USING REFORMULATIONS TO IMPROVE QUESTION ANSWERING

Jamileh Yousefi

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

September 2005

Your file Votre référence
ISBN: 0-494-10299-3
Our file Notre référence
ISBN: 0-494-10299-3

Abstract

Using reformulations to improve question answering

Jamileh Yousefi

The goal of open domain question answering is to find at least one possible form of an answer to a given question. Answer formulation is one strategy used to retrieve candidate answers in natural language documents. The ideal answer formulation should not retrieve incorrect answers but should also identify many candidate answers. While many research projects use a set of hand-made patterns for answer formulation, others try to automatically acquire answer formulation, using some linguistic rules.

This project has two major goals. In the first phase, we present a method for the automatic acquisition of reformulations from natural language questions. We hope to find useful and generic reformulation patterns, which can be used in our question answering system to find candidate answers. We use some examples of different types of questions and their right answers as training set and the system automatically extracts the patterns from retrieved documents. The system also calculates and assigns a weight to each of the extracted patterns. These patterns are added to the patterns set of the current Web-QA component. In the second part, we present a method to filter out noisy candidate answers and re-rank candidate answers in our Web-QA module by using semantic relations.

We use the original Web-QA module of QUANTUM and developed the Web-QA2 module for this thesis. We use the 1400 questions of the TREC-8, TREC-9, and TREC-10 competitions for training to develop the reformulation patterns and the filtering algorithm and use the 500 questions of TREC-11 for testing.

The experimental evaluation of our learning reformulations algorithm shows that using new generated patterns increase the precesion and MRR. The potential improvement is more than what is shown in the results because many of the generated patterns are not usable by Web-QA due to lack of well grained distinction of verbs.

Also the results from filtering out bad answers evaluation shows the significant improvment in MRR and precesion. It means although we provide less candidates, they are more likely to constitute correct answers than before.

Keywords

Question Answering, Web as a Linguistic Resource, Semantic Relations, Question Reformulation.

Acknowledgments

My sincere appreciation to many people who made this masters thesis possible. Special thanks to my supervisor, Dr. Leila Kosseim for her invaluable and continuous support. I would also like to thank the other committee members of my thesis who took effort in reading and providing me with valuable comments on the earlier version of this thesis: Dr. Sabine Bergler and Dr. Volker Haarslev.

Many thanks go to my husband, Mohammadreza, who gave all the encouragement and energy to work in a challenging academic environment. I would also acknowledge with much gratitute the crucial role of CLaC Laboratory members for their great ideas and extensive comments. Many more persons accompanied me and participated in various ways to ensure my project succeeded and I am thankful to them all.

I dedicate this thesis to my mother and father.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Information Retrieval (IR) systems, such as Google, are designed to help people find documents that match their information needs in large collections (ex. the Web). As the number of documents available on the Internet grows, finding necessary information is becoming a very difficult task. Although information retrieval systems are able to find relevant documents, they cannot extract useful information from these documents. Most of these IR systems work with keyword-based queries involving only a few words and not a complete question in natural language. In the past three decades, many research projects have investigated and developed systems which can deal with a precise question, and find the most accurate answer. Question answering is an example of systems that can perform this task.

In this chapter, we review some basic concepts of question answering and we describe the general architecture of most open domain question answering systems.

1.2 Question answering (QA)

Question answering (QA) tries to retrieve answers rather than documents in response to a fact-based question in natural language. With a QA system, the user is not responsible for analyzing the content of relevant documents to find an answer, it is the QA system's responsibility to find the best answer from the retrieved documents. In a question answering system, the user types a fact-based question (eg. *'Who is the prime minister of Canada?'*) and the system tries to produce one or more candidate answers for this question, from a corpus, or from the web. For example, given the following question, *'Who founded American Red Cross?'* the system searches the document collection and extracts *'Clara Barton'* from the retrieved documents and returns this answer to the user, instead of a set of retrieved documents.

1.3 Open-domain QA vs. Closed-domain QA

There are several factors that determine the complexity of the QA task. One of these factors is the scope of the questions. The scope of questions can be categorized into two types: closed-domain and open-domain.

- **Closed-domain:** Early question answering systems such as BASEBALL [Green et al.1961] and LUNAR [Woods.1973], for example, were closed-domain QA systems. This means that they only answered questions relating to a specific domain. For example, BASEBALL answered questions about baseball games that had been played in the American league over a single season and LUNAR was designed to answer geologist questions on moon rocks and soil composition gathered by Apollo 11. A closed-domain QA system accepts a question, analyzes it syntactically and semantically, by applying some domain-specific rules and heuristics, and then reformulates it into the form of a query that is then run on a specific database.

- **Open-domain:** More recently, most work has been performed on open-domain question answering systems. There are no restriction on the scope of questions that the user can ask in open-domain QA. These systems focus on extracting the best answer string for natural language questions from large text documents (ex. the Web).

  QUANTUM [Plamondon et al.2001] and Web-QA [Kosseim et al.2003], used in this thesis, are two examples of open-domain QA.

1.4 Resources

One other factor that plays an important role in the performance of QA systems is the resources they use. Resource data, Web search engines, encyclopedia, dictionaries, and natural language processing (NLP) tools are some resources that can help the QA task. In this section we briefly explain some of theses resources that have been used in our work.

1.4.1 Resource data

Resource data is defined as the document collection where the answer is found. The resource data could be structured data, such as a database, or unstructured data, such as free text or the Web. For example, the LUNAR [Woods.1973] system uses structured data in the form of a database while Web-QA uses the Web as a resource, to find answers.

1.4.2 Deriving semantic similarity using WordNet

WordNet [Miller1995] is a very common semantic resource that has been employed in many QA systems. WordNet is an on-line combination of dictionary and thesaurus, created by the Cognitive Science Laboratory, at Princeton University, to support automatic text analysis. English words including nouns, verbs, adjectives and adverbs, are grouped into a set of synonyms called synsets. A number of relations based on the types of words hold between synsets of the words. These relations include synonymy (similar-to), hypernymy/hyponymmy (ISA or above/below relation), meronymy/holonymy (the PART-OF or part/whole relation), and antonymy (opposite).
Here we list some of these relationships for the word *"board"* in WordNet:

1. Synset:

   There are eight synsets containing senses for this word, here we just mention two of them:

   ```
   SYNSET1: {board, plank}
   SYNSET2: {control panel, display panel, panel, board}
   ```

2. Hyponymy:

   ```
   SYNSET1 is a hyponym of {lumber, timber}
   SYNSET2 is a hyponym of {electrical device}
   ```

3. Meronymy:

   ```
   SYNSET1: {board, plank}
   ```

```
            HAS PART: knot

            HAS PART: knothole
```

"knot" and *"knothole"* are meronyms of *"board"* with the sense in SYNSET1

```
SYNSET2: {control panel, display panel, panel, board}

            HAS PART: idiot light
```

"idiot light" is meronym of *"board"* with the sense in SYNSET2

4. Hypernymy:

```
SYNSET1: {board, plank}
    => lumber, timber
        => building material
            => artifact, artefact
                => object, physical object
                    => entity
               => whole, whole thing, unit
                   => entity

SYNSET2:  {control panel, display panel, panel, board}
     => electrical device
         => device
            => instrumentality, instrumentation
               => artifact, artefact
                   => object, physical object
                       => entity
                   => whole, whole thing, unit
                       => entity
```

5. Polysemy count:

WordNet also provides a short definition for each word as well as polysemy count, i.e. the number of synsets that contain the word. For *"board"*, we find:

```
board used as a verb is uncommon (polysemy count = 4)
The verb board has 4 senses (first 2 from tagged texts):
1. (5) board, get on -- (get on board of (trains, buses, ships, aircraft, etc.))
2. (2) board, room -- (live and take one's meals (in a certain place))
3. board -- (lodge and take meals (at))
4. board -- (provide food and lodging (for))
```

1.4.3 NLP Tools

Natural language processing (NLP) tools are used as an important resource in developing QA. Most of them are employed to analyze the text, for instance a Named-Entity tagger is used to detect proper nouns in a sentence. There exist several NLP tools available to the research community. A Part-of-speech tagger, a Noun phrase chunker, a Named-Entity tagger, a Stemmer, and a Sentence Splitter are some NLP tools that have been used in our work.

Part-of-speech tagger

In most languages, words may have more than one part-of-speech. For example, in English, the word *'can'* may be an auxiliary verb or a noun. To determine the appropriate part-of-speech of a given word in a sentence, we need to consider the context in which the word appears. For instance, consider the sentence *'We can can the can'*, CAN could be either noun or verb or auxiliary verb. After classification and disambiguation of the word tokens of the sentence, a part-of-speech tagger will assign the following parts of speech tags to the words in the sentence:

```
We/Pron can/Aux can/Verb the/Det can/Noun.
```

Part-of-speech tagging could be done manually or automatically. To date, many automatic part-of-speech taggers have been developed in order to be used in NLP applications. Automatic part-of-speech taggers employ different approaches including stochastic and rule-based methods. Stochastic taggers use pre-tagged corpora to learn word/tag and tag sequence frequencies to be used in the tagging process. Rule-based taggers use context rules to disambiguate the words.

Noun phrase chunker

Text chunking is the process of dividing the sentence into grammatical constituents that do not overlap. For example, consider the following sentence:

```
He reckons the current account deficit will narrow to only 1.8 billion
in September.
```

This sentence can be chunked as follows:

```
[NP He ] [VP reckons ] [NP the current account deficit ] [VP will
narrow ] [PP to ] [NP only 1.8 billion ] [PP in ] [NP September ]
```

A noun phrase chunker identifies the chunks of the sentence that consist of noun phrases.

Named-Entity tagger

A Named Entity tagger is an NLP tool used to identify predictable forms corresponding to generic objects such as names (persons, organizations, locations), times (date, hour), and quantities (monetary values, percentages). For example, consider the sentence *'The earthquake which devastated the southeast Iranian city of Bam on December 26 killed 26,271 people, announced by Bam governor Ali Shafii.'* A named-entity tagger will tag *"Bam"* as a location, *"December 26"* as a date, *"26,271"* as a number and *"Ali Shafii"* as a person name. Information extraction systems use named-entity taggers extensively.

Stemmer

Stemming is the process of stripping the inflection and derivation (prefixes, suffixes) of a given word. A stemmer can reduce the variant forms of words (i.e. *"computer"*, *"computation"*) to the common form (i.e. *"compute"*). Stemmers are liable to make mistakes, for example, sometimes the result of the stemming process may not be a proper word and may not have a meaning. Stemmers are commonly used in information retrieval systems, because the stems are more common in documents than the infelcted/derivated form of the words from query. For example, if a query contains *"definition ..."*, the documents containing the term *"define"* may also be relevant to this query.

The Porter stemmer [Porter1980] is an algorithm for stripping the suffixes of English words. This algorithm has been written, in 1980, by Porter, and has been implemented in C++, Java, Perl as well as other languages. We use the Porter stemmer in this thesis to find relevant passages with a higher recall.

Sentence splitter

To split the documents into sentences, we used the sentence splitter developed by Paul Clough [Clough2000] at Sheffield University. The splitter uses very simple rules. This splitter considers the characters ".", "!", and "?" as a sentence terminal if they are followed by at least one whitespace and a capital letter. The splitter also uses some heuristics to check whether dot-ending words are abbreviation or not. Consider the following paragraph taken from [Office of the President1998]:

```
''PLANNING PROGRESS REPORT. September 18, 1998. The discussion today at
 the All-University Planning Retreat will officially launch the
 strategic planning process. So far, the work has been designing the
 process and garnering advice from various persons within the
 University and without.''
```

The splitter will divide the paragraph into the following four sentences:

```
1) PLANNING PROGRESS REPORT.
2) September 18, 1998.
3) The discussion today at the All-University Planning Retreat will officially launch
   the strategic planning process.
4) So far, the work has been designing the process and garnering
advice from various persons within the University and without.
```

1.5 Architecture of an open-domain QA

Every year, several open domain question answering systems are presented at the QA track of the TREC competition (see section 1.7). While each of the participating groups has a different approach and architecture, most of the systems are structured in four main modules. We can identify these modules as:

- Question analysis
- Passage retrieval

- Answer extraction
- Answer selection

1.5.1 Question analysis

The first step of a typical QA system is to analyze the question expressed in natural language in order to determine the expected semantic type of the answer and to extract keywords that help the system locate sentences where answers can probably be found. For instance, given the question *'How old is General Clark?'*, the most likely type of answer (Age) can be inferred by *'How old'*. For the question *'Who was the first Russian astronaut to walk in space?'*, the system gives five keywords *'first'*, *'Russian'*, *'astronaut'*, *'walk'*, and *'space'* that can be used to build the search query for the retrieval module.

The question analysis module may include some morphological, syntactic, and semantic analysis of the question. Different systems use different techniques in their question analysis modules. Some systems use the parts of speech of tokens to find the keywords and some systems determine the question type using regular expressions based on part-of-speech tags. Some systems also use shallow parsing and semantic information to determine the question type [Plamondon et al.2001].

For example WEBCLOPEDIA [Hovy et al.2000] used the CONTEX [Hermjakob1997] parser to parse the question and to find the desired semantic type of answer. Also the question analyzer extracts single and multi-word units (content words) and then uses these units to form a query.

QUANTUM [Plamondon et al.2001] performes this via a set of 60 patterns and rules based on words, part-of-speech tags and noun-phrase tags. For example, for the question *'How many people in U.S. do not have health insurance?'*, the following rule is used to analyze the question:

```
how many <noun-phrase NP1>, type = cardinality, focus = NP1
```

and the following results comes out from this module:

```
keyword: people, U.S., health, insurance
focus: people
answer type: cardinality
```

1.5.2 Passage retrieval

Almost all QA systems use various techniques to reduce the search space for potential answers and select the passages that most likely contains the answer. For example, consider the question *'Who is the prime minister of Australia?'* One would extract some keywords, for example, *'prime minister'* and *'Australia'*, search the collection of documents or the Web with a query which combines these keywords. The passage retrieval plays an important role in the overall performance of a QA system because if the passage retrieval cannot find relevant passages, even the perfect answer extraction module cannot extract any answer for the question. Okapi is an example of a passage retrieval engine [Robertson et al.1994]

We briefly describe some of the passage retrieval techniques employed in different systems:

- **Using external knowledge and tools**

  Most current question answering systems employ external knowledge and linguistic tools, such as Named-Entity taggers, WordNet [Miller1995], syntactic parsers, semantic relations, and ontology lists, to extract paragraph-sized chunks of the retrieved documents.

- **Hand-made patterns set:** Soubbotin et al. [Soubbotin and Soubbotin2001] used a fairly extensive list of surface patterns for passage retrieval. This approach searches in the document collection for an exact sentence that could be the formulation of the potential answer. Hermjakob [Hermjakob et al.2002] used semantically equivalent paraphrases of the question to reduce the gap between question and answer context. For example, given the question *"How did MAHATAMA GANDHI die?"* the system paraphrases it into 30 various forms such as *"Mohatma Gandie died < how >"* or *"< Who > killed Mohatma Gandie"* and then searched for these paraphrases in the document collection.

- **Automatically learned patterns:** A number of researchers have then implemented algorithms to acquire the optimal answer patterns automatically from a free text or the Web. They provide the system with a set of questions and their answers, then the system extracts patterns from the Web or a corpus for these examples. These patterns are then used as potential answer patterns to select top passages containing the answer.

1.5.3 Answer extraction

After running the document retrieval module, some passages are identified that are likely to contain the answer. Candidate answers are extracted from these selected passages. A variety of answer extraction techniques have been implemented in different question answering systems:

- **Pattern matching:** Pattern matching is a common technique that has been used in many systems for answer extraction. Clarke and Cormack [Clarke et al.2000] applied pattern matching techniques to scan and extract the potential answers that match the answer category. Extracted answers are then ranked using various quality heuristics.

- **Logic-based answer extraction technique:** In this technique, the core meaning of documents is identified by syntactic and semantic analysis and represented as a Minimal Logical Form (MLF). MLFs consist of a set of existentially closed formulas, expressed by Horn clauses. For example, consider the sentence *"Lambs eat grass"*, the MFL of this sentnce would be :

  object(Lambs, x1) , event(eat,[x1,x2]), object(grass, x2).

  where, "event" represents "eventuality" derived from the verb involving two objects and "x1" and "x2" represent the objects involved in the *"eating"* event.

- **Named entity recognition:** In this approach, the questions are classified into predefined categories based on answer type. The top ranked retrieved passages are scanned for potential entities. Then those entities that match the answer type of the question are considered as the answer. The answers are then ranked based on frequency and distance to the keywords of the question.

1.5.4 Answer selection

At the final stage of the question answering process, the QA system decides which one of the candidate answers should be returned by the system. Answer selection is the process of selecting and ranking the most probable answers from a set of potential answers. Many systems have investigated different approaches to answer selection. The BBN team [Xu et al.2003] ranked the candidate answers using the search engine that has been used in the document retrieval module. They selected the final answers using a few constraints on noun phrases and verb phrases. For example, they check if the candidate answer satisfies the verb arguments of the question. Moldovan, Pasca, and Harabagiu [Moldovan et al.2003], use a knowledge-intensive approach for the answer selection module. Knowledge-intensive approach uses structured and semi-structured data sources to determine the semantic type of answers with high confidence. In contrast to the knowledge-intensive approach, a redundancy-based approach employ the redundancy of the Web to re-rank (rather than filter out) candi-date answers found in the collection, by using web search engine searching for question and answer terms [Magnini2002].

The Web-QA component confirms the candidate answers by combining the answers coming from the Web and QUANTUM answers extracted from the TREC collection [Plamondon et al.2002]. The answer selection task is more challenging when different techniques and multiple information resources are used in the QA system. In this case the final answer must be selected from several sets of ranked candidate answers found by different modules [Jijkoun and de Rijke2004].

1.6 QA evaluation metrics

QA evaluation measures the effectiveness of the system. We refer to effectiveness of a system as measuring the ability of the system to retrieve relevant documents. Precision and recall are two metrics for measuring the effectiveness of a text retrieval system [Hirschman L1995]. Mean reciprocal rank (MRR) is another metric for evaluation the QA task which is conducted by TREC (see section 1.7).

In the following sections we briefly explain precision, recall and MRR measures.

1.6.1 Precision and recall

Before the detailed discussion of precision and recall, we explain the terms that have been used in the definition.

