Answer Ranking

The first time the system extracts an answer, it initializes its ranking to 0.65 points and for each subsequent occurrence of the same answer, the different to one divided by two is added to the ranking as described in the following equation:

$$\textbf{Answer Ranking} = \mu + ((1 - \mu) / 2)$$

Where μ is the current answer ranking point which is initialized to 0.65

The final answers are displayed in an output file that have *question ID, ranking, and the extracted answer* for each given question.

2.7.6 System Evaluation

As mentioned above, all experiments were performed on both TREC-10 and TREC-11, and the extracted answers were compared with the TREC answers. The ranking of each corrected answer is identified, then the MRR (Mean Reciprocal Rank) is calculated for each answer and for all the set as well by the MRR equations that was described in section 2.1. The original system was able, with the question reformulation, to obtain a correct answer for 24% of the total TREC questions compared to 20% without reformulation. The best results were obtained for <PERSON> type-question, "*who*", where there were about 100% increase from 16% without reformulation to 31% with reformulation.

Chapter 3

System Architecture

3.1 Programming Language

The original system was built mostly in Scheme. However after analyzing the scope of our work's requirements and studying the advantages and disadvantages of the programming languages available, we decided to work with Perl for the design and implementation. The main advantage of Perl over Scheme, beside its built-in features for character and string parsing and manipulating which are extensively used in our system, is its flexibility and its platform independence. Perl also is well known by many students and researchers which will make our work easily adopted and used in any future related work.

3.2 System Organization

We have used *GateInterface* (based on QUANTUM *gate.pm* Module) as an interface between our modules and the GATE system. GATE had to be enhanced, since it doesn't support all the searched entities that we need. These enhancements will be covered in details in section 4.2. We have therefore created our own *GateInterface* object interface for the missing entities. Our system uses the *GateInfo* module to access the *GateInterface* output, which is stored as an object in this module. Appendix B describes all the *GateInfo* functions and implementations.

3.3 System Modules

Our system consists of two major components:

3.3.1 Question analysis and retrieval process

These processes are part of the original Web-QA implementation and it is written in Scheme, the following are the two main system modules:

- `Patterns.scm` manipulates the question and reformulation process.

- `Get-text.scm` search engine interface, and documents retrieval process.

Guillemette [Guillemette, 2001] describes the Scheme system architecture, files installation and implementations.

3.3.2 Answer Extraction process

The answer extraction module was the emphasis of our work. It is written in Perl, and the `get-answer.pl` is the main module in this process. This module is called with several arguments as the following example:

```
"get-answer.pl -q question file -d directory -r results  file -p"
```

where:

- *`Question file`:* It contains the questions id and text in the following format:

  id question's text. e.g., *122 Who is the best Hockey player this year?*

- *`directory`:* parent of "*tmp*" directory where all retrieved documents are stored

- *`Results file`:* An output file with each question answer and its ranking value

- *p* if set, Link Grammar parser will be used to filter GATE output. If not, only GATE-NE tagging will be used.

Chapters 4 and 5 will describe in details both GATE and Link Parser modules installation and structure.

Figure 3.1 shows the answer extraction Data Flow.

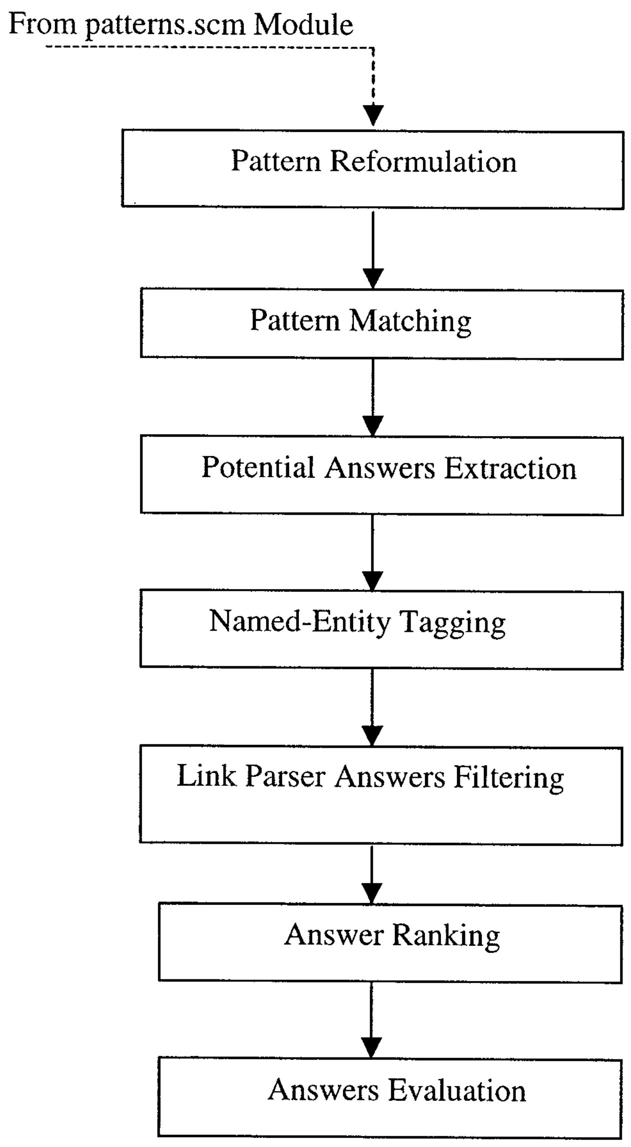

Figure 3.1 Answer Extraction Flow Chart of our system

3.4 Question Sets

As mentioned in Chapter 2, since the TREC test data collections are available to the researchers to evaluate their own retrieval systems, we used the TREC questions as our system training and validating sets. TREC-8 is used as our training set to develop the system. It consists of 200 questions. Table 3-1 shows the distribution of the named entities searched in the TREC-8 questions:

Entity	**# of occurrences**	**Distribution**	**Question type**
PERSON	73	36.5 %	Who
LOCATION	38	19.0 %	Where
MEASUREMENT (NUMBER)	32	16.0 %	How much, How far, etc.
TIME/DATE	30	15.0 %	When
OTHER	27	13.5 %	What, which
Total	**200**	**100 %**	

Table 3.1 Distribution of TREC-8 named entities

Table 3-1 shows that over 80% of the 200 TREC-8 questions required PERSON, LOCATION, MESUREMENT, TIME or DATE entities. Therefore, most of our developments and implementations were mainly focused on supporting these entities.

3.5 Answer Ranking

Since, our system is an extension of an already existing system, we have used the same ranking approach as per the original system (see section 2.3.6). Although, we prefer having the ranking initialization value to be determined experimentally, we decided to keep the same ranking process as per the original system for comparison reasons.

3.6 System Evaluation

As mentioned in chapter 2, we compared our extracted answers with the TREC answers. The ranking of each corrected answer is identified, then the MRR (Mean Reciprocal Rank) is calculated for each question and for all the questions set as well by the MRR equation (see section 2.1). Since the current system and the original system do not have the same number of questions that have an answer, we used the following two comparison measures:

$$\textbf{Actual MRR} = \sum_{i=1}^{n} (1/\text{rank}) / M$$

Where: n = Total number of questions to be answered
M = Total number of questions that have an answer

Normalized MRR = (Actual MRR * M) / Total number of questions in the set

This evaluation process is done on both GATE-NE results only and the combined GATE-NE and the Link Parser results. In each case, both actual and normalized MRR of our system are compared to the original system. Chapter Six presents in full details our final system results and evaluation.

Chapter 4

Named-Entity tagging (GATE-NE) Implementation

4.1 Introduction

Name Entity recognition involves processing a text and identifying certain occurrences of words or expressions as belonging to particular categories of Named Entity (NE). NE recognition software serves as an important preprocessing tool for tasks such as information retrieval, information extraction and other text processing applications. To improve our web-QA system, we decided to use the named entity module available within GATE (General Architecture for Text Engineering) [Cuningham, 1996]. As mentioned in chapter 1, GATE is an architecture, development environment, and framework for building systems that process human language. It has been in development at the University of Sheffield since 1995, and has been used for many research projects, including information extraction in multiple languages and media. It is an open source system available in C++/TCL (version 1, 1995-1997) and in Java (version 2 1999-2002). We used the latest version of GATE 2.0 (released on March 2002), which was completely written in Java and it was installed on our CLaC server in the Laboratory. (See Appendix B for more details).

4.2 Extracting entities

GATE 2.0 is able to identify only the following entities: *PERSON, PERCENT, LOCATION,* and *DATE* [Zhang et al, 2001]. Therefore, we had to create our own GATE interface for the other useful named entities such as *MEASUREMENT, DISTANCE, and DURATION,* which were all grouped under the same *NUMBER entity.* Our accessing and returning object approach was the same as the original GATE interface, which facilitated the implementation process. Section 4.2.3 gives an example of such entities identification limitations. In the following sections, we present some of the entities and how the GATE-NE system extracts them from a given sentence.