The term "relevant document" refers to a document that contains at least one sentence which is relevant to a specific query. In another way, in information retrieval a relevant document is a correct output.

The "term retrieved" documents refers to all documents that have been retrieved for a specific query.

Using the above terms, we define the precision as the proportion of retrieved documents which are relevant and recall is the proportion of relevant documents that are retrieved [Rijsbergen1999].

Assuming that:

Q1 = The specific query that has been sent to the system

RELEVANT = Number of relevant documents for Q1

RETRIEVED = Number of documents that the system has retrieved for Q1

RELEVANT-RETRIEVED = Number of relevant documents that have been retrieved for Q1

TOTAL = Total of documents

Precision = RELEVANT-RETRIEVED/ RELEVANT

Recall = RELEVANT-RETRIEVED / RETRIEVED

The following example demonstrates the computation of precision and recall.

Assuming that the collection contains 9 documents which 5 of them are relevant to the query (Q1).

```
TOTAL = 9

RELEVANT = 5
```

In the first run of the system, the following results are obtained:

```
RETRIEVED = 3

RELEVANT-RETRIEVED = 2


Precision = 2/3          Recall = 2/5
```

1.6.2 Mean Reciprocal Rank

To calculate the MRR, each question in the test set receives a score based on the rank of its first correct answer. For example, if the first four answers to a question are wrong and the fifth one is correct, then the question receives the score 1/5, and if no one correct answer has been found, the score for the question is zero. Then, the mean of this score over the set of questions in the test set is calculated and returned as MRR. Also, the system reports the number of questions that do not have any correct answer separately.

1.7 TREC

In 1992, TREC (The Text Retrieval Conference) was established by NIST[1] to encourage research and share improvements in information retrieval systems on large text collections. TREC provides the required resources and appropriate evaluation methodologies for evaluation of text retrieval systems. Since 1999, the TREC QA Track was added to the TREC program to investigate the evaluation of question answering systems. In each TREC track, participants are given a set of fact-based short answer questions and a set of documents. Each participant must run their own program with the given data and return a ranked list of the top five correct answer accompanied by their supporting document id. NIST then uses the PP-Eval[2] program to evaluate the results and judge the accuracy of the systems.

To judge the responses, PP-Eval uses answer patterns. The answer patterns consists of a ranked list of up to five responses per question.

The measurement for evaluating the QA task is the mean reciprocal rank (MRR).

1.8 Goals of the thesis

This thesis will work in the context of QA systems to target the specific problem domain of reformulation and improve the passage retrieval task.

The research in this thesis is concerned with improving an existing QA system, Web-QA Component.

The first goal is to learn question reformulations automatically from natural language questions. We hope to find useful reformulation patterns, which are general and can be used in any question answering system. We hope this task will improve the precision and recall rate of our QA system. Our approach also deals with synonyms or different possible spellings by expanding the keywords using different spellings of question terms. Our second goal is to improve the results by filtering out the *noisy* candidates and re-rank the remaining candidates better.

At first, we describe QUANTUM and the Web-QA Componet. Then we state the problems concerning the Web-QA Component performance, and we propose our solution for each of the

[1]National Institute of Standards and Technology

[2]PP-Eval is a Perl script used to evaluate the QA systems. PP-Eval calculates the score for each question based on the reciprocal of the rank at which the first correct answer is returned (MRR).

problems.

1.8.1 QUANTUM

The QUANTUM [Plamondon et al.2001] system, developed at the University of Montreal, participated in the TREC-10[3] competition with the goal of finding a short answer to well-formed natural language questions from a large document collection. This system uses some computational linguistics and information retrieval techniques.

The input to the system is a fact-based question and the output is a ranked list of five best candidate answers accompanied by their source-document number. Figure 1, taken from [Plamondon et al.2001], shows an example of QUANTUM input and output.

```
Question: Where are the British crown jewels kept?
Answer:
1 FT921-14782 are kept in Edinburgh Castle - together with jewel
2 AP901114-0171 kept in the Tower of London as part of the British
3 AP900620-0160 treasures in Britain's crown jewels. He gave the K
4 NIL
5 AP900610-0018 the crown jewel settings were kept during the war.
```

Figure 1: Example of a QUANTUM input and output

The architecture of the QUANTUM system can be described in five components: question analysis, passage retrieval and tagging, candidate extraction and scoring, expansion to 50 characters, and NIL insertion (for no-answer questions).

Given a question, for example *'How many people die from snakebite poisoning in the U.S. per year?'*, the system divides it into three parts: the question word, the focus, and the discriminant. The focus determines what the question is about and is used in the answer extraction phase. The discriminant, which is what remains of the question after having removed the question word and focus, helps to score the candidate answers. Figure 2, taken from [Plamondon et al.2001], shows this process on a sample question. The question is from TREC-9 (302) and the answer is from the TREC collection (document LA082390-0001).

In order to perform this task, QUANTUM uses a tokenizer, a part-of-speech tagger, and a noun phrase chunker.

Okapi [Robertson et al.1994] has been used as a document retrieval engine in QUANTUM. Okapi

[3]see section 1.7, for more information about TREC

$$\text{Q:}\quad \underbrace{\textit{How many}}_{\text{question word}} \quad \underbrace{\texttt{people}}_{\text{focus}} \quad \underbrace{\textit{die from snakebite poisoning in the U.S. per year?}}_{\text{discriminant}}$$

$$\text{A:}\quad \underbrace{\textit{About 10 people}}_{\text{candidate}} \quad \underbrace{\textit{die a year from snakebite poisoning in the United States.}}_{\text{variant of question discriminant}}$$

Figure 2: Example of question and answer decomposition in QUANTUM.

returns relevant paragraphs instead of entire documents. The system sends a question as a query to the search engine and the search engine returns 30 paragraph-sized documents. To extract candidate answers from these paragraphs QUANTUM uses 40 hand-made patterns called "extraction functions". For example, to find the answer for the question *'Who was the first woman to cross the Pacific Ocean?'*, the system uses the template 'A: < PERSON-NAME >' to extract candidate answers. It then ranks the candidate answers based on the extraction score, the passage score, and the proximity score. At the end, each candidate answer is expressed to meet the regulations for answer snippets in a TREC competition. The system takes 50-character substrings around the word and cuts the truncated words at both sides.

In the TREC-10 QA track, QUANTUM's best run received a MRR of 0.1911, while the average of all systems was 0.234 [4].

1.8.2 The Original Web-QA Component

For the TREC-11 competition, the Web-QA component was developed to improve the QUANTUM QA system. This Web module searches the Web for candidate answers that are then combined with QUANTUM answers found in the TREC collection to confirm a possible answer [Plamondon et al.2002]. Figure 3 shows the overall view of the Web-QA component.

The Web-QA component uses simple question reformulation rules to generate answer patterns and then searches for these answer patterns on the Web. For example, given the question *'Who is the prime minister of Canada?'*, the system generates the answer pattern `'The prime minister of Canada is` < PERSON-NAME >' and searches the Web for this exact phrase. The noun phrase

[4]see section 1.7 for a explanation of the MRR measure

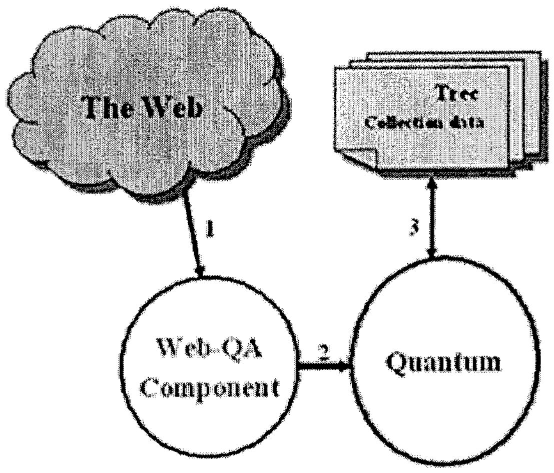

Figure 3: Web-QA Component architecture

Formulation Template	Example
When did ANY-SEQUENCE-WORDS-1 VERB-simple?	When did the Jurassic Period end?
ANY-SEQUENCE-WORDS-1 VERB-past TIME	the Jurassic Period ended TIME
TIME ANY-SEQUENCE-WORDS-1 VERB-past	`TIME the Jurassic Period ended`
TIME, ANY-SEQUENCE-WORDS-1 VERB-past	`TIME, the Jurassic Period ended`

Figure 4: Example of a reformulation template

following this pattern is then extracted and considered as a potential answer. Simple syntactic and semantic checks are also performed to ensure that the noun phrase is indeed a PERSON-NAME[Plamondon et al.2002].

To formulate an answer pattern from a question, the system turns the latter into its declarative form using a set of hand-made formulation templates that test for the presence of specific keywords, grammatical tags, and regular expressions. Figure 4 shows an example of a reformulation. The formulation template is composed of 2 sets of patterns: a question pattern that defines what the question must look like and a set of answer patterns that defines a set of possible answer formulations. The patterns take into account specific keywords (eg. `When did`), strings of characters (eg. `ANY-SEQUENCE-WORDS`), and part-of-speech tags (eg. `VERB-simple`). Answer patterns are specified using the same type of features plus a specification of the semantic class of the answer (eg. `TIME`). The semantic classes are used during answer extraction in order to validate the nature of the candidate answers from the document.

In total, 77 formulation templates are used. The templates are tried sequentially and all question patterns that are satisfied are activated.

1.8.3 Current problems with the Web-QA component

In order to identify areas of strength and weakness of the Web-QA component, and develop ways to improve the system, we evaluated the system with the standard TREC evaluation data and several measures.

In this section, we briefly describe the results of the evaluation, the problems that we found, and our proposed solution for each of those problems.

Evaluation of the reformulation:

- **Observation:** Table 1 shows that although pattern coverage is very high (89.9% and 90.0%), the candidate answer cannot be retrieved for many of the reformulated questions (9.1% and 51.3%). The pattern coverage is high, meaning that in most cases, for each question there exists at least one rule (question pattern) in the template table that could be triggered for this question, but still the document retrieval cannot find any occurrence of the answer, because the existing patterns for the triggered rule do not match with any answer context.

- **Problem:** The answer patterns in the pattern set drive the Web-QA component system to find answers with good linguistic quality, but the variety of these patterns are not enough to cover many forms of the answer context. For example given the question *'Who is Desmond Tutu?'*, there exists only one answer pattern as : 'x "is" TITLE' for the question pattern *'Who (v IN "is" "was") (x Proper)?'* , while the answer may occur in the answer context in one of the following forms:

```
(x TITLE )
(x "as" TITLE )
(x "known" "as" TITLE )
(x "became" TITLE )
```

In Tables 6, 7, 8, 9, 10, and 11, we show some formulation templates which are not implemented in the 77 existing templates. These templates are categorized based on question type.

Corpus	No. of questions	No. of questions with at least one reformulation	No. of reformulated questions with at least one answer
TREC-9	694	624 (89.9%)	63 (9.1%)
TREC-10	500	450 (90.0%)	256 (51.3%)

Table 1: Evaluation of the reformulation task

Corpus	No. of questions	No. of questions with at least one reformulation	No. of questions with a correct answer in the top 5 candidates	Precision of candidate list
TREC-9	694	624 (89.9%)	17 (2.4%)	0.270
TREC-10	499	450 (90.2%)	153 (30.7%)	0.598

Table 2: Results of the Web-QA component

- **Proposed solution:** Since manually entering the above templates into the system is a tedious task, we propose to learn reformulation patterns from natural language questions and then add the learned patterns to the Web-QA component system.

Evaluation of the answer extraction:

- **Observation:** Table 2 shows that only a small number of questions have a correct answer in the top five candidates and the precision of the list of candidate answers is rather low.
- **Problem:** Sometimes even with the right set of keywords, the system may still retrieve many pages containing noise, i.e., data that is not relevant to the question. For example, given the question *'Who founded the American Red Cross?'*, the system retrieved these two paragraphs:

  ' Jane Delano (1862-1919) worked in the nursing program of the American Red Cross.'

  'Despite life-threatening conditions, she provided supplies and care to troops in the American Civil War and became known as "The Angel of the Battlefield." Almost singlehandedly she founded the American Red Cross, which has provided comfort in times of crisis since 1882.'

  From the first paragraph, the proper noun *"Jane Delano"* is identified as an answer, while it is a wrong answer and from the second paragraph the pronoun *"she"* is identified as an answer. Also the system does not look for synonym words or different spellings, i.e., syntactically and semantically equivalent phrases.

- **Proposed solution:** Semantic checking of answers is one way to avoid giving an incorrect answer. To do so, we first need to classify the types of possible answers and then use Lexico Semantic Patterns (LSP) to determine the answer type of a question. For example, this will prevent the system from returning *"she"* as an answer. Using semantic checking, we can also determine that the verb *"work"* is not equivalent to the verb *"found"* and therefore reject *"Jane Delano"* as a possible answer.

1.9 Overview of the thesis

In chapter 2, we review some previous work related to question answering systems. Specifically, research that has been done in learning reformulations. In chapter 3, we give a detailed description of our algorithm which automatically learns reformulations.We also introduce the NLP tools such as the Brill POS tagger and the Base-NP chunker and show how we used them in the implementation of this thesis. In chapter 4, we describe our techniques for filtering out bad answers and re-ranking the results in the Web-QA component. In chapter 5, we describe the design and implementation of the Web-QA2 system. Finally, in chapter 6, we present conclusions and future work to be done.

Chapter 2

Literature survey

In this chapter we present some previous research that has been preformed on question answering systems, particulary some of the work that constitute an essential background to our work. At first, we explain some methologies that have been used to improve the Information Retrieval tasks. Then we survey some features of the most successful research that has been done in question reformulation.

To improve IR, several techniques have been used: Fact extraction, text data mining, and question answering.

2.1 Fact extraction

"Fact extraction is a process of extracting useful information from documents without complete understanding of meaning" [de Rijke and Webber2003].

In fact extraction systems, also known as information extraction, unstructured texts (ex. news, articles) are sent to the system and according to some user-built extraction rules, the system fills a predefined relational database (ex. news events). For example, documents about job offers are given as input to the system and the system fills a predefined table of all jobs available in one area. The extraction rules are generated manually or automatically by using some training data and the source documents can be a corpus of textbooks, papers, and journals.

SNOWBALL [Agichtein and Gravano2000] is an example of information extraction systems. This system extracts relations from a large collection of plain-text documents. Given a set of hand written example tuples, for instance $< organization, location >$, the system generates patterns from text documents and these patterns are then used to generate new tuples. For example, the user provides the system with tuples shown in Figure 5 [Agichtein and Gravano2000]:

Organization	**Location of Headquarters**
MICROSOFT	REDMOND
EXXON	IRVING
IBM	ARMONK
BOEING	SEATTLE
INTEL	SANTA CLARA

Figure 5: User-provided example tuples for SNOWBALL.

The system looks for instances of "organization" and "location" in the documents. Then it examines the surrounding text and extracts patterns. Figure 6 shows the sample of extracted patterns [Agichtein and Gravano2000].

$< ORGANIZATION >$ *headquarters* in$< LOCATION >$
$< LOCATION >$ *–based* $< ORGANIZATION >$
$< ORGANIZATION >, < LOCATION >$

Figure 6: Example of patterns that exploit Named Entity tags

Using the extracted patterns, the system generates new tuples, such as $< AIRCHINA, BEIJING >$. These are used in the next iteration to produce new patterns. Finally, a table of organizations and their locations will be produced. The algorithm can also be used for other attributes than organization and location. Figure 7 shows some tuples that are generated by the SNOWBALL system.

Organization	**Location of Headquarters**
3COM CORP	SANTA CLARA
3M	MINNEAPOLIS
AIR CHINA	BEIJING
FEDERAL EXPRESS CORP	MEMPHIS
FRUIT JELLIES	APPLE
MERRILL LYNCH and CO	NEW YORK
NETSCAPE	MOUNTAIN VIEW
NINTENDO CORP	TOKYO

Figure 7: Some tuples discovered during SNOWBALL's first iteration.

2.2 Text data mining (TDM)

> Text mining, also known as text data mining or knowledge discovery from textual databases, refers to the process of extracting interesting and non-trivial patterns or knowledge from text documents [Tan1999].

Text data mining goes beyond the surface of a document by looking for patterns and contextual relevance between documents. With text data mining, the user discovers new information that was not previously expressed in any individual document.

TDM receives unstructured data (text) and a well defined limited query (text based). Using a combination of linguistics and artificial intelligence, the system then analyzes the text, extracts the relevant information from multiple texts, links related information, and produces results in a predetermined format.

The system at first creates a semantic directory by assigning meaning to each word of the text. For example, all the following words may have the same meaning in the text.

```
jog    ---> run
sprint ---> run
hustle ---> run
```

In the second stage, the system forms concepts by determining the relationship between the content of multiple documents and linking this information together. Mostly this is done by comparing verb-noun relationships and by comparing word modifiers with the words they modify.

The system then analyzes the text and determines the intended communication of phrases, since the order of words or even insertion of just one word can change the meaning of the sentence. In this example we show some phrases with their intention:

```
Here is [x].      ---> Comment
Is [x] here?      ---> Question
[x] is here.      ---> Verification
[x] is not here. ---> Rejection
```

Using these results and linking them together, the system produces new information to the user. For example, a nutrition program plans to find new recipes for healthy diets. At first, the system scans databases containing different recipes. Recipes with specific calory, vitamin, and mineral amounts as well as specific types of meats, and starches, are extracted and classified. The system then creates optimal recipes for specific needs, using these classified recipes and linking them together.

2.3 QA Systems that use reformulations

Answer formulation deals with identifying possible forms of expressing answers to a given question. The ideal answer formulation should not retrieve incorrect answers but should also identify many candidate answers. For example given a question *'What is another name for the North Star?'*, a reformulation-based QA system searches for formulation like *'the North Star, also known as* $< NP >$*'* or *'*$< NP >$ *also known as the North Star'* in the document collection and instantiates $< NP >$ with the matching noun phrase.

As we saw in section 1.5.3, many researchers use a set of hand-made patterns for answer formulation while others try to automatically acquire answer formulations using some linguistic rules.

Here, we introduce some question answering systems that have used answer formulation.

2.3.1 TextRoller

Soubbotin et al. [Soubbotin and Soubbotin2001] participated in the TREC-10 competition with the TextRoller system with the idea of using a fairly extensive list of surface patterns as the core of their QA approach. This system uses positive and negative patterns for choosing and arranging text items in addition to its use of keywords.