4.2.1 PERSON ENTITY

Given the question "*Who is the new Prime Minister of Canada?*", the system should match "*PERSON is the new Prime Minister of Canada*", as it is given by our reformulation process. Accordingly "*Jean Paul declared that <u>Paul Martin is the new Prime Minister of Canada</u>*" will be considered as one of the potential matches. When sending to GATE: *Jean Paul declared that Paul Martin is the new Prime Minister of Canada,* GATE will identify all named entities in the given sentence and will return the following:

1- Entity type, which represent the entity name as *PERSON, LOCATION,* etc.

2- Entities start position, which represents the first character position in the given sentence.

3- Entity which represents the text that represent the identified entity

4- Entities stop position, which represents the last character position in the given sentence.

Figure 4.1, shows an actual GATE-NE system output for the above sentence:

```
'entitystop' => [
                  '71',
                  '9',
                  '35'
                 ],

'entitytypes' => [
                   'Location',
                   'Person',
                   'Person'
                  ],

'entitystart' => [
                   '65',
                   '0',
                   '24'
                  ],

 'entities' => [
                 'Canada',
                 'Jean Paul',
                 'Paul Martin'
                ]
```

Figure 4.1 GATE system PERSON entities Extraction

Since the actual GATE-NE system doesn't have an elegant interface (as we see in figure 4.1), we will present the GATE-NE system's output in a table format as follows:

Entity Type	Entity Position	Entity Text
Person	0-9	Jean Paul
Person	24-35	Paul Martin
Location	65-71	Canada

Table 4-1 GATE system PERSON entities Extraction

The GATE-NE system extracted all the possible entities in the given sentence but could not identify the correct answer (*Paul Martin*). Section 5.3.1 will present how the Link Parser could filter this answer extraction problem.

4.2.2 DATE (TIME) Entity

Given the question "*When did Beethoven die?*", the system will try to match *"Beethoven died TIME*", as it is given by our reformulation process "*Beethoven died on March 26 but was declared dead on June 11*" will be considered as one of the potential match. When sending to GATE *"Beethoven died on March 26 but was declared dead on June 11"*, GATE will identify and return all possible entities. Table 4.2 presents the reformatted GATE-NE system output for the above sentence:

Entity Type	Entity Position	Entity Text
DATE	19-27	March 26
DATE	53-60	June 11

Table 4-2 GATE system DATE entities Extraction

The GATE-NE system extracted all the possible entities in the given sentence including *June 11*, which is not the correct answer. Section 5.3.2 will present how the Link Parser could filter this answer extraction and only extract the correct answer.

4.2.3 NUMBER Entity

Assume that the question "*How many people live in Tokyo?*" will be analyzed by the GATE-NE system. Then, we will try to match the query reformulation as "*NUMBER people live in Tokyo*". The following sentence will be considered as one of the potential matches: "*Among 80 million, there are 12 million people living in Tokyo* ".

As mentioned above, the GATE-NE system interface cannot extract NUMBER entities. Therefore, we had to create our own GATE interface for NUMBER entities to overcome such limitation.

We have used regular expressions to identify any NUMBER representations in a given text; we have created the following four representations:

1- Digital number representation, such as *1, 200, 3001, 3,200*, etc.

2- Alphabetic representation such as: *One, three, twenty, hundred, million*, etc.

3- Compound representation such as: *Three thousands five hundreds and twenty-two.*

4- Measurement units such as: *feet, meter, kilo*, etc.

When sending to GATE: *Among 80 million, there are 12 million people living in Tokyo.* our module will parse the given sentence and identify all NUMBER expressions and output the results in the same format as the actual GATE-NE system does.

Table 4.3 presents the reformatted output of our system GATE-NE interface for the above sentence:

Obviously, *12 million* is the only valid answer in this question, but the GATE system extracts all possible entities in a given sentence.

Entity Type	Entity Position	Entity Text
NUMBER	6-15	80 million
NUMBER	49-59	12 million

Table 4-3 Our system Interface NUMBER entity Extraction

Section 5.3.3 will present how the Link Parser could filter this answer extraction.

4.2.4 Entity Extraction Limitation

The GATE system is able to recognize DATE entities, but only recognizes years in 4 characters format. Table 4.4 shows how GATE interpreted differently *March 1420, March 420* and *March 0420.* Table 4.4.a shows how the GATE-NE system recognizes the 4-character year string (1420), when sending to GATE: *March 1420.*

Entity Type	Entity Position	Entity Text
DATE	0-8	March 1420

Table 4-4.a GATE-NE system output for 4 characters year string

Table 4.4.b shows how the GATE-NE system was not able to recognize the 3 characters year string (*420*), when sending to GATE: *March 420.* GATE-NE system will identify the following:

Entity Type	Entity Position	Entity Text
DATE	0-4	March

Table 4-4.b GATE-NE system output for 3 characters year string

Table 4.4.c shows how the GATE system recognizes an invalid 4 characters year string (*0420*), when sending to GATE: *March 0420*.

Entity Type	Entity Position	Entity Text
DATE	0-8	March 0420

Table 4-4.c GATE-NE system output for invalid 4 characters year string

Our system did not address this issue, since it was never found in the training set, and it was just observed lately. Actually, we did not even face it in the validation process either. But definitely we consider it as a system deficiency and it should be eliminated.

4.3 GATE system Evaluation

Table 4-5 shows a summary of the testing results with all 1693 questions. The results show an overall 6% MRR improvement and about 9% correct answer improvement in our approach over the original system approach. The improvement is noticeable in each category.

As shown in table 4.5, our system extracts at least one answer for 536 questions, compared to 484 by the original system, which reflects a 10 % increase. Also the number of correctly extracted answers is also increased by the same percentage.

	Original System	GATE-NE System
Total number of questions	1693	1693
# of questions with no retrieved documents	583	583
# of questions with at least one retrieved doc.	1100	1100
# of questions with no answer found	616	564
# of questions with at least one extracted answer	484	536
# of questions with wrong extracted answers	214	236
# of questions with correct extracted answers	**270**	**300**
% of correct answer/total question number	**16.30 %**	**17.70 %**
% of correct answer /retrieved	**24.54 %**	**27.27 %**
$\sum$ (1 / answer ranking)	**169.4**	**186.52**
Actual MRR	**0.35**	**0.348**
Normalized MRR/ 1693 questions	**0.101**	**0.107**

Table 4-5 GATE-NE system final result comparison to the original system

As we will see in chapter 6, the training set entities distribution is not well represented in the validation set, and this could skew the system performance toward these entities that are well represented in both training and validation sets.

Table 4.6 shows a significant improvement in the number of questions with at least one extracted answer and number of questions with correctly extracted answer, when TIME entity is the searched entity. Although, the actual MRR is slightly higher, normalized MRR obtained a 80% increase compared to the original system.

Number of questions	**Original System**	**GATE-NE system**
With at least one extracted answer	**30**	**56**
With correct extracted answer	**18**	**37**
Actual MRR	**0.90**	**0.92**
Normalized MRR	**0.072**	**0.121**

Table 4-6 TREC-9, TREC-10, and TREC-2002: TIME entity performance comparison

In view of these results, we believe that a NE tagger is very useful tool for QA systems. But we also believe a QA system should not rely too heavily on NE tagging, as the tagging process fails in many cases to associate the proper semantic type to expressions. For example, proper names with the PERSON entity, specially, in the case of non-English names.

Therefore, we used a syntactical parser to improve the GATE-NE system. Chapter 5 will present the Link Parser implementation and performance.

Chapter 5

Link Parser Implementation

5.1 Introduction

The Link Parser Grammar is a syntactic parser of English, based on link grammar, an original theory of English syntax [Temperley, Sleator, and Lafferty, 1991]. The basic idea is to assign a syntactic structure to a given sentence. The system is written in generic C, and it can be compiled and run in any Platform with a C compiler.

The parser considers the words as blocks, which are connected through a set of pointers (connectors). Each word will be connected to the following word by a right connector (-) and to the previous word by a left connector (+). The pair of connectors between any two words forms a link. A valid sentence is one in which all words are used in a way that satisfies the following two types of rules:

- Word Rules: which are stored in a simple dictionary to identify different word meanings in a sentence (see Appendix C).
- Global Rules: which do not allow cross links (planarity) and ensure connectivity where all the words must be indirectly connected to each other.

The original version of the parser was designed around a standard interface, where the user types in a sentence, and the parser displays all the existing linkages for the sentence. This is fine for showing the result of the grammar and parser work, but in order to make actual use of the information that the parser provides, it is necessary to have access to its inner workings. The Link Parser API was written to give users flexibility in using the parser in their applications. The API is written in ANSI C, and runs in both UNIX and Windows environments

5.2 Perl Interface

The Link Grammar parser itself is a complex software that implements a complex theory of language. Therefore, there is a need for an interface that can easily access the parser API and return the parser links for analysis. Dan Brian developed the Lingua::LinkParser Module [Brian, 2000] that provides access to the parser API using Perl objects to easily analyse the parser links and linkages. The module organises data returned from the parser API into an object hierarchy consisting of, in order, sentence, linkage, sub-linkage, and link. Appendix D shows some of the Module's basic functions and commands. The current Perl implementation is based on version 4.0 of the Link Grammar parser API.