In the approach of Soubbotin, the question answering system checks the candidate snippets for the presence of certain predefined patterns with their associated scores. Candidate snippets containing the highest-scored patterns are chosen as final answers. If the system cannot find any pattern for the answer, then lexical similarity between the question and answer snippet will be used. This approach does not require NLP tools or knowledge based analysis of the question and the text. The accuaracy of this approach depends on the quantity and diversification of predefined patterns as well as on the recall of passages containing candidate answers.

The system was evaluated with the TREC-10 QA track, using 51 pattern elements, for each question category, five to fifteen of such patterns were applied for recognition of potential answers. From 492 questions, 372 questions had at least one candidate answer, and 289 of questions obtained correct answer. The MRR was 0.686.

2.3.2 Webstorm

Florence Duclaye, Francois Yvon, and Olivier Colin [Duclaye et al.2002] used the web as a linguistic resource for learning reformulations automatically and also take advantage of the results of an existing question answering system called Webstorm. Figure 8, from [Duclaye et al.2002], shows the architecture of Webstorm.

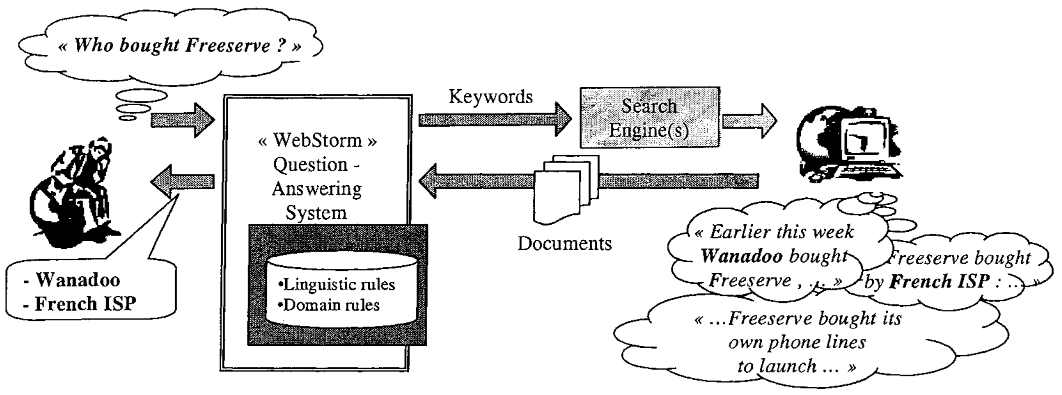
« Who bought Freeserve ? »
Keywords
Search Engine(s)
« WebStorm » Question - Answering System
•Linguistic rules
•Domain rules
Documents
- Wanadoo
- French ISP
« Earlier this week Wanadoo bought Freeserve , ... »
Freeserve bought by French ISP : ... »
« ...Freeserve bought its own phone lines to launch ... »

Figure 8: Webstorm system's architecture

Their approach focuses on the use of paraphrases as a potential way to improve question answering systems. They start with one single prototypical argument tuple of a given semantic relation and search for potential alternative formulations of the relation, then find new potential argument tuples and iterate this process to progressively validate the candidate formulations. The seed data consists of one instance of the target relationship, where both the linguistic expression of the relationship and the tuple of the arguments have been properly identified.

For example, the system is initialized with the seed sentence that is composed of a question *'Who bought Netscape?'*, and the answer assigned to this question by the Webstorm system is *'AOL bought Netscape'*. The system then builts the following corresponding parameters:

```
Formulation = {bought}  and  Arguments = {AOL, Netscape}.
```

In a two-level bootstrapping mechanism, at the first stage the system searches for new formulation for *(AOL, Netscape)*. To do this *'AOL'* and *'Netscape'* have been used as keywords and then the two following answer patterns are built to retrieve documents from the web:

```
AOL [verb] Netscape
Netscape [verb] AOL
```

For each of these patterns, only the top 100 retrieved documents are kept for further processing. The system then takes the verb that occurs between *'AOL'* and *'Netscape'* as new potential formulation. In the second stage of the bootstrapping mechanism, the system uses *'bought'* as a keyword to the search engine. The noun phrases occurring before and after *'bought'* are then considered as new argument tuples and are used in the next iteration to find new formulations. Table 3 shows an example of output of Webstorm.

Formulation	**Argument Tuples**
Tuer (kill)	Lee Harvey Oswald-Kennedy
Assassiner (assassinate)	Lincoln-joun Wilkes Bothh
Dominer (dominate)	Lee Harvey Hoswald-Kennedy
Innocenter (innocenting)	Anna Kournikova-jennifer Capriati
	Matta-papis
	Chris Carpenter-le Yankees

Table 3: Acquisition of potential reformulations for the semantic relation Kill(X,Y) - French seed sentence: Lee Harvey Oswald tua Kennedy

2.3.3 Tritus

Agichtein, Lawrence, and Gravano have developed the Tritus system [Agichtein et al.2001], that automatically learns to transform natural language questions into queries containing terms and phrases expected to appear in the retrieved documents. Tritus follows three goals: first; it classifies questions into question types, second; it generates question reformulations for each question type, and third it evaluates transformations on the different information retrieval systems and discovers which IR systems are more suitable for each type of question. Given a training set containing 30,000 question-answer pairs [1], the system generates question phrases to identify different categories of

[1]The training set was collected from 270 frequently asked questions(FAQ) files

questions based on n-grams appearing at the beginning of the questions. For example, consider the following question answer pair from the training corpus:

```
What is a near-field monitor? A near field monitor is one that is designed to be...
```

The phrase *"what is a"* in the question determines the goal of this question. The system then automatically produces candidate terms and phrases for each of the generated question phrases. The question phrases are ranked according to the frequency of matching question phrases and transformations. The Okapi weighting method has been used to weight n-grams. The weighting of n-grams is then extended to question phrases containing those n-grams in the answer. The candidate transformations are categorized into different groups based on their length and evaluated with the target information retrieval system. Table 4 shows sample of candidate transforms generated by the Tritus system.

Question type qt (question phrase)	**Candidate Transform t**	**qtft**	**wt**	**twt**
"what is a"	"refers to"	30	2.71	81.3
	"refers"	30	2.67	80.1
	"meets"	12	3.21	38.5
	"driven"	14	2.72	38.1
	"named after"	10	3.63	36.3
	"often used"	12	3.00	36.0
	"to describe"	13	2.70	35.1

Table 4: Sample Candidate Transforms where qtft = frequency of t in the relevant question type, wt = term selectivity/discrimination weight, twt = resulting candidate transform weight

The system did not participate in any TREC competition, hence no standard MRR is available.

2.3.4 The system of Ravichandran and Hovy

Ravichandran and Hovy [Rivachandram and Hovy2002] used a machine learning technique and a few hand-crafted examples of question-answer pairs to automatically learn patterns. This method allowed them to build a large tagged corpus of patterns along with the precision of each pattern.

This system can be used to learn patterns for other languages than English, however, the major limitation of this system is that it can handle only one question term in each query, while in some questions there are multiple words, possibly different from each other, that should be considered in the query to search the context of the answer.

2.4 Conclusion

All of the systems discussed in this section have some weaknesses and limitations. None of them use semantic type checking or handles long distance dependencies, and some do not use any external knowledge or matching of part-of-speeches. All of them use a bootstrapping technique starting with only a few examples of question-answer pairs.

Our approach uses a large corpus of question-answer pairs from each question category to learn reformulation patterns. We also take into account the semantic relation and part-of-speech tags as well as named-entity tags.

Chapter 3

Learning Reformulation Patterns

In this chapter, we briefly explain what question reformulation is and how it is employed in the Web-QA component. We then describe the rule-based patterns used in the Web-QA component, the weakness of current patterns, and our algorithm for the automatic acquisition of reformulation patterns.

3.1 Question reformulation

The idea behind question reformulation is, given a question Q, the system searches a document collection or the Web for a set of phrases P1, P2,...,Pn as possible answer contexts. For example, given the question *'Who wrote "Gone with the wind"?'*, most likely, the answer will have one of the following forms:

```
''Gone with the wind'' was written by <PERSON>
<PERSON> wrote ''Gone with the wind''
```

Having more linguistic formulations of the answer context means more possibilities to find documents that contain the answer. In fact the resulting possible answer contexts are more reliable if the system can guess how the answer appears in the text.

3.1.1 Reformulation patterns

In the current Web-QA component, a pattern is defined as a sequence of strings and special symbols that are used for the pattern matching function, such as the following sequences:

```
''When'' ''did'' (n ANY) (v Vsf)?
```

For example, the following sentence is matched with the above pattern:

```
When did the Jurassic Period end?


     mapped to
ANY -----------> ''the Jurassic Period''


      mapped to
Vsf ------------> ''end''
```

In each pattern, the strings (e.g. *"Who")* are matched exactly as literals in the matching sentence, and lists refer to "predicates" are functions that wrap specialised matching semantics. Predicates are special symbols (e.g (v Vsf) and (n Any)), where the first component of these lists is a variable name (e.g. n, v) and the second component is predicate arguments (e.g. ANY, Vsf).

Table 5 shows the special symbols that are used in the pattern language of the Web-QA component.

Name	Description
ANY	Matches any sequence of words
IN	Matches any one of a list of given words
Vsf	Matched a verb in simple form
Vpp	Matches a past participle
Vpast	Marches a verb in simple past
NOUN	Matches a noun
ADJ	Matches an adjective

Table 5: Special symbols used in the pattern language of the Web-QA component

- **Question patterns (Matching rules)**

  A question pattern, also called matching rule, is the sequence of strings that define what the question must look like (the order of strings is important), for example *' "Who" (Vsf) (x Proper)?'* is a question pattern that matches *'Who is George Bush?'*.

Each matching rule consists of the following elements:

```
Specific keyword (e.g When)
String of characters (e.g. ANY-SEQUENCE-WORDS)
Part-of-speech tags (e.g. VERB-simple).
```

The first element is the question keyword that is extracted from the question. The second element is the argument which could be object or subject. There may be several of these elements in the question's structure. The third element is the verb from the question which could be a main verb or an auxiliary verb.

- **Answer patterns**

  An answer pattern is the exact form of the sentence that may contain the possible candidate answer. For example, for the question *"Who is George Bush?"*, the system tries to find the sentences that match with one of these two patterns:

  ```
  ( <X> <Vsf> <answer> )
  ( <answer> <Vsf> by <X> )
  ```

  Where $< answer >$ is the candidate answer, $< X >$ is the Question term (i.e. "George Bush"), and $< Vsf >$ is the verb in simple form.

3.1.2 Rule based patterns in the Web-QA component

In the WebQA component, the reformulation invocation process is done in two steps, at first the question is mapped to the predefined reformulation templates. Then, the rule that satisfies the question is activated and the system returns the list of hand written answer patterns for this rule. For instance, consider the following question:

```
When did the Jurassic Period end?
```

The question matches with the following question pattern:

```
When did (n ANY) (v Vsf)?
```

Then, the system returns the following answer patterns for the above activated question pattern:

```
(n (past v) TIME)

(TIME n (past v) )

(TIME "," n (past v) )
```

Each question can be matched with only one question pattern. Also, some questions may not match with question pattern.

Figures 9, and 10 show all the hand written question patterns that are used in the Web-QA component.

Figures 11, 12, and 13 show all the hand written question patterns and their corresponding answer patterns that are used in the Web-QA component.

Question Pattern	Example
When did (n ANY) (v Vsf)?	When did the Jurassic Period end?
When did (n ANY) (v Vsf) (n2 ANY)?	When did the Jurassic Period end?
When was (n ANY) (p Vpp)?	When was Beethoven born?
When was the (n LNP) (n2 ANY)?	When was Beethoven born?
When was (n LNP)?	When was Hurricane Hugo?
When is (n ANY)?	When is Boxing Day?
Where did (n ANY) (v Vsf)?	Where did Dylan Thomas die?
Where (v IN is was were) (n ANY) (a Vpp)?	Where was Tesla born?
Where is (n ANY)?	Where is the Taj Mahal?
Where are (n LNP) (x Vpp)?	Where are diamonds mined?
Where are (n ANY)?	Where are diamonds mined?
Where did (n LNP) (v Vsf)?	Where did guinea pigs originate?
Where did (n ANY) (v Vsf) (x ANY)?	Where did Hillary Clinton graduate college?
Where do (n ANY)?	Where do lobsters like to live?
What does (n ANY) (v Vsf)?	What does the Peugeot company manufacture?
What does (n ANY) (v Vsf) (n2 ANY)?	What does El Nino mean in Spanish?
What are (n ANY) (x Vpp)?	What are the Black Hills known for?
What (t ANY) is (n ANY)?	What country is the biggest producer of tungsten?
Which (t ANY) is (n ANY)?	Which country is Australia 's largest export market?
Which (t ANY) (a ADJ) (n ANY)?	Which company created the Internet browser Mosaic?
What is the name of (n ANY)?	What is the name of the highest mountain in Africa?
What year was (x ANY) (a ADJ)?	What year was Janet Jackson's first album released?
What year did (x ANY) (a Vsf)?	What year did Hitler die?
What year did (x ANY) (a Vsf) (y ANY)?	What year did Montana become a state?
What year was (x ANY) (a ADJ) (y ANY)?	What year was Desmond Mpilo Tutu awarded the Nobel Peace Prize?
(x ANY) in what year ?	CNN began broadcasting in what year?
What did (n LNP) (v Vsf) (x ANY)?	What did Shostakovich write for Rostropovich?
What (v IN is are was) (n ANY)?	What is the brightest star visible from Earth?
What (n ANY) (v Vpast) (n2 ANY)?	What famous communist leader died in Mexico City?
What did (n ANY) (v Vsf) (n2 ANY)?	What did Richard Feynman say upon hearing he would receive the Nobel Prize in Physics?
Who (v IN is was) (x PROPER)?	Who is Desmond Tutu?
Who (v IN is was) (x ANY)?	Who is Desmond Tutu?
Who (a Vpast) (x ANY)?	Who developed potlatch?
Who (a Vpp) (x ANY)?	Who started the Dominos Pizza chain?
Whom did (x ANY) (v Vsf) (y ANY)?	Whom did the Chicago Bulls beat in the 1993 championship?
(n1 ANY) (v Vsf) (n2 ANY) what ?	CPR is the abbreviation for what?
What do (x ANY)?	What do penguins eat?
What (x ANY) does (y ANY) (v Vsf)?	What state does Charles Robb represent?
What makes (x ANY) a (y ANY)?	What makes Black Hills, South Dakota a tourist attraction?
Why are (n LNP) (n2 ANY) ?	Why are electric cars less efficient in the north-east than in California?
Why did (n LNP) (v Vsf) (x ANY)?	Why did David Koresh ask the FBI for a word processor?
Name (a IN a an the) (n ANY) ."	Name a flying mammal.
(n ANY) what (n2 LNP)?	Rider College is located in what city?
(n ANY) whom ?	(CNN) is owned by whom?

Figure 9: The matching rules for Web-QA Component

Question Pattern	Example
How many (n1 ANY) are there (n2 ANY)?	How many calories are there in a Big Mac?
How many (n ANY) are there?	How many types of lemurs are there?
How many (n1 ANY) were ever (a ADJ)?	How many Stradivarius violins were ever made?
How many (n1 ANY) are (n2 ANY)?	How many hexagons are on a soccer ball?
How many (n1 ANY) ago did (n2 ANY) (v Vsf)?	How many years ago did the ship Titanic sink?
How many (n1 ANY) did (n2 ANY) (v Vsf)?	How many films did Ingmar Bergman make?
How many (n1 ANY) did (n2 ANY) (v Vsf) (n3 ANY)?	How many home runs did Lou Gehrig have during his career?
How many (n1 ANY) does (n2 ANY) (v Vsf)?	How many moons does Jupiter have?
How many (n1 ANY) does (n2 ANY) (v Vsf) (n3 ANY)?	How many people does Honda employ in the U.S.?
How many (n1 ANY) were there (n2 ANY)?	How many Vietnamese were there in the Soviet Union?
How many (n1 ANY) (v Vsf) (n2 ANY)?	How many people live in Tokyo?
How much did (n1 ANY) spend (n2 ANY)?	How much did Mercury spend on advertising in 1993?
How much (n1 ANY) should (n2 ANY) (v Vsf) (n3 ANY)?	How much folic acid should an expectant mother get daily?
How much in (n1 ANY) is (n2 ANY)?	How much in miles is a 10 K run?
How much does (n1 ANY) cost?	How much does one ton of cement cost?
How much does (n1 ANY) cost (n2 ANY)?	How much does one ton of cement cost?
How much money?	How much money does the Sultan of Brunei have?
How do you (v Vsf) (n ANY)?	How do you abbreviate Original Equipment Manufacturer?
How is (n ANY) (p Vpp)?	How is Original Equipment Manufacturer abbreviated?
How tall is (n2 ANY)?	How tall is Mt. Everest?
How did (n ANY) (v Vsf)?	How did Bob Marley die?
How long did (n ANY) (v Vsf)?	How long did the Charles Manson murder trial last?
How far is (x ANY) from (y ANY)?	How far is Yaroslavl from Moscow?
How far away is (x ANY)?	How far away is the moon?
How far is (x ANY)?	How far away is the moon?
How long is (x ANY)?	How long is human gestation?
How (a ADJ) is (x ANY)?	How old is the sun?

Figure 10: The matching rules for Web-QA Component (Cont.)

3.1.3 Weakness of current patterns

As mentioned in Section 1.8.2, QUANTUM's Web-QA component uses 77 hand written reformulation templates to generate possible forms of answer contexts to a question.

The reformulation templates in the pattern set drive the Web-QA component system to find answer contexts of good linguistic quality, but theses templates only cover specific types of questions. In order to determine the coverage of these predefined templates, we evaluated the Web-QA component with the TREC-9, and TREC-10 collection data. In our investigation, we found two major problems:

1. The question is not reformulated (Incompleteness).

   Some questions are not matched with any question pattern in the reformulation templates.

Class	Matching Rule (Question Pattern)	Answer Formulation
WHEN	When did (n ANY) (v Vsf)?	n (past v) TIME
		TIME n (past v)
		TIME , n (past v)
	When did (n ANY) (v Vsf) (n2 ANY)?	n (past v) n2 TIME
	When was (n ANY) (p Vpp)?	n was p TIME
	When was the (n LNP) (n2 ANY)?	the n was n2 TIME
	When was (n LNP)?	n was TIME
	When is (n ANY)?	n is TIME
		TIME is n
WHERE	Where (v IN is was were) (n ANY) (a Vpp)?	n v a LOCATION
	Where is (n ANY)?	n is located LOCATION
		n , LOCATION
		n , located LOCATION
		n is LOCATION
	Where are (n LNP) (x Vpp)?	n are x LOCATION
	Where are (n ANY)?	n are LOCATION
	Where did (n LNP) (v Vsf)?	n (past v) LOCATION
	Where are (n ANY)?	n are LOCATION
	Where did (n LNP) (v Vsf)?	n (past v) LOCATION
	Where did (n LNP) (v Vsf)?	n (past v) LOCATION
	Where did (n ANY) (v Vsf) (x ANY)?	n (past v) x LOCATION
	Where do (n ANY)?	n LOCATION
WHO	Who (v IN is was) (x PROPER)?	x v TITLE
	Who (v IN is was) (x ANY)?	x v PERSON
		x , PERSON
		PERSON , x
	Who (a Vpast) (x ANY)?	PERSON a x
	Who (a Vpp) (x ANY)?	PERSON a x
	Whom did (x ANY) (v Vsf) (y ANY)?	x (past v) CLAUSE y
WHAT	(n1 ANY) (v Vsf) (n2 ANY) what?	n1 v n2 CLAUSE
	What do (x ANY)?	(x CLAUSE)
	What (x ANY) does (y ANY) (v Vsf)?	y (third-person v) CLAUSE
		AND CLAUSE is a x
	What makes (x ANY) a (y ANY)?	CLAUSE makes x a y

Figure 11: Question patterns and answer patterns for Web-QA component

Class	Matching Rule	Answer Formulation
HOW MANY	How many (n1 ANY) are there (n2 ANY)?	there are NUMBER n1 n2
	How many (n ANY) are there?	there are NUMBER n
	How many (n1 ANY) were ever (a ADJ)?	NUMBER n1 were a
	How many (n1 ANY) are (n2 ANY)?	there are NUMBER n1 n2
	How many (n1 ANY) ago did (n2 ANY) (v Vsf)?	NUMBER n1 ago n2 (past v)
	How many (n1 ANY) did n2 ANY) (v Vsf) (n3 ANY)?	n2 (third-person v) NUMBER n1 n3
	How many (n1 ANY) did (n2 ANY) (v Vsf)?	n2 (past v) NUMBER n1
	How many (n1 ANY) does(n2 ANY) (v Vsf)?	n2 (third-person v) NUMBER n1
	How many (n1 ANY) does (n2 ANY) (v Vsf) (n3 ANY)?	n2 (third-person v) NUMBER n1 n3
		n3 , n2 (third-person v) NUMBER n1
	How many (n1 ANY) were there (n2 ANY)?	there were NUMBER n1 n2
	How many (n1 ANY) (v Vsf)(n2 ANY)?	NUMBER n1 v n2
HOW MUCH	How much did (n1 ANY) spend (n2 ANY)?	n1 spent MONETARY n2
	How much (n1 ANY) should (n2 ANY) (v Vsf) (n3 ANY)?	n2 should v AMOUNT of n1 n3
	How much in (n1 ANY) is (n2 ANY)?	n2 is NUMBER n1
	How much does (n1 ANY) cost?	n1 costs MONETARY
HOW	How do you (v Vsf) (n ANY)?	(n is (past v) CLAUSE)
	How is (n ANY) (p Vpp)?	n is p CLAUSE
	How tall is (n2 ANY)?	n2 is DISTANCE high
		n2 is DISTANCE tall
	How did (n ANY) (v Vsf)?	n (past v) of CLAUSE
		n (past v) by CLAUSE
	How long did (n ANY) (v Vsf)?	n (past v) DURATION
	How far is (x ANY) from (y ANY)?	x is DISTANCE away from y
		x and y are DISTANCE
		y and x are DISTANCE
	How far away is (x ANY)?	(x is DISTANCE away)
	How far is (x ANY)?	x is DISTANCE away
	How long is (x ANY)?	x lasts DURATION
	How (a ADJ) is (x ANY)?	x is CLAUSE a

Figure 12: Question patterns and answer patterns for Web-QA component (Cont.)

Class	Matching Rule	Answer Formulation
WHAT	What does (n ANY) (v Vsf)?	n (third-person v) CLAUSE
	What does (n ANY) (v Vsf) (n2 ANY)?	n (third-person v) CLAUSE n2
	What are (n ANY) (x Vpp?	n are x CLAUSE
IMP	What (t ANY) is (n ANY)?	CLAUSE is n AND CLAUSE is a t
	Which (t ANY) is (n ANY)?	CLAUSE is n AND CLAUSE is a t
	Which (t ANY) (a ADJ) (n ANY)?	CLAUSE a n AND CLAUSE is a t
	What is the name of (n ANY)?	n is NAME
	What year was (x ANY) (a ADJ)?	x was a in NUMBER
	What year was (x ANY) (a ADJ) (y ANY)?	x was a y in NUMBER
	(x ANY) in what year?	x in NUMBER
	What did (n LNP) (v Vsf) (x ANY)?	n (past v) CLAUSE x
	What (v IN is are was) (n ANY)?	n v CLAUSE
	What (n ANY) (v Vpast) (n2 ANY)?	CLAUSE v n2 AND CLAUSE is a n
	What did (n ANY) (v Vsf) (n2 ANY)?	n (past v) CLAUSE n2
WHY	Why are (n LNP) (n2 ANY)?	n are n2 REASON
	Why did (n LNP) (v Vsf) (x ANY)?	n (past v) x REASON

Figure 13: Question patterns and answer patterns for Web-QA component (Cont.)

Thus the system cannot find any formulation pattern for the answer context. For example, no rule matches the following question:

```
''What caused the Lynmouth floods?''
```

In Chapter 1, we showed a sample of questions from the TREC-9 and TREC-10 collection data that do not have any matching rule. These samples are illustrated in Tables 6, 7, 8, 9, 10, 11.

2. The question is reformulated, but the system can not find any answer for the question.

   Some questions are reformulated, meaning there exists at least one rule in the template table that could be triggered for this question, but still the document retrieval system cannot find any occurrence of the answer, because the existing answer patterns for the triggered rule do not match with any answer context. Consider the question *'Who is Desmond Tutu?'*. The system returns **(< PERSON > V TITLE)** as the only answer pattern for this question, while in many documents, the answer is *'Nobel Peace Prize winner, Desmond Tutu'*, that have the form of **(TITLE, < PERSON >)** that is not covered by the answer pattern templates.

Due to these two problems, we not only need to expand the coverage of the question patterns, but also we must expand the matching rules to increase the number of answer formulations.

Template	Example
Which (n1 ANY) was (n2 ANY)?	Which president was unmarried?
(n1 ANY) Which (n2 ANY) do (n3 ANY) (v Vsf)?	During which season do most thunderstorms occur?
(n1 ANY) Which (n2 ANY) would (n3 Any) find (n4 ANY)?	In which state would you find the Catskill Mountains?

Table 6: The questions of type *'WHICH'* that are not covered by the existing templates

Template	Example
Where does (n1 ANY) come from?	Where does chocolate come from?
Where does (n1 ANY) (v Vsf)?	Where does Mother Angelica live?
Where does (n1 ANY)?	Where does Buzz Aldrin want to build a permanent, manned space station?
Where can (n1 ANY) find (n2 ANY)?	Where can you find the Venus flytrap?
Where Could (n1 ANY) (v Vsf) (n2 ANY)?	Where could I go to take a ride on a steam locomotive?
Where is (n1 ANY)?	Where is Logan Airport in?
(n1 ANY) Where do (n2 ANY) (v Vsf)?	In Poland, where do most people live?
Where (n1 ANY) is (n2 ANY)?	Where on the body is a mortarboard worn?
Where was (n1 ANY)?	Where was the first golf course in the United States?

Table 7: The questions of type *'WHERE'* that are not covered by the existing templates

To do the reformulation learning task, we trained using questions and answers from the TREC collection data and we expanded the queries using WordNet synonyms, hypernyms, and one-level hyponyms.

3.1.4 An overview of the learning algorithm

Our learning algorithm is mainly motivated by linguistic approaches. At first, we categorize the question types in six categories used in QUANTUM's question classification. Then, for each question category we build a question-answer pair training corpus, based on questions and their corresponding answers in the TREC collection data. Each question-answer pair is then analyzed to extract the arguments and the semantic relation between the question arguments and the answer. A query is then formulated using the arguments extracted from the question-answer pair. The formulated query is sent to a search engine which returns the first N retrieved documents. These are passed to a sentence splitter, a part-of-speech tagger, and a noun phrase chunker, to select the sentences that contain all the query terms. A semantic filter is applied to the selected sentences to keep only the sentences that contain the same semantic relation. Each sentence is then generalized to an answer

Template	Example
How Long does it take to (v Vsf) from (n2 ANY) to (n2 ANY)?	How long does it take to travel from Tokyo to Niigata?
How Long would it take to (v Vsf) from (n2 ANY) to (n2 ANY)?	How long would it take to get from Earth to Mars?
How Long would it take for (n1 ANY)?	How long would it take for a 50 savings bond to mature?
How Long do (n1 ANY) (v Vsf)?	How long do hermit crabs live?
(n1 ANY) How long is (n1 ANY)?	For how long is an elephant pregnant?
How hot does (n1 ANY)(v Vsf)?	How hot does the inside of an active volcano get?
How old was (n1 ANY) when (n2 ANY) (past v)?	How old was Elvis Presley when he died?
How old do (n1 ANY) have to be to (v Vsf)(n2 ANY)?	How old do you have to be in order to rent a car in *Italy?*
How old was (n1 ANY)?	How old was the youngest president of the United *States?*
How many (n1 ANY) (past v) (n2 ANY)?	How many people lived in Nebraska in the mid 1980s?
About How many (n1 ANY) (past v) (n2 ANY)?	About how many soldiers died in World War II?
How many (n1 ANY) (past v) when (n2 ANY)?	How many people died when the Estonia sank in 1994?
How many (n1 ANY) IN (n2 ANY)?	How many people in Tucson?
How many (n1 ANY) is it from (n2 ANY) to (n3 ANY)?	How many miles is it from London to Plymouth?
(n1 ANY) (v Vsf) How many (n2 ANY)?	A normal human pregnancy lasts how many months?
How much could (n1 ANY) (v Vsf) (n2 ANY) for in NUMBER?	How much could you rent a Volkswagen bug for in 1966?
How much was (n1 ANY) for (n2 ANY)?	How much was a ticket for the Titanic?
How much does (n1 ANY)?	How much does the human adult female brain weigh?
How much of (n1 ANY) is (n2 ANY)?	How much of an apple is water?
How much stronger is (n1 ANY) (past v) compared with (n2 ANY)?	How much stronger is the new vitreous carbon material invented by the Tokyo Institute of Technology compared with the material made from cellulose?
How fast can (n1 ANY) (v Vsf)?	How fast can a Corvette go?
How was (n1 ANY) (past v) (n2 ANY)?	How was Teddy Roosevelt related to FDR?
How often does old (n1 ANY)?	How often does old Faithful erupt at Yellowstone *National Park?*

Table 8: The questions of type *'HOW'* that are not covered by the existing templates

Template	Example
(n1 ANY) What (n2 ANY) did (n3 ANY) (v Vsf) (n4 ANY)?	In 1990, what day of the week did chrismas fall on?
What (n1 ANY) (past v) (n2 ANY)?	What culture developed the idea of potlatch?
AT What (n1 ANY) did (n2 ANY) (v Vsf) (n3 ANY)?	At what age did Rossini stop writing opera?
AT What (n1 ANY) does (n2 ANY) (v Vsf) (n3 ANY)?	At what speed does the Earth revolve around the sun?
What (n1 ANY) are there in (n2 ANY)?	What tourist attractions are there in Reims?
For What (n1 ANY) is (n2 ANY) used (n3 ANY)?	For what disease is the drug Sinemet used as a treatment?
What did (n1 ANY) (v Vsf)?	What did Vasco da Gama discover?
What (n1 ANY) (v Vsf) (n2 ANY)?	What company sells the most greeting cards?
What (n1 ANY) has (n1 ANY)?	What state has the most Indians?
What (n1 ANY) do (n2 ANY) (v Vsf)?	What sport do the Cleavland Cavaliers play?
What (n1 ANY) do (n2 ANY) (v Vsf) (n3 ANY)?	What primary colors do you mix to make orange?
What (n1 ANY) does (n2 any) (v Vsf) (n3 any)?	What store does Martha Stewart advertise for?
What (past v) (n1 ANY)?	What caused the Lynmouth floods?
What (v Vsf) (n1 ANY)?	What attracts tourists to Reims?
What is (n1 ANY)?	What is the Milky Way?
What were (n1 ANY)?	What were Christopher Columbus' three ships?
WHAT (n1 ANY) are (n1 ANY)?	What gasses are in the troposphere?
What (n1 ANY) (v Vsf) (n2 ANY)?	What type of horses appear on the Budweiser commercials?
(n1 ANY), what is (n2 ANY)?	At Christmas time, what is the traditional thing to do under the mistletoe?
What is the name of (n1 ANY)?	What is the name of the satellite that the Soviet Union sent into space in 1957?
What (n1 ANY) can I use to (n2 ANY)?	What hair color can I use to just cover a little gray?
What Can (n1 ANY) (v Vsf) (n2 ANY)?	What can one see in Reims?
(n1 ANY) What (n1 ANY) Can I find (n2 ANY)?	In what book can I find the story of Aladdin?
What could (n1 ANY) (v Vsf) (n2 ANY)?	What could I see in Reims?
What (n1 ANY) would (v Vsf) (n2 ANY)?	What soft drink would provide me with the biggest intake of caffeine?
WHat (n1 ANY) should (n2 ANY) (v Vsf) (n3 ANY)?	What amount of folic acid should an expectant mother take daily?
WHat (n1 ANY) need to be (v Vsf) to (n2 ANY)?	What colors need to be mixed to get the color pink?
Tell me WHAT (n1 ANY) is (n2 ANY)?	Tell me what city the Kentucky Horse Park is near?
(n1 ANY) is known by what (n2 ANY)?	Aspartame is known by what other name?
To get (n1 ANY) What (n2 ANY) should (n3 ANY) (v Vsf)?	To get the most caffeine, what soda should I drink?
(n1 ANY) is in What (n2 ANY)?	The corpus callosum is in what part of the body?
In What (n1 ANY) is (n2 ANY)?	In what city is the US Declaration of Independence located?
(n1 Any) (v Vsf) What (n2 Any)?	Developing nations comprise what percentage of the world's population?

Table 9: The questions of type *'WHAT'* that are not covered by the existing templates

Template	Example
Why does (n1 ANY) (v Vsf) (n2 ANY)?	Why does the moon turn orange?
Why can't (n1 ANY) (v Vsf)?	Why can't ostriches fly?
Why is (n1 ANY) (a ADJ)?	Why is Jane Goodall famous?

Table 10: The questions of type *'WHY'* that are not covered by the existing templates

Template	Example
Can you give me (n1 ANY)?	Can you give me the name of a clock maker in London, England?
Define (n1 ANY)?	Define thalassemia.

Table 11: The questions of other types that are not covered in existing template

pattern. A weighting module finally assigns a weight to each generated pattern according to its semantic distance from the question and the frequency of that pattern.

In order to perform the above algorithm we used several natural language processing (NLP) tools available to the research community. These will be described before the details of our learning algorithm are given.

3.2 The NLP tools used

3.2.1 Brill Tagger

In this thesis we used the Brill Tagger [Brill1992] that has been developed at the Massachusetts Institute of Technology (M.I.T.) and the University of Pennsylvania, in 1993. The Brill tagger is a transformation-based part-of-speech tagger that automatically learns its rules and tags from the tagged Brown Corpus [Francis and Kucera1982]. The Brill tagger was trained on 90% Penn TreeBank corpus and tested on a distinct 5%. The accuracy of Brill tagger is 95-97% (i.e. 95-97% of the word tokens in the text receive the correct tag). The Brill tagger can be downloaded from Brill's webpage:

`http://www.cs.jhu.edu/brill`

3.2.2 BaseNP chunker

In this work we used the *C++/Perl* version of the BaseNP chunker[1] that is released by Ramshaw

[1]The BaseNP chunker is available at the following address:
ftp:*//ftp.cis.upenn.edu/pub/chunker/basenp – chunker – 1/chunker.tar.gz*

and Marcaus [Ramshaw and Marcus1995].

The application takes the input sentence with part-of-speech tags in the same format of Eric Brill's transformational tagger and then attempts to insert brackets marking the noun phrases in the sentence. This application uses a transformation-based learning approach. The heuristic transformation rules were trained on the Wall Street Journal text, from the UPenn Treebank. The recall of the BaseNP chunker (percentage of correct chunks found) for a training set with 200K is 92.3%, the precision (percentage of chunks found that are correct) is 91.8%, when the percentage of words with correct tag is 97.4%.