5.3 Analysis Process

We used the parser process in our implementation as a filter to the extracted answers from the GATE-NE process. The main role of the parser is to try to verify if the extracted answers from GATE-NE are the head of a noun phrase or not. Our system sends the potential text to the Link Parser to extract the tokens, then tags them with parts-of-speech, and identifies the syntactic constituents. The Link Parser often returns more than one parse tree for a given sentence, along with a link cost "confidence level". In our implementation we only process the lower link cost or "higher confidence level".

In the following sections, we present some of the entities and how the Link Parser extracts them from a given sentence.

5.3.1 Person Entity

Figure 5.1 shows few possible trees that Link Parser creates for the sentence, *"Jean Paul declared that Paul Martin is the new Prime Minister of Canada"* from section 4.2.1.

```
                                                +-----------Ost-----------+
                                                |   +----------Ds---------+
                                +----Cet----+   |   |    +--------A-------+
  +--G-+---Ss--+---TH---+     +--G--+--Ss-+   |   |      +----A----+--
Mp--+-
   |    |      |       |     |     |     |   |    |      |          |
|
Jean Paul declared.v that.c Paul Martin is.v the new.a prime.a minister
.n

Js-+
   |
of Canada
```

Figure 5.1.a PERSON entity parsing Tree (Linkage) # 1 and cost =21

```
                                                +-----------Ost-----------+
                                                |   +----------Ds---------+
                                +----Cet----+   |   |    +--------A-------+
  +--G-+---Ss--+---TH---+     +--G--+--Ss-+   |   |      +----AN---+--
Mp--+-
   |    |      |       |     |     |     |   |    |      |          |
|
Jean Paul declared.v that.c Paul Martin is.v the new.a prime.n
minister.n of


Js-+
   |
Canada
```

Figure 5.1.b PERSON entity parsing Tree (Linkage) # 2 and Cost =21

```
                                                  +-----------Ost-----------+
                                                  |   +----------Ds---------+
                                  +----Cet----+   |   |    +--------A-------+
   +--GN-+---Ss--+---TH---+     +--G--+--Ss-+   |   |      +----AN---
+--Mp-
   |     |      |        |     |     |    |   |    |      |          |
jean.n Paul declared.v that.c Paul Martin is.v the new prime.n miniter

-+-Js-+
 |    |
of Canada
```

Figure 5.1.c PERSON entity parsing Tree (Linkage) #4 and Cost =23

Figure 5.1.a. shows the higher confidence tree (lower cost), which is the only one processed by our system.

By knowing that we are searching a *PERSON* and knowing that the Link Parser connects proper nouns together in series by the "G" connector, then we could identify both *Jean Paul* and *Paul Martin* as valid answers (the same as GATE-NE in section 4.2). But, the Link Parser identifies the subject of the sentence as well, which is identified by the connector "S", and it reflects the existence of a link (relation) between *"Martin"* and *"is the new prime Minister of Canada"*. Therefore, only *Paul Martin* will be considered as the correct extracted answer for this question.

5.3.2 DATE (Time) Entity

Figure 5.2 shows the possible trees that the Link Parser creates for the example in section 4.2.2, where the question was "*When did Beethoven die?*" And one of the sentences was "*Beethoven died on March 26 but was declared dead on June 11*".

The subject of the sentence is Beethoven and "S" is its Parser connector, knowing one or a combination of the following "TA, TD, TM, TW, TH, ON. or DT" is the TIME parser's connectors. Then we could identify and extract the answer as "*on March 26*".

```
    +-------------------Ss--------------+
    +---Ss---+-MVp+-ON+-TM-+   +---Pvf--+----MVp----+-ON+-TM+
    |        |       |      |       |  |          |               |      |     |
Beethoven died.v on March 26 but was.v declared.v [dead] on   June    11
```

Figure 5.2 Date entity Parsing Tree (Linkage 1, Cost = 31)

5.3.3 Number Entity

Figure 5.4 shows one of the possible trees that the Link Parser creates for the example of section 4.2.3, where the question was *"How many people live in Tokyo?"* and one of the excepts was *"Among 80 million, there are 12 million people living in Tokyo".* Knowing that we are looking for a NUMBER and the connectors that are used by the Link Parser to identify NUMBER is one or a combination of the following "NN, NI, NF, or Dmcn ", we can identify our MVp phrase then we could identify and extract the answer as "12 million".

```
+--------CO--------+
+-------Xc------+   |
+----Jp---+   |   |   +--------Opt-------+        +------MVp------+
|   +- NN-+  |   +-SFp-+   +-NN-+--Dmcn--+---Ma--+-MVp+-Js+ |
|   |      |  |  |    |         |      |        |    |    |   |
Among 80 million , there is.v 12 million people.p live.a in Tokyo alone.a
```

Figure 5.3 Number entity Parsing Tree (Linkage=1 cost=27)

The same concept has been used for all the other entities.

Perl could not parse any of the above-mentioned diagrams. The following section will explain how we used pattern matching, to parse and implement our Link Parser approach.

5.4 Accessing the Link Parser Connectors

The Link Parser diagrams help the user understand the parse tree, but to use this information within a program requires access to the links and sub-links themselves. The Lingua Module [Brian, 2000] gives the possibility to overload the parser print command to display the diagram as a string. Having the diagram as a string, helps users use pattern matching to identify the links (connectors) between words.

5.4.1 PERSON Entity

Figure 5.4 shows how the example in section 5.3.1 (Figure 5.1.a) could be represented as a string. Since we are looking for a PERSON entity (G or GN) and a Subject (S) then we can use pattern matching to extract the required answer from the string.

Sentence

Jean Paul declared that Paul Martin is the new Prime Minister of Canada.

Parsing Tree

```
(( "LEFT-WALL" RW:14:RIGHT-WALL Wd:2:Paul )
( "Jean" G:2:Paul )
(Wd:0:LEFT-WALL G:1:Jean  "Paul" Ss:3:declared.v G:6:Martin )
(Ss:2:Paul  "declared.v" TH:4:that.c )
(TH:3:declared.v  "that.c" Cet:6:Martin )
(Wd:0:LEFT-WALL G:1:Jean Ss:3:declared.v  "Paul" G:6:Martin )
(Cet:4:that.c G:5:Paul  "Martin" Ss:7:is.v )
(Ss:6:Martin  "is.v" Ost:11:Minister.n )
( "the" Ds:11:Minister.n )( "new.a" A:11:Minister.n )
( "Prime.n" AN:11:Minister.n )
(Ost:7:is.v Ds:8:the A:9:new.a AN:10:Prime.n  "Minister.n" Mp:12:of )
(Mp:11:Minister.n  "of" Js:13:Canada )(Js:12:of  "Canada" )
(RW:0:LEFT-WALL  "RIGHT-WALL" ))
```

Matched string

```
(Cet:4:that.c G:5:Paul  "Martin" Ss:7:is.v )
```

Extracted Answer:

Paul Martin

Figure 5.4 Extracting PERSON entities from the Link Parser's output

5.4.2 DATE (TIME) entity implementation

As mentioned above, we display the parsing tree as one full string. Then we use pattern matching to match and "grep" the given sentence Subject connector (S), verb-phrase connector (MV) and the related TIME connectors (Js, TM,). Figure 5.5 shows the matched string in a given tree.

Sentence:

Beethoven died on March 26 but was declared dead on June 11.

Parsing Tree:

```
(( "LEFT-WALL" RW:12:RIGHT-WALL Wd:1:Beethoven )(Wd:0:LEFT-WALL
"Beethoven" Ss:2:died.v Ss:7:was.v )(Ss:1:Beethoven "died.v" MVp:3:on
)(MVp:2:died.v "on" ON:4:March MVp:8:declared.v ON:10:June )(ON:3:on "March"
TM:5:26 )(TM:4:March "26" )( "but" )(Ss:1:Beethoven "was.v" Pvf:8:declared.v
)(Pvf:7:was.v "declared.v" MVp:9:on )(MVp:2:died.v ON:4:March MVp:8:declared.v
"on" ON:10:June )(ON:9:on "June" TM:11:11 )(TM:10:June "11" )(RW:0:LEFT-
WALL "RIGHT-WALL" ))
```

Matched string (Ss:1:Beethoven "died.v" MVp:3:on)(ON:3:on TM:4:March "26")

Extracted Answer: March 26.

Figure5.5 Extracting DATE entities from the Link Parser's output

5.4.3 NUMBER entity implementation

Figure 5.6 shows how the example in section 5.3.3 (Figure 5.3) could be represented as a string. Since we are looking for a NUMBER entity (NN, NI, NF, or Dmcn) and a Subject (S), we can use pattern matching to extract the required answer from the string.

Sentence

Among 80 million, there are 12 million people living in Tokyo.