The following is an example of input and output of the BaseNP chunker:

```
Input: Georgia-Pacific/NNP close0d/VBD down/RB $/$ 2.50/CD ,/, at/IN $/$
  50.875/CD in/IN Big/NNP Board/NNP trading/NN.
Output: [ Georgia-Pacific/NNP ] closed/VBD down/RB [ $/$ 2.50/CD ] ,/,
  at/IN [ $/$ 50.875/CD ] in/IN [ Big/NNP Board/NNP trading/NN ].
```

Note: Some BaseNP chunker errors

During the process of noun phrase chunking, the most common errors appeared where words are tagged VBG (Verb gerund) and VBN (Verb, past participle) but they occur frequently inside brackets as noun phrase. The other common error involves conjunctions; for example, the words *"and"* and *","* with the part-of-speech tag CC (Coordinative conjunction), which occur outside of a bracket. One other error that appears in many cases is where the noun phrase is quoted, for example:

```
the/DT movie/NN  ``Sleepless/JJ in/IN Seattle''/NNP
```

is chunked to:

```
[the/DT movie/NN ] ``Sleepless/JJ in/IN [ Seattle''/NNP].
```

To fix this error, we grouped the quoted noun phrases within the same sequence brackets to form them as a single noun phrase. For example, in the above case, the noun phrase chunks results became:

```
[ the/DT movie/NN] [``Sleepless/JJ in/IN  Seattle''/NNP].
```

We also grouped the noun phrases conjucated with *"of"*, within the same sequence brackets as a single noun phrase. For example, consider the phrase *"The prime minster of Canada"* and the chunker result for this phrase:

```
[the/DT prime/NN mister/NN] of/IN [Canada/NNP]
```

In this case, the noun phrase chunks results became:

```
[the/DT prime/NN mister/NN of/IN Canada/NNP]
```

3.2.3 GateNE Named-entity tagger

In this project, we used GateNE [Cunningham et al.1991] as our named-entity tagger to examine the noun phrases appearing in an answer context as a possible answer. Using a named-entity tagger, all named-entities in the relevant passages are identified. The identified named-entity fields that satisfy the answer type are then extracted as possible answers. For example, consider the following question and the candidate passage containing a possible answer:

```
Question: How high is the Everest?
Relevant passage: At 29,035 feet the summit of Everest is the highest...
```

The question keyword **"how high"** yields the answer type **Number**, which is determined in the question analysis module. After applying the Gate named entity tagger, the answer context may look like this:

```
At <Number=  29,035>  feet the summit of <Location = Everest> is the highest...
```

"29,035" is considered as a candidate answer because its named entity tag is the same as the expected answer type for the question.

3.2.4 Porter stemmer

The Porter stemmer [Porter1980] is an algorithm for stripping the suffixes of English words. This algorithm has been written, in 1980, by Porter, and has been implemented in C++, Java, Perl as well as other languages.

We use the Porter stemmer in this thesis to find relevant passages with a higher recall (see section 1.4.3).

3.3 The algorithm for learning of reformulation patterns

Our reformulations learning task is a four stages process:

1. **Question classification**

2. **Building a training corpus**

3. **Learning reformulation patterns**

4. **Assigning a weight to each pattern**

Figure 14 shows the system architecture. Arrows indicate flow of data.

In the following sections, we explain in detail the four stages of the algorithm.

3.3.1 Question classification

Questions are formed depending on the kind of information sought. In fact, one major factor to guessing the answer type is to know the question type.

The QUANTUM system [Plamondon et al.2001] categorized questions into 6 classes and 20 sub-classes. We borrow this idea and use this classification in our approach. This categorization is shown in Figure 15.

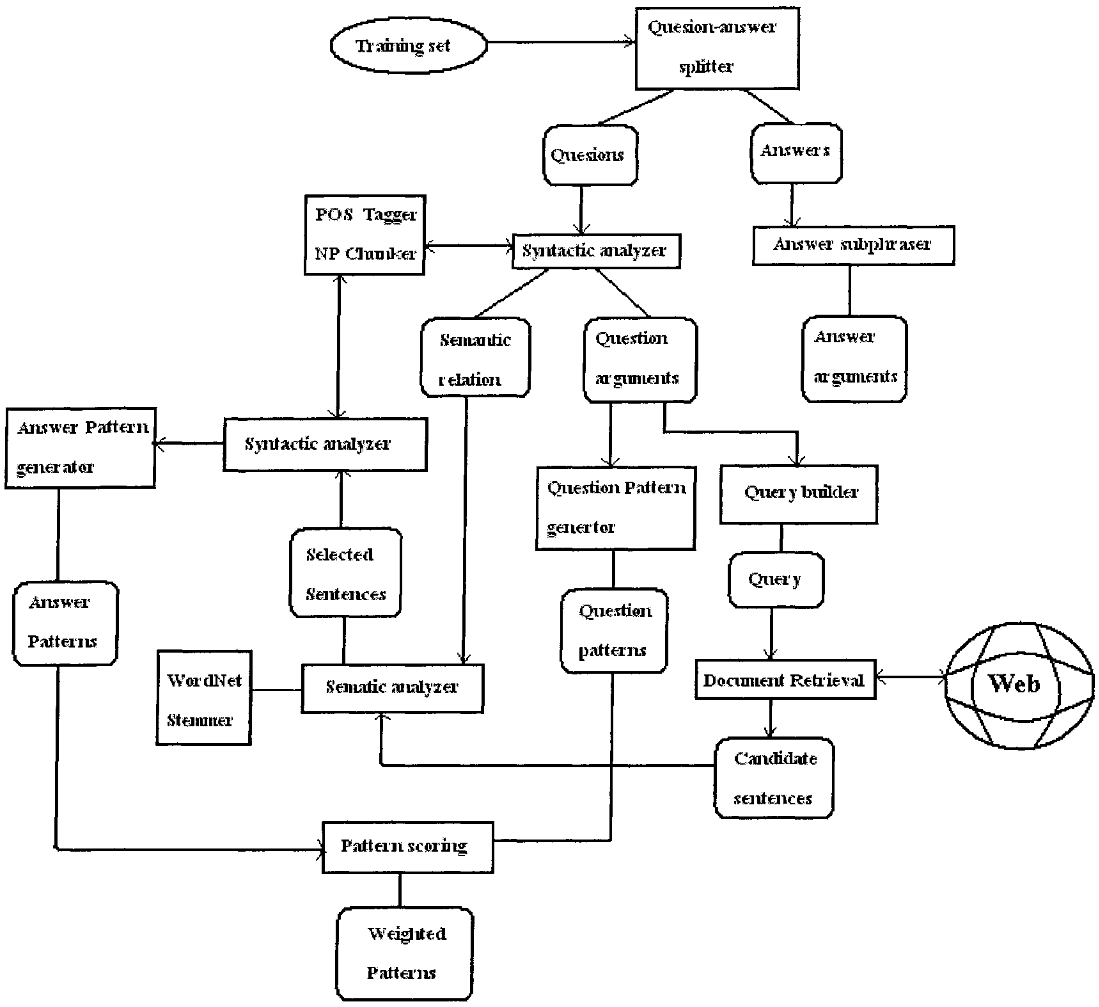

Figure 14: Architecture of the reformulation learning system

3.3.2 Building the training corpus

Our learning algorithm starts with a set of question-answer pairs as training corpus for each question type. To build the training corpus, we first classified the questions using the classification described in the previous section and then we built a training corpus for each type of question.

The training corpus contains 1343 question-answer pairs taken from the TREC-8, TREC-9, and TREC-10 collection data. Each question-answer pair of the training corpus is composed of one question and its corresponding answer. The following are some examples of question-answer pairs in the training corpus:

Class	Subclass	Example
What	basic-what	What was the monetary value of the Nobel Peace Prize in 1989?
	what-who	What custom designer decided that Micheal Jackson should only wear one glove?
	what-when	In what year did Ireland elect its first woman president?
	what-where	What is the capital of Uruguay?
Who	who-person	Who is the author of the book "The Iron Lady: A Biography of Margaret Thatcher"?
	who-pronoun	Who was Martin Luther King?
How	basic-how	How did Socrates die?
	how-many	How many people died when the Estonia sank in 1993?
	how-long	How long does it take to travel from Tokyo to Niigata?
	how-much	How much fiber should you have per day?
	how-far	How far is Yaroslav from Moscow?
	how-tall	How tall is Mt. Everest?
	how-rich	How rich is Bill Gates?
	how-large	How large is Missouri's population?
Where		Where is Taj Mahal?
When		When did the Jurassic Period end?
Which	which-who	Which former KLU Klux Klan member won an elected office in the U.S.?
	which-where	Which city has the oldest relationship as sister-city with Los Angeles?
	which-when	In which year was New Zealand excluded from the ANZUS alliance?

Figure 15: Example of a question classification taken from [Plamondon et al., 2001]

```
34   Where is the actress, Marion Davies, buried? Hollywood Memorial Park

54   When did Nixon die? April 22, 1994

220  Who is the prime minister of Australia? Paul Keating

232  Who invented television? Philo Farnsworth, Vladimir Zworykin

255  Who thought of teaching people to tie their shoe laces? Starlace

1071 When was the first stamp issued? 1840
```

We divided the training corpus into six individual files, each file represents question-answer pairs for one question category. Table 12 shows our corpus files along with their sizes (number of question-answer pairs).

Note: Question Simplification

In some questions, the auxiliary verb appears in the contracted form. For example, *"What's"* is the contracted form of *"What is"*. This form increases the number of cases in each question category. To restrict the number of cases in each question category we adapted some simplification rules from the Web-QA component called Canon rules. These rules transform the questions that are in the form of *"What's"*, *"Who's"*, *"Where's"*, *"How's"*, and *"When's"* into a canonical form. Consider the question *'What's the farthest planet from the sun?'*, after applying the rules, the question is

Corpus	Size (No. question-answer pair)
Who-Corpus.txt	208
Where-Corpus.txt	119
When-Corpus.txt	88
What-Corpus.txt	747
Why-Corpus.txt	8
Which-Corpus.txt	32
How-Corpus.txt	111
Other-Corpus.txt	30

Table 12: Training corpus files according to the type of question

replaced with *'What is the farthest planet from the sun?'*.

The canon rules also identify the **"What is the name of"** questions and simplify them to the form of **"Name X"**. For example, the question *'What is the name of the president of Madagascar?'* is simplified to the form *'Name the president of Madagascar?'*

Question	Simplified Question
What's (n ANY)?	What is n?
Who's (n ANY)?	Who is n?
Where's (n ANY)?	Where is n?
How's (n ANY)?	How is n?
When's (n ANY)?	When is n?
(n1 ANY) which (n2 ANY)?	n1 what n2?
In what year (n ANY)?	What year n?
(n1 ANY) used to be (n2 ANY)?	n1 were n2?
Name (n ANY)?	Name n.
What was the name of (n ANY)?	Name n.
What were the names of (n ANY)?	Name n.
What is the name of (n ANY)?	Name n.
What are the names of (n ANY)?	Name n.
What was (n ANY) called?	Name n.
What was (n ANY) named?	Name n.
How was (n ANY) called?	Name n.
How was (n ANY) named?	Name n.

Figure 16: Canon rules for question simplification

Figure 16 shows the canon rules taken from the Web-QA component used for question simplification.

3.3.3 Learning patterns

To generate question patterns and answer patterns, we applied the following algorithm to each individual question-answer pair from the training corpus. Each step will be explained in detail:

1. Split question-answer pair into question and answer part
2. Extract the arguments
    (a) Extract the question arguments
    (b) Answer sub-phrasing
    (c) Extract the answer arguments
3. Extract the relation between the question arguments and the answer
4. Generate question patterns (matching rules)
5. Generate answer patterns
    (a) Construct the query using question and answer arguments
    (b) Submit the query to a search engine
    (c) Convert the documents from HTML to TEXT
    (d) Apply a sentence splitter to the documents
    (e) Pass each sentence to syntactical sentence selection
    (f) Pass each selected sentence to semantic sentence selection
    (g) Eliminate redundancy
    (h) Detect the sentence noun phrases
    (i) Construct the answer pattern

We describe this procedure in detail with an example:

1. **Split the question-answer pair**

    Each question-answer pair is split into two parts: the question part and the answer part. For simplicity, we refer to the question part as Q-PART and to the answer part as ANS-PART.

    For example, the question-answer pair:

    ```
    479 Who provides telephone service in Orange County, California? Pacific Bell
    ```

is splited into the following parts:

```
Q-PART :  Who provides telephone service in Orange County, California?
ANS-PART: Pacific Bell
```

To do this task we used the sentence splitter presented in section 1.4.3.

2. **Extracting the arguments**

   We define an argument set for each question-answer pair as the set of terms which we believe a relevant document should contain. To give an example, consider the question-answer pair:

   ```
   479 Who provides telephone service in Orange County, California? Pacific Bell
   ```

   Any relevant document to this question-answer pair must contains the terms *"telephone service"*, *"Orange County"*, *"California"*, and *"Pacific Bell"*.

   Therefore to search documents on the Web, we formulate a query made up of all the arguments found in the question-answer pair.

   To obtain the argument sets, at first the question is syntactically analyzed to detect its noun phrases. All the noun phrases detected in the Q-PART are grouped in the set called Q-ARGUMENT. Then we form all the possible sub-phrases of the candidate answer, and we group the original candidate answer and all its sub-phrases in the ANS-ARGUMENT set.

   (a) **Extracting the question arguments**

      To extract noun phrases, at first, we need to assign part-of-speech tags (noun, verb, preposition, etc.) to the individual words. Then, we use an NP chunker to select and group the word sequences that are recognized as noun phrases in brackets. Eric Brill's part-of-speech tagger (see section 1.4.3) is used to assign the part-of-speech tags, and NP chunking is done with the BaseNP chunker (see section 1.4.3).

      We explain this process with the following example:

      i. **Apply part-of-speech tagger on the Q-PART**

         After applying Brill's part-of-speech tagger to the Q-PART, the result is:

```
479/NN Who/WP provides/VBZ telephone/NN service/NN in/IN Orange/NNP
County,/NNP California/NNP
```

ii. **Apply noun phrase chunker**

The result from part of specch tagging is passed to noun phrase chunker, to identify noun phrase chunks. The results after applying the NP chunker is:

```
[ 479/NN Who/WP ] provides/VBZ [ telephone/NN service/NN ] in/IN [
Orange/NNP County,/NNP California/NNP]
```

In this result, each word sequence in brackets represents one noun phrase.

iii. **Group the noun phrases**

We extract all the word sequences in the brackets from the chunked sentence:

```
Q-ARGUMENT1 = [ telephone/NN service/NN ]
Q-ARGUMENT2 = [ Orange/NNP County,/NNP California/NNP ]
```

Then the part-of-speech tags are removed and the noun phrases from the Q-PART are kept in the Q-ARGUMENTS. For example, this step produces the following set :

```
Q-ARGUMENTS = {``telephone service'', ``Orange County, California''}
```

(b) **Answer sub-phrasing**

The candidate answer (ANS-PART) is a noun phrase that can contain one or more words. Some supporting documents may only contain a part of this noun phrase. To increase the precision of document retrieval we search for a combination of question arguments and each sub-phrase of the answer. We restrict each sub-phrase to contain less than four[2] words and it also does not contain any stop word. To give an example of answer sub-phrasing, consider the question-answer pair from the previous step:

```
479 Who provides telephone service in Orange County, California? Pacific Bell
```

The ANS-PART for this question-answer pair is:

```
Pacific Bell
```

We break down the ANS-PART into the following sub-phrases:

[2]This limit was set arbitrary.

```
sub-phrase1 = { Pacific }

sub-phrase2 = { Bell }
```

For example, the following sentence contains all the question arguments, but only one of the answer sub-phrases and not the complete ANS-PART:

```
''Southwestern Bell company, the subsidiary that provides telephone
network service, is bringing high tech home to millions of people in
Orange County, California.''
```

We assign a score to each sub-phrase according to the number of words in the sub-phrase divided by the number of words in the candidate answer. Figure 17 shows the score assigned to each of the above sub-phrases:

Answer Sub-phrase	**Sub-phrase Score**
Pacific Bell	1
Pacific	1/2
Bell	1/2

Figure 17: Example of answer sub-phrases and their score.

(c) **Extract the answer arguments**

The answer arguments set is built using the original candidate answer and its sub-phrases. The set is ordered by the rank of sub-phrases.

The following set is the answer arguments set for the given question-answer pair in the previous example:

```
ANS-ARGUMENT = { ''Pacific Bell'', ''Pacific'', ''Bell'' }
```

3. **Extraction of relation between arguments**

The key aspect of this research is to find equivalent semantic relations which is defined as the relation held between question entities and the answer. In fact semantic relations is used in measuring the relevance of documents with the question-answer pair. We assumed that the semantic relation generally appears as a main verb in the question. For example, the verb *'provide'* is considered as the semantic relation in the following question-answer pair.

```
479 Who provides telephone service in Orange County, California? Pacific Bell
```

If the main verb that occurred in the question is an auxiliary verb, then we ignore the semantic relation. For example, given the question:

```
Who is the president of Stanford University? Donald Kennedy
```

The semantic relation is ignored and the validation of the relevant document is based on the frequency of the answer context.

The representation of the above concepts is done in the following constructs:

```
479 Who provides telephone service in Orange County, California? Pacific Bell
Relation = X(argument1) provides(Verb) Y(argument2)
Relation = provide
```

Once the semantic relation is determined, we generate a semantic representation vector. This vector is composed of the relation, its synonyms, one-level hyponyms and all hypernyms that are obtained from WordNet (see section 1.4.2). We assign a weight to each term in the vector based on the semantic distance of the term to the original verb in WordNet. Since there can be many senses for the linguistic expression of the relation, we consider the most frequent sense among them.

The representation of above concepts is done in the following constructs:

```
Relation: provide
Sense = provide
Synonyms(provide) = { supply, provide, render, furnish, ply, cater,
give, stipulate, qualify, condition, specify,put up, provide, offer,
engage, wage}

Hyponyms(provide) = {charge, date, feed, calk, fund, stint, skimp,
scant, terrace, terrasse, innervate, offer, signalize, extend, offer,
stock, stock, buy in, caption, rail, grate, capitalize, capitalise,
alphabetize, wharf, air-condition, uniform, railroad }
```

```
Hypernyms(provide) = { give, transfer stipulate, qualify, condition,
specify, contract, undertake, promise, assure, declare, state, say,
tell, express, utter, give tongue to }
```

```
Semantic representation vector = { provide, supply, render, furnish, ply, cater,
give, stipulate, qualify, condition, specify,put up, offer,
engage, wage, charge, date, feed, calk, fund, stint, skimp,
scant, terrace, terrasse, innervate,signalize, extend,,
stock, stock, buy in, caption, rail, grate, capitalize, capitalise,
alphabetize, wharf, air-condition, uniform, railroad, give, transfer,
contract, undertake, promise, assure, declare, state, say, tell,
express, utter}
```

4. **Generating Question patterns (matching rules)**

In the previous stage we identified the noun phrase chunks in the question. To build a general form for each question, we replace the proper nouns [NPP] in the chunked question by $< PROPER >$ and the other noun phrases by $<$ *any* sequence of *words* $>$.