Parsing Tree

```
=(( "LEFT-WALL" RW:13:RIGHT-WALL Wq:1:among )(Wq:0:LEFT-WALL
"among" PF:6:are.v Jp:3:million )( "80" NN:3:million )(Jp:1:among NN:2:80  "million"
MXp:5:There Dmcn:9:people.p NN:7:12 )( "," Xd:5:There )(MXp:3:million Xd:4:,
"There" )(PF:1:among  "are.v" SIpx:9:people.p )( "12" NN:8:million )(Jp:1:among
NN:2:80 MXp:5:There NN:7:12  "million" Dmcn:9:people.p )(SIpx:6:are.v
Dmcn:8:million  "people.p" Mg:10:living.v )(Mg:9:people.p  "living.v" MVp:11:in
)(MVp:10:living.v  "in" Js:12:Tokyo )(Js:11:in  "Tokyo" )(RW:0:LEFT-WALL
"RIGHT-WALL" ))
```

Matched string

```
(NN:7:12 "million" Dmcn:9:people.p )( "12" NN:8:million )
```

Extracted Answer

12 million

Figure 5-6. Extracting NUMBER entities from the Link Parser's output

5.4.4 Entity Extraction Limitation

We have observed the same date entity extraction issue as in the GATE system. The Link Parser could not recognize any 2 or 3 digit word that represents a year. Even any year prior to 1900, will not be recognized as a DATE (year) by the Link Parser. This shown in Figures 5.7.a and 5.7.b

Sentence

He died on March 1885.

Parsing Tree

```
+-Ss-+-MVp+-Js+
|    |     |   |
he died.v on March [1885]
```

Figure 5-7.a Link Parser System year string Limitations

Sentence

He died on March 85.

Parsing Tree

```
+-Ss-+-MVp+-Js+
|    |     |  |
he died.v on March [85]
```

Figure 5-7.b Link Parser System year string Limitations

5.5 The Link Parser system Evaluation

Chapter 6 will present a summary of the testing results with all 1693 questions. The results show an overall 5% MRR improvement with the Link Parser approach compared to the GATE-NE system only.

As per section 4.3, and due to the fact that the training set entities distribution is not well representing the validation set, Table 5.1 shows a slight improvement in normalized MRR compared to the GATE-NE.

Number of questions	Original System	GATE-NE system	GATE-NE and Link Parser system
With at least one extracted answer	30	56	56
With correct extracted answer	18	37	37
Actual MRR	0.90	0.92	0.922
Normalized MRR	0.072	0.121	0.131

Table 5-1 TREC-9, TREC-10, and TREC-2002: TIME entity performance comparison

Chapter 6

Experiments

6.1 Evaluation

The GATE-NE and Link Parser Pert modules were developed on a test-set of 200 questions (TREC-8). In each TREC experiment, the results from the original Web-QA system [Guillemette, 2001] were compared with our GATE-NE and Parser results. Our analysis showed that over 80% of our TREC-8 200 questions were tagged as TIME or DATE, PERSON, LOCATION, MEASUREMENT. Therefore, most of our developments and implementations were mainly supporting these entities. Table 6-1 shows the results with the TREC-8 training set. Although, the original system has a higher number of questions that have at least one extracted answer our approaches have a higher extracted correct answers and accordingly higher MRR.

	Original System	GATE-NE ONLY	GATE-NE And Parser
Total number of questions	200	200	200
# of questions with no retrieved documents	44	44	44
# of questions with at least one retrieved doc.	156	156	156
# of questions with no Answer found	95	110	110
# of questions with at least one extracted answer	61	46	46
# of questions with wrong extracted answers	34	17	17
# of correct extracted answers	**27**	**29**	**29**
% of correct answer/total question number	**13.43 %**	**14.43 %**	**14.43 %**
% of correct answer /retrieved	**44.26 %**	**63.04 %**	**63.04 %**
$\sum$ (1 / answer ranking)	**21.22**	**19.69**	**21.76**
Actual MRR	**0.348**	**0.428**	**0.473**
Normalize MRR/ 200 questions	**0.105**	**0.100**	**0.109**

Table 6-1 TREC-8 (Training set) Results analysis.

The low performance of the extracted relevant answer (14.43 %) is mainly due to quality of the document retrieved by the IR process. Although, our goal is improving an answer extraction process, a higher number of extracted documents would give more confidence of the system overall.

6.2 Validation set Entities distribution

As Table 3-1 showed, more than 80% of the 200 TREC-8 questions required PERSON, LOCATION, MEASUREMENT, and TIME or DATE entities. Therefore, most of our developments and implementations were mainly supporting these entities.

Table 6-2 shows the distribution of the searched named entities for TREC-9, TREC-10 and TREC-2002, which were used in our validation process.

Entity	**TREC-9**	**TREC-10**	**TREC-2002**	**TOTAL**	**Distribution**
PERSON	230	100	105	435	26.0 %
LOCATION	115	70	62	247	14.5 %
MEASUREMENT	55	30	35	120	7.0 %
TIME/DATE	75	55	104	234	14.0 %
Other	218	245	194	637	38.5 %
Total	**693**	**500**	**500**	**1693**	**100 %**

Table 6.2 Distribution of TREC-9, TREC-10, and TREC2002 entities

From the above table, we can conclude that the training set (TREC-8) entity distribution (see Table 3.1) is not a good representation for the test set (TREC-9, TREC-10, and TREC-2002). The best-represented entities are TIME/DATE, which represents about 15 % of the total searched entities in both training and evaluation questions sets, followed by LOCATION and then PERSON entities. This representation was reflected in our system performance as shown in Tables 4-6 and 5-1.

6.3 Evaluation of the test set

The system was validated on TREC-9 (693 questions), TREC-10 (500 questions), and TREC-2002 (500 questions), for a total of 1693 questions. Appendix A presents an example of the system internal process. Table 6-3 shows the result with the TREC-9 question set. The actual MRR of the original system is about 0.30 which is slightly higher than the GATE-NE approach MRR, but the Parser results improved the overall MRR.

	Original System	GATE-NE ONLY	GATE-NE And Parser
Total number of questions	693	693	693
# of questions with no retrieved documents	73	73	73
# of questions with at least one retrieved doc.	620	620	620
# of questions with no answer found	450	450	450
# of questions with at least one extracted answer	170	170	170
# of questions with wrong extracted answers	97	93	93
# of correct extracted answers	**73**	**77**	**77**
% of correct answer/total question number	**10.53 %**	**11.11 %**	**11.11 %**
% of correct answer /retrieved	**42.94 %**	**45.29 %**	**45.29 %**
∑ (1 / answer ranking)	**51**	**48.45**	**55.25**
Actual MRR	**0.3**	**0.285**	**0.325**
Normalize MRR/ 693 questions	**0.073**	**0.07**	**0.08**

Table 6-3 TREC-9 Result.

Table 6-4 shows the result with TREC-10 question set. We consider TREC-10 as our best test set, and this is mainly due to the higher percentage of extracted answers by all three approaches. The improvement is noticeable in each category. Our system extracted 241 answers out of 301 available (80%) compared to 67% by the original system. We extracted 144 correct answers with an actual MRR and normalize MRR of 0.322 and 0.155 respectively, compared to 0.311 and 0.125 for the original approach.

	Original System	GATE-NE ONLY	GATE-NE And Parser
Total number of questions	500	500	500
# of questions with no retrieved documents	199	199	199
# of questions with at least one retrieved doc.	301	301	301
# of questions with no Answer found	100	60	60
# of questions with at least one extracted answer	201	241	241
# of questions with wrong extracted answers	78	93	93
# of correct extracted answers	**123**	**144**	**144**
% of correct answer/total question number	**24.60 %**	**28.80 %**	**28.80 %**
% of correct answer /retrieved	**61.19 %**	**59.75 %**	**59.75 %**
∑ (1 / answer ranking)	**62.51**	**76.88**	**77.6**
Actual MRR	**0.311**	**0.319**	**0.322**
Normalize MRR/ 500 questions	**0.125**	**0.153**	**0.155**

Table 6-4 TREC-10 Result.

Table 6-5 shows the results with TREC-2002 question set. The actual MRR of the original system is about 0.50, which is slightly higher than our approach's MRR. But our approach has a higher correct answer extraction. Again as with the TREC-9 set, the number of the retrieved documents is very low and we could not fairly judge the system with these results.

	Original System	GATE-NE ONLY	GATE-NE And Parser
Total number of questions	500	500	500
# of questions with no retrieved documents	321	321	321
# of questions with at least one retrieved doc.	179	179	179
# of questions with no Answer found	66	54	54
# of questions with at least one extracted answer	113	125	125
# of questions with wrong extracted answers	39	46	46
# of correct extracted answers	74	79	79
% of correct answer/total question number	14.80 %	15.80 %	15.80 %
% of correct answer /retrieved	**65.49 %**	**63.20 %**	**63.20 %**
∑ (1 / answer ranking)	**57.6**	**57.87**	**59.0**
Actual MRR	**0.509**	**0.463**	**0.472**
Normalize MRR/ 500 questions	**0.116**	**0.116**	**0.118**

Table 6-5 TREC-2002 Result.

6.4 Summary of results

Table 6-6 shows a summary of the testing results with all 1693 questions.

	Original System	GATE-NE ONLY	GATE-NE And Parser
Total number of questions	1693	1693	1693
# of questions with no retrieved documents	583	583	583
# of questions with at least one retrieved doc.	1100	1100	1100
# of questions with no Answer found	616	564	564
# of questions with extracted answers	484	536	536
# of questions with wrong extracted answers	214	236	236
# of correct extracted answers	**270**	**300**	**300**
% of correct answer/total question number	**16.30 %**	**17.70 %**	**17.7 %**
% of correct answer /retrieved	**24.54 %**	**27.27 %**	**27.27%**
$\sum$ (1 / answer ranking)	**169.4**	**186.52**	**192.96**
Actual MRR	**0.35**	**0.348**	**0.36**
Normalize MRR/ 1693 Questions	**0.101**	**0.107**	**0.113**

Table 6-6 Summary of the results with TREC-9, TREC-10, and TREC-2002.

The results show an overall 11% MRR improvement and about 9% correct answer improvement in our approach over the original system approach. The improvement is noticeable in each category. Our system extracted 536 answers out of 1100 available question (~49%) compared to 44% extracted by the original system. Our approach's correct answer extraction was about 27 % of the total retrieved documents, compared to 24.5% in the original system.

After the analysis of the results, we concluded that our approach is giving slightly higher and better results than the original system and performs well with respect to our main research objectives as shown in Figure 6.1.

1- Increase number of extracted answers

2- Increase answer extraction accuracy (MRR)

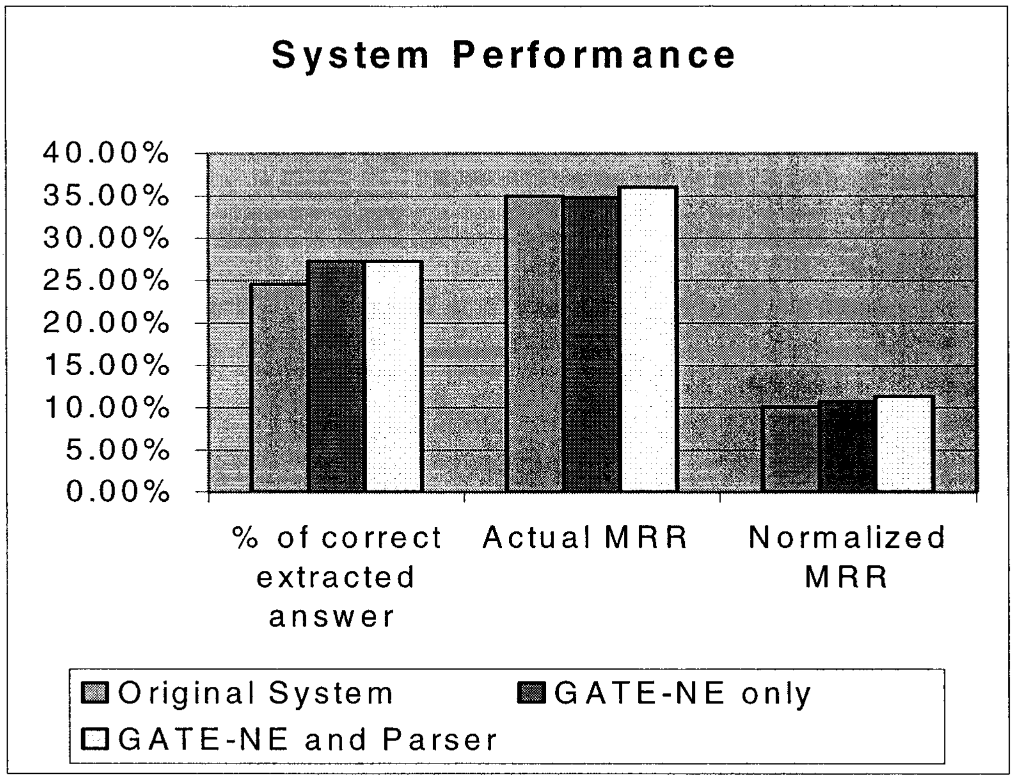

Figure 6-1. System Performance

6.5 Remaining Errors

Although, the MRR has improved slightly, many errors still remain. The following sections will present some of these errors.

6.5.1 Entities Type

As mentioned above, our training set entities representation was skewed in favor of the following entities, PERSON, TIME or DATE, LOCATION, and MEASUREMENT (NUMBER). Therefore, most of our developments and implementations were mainly supporting these entities, and less to some other as CLAUSE, ORGANIZATION, TITLE, REASON, that accounted for 38.5% for the answers in the test questions (see Table 6.2) compared to only 13.5% in the training questions (see Table 3.1). Having a training set that is well representing the validation set would have helped us to fully and fairly judge our system results.

6.5.2 Redundancy Answer Ranking

One of the issues that we addressed in the system is the repetitive answer ranking. We increased the ranking of such a case by a percentage of the existing ranking. This technique works most of the time, but we believe that there is still room for improvements on such a technique. Question 945 (TREC-10) is a best example of this where the question was "*Who discovered x-rays?*". Table 6-7 presents our system's extracted answers and their ranking. Although, all the answers are referring to the same person, our system was dealing with each answer as a separate answer. This is mainly due to the spelling and abbreviations mismatch. The correct answer was "*Wilhelm Conrad Roentgen*" and was ranked second by our system.

Rank	Score	Answer
1	0.90000	Wilhelm Conrad Röntgen
2	0.80000	Wilhelm Conrad Roentgen
3	0.60000	Wilhelm Conrad Rontgen
4	0.60000	Wilhelm Rontgen
5	0.60000	Wilhelm Roentgen
6	0.60000	Wilhelm C. Roentgen
7	0.60000	Wilhelm Röntgen

Table 6.7 Effect of a simple string matching

Chapter 7

Conclusion and Future Work

The extraction techniques presented here show that using a named-entity tagger combined with a syntactic parser can effectively increase the overall MRR of a QA system. The improvement was very low for TREC-9 and TREC-2002, and relatively high for TREC-10. Overall we have an average of 0.113 MRR compared to 0.101 for the original system, and we were able to extract correct answers for 300 questions, 24 more than the original system. One of the factors that effected our validation is the low recall of the IR process (65%), only 1100 questions out of a total of 1693 questions have retrieved documents. Although, our work should be measured relatively to the original system, the low number of documents does not give an accurate evaluation of the system. Therefore, we believe that we should re-evaluate this system with a higher number of documents, to be able to fully and fairly judge this system. At the same time, we feel that more improvements and actions are needed in our approach and should be considered in future work. The following sections present some of these needed actions.

7.1 Co-reference Chains

As we mentioned in Section 6.5.2, using a simple string matching in answer redundancy ranking will not be an effective technique all the time (see Table 6.7), therefore, we suggest the use of co-reference chains to solve this issue.

7.2 Link Parser confidence level

As we mentioned in Chapter 5, the Link Parser usually creates more than one tree for a given sentence. Each tree has a cost or level of confidence associated with it. We only considered the lower cost tree, but we may want to consider all the possible trees. More work is needed to validate this assumption. Another issue that is not solved in our approach is the conjunction links, the Link Parser could create crossing links between words due to the existence of "AND", "OR", "but", "either", or "neither".

7.3 Nature of the question

General questions like "*Where can l find the best car in the Market?*", or "*Where can I find more information about QA answer extraction?*" might be easier to answer by a system based on a well-maintained database. Since our system is based on the processing of the underlying documents, no correct answer can be provided if there is no such answer (explicitly expressed in English) in the processed documents.

7.4 More than one correct answer

Some questions could have more than one correct answer, and these answers could be divided over the text and not in the same sentence, or even over more than one document. In the first case the system will only extract the first answer, and in the second the system will have to choose the correct answer by the ranking approach.

Appendix A: System internal running process

This Appendix shows an example of an internal running process of our system. Given the system the following three questions form TREC-9:

Questions

Question # 221:

Who killed Martin Luther King?

Question # 244:

Who invented baseball?

Question # 430:

Where is Basque country located?

System Results (output):

The following is the output of both the original and the current systems. The output is formatted in three columns, which represent the following:

Question ID   Answer Ranking   Answer Extracted

1- Original System Output

```
221 0.9998046875 James Ray
221 0.9998046875 Prime Suspects Who
244 0.9998046875 Abner Doubleday
244 0.9998046875 The Man Who
244 0.9998046875 Who
244 0.999999999988358 Russians
```

2- GATE-NE output

```
221 0.987500 James Ray
221 0.900000 Isaac Clark Tele
221 0.999805 James Earl Ray An
244 0.999805 Abner Doubleday
244 0.999805 John Leos
244 0.987500 B Find
```

```
244 0.900000 Alexander Cartwright
244 0.987500 Mr. Spaulding
244 0.987500 Abner Graves
244 0.987500 Major Abner Doubleday
244 0.987500 Jim Becker
244 0.987500 Queen
430 0.999219 Spain
430 0.987500 France
430 0.800000 western Europe
```

3- Link Parser and GATE-NE output

```
221 0.987500 James Ray
244 0.900000 Invented
244 0.987500 Abner Doubleday
244 0.987500 The Man Who
430 0.950000 in the north coast
430 0.800000 Spain
430 0.950000 between France and Spain
```

Results evaluation

The following is the evaluation results compared to the actual TREC expected answers, and the Mean Reciprocal Rank (MRR) over the total number of answers:

MRR = $\sum$ (1/rank) / Total number of questions

The output is formatted in three parts, which represent the following:

Question ID: Extracted answer, correct answer ranking

Mean reciprocal rank over Total questions question(s) is MRR

Number of questions had no answers found in top 5 responses

1- Original System Evaluation

Question 221: Correct answer James Ray, found at rank 2 (0.50).