Consider this chunked question from the example in the previous step:

```
[ 479/NN Who/WP ] provides/VBZ [ telephone/NN service/NN ] in/IN [
Orange/NNP County,/NNP California/NNP]
```

After replacing each noun phrase with $<$ *any* sequence of *words* $>$, the following string is produced:

```
479 Who/WP provides/VBZ <any sequence of words> in/IN <any sequence of
words>?
```

To achieve a more general form of the question pattern, the lexicon words are removed from the above string (except for the question word and prepositions):

```
479 Who/WP VBZ <any sequence of words> in/IN <any sequence of words>?
```

We remove grammatical words from the string. To remove these words we use a stop word list that contains 600 entries. The pattern after removing stop-words is:

```
479 ''Who'' VBZ <any sequence of words> ''in'' <any sequence of words>?
```

Then, the sequence of words that are adjacent are merged.

```
''Who'' VBZ <any sequence of words> ''in'' <any sequence of words>?
```

Because later we will add these patterns to the Web-QA component reformulation pattern set, for compatibility we replace each tag with one of the symbols in Figure 18 that have been used in the Web-QA component matching rules:

Name	**Description**
ANY	Matches any sequence of words
IN	Matches any one of a list of given words
Vsf	Matched a verb in simple form
Vpp	Matches a past participle
Vpast	Matches a verb in simple past
NOUN	Matches a noun
ADJ	Matches an adjective
PROPER	Matches the proper noun

Figure 18: Symbols used in Web-QA matching rules

We replace each $< any$ sequence of $words >$ by $< n\ ANY >$. If there is more than one $< any$ sequence of $words >$, then each one is replaced with $< nx\ ANY >$, where x is the number of the tag. For instance, if the pattern contains two $< any$ sequence of $words >$ then the first is replaced by $< n1\ ANY >$ and the second with $< n2\ ANY >$.

At the end, other tags are replaced with the corresponding matching symbol. Therefore the final question pattern for our example is:

```
479 "Who" (a Vsf) (x ANY) ''in'' (y PROPER)?
```

Figure 19 shows the question patterns generated from the Who_Corpus.txt training set.

5. **Generating answer patterns**

```
"Who" (v IN "is" "was") (x Proper)?
"Who" (v IN "is" "was") (x ANY)?
"Who" (a Vsf) (x ANY)?
"Who" (v IN "is" "was") (x ANY) "in" (y ANY)?
"Who" (a Vsf) (x ANY) "in" (y PROPER)?
"Who" (v IN "is" "was") "Who" (a Vpast) (x ANY)?
"Who" (v IN "is" "was") (x ANY) "to" (v Vsf) "in" (y ANY)?
"Who" (v IN "is" "was") (x ANY) "to" (v Vsf) "on" (y ANY)?
"Who" (v IN "is" "was") (a Vpp) (x ANY) "in" (y ANY)?
"Who" (v IN "is" "was") "Who" (a Vpast) (x ANY) "on" (y ANY) "in" (n3 ANY)?
"Who" (v IN "is" "was") "Who" (v IN "is" "was") (a Vpp) "to" (v Vsf) "to" (x ANY)?
```

Figure 19: Question patterns generated from training corpus for 'WHO' questions

In this step we acquire one or more answer patterns for each of the question patterns generated by the previous step. Here, we discuss the complete process of acquiring answer patterns (also shown in Figure 20):

(a) **Construct the query using question and answer arguments**

The query is a representation form of the user's information needs in the format accepted by the search engine. The query is a combination of the query terms and boolean or arithmetic operations. An Alta Vista [Digital Equipment Corporation's Research lab in Palo Alto1995] the query terms must connect with Boolean operators AND, OR, NEAR, while in Google the query terms are connected by arithmetic operations such as + and - (equivalent to 'AND' and 'NOT').

At this stage, we constructed the query in the format accepted by the Google search engine. The queries formed using all the arguments extracted from the question, and the original candidate answer or one of its sub-phrases, are conjugated with arithmetic operators. Consider the following question-answer pair:

```
479 Who provides telephone service in Orange County, California? Pacific Bell
```

The argument sets extracted from question-answer pair are:

```
Q-ARGUMENTS = { ''telephone service'', ''Orange County, California'' }
ANS-ARGUMENTS = { ''Pacific Bell'', ''Pacific'', ''Bell'' }
```

In our example, we use each of the following combination in building the query:

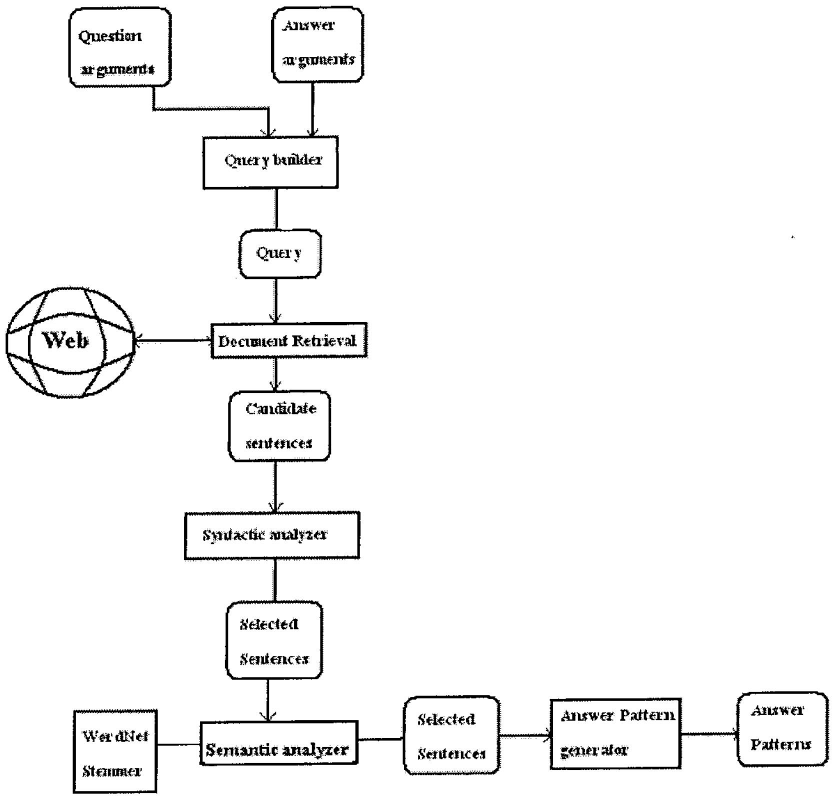

Figure 20: The flowchart of generating answer patterns.

```
{ ''telephone service'', ''Orange County, California'', ''Pacific Bell'' }

{ ''telephone service'', ''Orange County, California'', ''Pacific'' }

{ ''telephone service'', ''Orange County, California'', ''Bell'' }
```

Using these arguments, our system formulated the following queries to be used in the search engine:

```
''telephone service'' + ''Orange County, California'' + ''Pacific Bell''
```

which were formulated into the URL [3]:

```
http://www.google.ca/search?num=100&hl=en&as_qdr=all&q=%22telephone+service%22%2B
%22Orange+County,+California%22%2B%22Pacific+Bell%22+-trec+-quiz&
btnG=Google+Search&meta=
```

[3]+-trec+-quiz has been used to filter the webpages contain the trec or quiz documents

(b) **Submit the query to a search engine**

The document retrieval module retrieves the most relevant documents to the query from the document collection. We implemented this module using the Google search engine. We post the structured query to this search engine and then we scan the first 500 retrieved documents to identify the sentences that are likely to have the answer.

(c) **Convert the documents from HTML to TEXT**

In this module, we remove all HTML tags from the selected documents. To do this we use an HTML2TEXT converter to convert the HTML files to plain text format.

(d) **Apply a sentence splitter to the documents**

We use the sentence splitter to split each document into sentences. For example, the reults after applying sentence splitter is:

```
Sentence 1: ‘‘But few people are happy with the way the law has worked
out.’’
Sentence 2: ‘‘California’s Baby Bell, SBC Pacific Bell, still provides nearly
all of the local phone service to residents in Orange County,
California.’’
Sentence 3: ‘‘However, the company says its profits are being hurt by
unreasonably low lease rates.’’
```

(e) **Pass each sentence to syntactical sentence selection**

We post each of the above sentence to the syntactical sentence selection modul. This module selects only those sentences that contain all of the question arguments and at least one answer argument. In our example, the following sentence is retained:

```
Sentence 2: ‘‘California’s Baby Bell, SBC Pacific Bell, still provides nearly
all of the local phone service to residents in Orange County,
California.’’
```

(f) **Pass each selected sentence to semantical sentence selection**

Here, we select our final sentences for further processing according to the validity of the semantic relation that they contain.

From the sentences selected in the previous step, we only choose those that have the same relation as the relation extracted from the question-answer pair.

To do so, we examine all verbs in the selected sentences for a possible semantic relation. We take into account the part-of-speech tags of the words in the sentence. We check if the main verb of the sentence is a synonym, hypernym, or hyponym of the original verb in the question. The verb is valid if it occurrs in the semantic representation vector.

For example, given the following question-answer pair:

```
479 Who provides telephone service in Orange County, California? Pacific Bell
```

The semantic sentence selection module selects the following sentences:

```
Sentence 1: ‘‘California’s Baby Bell, SBC Pacific Bell, still provides nearly
all of the local phone service to residents in Orange County, California,
California. ’’
Sentence 2: ‘‘Pacific Bell Telephone Services today offers the best long
distance rate in Orange County, California.’’
```

Because both sentnces contain a verb (*"provide" and "Offer"*) that is included in the semantic representative vector of the question's verb (*"provide"*).

At first, we only attempt to validate verbs but if the semantic relation is not found through the verbs, then we also validate nouns and adjectives because the semantic relation may occur as a noun or adjective in the paragraph. In such case we use the Porter stemmer to find the stem of the adjectives and nouns and then we check if it has the same stem as the original verb or another verb from its semantic representative vector. For example, for the phrase *"provider of"* we check if the stem of the original verb *"provide"* or one of its synonyms is same as *"provide"* (the stem of *"provider"*).

For example, the following sentence is also selected by the semantic sentence selection module for the above question:

```
Sentence 3: ‘‘Pacific Bell, major provider of telephone service in
Orange County, california.’’
```

(g) **Eliminate redundancy (Eliminate repeated sentences)**

Some sentences may appear more than once in the selected sentences because the contents of two or more websites may be the same, or the sentence is repeated in many news articles. This step eliminates such redundancy in the candidate sentence collection.

(h) **Detect the sentence noun phrases**

Here, each sentence obtained from the previous step is syntactically analyzed to detect its noun phrases, using the same strategy employed for generating matching rules. For example, consider the following sentence:

```
``California's Baby Bell, SBC Pacific Bell, still provides nearly
all of the local phone service to residents in Orange County,
California.''
```

After applying Brill's part-of-speech tagger to the sentence, the result is:

```
479/CD California's/NNP Baby/NNP Bell,/NNP SBC/NNP Pacific/NNP
Bell,/NNP still/RB provides/VBZ nearly/RB all/DT of/IN the/DT local/JJ
phone/NN service/NN to/TO residents/NNS in/IN Orange/NNP County,/NNP
California./NNP
```

The results after applying the noun phrase chunker is:

```
479 [ California's/NNP Baby/NNP Bell,/NNP SBC/NNP Pacific/NNP
Bell,/NNP ] still/RB provides/VBZ [ nearly/RB all/DT ] of/IN [ the/DT
local/JJ phone/NN service/NN ] to/TO [ residents/NNS ] in/IN [
Orange/NNP County,/NNP California./NNP]
```

We remove all tags from the brackets. The result will be:

```
479 [ California's Baby Bell, SBC Pacific Bell,] still/RB provides/VBZ
[ nearly all] of/IN [ the local phone service ] to/TO [ residents ]
in/IN [ Orange County, California.]
```

Each chunked sentence is then passed to the answer pattern constructor for further processing.

(i) **Construct answer patterns**

To construct a general form for answer patterns, we replace the noun phrase for ANS-ARGUMENT by the string $< ANSWER >$ and the noun phrases for Q-ARGUMENT by the string $< QTERM >$. If more than one noun phrase has been identified in the Q-PART, then the same strategy that was used for generating matching rules is used and each noun phrase is replaced with $< QTERMx >$, where x is the number of QTERM. We replace the other noun phrases that are neither question argument nor answer argument, with $< nx\ ANY >$, where x is the noun phrase counter.

Consider this NP chunked from the example in previous step:

```
479 [ California's Baby Bell, SBC Pacific Bell,] still/RB provides/VBZ
[ nearly all] of/IN [ the local phone service ] to/TO [ residents ]
in/IN [ Orange County, California.]
```

After replacing the noun phrases by the appropriate tag, the following string is produced:

```
479 <ANSWER0> provides/VBZ  <QTERM0> ''to'' <QTERM1>
```

To achieve a more general form of the answer pattern, the words and the stop-words except the prepositions (*in, on, to, etc.*) along with their POS tags are removed from the above string:

```
479 <ANSWER> (v Vsf) <QTERM0> ''to'' <QTERM1>
```

We then replace each tag with one of the symbols in Figure 18 that have been used in the Web-QA component answer patterns:

```
479 <ANSWER> v x ''to'' y
```

At the end we replace the $< ANSWER >$ tag with the corresponding named-entity tagger for the answer.

```
479 ORGANIZATION v x ''to'' y
```

where, x and y are the first and second question arguments, and v is the question's main verb in present tense.

We apply the above algorithm to all candidate patterns obtained from the previous step. from this question with its different answer pairs: Figure 21 shows the question and answer patterns generated for our example question-answer pairs.

479 "Who" (v Vsf) (x ANY)?	479 ORGANIZATION (v Vsf) x y
	479 ORGANIZATION x y
	479 ORGANIZATION x v y

Figure 21: Question and answer patterns generated from the question "479 Who provides telephone service in Orange County, California?"

3.3.4 Assigning a weight to each pattern

The last challenge is to assign a weight to each candidate pattern. This helps us to rank the pattern list by the quality and precision of them. From our experiments we found that the length of a pattern, the answer sub-phrase score, and the level of semantic similarity between the main verbs of the pattern and the question are the most indicative factors in the quality of each pattern. We should also consider the frequency of each pattern. We set up a function to produce a weight for each pattern over the above major factors; these weights are defined to have values between 0, and 1.

At first we define how we compute each factor and then we will define a function to calculate a weight for the pattern. Note that the function we come up with is not the optimal way for combining these contributing factors, nor the factors are complete by any means; however, as long as it produces an acceptable ranking, we can apply them to the patterns; finding the best function would involve mathematical optimization which is out of the scope of this project. The error produced by just a simple acceptable ranking function is negligible compared to the error present in other modules of the Web-QA system, such as Named Entity Recognizer.

Let P_i be the ith pattern of the pattern set P extracted for a question-answer pair; we compute each factor as the following:

- $length(P_i)$ is the length of the pattern P_i measured in words; so, a shorter pattern will have a better rank.

- sub_phrase_score is the score of the candidate answer sub-phrase.

The score of each answer sub-phrase depends on its similarity to the candidate answer. Here we have used the simple heuristic method to score a sub-phrase defined as the following: $Sub - phrase - score = \frac{number\ of\ words\ that\ are\ present\ into\ both\ p_i\ and\ candidate\ answer}{total\ number\ of\ words\ in\ the\ candidate\ answer}$

- $semantic\_similarity(V_Q, V_{P_i})$: Given a verb of candidate pattern (V_{P_i}) and the original verb in the question (V_Q), we want to estimate the likelihood that the two verbs are paraphrases of the same fact or event. We thus assign a weight for a given verb (V_{P_i}) which is based on the semantic distance of that verb to the original verb in WordNet. We compute the semantic-similarity by the following way:
  - 1 If the verb in the answer is same as the original verb in the question.
  - 1/2 If the verb is a strict synonym of the original verb, i.e. a verb is in the same synset.
  - 1/4 If the verb is extended synonym of the original verb, i.e a verb in the second level of synonymy.
  - 1/8 If the verb is a hyponym or a hypernym of the original verb.

The final weight for the pattern is based on the combined score of the previous four factors. We used multiplication to maximize the contribution of each factor. In other words, a high weight is produced only if all factors indicate high quality.

The combined score is computed as:

$Weight(P_i) = \frac{count(P_i)}{count(P)} \times \frac{1}{length(P_i)} \times \frac{1}{distance} \times sub\_phrase\_score(used) \times semantic\_similarity(V_Q, V_{P_i})$

Figure 22 shows an example of some candidate answer patterns along with their weight.

Weight	Candidate Pattern
1.00	(x TITLE)
0.77	(x (past v) TITLE)
0.59	(x v TITLE)
0.43	(x "is" TITLE)
0.24	(x "was" TITLE)
0.29	(x (a Vpp) TITLE)

Figure 22: Example of candidate answer patterns and their weight.

Figure 23 shows an example of a question pattern along with its ranked answer patterns. Appendix 7 provides original and generated patterns.

Question Pattern	Answer Pattern
((”Who” (v IN ”is” ”was”) (x PROPER)?)	1.00 (x TITLE)
	0.77 (x (past v) TITLE)
	0.59 (x v TITLE)
	0.43 (x ”is” TITLE)
	0.24 (x ”was” TITLE)
	0.29 (x (a Vpp) TITLE)

Figure 23: Example of question pattern and its ranked answer patterns.

3.4 Evaluation

In this section we discuss the results of the evaluation of our learning patterns.

To measure the effects of our learning algorithm on the performance of the Web-QA component, we employed two methods: Precision and MRR (see section 1.6).

We tested our learning approach using the 493 questions from the TREC-11 collection data. We submitted these questions to the Web-QA component system which uses our learned patterns. The system was evaluated by MRR and precision. Then the answers from both runs were compared. The results are reported in Tables 13 and 14 and 15.

Question type	Nb of questions	Nb of questions with at least one candidate answer	Nb of questions with a correct answer in the top 5 candidates	Precision of candidate list
who	52	35	20	0.571
what	266	86	42	0.500
where	39	15	8	0.533
when	71	16	11	0.687
how + adj/adv	53	18	5	0.277
which	12	3	0	0
why	0			
Total	493	171	92	0.538

Table 13: Results with the original version of the Web-QA for each category

Tables 13 and 14 show the result of this comparision baed on precession and the number of questions with at least a candidate answer.

Table 15 shows the MRR for each type of question.