Question 244: Correct answer Abner Doubleday , found at rank 4 (0.25).

Mean reciprocal rank over 2 question(s) is 0.375

0 questions had no answers found in top 5 responses

2- GATE-NE Evaluation

Question 221: Correct answer James Earl Ray , found at rank 1 (1.00).

Question 244: Correct answer Abner Doubleday, found at rank 2 (0.50).

Question 430: Correct answer Spain, found at rank 1 (1.00).

MRR(Mean reciprocal rank) over 3 question(s) is 0.833

no answers found in top 5 responses.

3- Link Parser and GATE-NE Evaluation

Question 221: Correct answer James Earl Ray, found at rank 1 (1.00).

Question 244: Correct answer Abner Doubleday, found at rank 2 (0.50).

Question 430: Correct answer Spain, found at rank 1 (1.00).

MRR(Mean reciprocal rank) over 3 question(s) is 0.833

0 questions had no answers found in top 5 responses.

Appendix B: GATE SYSTEM FUNCTIONS

This Appendix presents the Gate system structure and its main modules and sub-functions. It is based on Lavalée's Perl implementation paper [Lavalée, - 2003].

The GateInterface module represents our module interface to the GATE system. *ExtractNE* is the Module's main function, it takes an input sentence string and returns back a hash of arrays of all entities found in the given sentence:

- {entities}: is the array of entities' text
- {entitytypes}: is the array of entities' type such as (PERSON, ORGANIZATION, LOCATION, MONEY, DATE, PERCENT, TIME)
- {entitypos}: is an array of entities' offset in the original text

GateInterface's returns Hash arrays are stored in Gateinfo's object. Gateinfo is an encapsulation (i.e data structure) to store the GATE system output in a structured way. The following are the main functions of this module:

getSentence:	Returns original sentence that was sent to the GATE system for analysis
numElements:	Returns the number of elements that have been tagged (0 or more)
hasEntity:	Returns if there are entities in our sentence, and how many are there?
getText:	Returns the text that represents the retrieved entity.
getType:	Returns the entity type as PERSON, TIME
getRange:	Returns the text position in the sentence, where it starts and stops.

The following is an example of a call to GATE functions, it is based on question # 945 (TREC-10): "*Who discovered x-rays?*" Possible match will be: *Wilhelm Conrad Roentgen discovered x-rays on November 8 1895*. The following is the GATE output:

```
'entitystart' => [
                   '0',
                   '7',
                   '78',
                   '33'
                 ],

 'entitystop' => [
                   '6',
                   '11',
                   '93',
                   '56'
                 ],


 'entitytypes' => [
                   'Person',
                   'Date',
                   'Date',
                   'Person'
                 ],
 'entities' => [
                 'Thomas',
                 '1965',
                 'November 8 1895',
                 'Wilhelm Conrad Roentgen'  ]
```

Appendix C: LINK PARSER FUNCTIONS

This Appendix gives a brief description of all the link-types of the Link Parser. Please, refer Link Grammar Parser 4.1 web site for more details [http://www.link.cs.cmu.edu]:

A connects pre-noun ("attributive") adjectives to following nouns: "The BIG DOG chased me", "The BIG BLACK UGLY DOG chased me".

AA is used in the construction "How [adj] a [noun] was it?". It connects the adjective to the following "a".

AF connects adjectives to verbs in cases where the adjective is fronted, such as questions and indirect questions: "How BIG IS it?"

AL connects a few determiners like "all" or "both" to following determiners: "ALL THE people are here".

AM connects "as" to "much" or "many": "I don't go out AS MUCH now".

AN connects noun-modifiers to following nouns: "The TAX PROPOSAL was rejected".

AZ connects the word "as" back to certain verbs that can take "[obj] as [adj]" as a complement: "He VIEWED him AS stupid".

B serves various functions involving relative clauses and questions. It connects transitive verbs back to their objects in relative clauses, questions, and indirect questions ("The DOG we CHASED", "WHO did you SEE?"); it also connects the main noun to the finite verb in subject-type relative clauses ("The DOG who CHASED me was black").

BI connects forms of the verb "be" to certain idiomatic expressions: for example, cases like "He IS PRESIDENT of the company".

BT is used with time expressions acting as fronted objects: "How many YEARS did it LAST?".

BW connects "what" to various verbs like "think", which are not really transitive but can connect back to "what" in questions: "WHAT do you THINK?"

C links conjunctions to subjects of subordinate clauses ("He left WHEN HE saw me"). it also links certain verbs to subjects of embedded clauses ("He SAID HE was sorry").

CC connects clauses to following coordinating conjunctions ("SHE left BUT we stayed").

CO connects "openers" to subjects of clauses: "APPARENTLY / ON Tuesday , THEY went to a movie".

CP connects paraphrasing or quoting verbs to the wall (and, indirectly, to the paraphrased expression): "///// That is untrue, the spokesman SAID."

CQ connects to auxiliaries in comparative constructions involving s-v inversion: "SHE has more money THAN DOES Joe".

CX is used in comparative constructions where the right half of the comparative contains only an auxiliary: "She has more money THAN he DOES".

D connects determiners to nouns: "THE DOG chased A CAT and SOME BIRDS".

DD connects definite determiners ("the", "his") to certain things like number expressions and adjectives acting as nouns: "THE POOR", "THE TWO he mentioned".

DG connects the word "The" with proper nouns: "the Riviera", "the Mississippi".

DP connects possessive determiners to gerunds: "YOUR TELLING John to leave was stupid".

DT connects determiners to nouns in idiomatic time expressions: "NEXT WEEK", "NEXT THURSDAY".

E is used for verb-modifying adverbs which precede the verb: "He is APPARENTLY LEAVING".

EA connects adverbs to adjectives: "She is a VERY GOOD player".

EB connects adverbs to forms of "be" before an object or prepositional phrase: "He IS APPARENTLY a good programmer".

EC connects adverbs to comparative adjectives: "It is MUCH BIGGER"

EE connects adverbs to other adverbs: "He ran VERY QUICKLY".

EF connects the word "enough" to preceding adjectives and adverbs: "He didn't run QUICKLY ENOUGH".

EI connects a few adverbs to "after" and "before": "I left SOON AFTER I saw you".

EL connects certain words to the word "else": something / everything / anything / nothing , somewhere (etc.), and someone (etc.).

EN connects certain adverbs to expressions of quantity: "The class has NEARLY FIFTY students".

ER is used the expression "The x-er..., the y-er...". it connects the two halfs of the expression together, via the comparative words (e.g. "The FASTER it is, the MORE they will like it").

EZ connects certain adverbs to the word "as", like "just" and "almost": "You're JUST AS good as he is."

FL connects "for" to "long": "I didn't wait FOR LONG".

FM connects the preposition "from" to various other prepositions: "We heard a scream FROM INSIDE the house".

G connects proper noun words together in series: "GEORGE HERBERT WALKER BUSH is here."

GN (stage 2 only) connects a proper noun to a preceding common noun which introduces it: "The ACTOR Eddie MURPHY attended the event".

H connects "how" to "much" or "many": "HOW MUCH money do you have".

I connects infinitive verb forms to certain words such as modal verbs and "to": "You MUST DO it", "I want TO DO it".

ID is a special class of link-types generated by the parser, with arbitrary four-letter names (such as "IDBT"), to connect together words of idiomatic expressions such as "at_hand" and "head_of_state".

IN connects the preposition "in" to certain time expressions: "We did it IN DECEMBER".

J connects prepositions to their objects: "The man WITH the HAT is here".

JG connects certain prepositions to proper-noun objects: "The Emir OF KUWAIT is here".

JQ connects prepositions to question-word determiners in "prepositional questions": "IN WHICH room were you sleeping?"

JT connects certain conjunctions to time-expressions like "last week": "UNTIL last WEEK, I thought she liked me".

K connects certain verbs with particles like "in", "out", "up" and the like: "He STOOD UP and WALKED OUT".

<u>**L**</u> connects certain determiners to superlative adjectives: "He has THE BIGGEST room".

<u>**LE**</u> is used in comparative constructions to connect an adjective to the second half of the comparative expression beyond a complement phrase: "It is more LIKELY that Joe will go THAN that Fred will go".

<u>**LI**</u> connects certain verbs to the preposition "like": "I FEEL LIKE a fool."

<u>**M**</u> connects nouns to various kinds of post-noun modifiers: prepositional phrases ("The MAN WITH the hat"), participle modifiers ("The WOMAN CARRYING the box"), prepositional relatives ("The MAN TO whom I was speaking"), and other kinds.

<u>**MF**</u> is used in the expression "Many people were injured, SOME OF THEM children".

<u>**MG**</u> allows certain prepositions to modify proper nouns: "The EMIR OF Kuwait is here".

<u>**MV**</u> connects verbs and adjectives to modifying phrases that follow, like adverbs ("The dog RAN QUICKLY"), prepositional phrases ("The dog RAN IN the yard"), subordinating conjunctions ("He LEFT WHEN he saw me"), comparatives, participle phrases with commas, and other things.

<u>**MX**</u> connects modifying phrases with commas to preceding nouns: "The DOG, a POODLE, was black". "JOHN, IN a black suit, looked great".

<u>**N**</u> connects the word "not" to preceding auxiliaries: "He DID NOT go".

<u>**ND**</u> connects numbers with expressions that require numerical determiners: "I saw him THREE WEEKS ago".

<u>**NF**</u> is used with NJ in idiomatic number expressions involving "of": "He lives two THIRDS OF a mile from here".

<u>**NI**</u> is used in a few special idiomatic number phrases: "I have BETWEEN 5 AND 20 dogs".

<u>NJ</u> is used with NF in idiomatic number expressions involving "of": "He lives two thirds OF a MILE from here".

<u>NN</u> connects number words together in series: "FOUR HUNDRED THOUSAND people live here".

<u>NO</u> is used on words which have no normal linkage requirement, but need to be included in the dictionary, such as "um" and "ah".

<u>NR</u> connects fraction words with superlatives: "It is the THIRD BIGGEST city in China".

<u>NS</u> connects singular numbers (one, 1, a) to idiomatic expressions requiring number determiners: "I saw him ONE WEEK ago".

<u>NT</u> connects "not" to "to": "I told you NOT TO come".

<u>NW</u> is used in idiomatic fraction expressions: "TWO THIRDS of the students were women".

<u>O</u> connects transitive verbs to their objects, direct or indirect: "She SAW ME", "I GAVE HIM the BOOK".

<u>OD</u> is used for verbs like "rise" and "fall" which can take expressions of distance as complements: "It FELL five FEET".

<u>OF</u> connects certain verbs and adjectives to the word "of": "She ACCUSED him OF the crime", "I'm PROUD OF you".

<u>ON</u> connectors the word "on" to dates or days of the week in time expressions: "We saw her again ON TUESDAY".

<u>OT</u> is used for verbs like "last" which can take time expressions as objects: "It LASTED five HOURS".

OX is an object connector, analogous to SF, used for special "filler" words like "it" and "there" when used as objects: "That MAKES IT unlikely that she will come".

P connects forms of the verb "be" to various words that can be its complements: prepositions, adjectives, and passive and progressive participles: "He WAS [ANGRY / IN the yard / CHOSEN / RUNNING]".

PF is used in certain questions with "be", when the complement need of "be" is satisfied by a preceding question word: "WHERE are you?", "WHEN will it BE?"

PP connects forms of "have" with past participles: "He HAS GONE".

Q is used in questions. It connects the wall to the auxiliary in simple yes-no questions ("///// DID you go?"); it connects the question word to the auxiliary in where-when-how questions ("WHERE DID you go").

QI connects certain verbs and adjectives to question-words, forming indirect questions: "He WONDERED WHAT she would say".

R connects nouns to relative clauses. In subject-type relatives, it connects to the relative pronoun ("The DOG WHO chased me was black"); in object-type relatives, it connects either to the relative pronoun or to the subject of the relative clause ("The DOG THAT we chased was black", "The DOG WE chased was black").

RS is used in subject-type relative clauses to connect the relative pronoun to the verb: "The dog WHO CHASED me was black".

RW connects the right-wall to the left-wall in cases where the right-wall is not needed for punctuation purposes.

S connects subject nouns to finite verbs: "The DOG CHASED the cat": "The DOG [IS chasing / HAS chased / WILL chase] the cat".

SF is a special connector used to connect "filler" subjects like "it" and "there" to finite verbs: "THERE IS a problem", "IT IS likely that he will go".

SFI connects "filler" subjects like "it" and "there" to verbs in cases with subject-verb inversion: "IS THERE a problem?", "IS IT likely that he will go?"

SI connects subject nouns to finite verbs in cases of subject-verb inversion: "IS JOHN coming?", "Who DID HE see?"

SX connects "I" to special first-person verbs lke "was" and "am".

SXI connects "I" to first-person verbs in cases of s-v inversion.

TA is used to connect adjectives like "late" to month names: "We did it in LATE DECEMBER".

TD connects day-of-the-week words to time expressions like "morning": "We'll do it MONDAY MORNING".

TH connects words that take "that [clause]" complements with the word "that". These include verbs ("She TOLD him THAT..."), nouns ("The IDEA THAT..."), and adjectives ("We are CERTAIN THAT").

TI is used for titles like "president", which can be used in certain cirumstances without a determiner: "AS PRESIDENT of the company, it is my decision".

TM is used to connect month names to day numbers: "It happened on JANUARY 21".

TO connects verbs and adjectives which take infinitival complements to the word "to": "We TRIED TO start the car", "We are EAGER TO do it".

TQ is the determiner connector for time expressions acting as fronted objects: "How MANY YEARS did it last".

TS connects certain verbs that can take subjunctive clauses as complements - "suggest", "require" - to the word that: "We SUGGESTED THAT he go".

TW connects days of the week to dates in time expressions: "The meeting will be on MONDAY, JANUARY 21".

TY is used for certain idiomatic usages of year numbers: "I saw him on January 21 , 1990 ". (In this case it connects the day number to the year number.)

U is a special connector on nouns, which is disjoined with both the determiner and subject-object connectors. It is used in idiomatic expressions like "What KIND_OF DOG did you buy?"

UN connects the words "until" and "since" to certain time phrases like "after [clause]": "You should wait UNTIL AFTER you talk to me".

V connects various verbs to idiomatic expressions that may be non-adjacent: "We TOOK him FOR_GRANTED", "We HELD her RESPONSIBLE".

W connects the subjects of main clauses to the wall, in ordinary declaratives, imperatives, and most questions (except yes-no questions). It also connects coordinating conjunctions to following clauses: "We left BUT SHE stayed".

WN connects the word "when" to time nouns like "year": "The YEAR WHEN we lived in England was wonderful".

WR connects the word "where" to a few verbs like "put" in questions like "WHERE did you PUT it?".

X is used with punctuation, to connect punctuation symbols either to words or to each other. For example, in this case, POODLE connects to commas on either side: "The dog , a POODLE , was black."

<u>**Y**</u> is used in certain idiomatic time and place expressions, to connect quantity expressions to the head word of the expression: "He left three HOURS AGO", "She lives three MILES FROM the station".

<u>**YP**</u> connects plural noun forms ending in s to "'" in possessive constructions: "The STUDENTS ' rooms are large".

<u>**YS**</u> connects nouns to the possessive suffix "'s": "JOHN 'S dog is black".

<u>**Z**</u> connects the preposition "as" to certain verbs: "AS we EXPECTED, he was late".

Appendix D: Lingua::LinkParser Perl Module

Requirements

To install Lingua::LinkParser you must have already downloaded, compiled and install Link Parser package from http://www.link.cs.cmu.edu/link/.

This module has been compiled and tested with Perl 5.6 and 5.8 on Linux 2.2.13 & 2.2.14, and Perl 5.6 and 5.8 on OS X. Any incompatibilities *should* be the result of lib issues within the Link Parser itself, but these seem very stable.

Installation

To begin installation type: `perl Makefile.PL`

This will ask you where your Link Parser package directory is located, and must contain the distribution obj/, include/, and data/ directories, with obj/ containing compiled object files. This might look something like "/home/username/system-4.1/link-4.1".

Once the Makefile is written, you can build and test with:

```
make
make test
```

On Linux, the make displays several warnings about redefined macros, these messages may be ignored. The test will load the parser dictionary files and parse a sample sentence. If they do not, back up and figure out why before installation. To install:

```
make install
```

Documentation and help

Type perldoc linkparser.pm

Modules objects and commands

$parser = new Lingua::LinkParser(DictFile => 'PATH', KnowFile => 'PATH', ConstFile => 'PATH', AffixFile => 'PATH')

This returns a new Lingua::LinkParser object, loads the specified dictionary files, and sets basic configuration. If no dictionary files are specified, the parser will attempt to load the files using the path in global $DATA_DIR. This is a change from the Link Parser 3.0 implementation, where defaults were stored in the C API. The hash passed may also contain keys equivalent to the link parser options, in order to set these before a parser object is returned.

opts(OPTION_NAME_=_OPTION_VALUE,_...)"$parser->opts (OPTION_NAME => OPTION_VALUE, ...)