We then compared the answers found by the original Web-QA component and the Web-QA

Question type	Nb of questions	Nb of questions with at least one candidate answer	Nb of questions with a correct answer in the top 5 candidates	Precision of candidate list
who	52	37	24	0.648
what	266	76	42	0.552
where	39	19	11	0.578
when	71	25	18	0.720
how + adj/adv	53	13	6	0.462
which	12	5	0	0
why	0	0	0	0
Total	493	175	113	0.646

Table 14: Results with the generated patterns for each category

Question Type	Frequency	Before		After	
		MRR	Precision	MRR	Precision
who	52 (10.4%)	0.301	0.571	0.396	0.648
what	266 (53.2%)	0.229	0.500	0.317	0.552
where	39 (7.8%)	0.500	0.533	0.348	0.578
when	71 (14.2%)	0.688	0.687	0.643	0.720
how + adj/adv	53 (10.6%)	0.194	0.277	0.310	0.462
which	12 (2.4%)	0	0	0	0
why	0 (0.0%)	0	0	0	0

Table 15: The results based on question categories.

component with new generated patterns. The table 15 shows high improvement for questions like "Who is x?" and "Who (verb) x?" has been obtained.

Table 17 shows the results for original Web-QA component and the results for Web-QA component with new generated patterns. We can see the Web-QA component performs higher precesion and recall using new generated patterns. We believe the potential improvement in precision and recall is more than what is shown in the results table; for now our results are limited to the Web-QA component: the tags in reformulation patterns that are recognized by this system is very limited,

Corpus	System	Nb of questions	Nb of questions with at least one candidate answer	Nb of questions with a correct answer in the top 5 candidates	Precision of candidate list
TREC-11	Original system	493	171	92	0.538
TREC-11	New system	493	175	113	0.646

Table 16: Results with the original version of the Web-QA and web-QA2

preventing a greater granularity of patterns; for example, currently there is no differentiation between past, past participle, or third person verbs. This makes most of the new reformulated patterns we extracted not recognizable by the Web-QA component. Another problem is that noun phrases that are not in the question (refered to as extra noun phrase in our work) can not be included in the pattern repository of current Web-QA (see section 1.8.2). Also since the Web-QA component only employed some simple syntactic check to approximate the type of the expected answer, the new patterns generate a high percentage of noisy candidate answers which lead us to have a lower MRR.

3.5 Conclusion

In this chapter we presented a method for learning reformulations patterns as well as techniques for evaluating the quality of the generated patterns.

The experimental evaluation of our learning reformulations algorithm shows that using new generated patterns does not increase the precesion and MRR significantly, mainly due to the fact that many of the generated patterns are not usable by Web-QA.

There is some possibility to improve the results by a number of modifications to the Web-QA component. For example, verb granularity should be changed and noun phrases that are not in the question should be included in the pattern repository of current Web-QA (see section 1.8.2).

Chapter 4

Filtering out bad answers

4.1 Introduction

Although the Web component was meant to complement the core QA system of QUANTUM, when evaluated on its own, its results were very low. Table 2 in Section 1.8.2 shows the evaluation of the Web-QA component alone with the TREC-9 and TREC-10 data. Although most questions from the data set were actually covered by the reformulation patterns and generated at least one reformulation, only a small number of reformulations actually retrieved a correct answer in the top 5 candidates. The precision of the list of candidate answers is rather low. For example, with the TREC-9 question set, only 27% of the questions for which a list of candidates was retrieved contain a correct answer in the top 5 candidates. Our next goal was then to improve these results by filtering out the *noisy* candidates and re-rank the remaining candidates better. This chapter is explained in detail in [Kosseim and Yousefi2004].

4.2 An overview of the filtering and re-ranking algorithm

To filter and better rank the results of the Web-QA component, we were inspired by the approach used in the WebStorm System [Duclaye et al.2002, Duclaye et al.2003]. As we mentioned in section 2.3.2, in WebStorm, the authors use the web as a linguistic resource to learn similar semantic relation holding between entities automatically. They start with one single prototypical argument tuple of a given semantic relation and search for potential alternative formulations of the relation, then find new potential argument tuples and iterate this process to progressively validate the candidate formulations. Their approach focuses on the use of paraphrases as a potential way to improve question answering systems.

Similarly to their work, we extract the semantic relation expressed in the question, and create argument tuples from the noun phrases expressed in the question and the noun phrases that the Web-QA module identifies as candidates. We then run the Web-QA module again, but this time, only these tuples are used to try to find new evidence of the original relation, expressed in various linguistic forms. To do so, we use WordNet [Miller1995] and the Porter stemmer [Porter1980] to identify semantically related verbs, nouns, and adjectives.

4.3 The filtering and re-ranking algorithm

Our filtering and re-ranking algorithm is made up of the following steps:

1. Running the Web-QA component and retrieving its top candidates
2. Running a named entity tagger
3. Extracting the semantic relation and argument tuples for each candidate
4. Document retrieval
5. Validating the semantic relation
6. Re-ranking the candidate answers by their frequency

In the following section, this process will be described in detail.

4.3.1 Running the Web-QA Component

We first run the Web-QA component and retrieve its top 200 [1] candidates. For example, with the question *'Who killed Martin Luther King?'*, the Web-QA module retrieves the following candidates:

```
Bobby Kennedy
```

[1]This number was set arbitrary.

```
Activism Exit RW ONLINE Who
Who Really
Crawford The Real Reason They
James Earl Ray
Dream Who n
```

4.3.2 Running a named entity tagger

Since the Web-QA Component only employed some simple syntactic check to approximate the type of the expected answer, the system generates a high percentage of noisy candidate answers. In order to filter such noisy candidates, all the candidates are tagged using the GATE-NE named entity tagger, and only the candidates that satisfy the predicted constraints of the answer are kept. For example, among the candidates generated in the previous step, only the following candidates will be kept.

```
Bobby Kennedy
James Earl Ray
```

4.3.3 Building the argument tuples and the semantic relation

Similarly to the previous chapter, we build a set of argument tuples composed of the candidate answers and the argument expressed in the question. In order to do this task, we decompose the original question into two parts: the main semantic relation expressed (ex. *killed*) and the argument(s) of the relation (ex. *Martin Luther King*). A set of argument tuples are then created from the noun phrases of the question and the candidate found by the Web-component. In our case, the following tuples are created:

```
(Martin Luther King, Bobby Kennedy)
(Martin Luther King, James Earl Ray)
```

4.3.4 Document retrieval

Once we have built the set of argument tuples, we search them in the document collection to identify the possible semantic relations relating them, and make sure that the relation that relates

$$\boxed{\underbrace{\textit{pre-window}}_{\text{N words or less}} \quad \underbrace{\texttt{Martin Luther King}}_{\text{argument 1}} \quad \underbrace{\textit{in-window}}_{\text{N words or less}} \quad \underbrace{\texttt{Bobby Kennedy}}_{\text{argument 2}} \quad \underbrace{\textit{post-window}}_{\text{N words or less}}}$$

Figure 24: Example of a context window for the argument tuple
(`Martin Luther King, Bobby Kennedy`)

them in the documents is equivalent to what we were originally looking for in the question.

In our experiment, we submitted all the tuples to both the TIPSTER collection used in TREC 8-10 and the Web to find paragraphs that contain these tuples. Then we extracted only the paragraphs where both tuple elements are at a distance of N[2] words or less. We used a context window size of N words between the tuple elements and N words on each side of them in the extracted paragraphs and then examined the words in these context windows for a possible similar semantic relation. This is shown in Figure 24.

For example, the excerpts found in the TIPSTER document collection for the tuple `(Martin Luther King, James Earl Ray)` are:

`James Earl Ray killed Martin Luther King.`

`James Earl Ray shot Martin Luther King.`

`James Earl Ray assassinated Dr. Martin Luther King.`

4.3.5 Examining the semantic relation

Finally, we evaluate the relations expressed in the context windows to identify if at least one is semantically equivalent to the original semantic relation expressed in the question. To verify the semantic relation, we first check if any verb found in any context window is a synonym, a hypernym or a hyponym of the original verb in the question.

If no verb has an equivalent semantic relation, we then backup to analyze other parts of speech. We try to validate nouns and adjectives by using the Porter stemmer [Porter1980] to find the stem of the words and we check if it has the same stem as the original verb or one of its synonyms. For example, if the phrase *the assassination of* appears in a context window, we check if the original verb `kill` in the question or one of its synonyms share the same stem with *assassination*. The stem

[2]In our experiment, N was set to 5.

assassin is indeed the same as the stem of the synonym *assassinate*. This process is similar to that already explained in chapter 3.

For example, the excerpts found in the Web for the tuple `(Martin Luther King, James Earl Ray)`, with other parts of speech carrying the semantic relation, are:

> `because the Mob contract was successful in killing` **Martin Luther King** `and framing` **James Earl Ray.**
>
> `Killing the Dream:` **James Earl Ray** `and the Assassination of` **Martin Luther King**`:Gerald Posner .`

Any tuple that cannot be found to have a similar semantic relation in the question and in the documents is thrown out.

4.3.6 Re-ranking the candidate answers

The remaining candidates are re-ranked according to the number of passages in the collection containing the same relation. For example, when we submitted the tuple `(Martin Luther King, James Earl Ray)` to the TIPSTER collection, we found 110 passages containing the elements of the tuple. Among these passages, only 24 contained the tuples and the relation `kill` within 5 words of each other. We therefore gave a rank of (24/110) to the candidate `James Earl Ray`. By applying this procedure to all the argument tuples, the five best ranked candidates can be easily found and selected.

4.4 Evaluation

We evaluated our approach with the questions from the TREC-9 and TREC-10 collection and compared the answers found this way with the original candidates. Table 17 shows the results of this evaluation. Although the number of questions with candidate answers (column 4) is inferior in the new system, the number of correct candidates (or actual answers) is greater or similar and the precision is higher. This means that, although we provide less candidates, they are more likely to constitute correct answers than before.

In TREC-9, MRR is increased by 40% and 25% of correct answers were ranked better; i.e. they moved up the list by 2.2 on average. At the same time, 5% of correct answers got worse ranks;

Corpus	System	Nb of questions	Nb of questions with at least one candidate answer	Nb of questions with a correct answer in the top 5 candidates	Precision of candidate list
TREC-9	Original system	694	63 (9.1%)	17	0.270
TREC-9	New system	694	28 (4.0%)	20	0.714
TREC-10	Original system	499	255 (51.1%)	154	0.603
TREC-10	New system	499	189 (37.8%)	152	0.804

Table 17: Results with the original version of the Web-QA and Web-QA2

moved down by 6 on average, and 5% of good answers were removed.

In TREC-10 however, MRR is increased by 12% when 31% of correct answers were ranked better by 4.5 on average while 9% were ranked worse by 3.5. Only one good answer was lost during the process.

4.5 Conclusion

The method described in this chapter improves the accuracy of our Web-QA module by re-ranking the candidates and by discarding those that do not contain the correct semantic relation.

As opposed to several other approaches that reinforce their candidate answers by looking on the Web; our approach is less strict as it looks for reinforcement of the semantic relation between the arguments, rather than looking only for lexically similar evidence. In this respect, our approach is much more tolerant and allows us to find more evidence. On the other hand, as we look for evidence in a window of N words, rather that a strict string match, we are more sensitive to mistakes and wrong interpretations. Indeed, we are only interested in finding a word that caries a similar sense without doing a full semantic parse of the sentence. Negations and other modal words may completely change the sense of the sentence, and we will not catch it. When looking in a very large corpus such as the Web, this may lead to more noise than a strict lexical string match approach. However, if we perform the QA task on a much smaller corpus, such as in closed-domain QA, looking for semantic equivalences may be more fruitful.

The current implementation of our approach only looks at semantic relations holding between pairs of arguments. However, it can easily be extended to consider variable-size relations. However, as more constraints are taken into account, the precision of the candidate list is expected to increase, but recall is expected to decrease. A careful evaluation would be necessary to ensure that the does not introduce too many constraints and consequently filters out too many candidates.

Chapter 5

Design and Implementation

This chapter briefly describes the design and implementation of the Web-QA2 system.

5.1 Implementation

The system has been written in the Perl programming language on the Linux platform. The user interacts with the system through the command line.

The system takes a file containing question-answer pairs as input, and returns a file containing a list of question-answer patterns along with their rank.

Figures 25 and 26 illustrate example of input and output files used in the system.

```
220 Who is the prime minister of Australia? Paul Keating
232 Who invented television? Philo Farnsworth, Vladimir Zworykin
255 Who thought of teaching people to tie their shoe laces? Starlace
54 When did Nixon die? April 22, 1994
1071 When was the first stamp issued? 1840
34 Where is the actress, Marion Davies, buried? Hollywood Memorial Park
```

Figure 25: Example of input file: **corpus.txt**

("Who" VBD $< n1ANY >$ VBN $< n3ANY >$ "?")	0.98 ($< PERSON >$ VBN n1)
	0.95 ($< PERSON >$ n1 VBN n2)
	0.80 (n1 VBN $< PERSON >$))

Figure 26: Example of output file: **pattern.txt**

Module Name	Description
analyzer	Analyzes each question-answer pair
brilltagger	Tags the question parts and save output into brilfile
chunker	Chunks the tagged questions and save output into chunkfile
NPextractor	Extracts noun phrases and save them in NP hash table
entitydeterminer	Determines the entities and save them in ENTITY hash table
namedentity	Determines the named-entity for each entity
anssubphraser	Extracts all the possible subpharses of answer and calculate their rank
nporganizer	organizes all the NPs extracted from question and answer for building query
documentretreival	Builds queries and search the web for relevant documents and save them in a file
Qualify	Extracts the sentences that contains all the NPs and at least one answer subphrase
brilltagger	Tags the qualify sentences and save output into brilparfile
chunker	Chunks the tagged sentences and save output into chunkparfile
questionpatterngenerator	Generates question patterns from the chunked questions
generalpatterngenerator	Generates answer patterns from the chunked sentences

Table 18: List of program modules

5.2 Modules and files

The implementation of our system consists of a main program and 15 modules. Each question of the training corpus is processed separately, and for each of them, the system calls all modules. Table 18 shows these modules and a brief description for each of them.

Table 19 shows the list of input and output files with their description.

File Name	Description
questionanswerfile	The file containing question-answer pairs from one category
questionfile	The file that saves all question parts of the question-answer pairs from one category
brilfile	The file that saves all the tagged question that have been tagged by Brill's tagger
chunkfile	The file that saves all the chunked questions that have been chunked by Base-np-chunker
docsfile	The file that saves all documents retrieved for the input file
brilparfile	The file that saves all the paragraphs that have been tagged by Brill's tagger
chunkparfile	The file that saves all the paragraphs that have been chunked by Base-np-chunker
questionpatternfile	The file that saves all the patterns generated for questions
answerpatternfile	The file that saves all the patterns generated for answers
patterns	The file that saves all the question patterns company with their answer patterns
answerfile	The file that keeps all the answers for identifying named entity
entityfile	The file that saves all the named entity for answers

Table 19: List of input and output files

Chapter 6

Conclusion and future work

The goal of this thesis was to improve the precision and MRR of the Web-QA Component synchronously. We combined our learning algorithm and the filtering methods to achieve our goal. In chapter 3 we presented a method for learning reformulations patterns as well as techniques for evaluating the quality of the generated patterns.

The experimental evaluation of our learning reformulations algorithm shows that using new generated patterns does not increase the precesion and MRR high enough, mainly due to the fact that many of the generated patterns are not usable by Web-QA. There is some possibility to improve the results of by a number of modifications to the Web-QA component.

The method described in chapter four improves the accuracy of our Web-QA module by re-ranking the candidates and by discarding those that do not contain the correct semantic relation.

As future work, a number of modifications to the Web-QA component are suggested: First, verb granularity should be changed; currently, present third person verbs (VBZ) are distinguished from other verbs (shown by VB). Many patterns realized in our work are not usable by Web-QA due to lack of a well grained distinction of verbs.

Second, noun phrases that are not in the question (refered to as extra noun phrase in our work) can not be included in the pattern repository of current Web-QA (see section 1.8.2). This shold defintily be improved.

Third, verbs that are not in the question can not be included in the pattern repository of current Web-QA. This shold defintily be improved.

Finally, for improvement of the quality of the patterns, we suggest a systematic evaluation and adjustment of the parameters that take part in weighting of the patterns. For example, the size of the windows and the sub-phrase scoring.

Bibliography

[Agichtein and Gravano2000] Eugene Agichtein and Luis Gravano. 2000. Snowball: Extracting relations from large plain-text collections. In *Proc. 5th ACM International Conference on Digital Libraries*, San Antonio, Texas, USA.

[Agichtein et al.2001] Eugene Agichtein, Steve Lawrence, and Luis Gravano. 2001. Learning search engine specific query transformations for question answering. In *Proceedings of WWW10*, pages 169–178, Hong Kong.

[Brill1992] Eric Brill. 1992. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing, ACL*, pages 152–155, Trento, Italy.

[Clarke et al.2000] Charles L. A. Clarke, Gordon V. Cormack, D. I .E. Kisman, and Tomas R. Lynam. 2000. Question answering by passage selection (multitext experiments for trec-9). In *The Ninth Text REtrieval Conference (TREC-9)*, pages 673–683.

[Clough2000] Paul Clough. 2000. Evaluation of a perl regular expression sentence splitter. In *Internal Report, Department of Computer Science, University of Sheffield.*

[Cunningham et al.1991] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Cristian Ursu, Marin Dimitrov, Mike Dowman, Niraj Aswani, and Ian Roberts. 1991. Gate: General architecture for text engineering. http://gate.ac.uk.

[de Rijke and Webber2003] Maarten de Rijke and Bonnie Webber. 2003. Knowledge-intensive question answering. In *Course notes, School of Informatics, Edinburg,* http://staff.science.uva.nl/m̃dr/Teaching/ESSLLI2003/qa-day-02-8up.pdf.

[Digital Equipment Corporation's Research lab in Palo Alto1995] CA Digital Equipment Corporation's Research lab in Palo Alto. 1995. A view from above. http://Altavista.com.

[Duclaye et al.2002] Florence Duclaye, Francois Yvon, and Olivier Collin. 2002. Using the Web as a Linguistic Resource for Learning Reformulations Automatically. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, pages 390–396, Las Palmas, Spain.

[Duclaye et al.2003] Florence Duclaye, Francois Yvon, and Olivier Collin. 2003. Learning paraphrases to improve a question-answering system. In *Proceedings of the Natural Language Processing for Question Answering Workshop at EACL (EACL'03)*, Budapest, Hungary, April.

[Francis and Kucera1982] W. Francis and H. Kucera. 1982. Frequency analysis of english usage: Lexicon and grammar. Boston: Houghton Mifflin.