This sets the parser option OPTION_NAME to the value specified by OPTION_VALUE. A list of these options is found in the Link Parser documentation.

create_sentence(TEXT)"$sentence = $parser->create_sentence(TEXT)

Creates and assigns a sentence object (Lingua::LinkParser::Sentence) using the supplied value. This object is used in subsequent creation and analysis of linkages.

length"$sentence->length

Returns the number of words in the tokenized sentence, including the boundary words and punctuation.

num_linkages"$sentence->num_linkages

Returns the number of linkages found for $sentence.

num_valid_linkages"$sentence->num_valid_linkages

Returns the number of valid linkages for $sentence

num_linkages_post_processed"$sentence->num_linkages_post_processed

Returns the number of linkages that were post-processed.

null_count"$sentence->null_count

Returns the number of null links used in parsing the sentence.

num_violations"$sentence->num_violations

Returns the number of post processing violations for $sentence.

get_word(NUM)"$sentence->get_word(NUM)

Returns the word (with original spelling) at position NUM.

linkage(NUM)"$linkage = $sentence->linkage(NUM)

Assigns a linkage object (Lingua::LinkParser::Linkage) for linkage NUM of sentence

linkages"@linkages = $sentence->linkages

Assigns a list of linkage objects for all linkages of $sentence.

num_words"$linkage->num_words

Returns the number of words within $linkage.

get_words"$linkage->get_words

Returns a list of words within $linkage

words"$linkage->words

Returns a list of ::Word objects for $linkage.

num_sublinkages"$linkage->num_sublinkages

Returns the number of sublinkages for linkage $linkage.

compute_union"$linkage->compute_union

Combines the sublinkages for $linkage into one, possibly with crossing links.

violation_name"$linkage->violation_name

Returns the name of a rule violated by post-processing of the linkage.

sublinkage(NUM)"$sublinkage = $linkage->sublinkage(NUM)

Assigns a sublinkage object (Lingua::LinkParser::Linkage::Sublinkage) for sublinkage NUM of linkage $linkage.

sublinkages"@sublinkages = $linkage->sublinkages

Assigns an array of sublinkage objects.

get_word(NUM)"$sublinkage->get_word(NUM)

Returns the word for the sublinkage at position NUM.

words"$sublinkage->words

Returns a list of ::Word objects for $sublinkage.

num_links"$sublinkage->num_links

Returns the number of links for sublinkage $sublinkage.

text"$word->text

Returns the post-parse word text.

position"$word->position

Returns the number for the word's position in a sentence.

links"@links = $word->links

Returns a list of link objects for the word.

link(NUM)"$link = $sublinkage->link(NUM)

Assigns a link object (Lingua::LinkParser::Link) for link NUM of sublinkage

links"@links = $sublinkage->links

Assigns an array of link objects.

num_domains"$link->num_domains

Returns the number of domains for the sublinkage.

domain_names"$link->domain_names

Returns a list of the domain names for $link.

label"$link->label

Returns the "intersection" label for $link.

llabel"$link->llabel

Returns the left label for $link.

rlabel"$link->rlabel

Returns the right label for $link.

lword"$link->lword

Returns the number of the left word for $link.

rword"$link->rword

Returns the number of the right word for $link.

length"$link->length

Returns the length of the link.

linklabel"$link->linklabel

Only for link objects created via a word object, this returns the label for the link from the word object that created it.

linkword"$link->linkword

Only for link objects created via a word object, this returns the word text which the link points *to* from the object that created it.

linkposition"$link->linkposition

Only for link objects created via a word object, this returns the number of the word which the link points *to* from the object that created it.

get_diagram($linkage)"$parser->get_diagram($linkage)

Returns an ASCII pretty-printed diagram of the specified linkage or sublinkage.

get_postscript($linkage,_MODE)"$parser->get_postscript($linkage, MODE)

Returns Postscript code for a diagram of the specified linkage or sublinkage.

get_domains($linkage)"$parser->get_domains($linkage)

Returns formatted ASCII text showing the links and domains for the specified linkage or sublinkage.

print_constituent_tree($linkage,_MODE)"$parser-> print_constituent_tree ($linkage, MODE)

Returns an ASCII formatted tree displaying the constituent parse tree for $linkage. MODE is an integer with the following meanings: '1' will display the tree using a nested Lisp format, '2' specifies that a flat tree is displayed with brackets, and '0' results in no structure, a null string being returned

Bibliography

Brill, Eric. A Simple Rule-Based Part Of Speech Tagger. *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing.* Trento, Italy. 1992. *pages 152-155.*

Brill E., Lin J., Banko M., Dumais S. and Ng A., Data-Intensive Question Answering, *In Proceedings of the Eleventh Text Retrieval Conference.* NIST, Gaithersburg, MD, *(TREC-2002),* 2002.

Chalendar, G., T. Dalmas, F. Elkateb-Gara, O. Ferret, B. Grau, M. Hurault-Plantet, G. Illouz, L. Monceaux, I. Robba, and A. Vilnat, The question answering system QALC at LIMSI: experiments in using Web and WordNet, *In Proceedings of the 11th Text Retrieval Conference.* NIST, Gaithersburg, MD, *(TREC-2002),* 2002.

Cunningham, H., Y. Wilks, and R. Gaizauskas, GATE- a general Architecture for text Engineering. *In proceedings of the 16th International Conference on Computational Linguistics* (Coling-1996), Copenhagen, August, 1996.

Cunningham, H., Maynard, D., and Bontcheva, K., and Tablen V, GATE: A framework and graphical development environment for robust NLP tools and applications. *In proceedings of the 40t Anniversary meeting of the Association for Computational Linguistics.* 2002.

Dan, Brian. Parsing Natural Lnguage with Lingua::LinkParser - *The Perl Journal*, a *commercial publications*, Volume 5, Number 3 (#19), Fall 2000.

Giuseppe A., Cisternino A., Formica F., Simi M., Tommasi A., PiQASso:Pisa Question Answering system. *Technical Report* Dipartimento di Informatica, Università di Pisa, Italy, 2001

Harabagiu S., Pasca M. and Maiorano Experiments with open-domain textual question answering, *In proceedings of the 18th International Conference on Computational Linguistics* (Coling-2000), Saarbrucken, July 31st to August 4th 2000.

Hirschman, L. and R. Gaizauskas. Natural language question answering; The view from here. *Journal of Natural Language Egineering. Special Issue on Question Answering Engineering*, Fall-Winter 2001.

Katz, B. Using English for indexing and retrieving. In P.H. Winston and S.A. Shellard, editors, *Artificial Intelligence at MIT: Expanding Frontiers, volume 1. MIT Press.* 1990.

Katz, B. Annotating the World Wide Web using natural language. *In Proceedings of the 15th RIAO conference on Computer Assisted Information Searching on the Internet* (RIAO'97). Montreal, Canada, June 1997.

Kupiec, J. MURAX: A robust linguistic approach for question answering using an on-line encyclopedia. *In 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp 181-190, Pittsburgh, 1993.

Light, G. Mann, E. Riloff, and E. Breck. Analyses for elucidating current question answering technology. Technical Report. *Journal of Natural Language Engineering* Fall-Winter 2001.

Lin, J. and Katz, B. REXTOR: A System for Generating Relations from Natural Language. *MIT Artificial Intelligence Laboratory* Cambridge MA. 2001.

Guillemette L. QUANTUM's Web-QA Component. *Technical Report,* Concordia University, Montreal Canada, 2002.

MUC-7 (1998) *Proceedings of the Seventh Message Understanding Conference* (MUC-7), published on the website _http://www.muc.saic.com/

Plamondon L, Kosseim L, Lapalme G. The QUANTUM Question-Answering System at TREC-11. *In Proceedings of the Eleventh Text Retrieval Conference (TREC-11).* pp 670-677. Gaithersburg, Maryland, 2002.

Robin L. Perl implementation of the QUANTUM's Web-QA Component. *Technical Report,* Concordia University, Montreal Canada, August 2003.

Temperley, D., Sleator, D. and Lafferty, J. Parsing English with Link Grammar, technical report CMU-CS-91-196, Department of Computer Science, Carnegie Mellon University, 1991.

Scott, S. and Gaizauskas R. QA_LaSIE: A Natural Language QA system, *Advances in Artificial Intelligence*, 14th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI 2001, Ottawa Canada, June 7-9, 2001. pp 172—182.

Srihari, R. and W. Li: Information extraction supported question answering. *In Proceedings of the Eight Test Retrieval Conference (TREC-8),* NIST, 1999.

Stratica, N. A natural language processor for querying. *Master's thesis Concordia University*, Montreal, Canada 2002.

Voorhees, E. and D. M. Tice. Overview of the TREC-9 question answering track. *In proceedings of the Ninth Text Retrieval Conference. (TREC-9)*, pp 71—80, NIST, Gaithersburg, MD, 2000.

Voorhees, E. Overview of the TREC-2001 question answering track. *In Proceedings of the Tenth Text Retrieval Conference.* pp. 157--165, NIST, Gaithersburg, MD, *(TREC-2001)*, 2001.

Voorhees, E. Overview of the TREC-2002 question answering track. *In Proceedings of the Eleventh Text Retrieval Conference.* pp. 57--66, NIST, Gaithersburg, MD, *(TREC-2002),* 2002.

Zhang, H. ICT Experiments in TREC-11 QA Main Task, Institute of Computing Technology, *In Proceedings of the Tenth Text Retrieval Conference*, NIST, Gaithersburg, MD, *(TREC-2001)*, 2001.

Zheng, Z. AnswerBus AnswerBus Question Answering System. *Proceeding of HLT Human Language Technology Conference (HLT 2002).* San Diego, CA. March 24-27, 2002.