[Green et al.1961] B. F. Green, A. K. Wolf, C. Chomsky, and K. Laughery. 1961. BASEBALL: An Automatic Question Answerer. In *Proceedings of the Western Joint Computer Conference 19*, pages 219–224.

[Hermjakob et al.2002] U. Hermjakob, A. Echihabi, and D. Marcu. 2002. Natural language based reformulation resource and wide exploitation for question answering. In *Proceedings of the TREC-2002 Conference.*

[Hermjakob1997] U. Hermjakob. 1997. Learning parse and translation decisions from examples with rich context. In *Ph.D. dissertation, University of Texas Austin.*, file://ftp.cs.utexas.edu/pub/mooney/papers/hermjakob-dissertation.

[Hirschman L1995] Thomson H.S. Hirschman L. 1995. Overview of evaluation in speech and natural langage processing. http://cslu.cse.ogi.edu/HLTsurvey/HLTsurvey.html.

[Hovy et al.2000] Eduard Hovy, Laurie Gerber, Ulf Hermjakob, Michael Junk, and Chin-Yew Lin. 2000. Question Answering in Webclopedia. In *Proceedings of the TREC-9 Conference. NIST.*, pages 655–673, Gaithersburg, MD.

[Jijkoun and de Rijke2004] Valentin Jijkoun and Maarten de Rijke. 2004. Answer selection in a multi-stream open domain question answering system. In *26th European Conference on Information Retrieval*, pages 99–111.

[Kosseim and Yousefi2004] L. Kosseim and J. Yousefi. 2004. Filtering out bad answers with semantic relations in a web-based question-answering system. In *Proceedings of the QA workshop ot Traitement Automatique de la Langue Naturelle (TALN-2004)*, pages 423–430 (volume RECITAL), Fez, Morocco.

[Kosseim et al.2003] L. Kosseim, L. Plamondon, and L.J. Guillemette. 2003. Answer formulation for question-answering. In Y. Xiang and B. Chaib-draa, editors, *Proceedings of The Sixteenth Conference of the Canadian Society for Computational Studies of Intelligence (AI'2003).*, Lecture Notes in Artificial Intelligence no. 2671, pages 24–34, Halifax, Canada, June. Springer-Verlag.

[Magnini2002] B. Magnini. 2002. Is it the right answer? exploiting web redundancy foranswer validation. In *Proceedings ACL 2002*, pages 425–432, Philadel-phia.

[Miller1995] G. Miller. 1995. WordNet: a Lexical Database for English. *Communications of the ACM*, 38(1):39–41, Nov, 2004.

[Moldovan et al.2003] D. Moldovan, M. Pasca, S. Harabagiu, and M. Surdeanu. 2003. Performance issues and error analysis in an open-domain question answering system. In *ACM Transactions on Information Systems*, pages 133–154.

[Office of the President1998] University of Virginia Office of the President. 1998. Planning progress report, september 18, 1998. http://www.virginia.edu/virginia2020/h2progress91898.htm.

[Plamondon et al.2001] L. Plamondon, G. Lapalme, and L. Kosseim. 2001. The QUANTUM Question Answering System. In *Proceedings of The Tenth Text Retrieval Conference (TREC-10)*, pages 157–165, Gaithersburg, Maryland.

[Plamondon et al.2002] L. Plamondon, G. Lapalme, and L. Kosseim. 2002. The QUANTUM Question Answering System at TREC-11. In *Notebook Proceedings of The Eleventh Text Retrieval Conference (TREC-11)*, pages 157–165, Gaithersburg, Maryland.

[Porter1980] M.F. Porter. 1980. An algorithm for suffix stripping. *Program*, 14(3):130–137.

[Ramshaw and Marcus1995] Lance Ramshaw and Mitchell Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*, pages 82–94, MIT.

[Rijsbergen1999] C.J. Van Rijsbergen. 1999. Information retrieval. In *Information Retrieval (Second Edition)*, pages 144–160, Glasco, Scotland.

[Rivachandram and Hovy2002] Deepak Rivachandram and Eduard Hovy. 2002. Learning surface text patterns for a question answering system. In *Proceeding of ACL Conference*, pages 41–47, Philadephia.

[Robertson et al.1994] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. 1994. Okapi at trec-3. In *The Third Text REtrieval Conference (TREC-3)*, pages 109–126.

[Soubbotin and Soubbotin2001] M.M. Soubbotin and S.M. Soubbotin. 2001. Patterns of potential answer expressions as clues to the right answers. In *Proceedings of The Tenth Text Retrieval Conference (TREC-X)*, pages 175–182, Gaithersburg, Maryland.

[Tan1999] Ah-Hwee Tan. 1999. Text mining: The state of the art and the challenge. In *Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*, pages 65–70, Beijing, China.

[Woods.1973] W. Woods. 1973. Progress in natural language understanding: An application to lunar geology. In *AFIPS Conference Proceedings, volume 42*, pages 441–450.

[Xu et al.2003] J. Xu, A. Licuanan, J. May, S. Miller, and R. Weischedel. 2003. TREC 2002 QA at BBN: Answer Selection and Confidence Estimation. In *Proceedings of TREC 2002*, pages 96–101.

Chapter 7

Appendix A

7.1 Example of generated patterns

7.1.1 "Who" Question Patterns

```
( ("Who" (v IN "is" "was") (x ANY) "to" (v Vsf) "on"  (y ANY)?)

(x "on" y PERSON )

(x "to" v "on" y PERSON )

(PERSON x "on" y )

(x "to" v "on" y PERSON )

(x "on" y PERSON ))


( ("Who" (v IN "is" "was") "Who" (a Vpast) (x ANY) "on"  (y ANY) "in"  (n3 ANY)?)

 (x "on" PERSON (past v) "in" y ))


( ("Who" (v IN "is" "was") "Who" (v IN "is" "was") (a Vpp) "to" (v

 Vsf) "to"  (x ANY)?)

 (x (past v) PERSON )

 (x v PERSON )

 (x a PERSON )
```

```
(x "that" PERSON )

(x "during" PERSON )

(x "was" PERSON )
```

7.1.2 "When" Question Patterns

```
( ("When" "did" (n ANY) (v Vsf) "?" )
(n (past v) TIME)
(TIME n (past v) )
(TIME "," n (past v) )
(n "as" TIME )
(n (past v) "on" TIME )
(n "was" (Vpp v) "in" TIME )
(n "was" (Vpp v) TIME )
(n "was" (Vpp v) "about" TIME )
(n (past v) "in" TIME )
)

( ("When" "did" (n ANY) (v Vsf) (n2 ANY) "?" )
(n (past v) n2 TIME) )

( ("When" "was" (n ANY) (p Vpp) "?" )
(n "was" p TIME)
(n "was" p "on" TIME )
(n "was" p "in" TIME )
(n p TIME )
(n p "in" TIME )
)
```

```
(  ("When" "was" (n LNP) (p Vpp) "?" )
(n p TIME )
(n p "from" TIME )
(n "from" TIME )
(n "was" p "on" TIME )
(n "was" p "in" TIME )
)

(  ("When" "was" "the" (n LNP) (n2 ANY) "?" )
("the" n "was" n2 TIME) )

(  ("When" "was" (n LNP) "?" )
(n "was" TIME)
(n "in" TIME )
(n "On" TIME )
(n "on" TIME )
)

(  ("When" "is" (n ANY) "?" )
(n "is" TIME)
(TIME "is" n)
(n "in" TIME )
(n "on" TIME )
(n "across" TIME )
(n "after" TIME )
)


(  ("When" "did" (n LNP) (v Vsf) "for"  (n2 ANY) ?)
(n n2 "in" TIME )

(n "as" n2 "in" TIME )

(n2 "is" n TIME )

(n "was" n2 "in" TIME )

(n (past v) "for" n2 "in" TIME )

(n "was" (Vpp v) "for" n2 "at" TIME )

)


(  ("When" "did" (n LNP) ?)

(n "about" TIME )

(n "to" TIME )

(n "by" TIME )

(n "between" TIME )

(n "than" TIME )

(n "around" TIME )

(n "was" TIME )

)
```

7.1.3 "Where" Question Patterns

```
(  ("Where" "did" (n ANY) (v Vsf) "?" )

(n (past v) LOCATION)

(n "on" LOCATION)

(n v "to" LOCATION )

(n "as" LOCATION )

(n "is" LOCATION )

(n "in" LOCATION )

)
```

```
(  ("Where" (v IN "is" "was" "were") (n ANY) (a Vpp) "?" )
 (n v a LOCATION)
 (n "was" LOCATION )
 (n "was" a "in" LOCATION )
 (n "is" a "in" LOCATION )
 (n "were" a "in" LOCATION )
 (n a LOCATION )
 (n "for" LOCATION )
 (n a "in" LOCATION )
 (n a "to" LOCATION )
 (n "were" LOCATION )
 (n a "in" LOCATION )
 (n "with" LOCATION )
 (n "was" a "throughout" LOCATION )
 (n "was" a "on" LOCATION )
)


(  ("Where" "is" (n ANY) (a Vpast) ?)
 (n "in" LOCATION )
 (n "is" a "in" LOCATION )
 (n "within" LOCATION )
 (n a "with" LOCATION )
 (n a "by" LOCATION )
 (n "is" LOCATION )
 (n "as" "in" LOCATION )
 (n a "in" LOCATION )
 (n "is" "in" LOCATION )
 (n "is" a "to" LOCATION )
)
```

```
 (  ("Where" (p Vpp) (n ANY) (a Vpp) ?)
  (n "from" LOCATION )
  (n a LOCATION )
  (n "at" TITLE )
 )

(  ("Where" "is" (n ANY) "?" )
  (n "is" "located" LOCATION)
  (n "," LOCATION )
  (n "," "located" LOCATION)
  (n "is" LOCATION )
  (n "on" LOCATION )
  (n "in" LOCATION )
  (n "was" LOCATION )
  (n "is" "on" LOCATION )
  (n "locate" "at" LOCATION )
  (n "are" LOCATION )
  (n "is" "located" "in" LOCATION )
)

(  ("Where" "is" (n LNP) ?)
 (n "is" LOCATION)
 (n "at" LOCATION )
 (n "in" LOCATION )
 (n "is" "located" "in" LOCATION )
 (n "on" LOCATION )
 (n "is" LOCATION )
 (n "with" LOCATION )
```

```
  (n "is" "in" LOCATION )

  (n "Down" LOCATION )

  (n "though" LOCATION )

  )


 (  ("Where" "are" (n LNP) (a Vpp) "?" )

  (n "are" a LOCATION) )


 (  ("Where" "are" (n ANY) "?" )

  (n "are" LOCATION) )



 (  ("Where" "did" (n LNP) (v Vsf) "?" )

  (n (past v) LOCATION) )


 (  ("Where" "did" (n LNP) (v Vsf) "?" )

  (n (past v) LOCATION) )


 (  ("Where" "did" (n ANY) (v Vsf) (x ANY) "?" )

  (n (past v) x LOCATION)

  (x "in" n LOCATION )

  )


 (  ("Where" "do" (n ANY) "?" )

  (n LOCATION) )


 (  ("Where"  "could"  (n ANY) (v Vsf) "to" (v Vsf) (n2 ANY) "on"  (n3 ANY) ?)

  (n v n2 "through" n3 v "with" LOCATION )

  )
```

```
( ("Where" "is" (n LNP) (a Vpast) ?)
 (n "in" LOCATION )
 (n LOCATION )
 (n "is" LOCATION )
 (n "to" a n LOCATION )
 (n a  LOCATION )
 (n "from" LOCATION )
)

( ("Where" "did" (n LNP) (p Vpp) ?)
 (n LOCATION )
 (n "in" LOCATION )
 (n p "in" LOCATION )
 (n "from" "across" LOCATION )
 )
```

7.1.4 "What" Question Patterns

```
( ("What" "does" (n ANY) (v Vsf) "?")
  (n (third-person v) CLAUSE))

(("What" "does" (n ANY) (v Vsf) "for"  ?)
 (n CLAUSE )
 (n "is" CLAUSE )
 (n (third-person v) CLAUSE )
 (n "can" v CLAUSE )
 (n "on" CLAUSE )
 (n v "for" CLAUSE )
```

```
  (n (past v) CLAUSE )
  (n "as" CLAUSE )
  (n "in" CLAUSE )
  (n "is" (past v) "in" CLAUSE )
  (n "from" CLAUSE )
  (n "was" (past v) "under" CLAUSE )
 )

( ("What" "are" (n ANY) (x Vpp) "?")
  (n "are" x CLAUSE) )

( ("What" (t ANY) "is" (n ANY) "?")
  (CLAUSE "is" n AND CLAUSE "is" "a" t))

( ("What" "is" "the" "name" "of" (n ANY) "?")
  (n "is" NAME))

( ("What" "year" "was" (x ANY) (a ADJ) "?")
  (x "was" a "in" NUMBER) )

( ("What" "year" "did" (x ANY) (a Vsf) "?")
  (x (past a) "in" NUMBER))

( ("What" "year" "did" (x ANY) (a Vsf) (y ANY) "?")
  (x (past a) y "in" NUMBER))

( ("What" "year" "was" (x ANY) (a ADJ) (y ANY) "?")
  (x "was" a y "in" NUMBER))
```

```
( ((x ANY) "in" "what" "year" "?")
  (x "in" NUMBER))


( ("What" "did" (n LNP) (v Vsf) (x ANY) "?" )
  ( n (past v) CLAUSE x ) )


( ("What" (v IN "is" "are" "was") (n ANY) "?")
  (n v CLAUSE))


( ("What" (n ANY) (v Vpast) (n2 ANY)"?")
  (CLAUSE v n2 AND CLAUSE "is" "a" n) )


( ("What" "did" (n ANY) (v Vsf) (n2 ANY) "?")
  (n (past v) CLAUSE n2) )
```

7.1.5 "Which" Question Patterns

```
( ("Which" (t ANY) "is" (n ANY) "?")
  (CLAUSE "is" n AND CLAUSE "is" "a" t)
  (n "are" CLAUSE))


( ("Which" (t ANY) (a ADJ) (n ANY) "?")
  (CLAUSE a n AND CLAUSE "is" "a" t)
 )


(  ("Which" (n1 ANY) (a Vpast) (n2 ANY) "in"  (n3 ANY) ?)
 (TITLE a n2 "in" n3)
 (TITLE "who" a n2)
 (n3 n2 TITLE)
 (n2 n3 TITLE)
 )

(  ("Which" (n1 ANY) "is" (a Vpp) "in"  (n2 ANY) ?)
 (n1 n2 PERSON)
 (PERSON "is" a "in" n2)
 (PERSON "is" "the" "only" n1 a "in" n2)
 (n1 PERSON "at" n2)
 )



(  ("Which" (n1 ANY) (a Vpast) (n2 ANY) ?)
 (PERSON "and" n2)
 (PERSON "known as" n2)
 (n2 a "by" PERSON)
 ("with" n2, PERSON)
 )

(  ("Which" (n1 ANY) (v Vsf) (n2 ANY) ?)
 ("in" LOCATION, n2)
 (LOCATION v n2)
 )

(  ("Which" (n1 ANY) (a Vpast) (n2 ANY) (n3 ANY)?)
 ("located" "in" n1, n2 "is" "a gift from" LOCATION)
 (n2 "in" n1 "was" a "by" LOCATION")
 (LOCATION a n2 "to" n1)
 (n2 "was" "a gift to" n1 "from" LOCATION")
 )

(  ((n1 ANY) "is" "close" "to" "which" (n2 ANY)  ?)
 (n2 "in" LOCATION )
 )
```

7.1.6 "How" Question Patterns

```
(  ("How" "many" (n1 ANY) "are" "there" (n2 ANY) "?" )
   ("there" "are" NUMBER n1 n2))

(  ("How" "many" (n ANY) "are" "there" "?" )
   ("there" "are" NUMBER n))

(  ("How" "many" (n1 ANY) "were" "ever" (a ADJ) "?" )
   (NUMBER n1 "were" a))
(  ("How" "many" (n1 ANY) "were" (a ADJ) "?" )
   (NUMBER n1 "were" a))

(  ("How" "many" (n1 ANY) "are" (n2 ANY) "?" )
   ("there" "are" NUMBER n1 n2))

(  ("How" "many" (n1 ANY) "ago" "did" (n2 ANY) (v Vsf) "?" )
   (NUMBER n1 "ago" n2 (past v)))

(  ("How" "many" (n1 ANY) "did" (n2 ANY) (v Vsf) (n3 ANY) "?" )
   (n2 (third-person v) NUMBER n1 n3 )) ; other cases
```

```
(  ("How" "many" (n1 ANY) "did" (n2 ANY) (v Vsf) "?" )

   (n2 (past v) NUMBER n1 )

   (n2 v "of" NUMBER )

   (n2 "Below" "are" NUMBER )

   (n2 "with" n1 (past v) "in" NUMBER )

 )


(  ("How" "many" (n1 ANY) "does" (n2 ANY) (v Vsf) "?" )

   (n2 (third-person v) NUMBER n1)

   (n2 NUMBER "are" n1 )

   (n2 (past v) NUMBER )

   (n2 v NUMBER )

)


(  ("How" "many" (n1 ANY) "does" (n2 ANY) (v Vsf) (n3 ANY) "?" )

   (n2 (third-person v) NUMBER n1 n3 )

   (n3 "," n2 (third-person v) NUMBER n1 )  ) ; other cases


(  ("How" "many" (n1 ANY) "were" "there" (n2 ANY) "?" )

   ("there" "were" NUMBER n1 n2 ))


(  ("How" "many" (n1 ANY) (v Vsf) (n2 ANY) "?" )

   (NUMBER n1 v n2))


(  ("How" "many"  (n1 ANY) "in"  (n2 ANY) ?)

 (n2 MONETARY )

 (n2 "is" MONETARY )

 (n2 MONETARY n2 )

 (n2 "was" MONETARY n1 )
```

```
  (n2 "by" MONETARY )

  (y "is" MONETARY "in" x )

 )


 (  ("How" "many"  (n1 ANY) (p Vpp) "on"  (n2 ANY) ?)

  (n2 NUMBER )

  (n2 v NUMBER )

  (n2 "is" NUMBER )

  (n2 "with" NUMBER )

)


 (  ("How" "many"  (n1 ANY) (p Vpp) (a Vpp) "on"  (n2 ANY) ?)

  (n1 n a NUMBER )

  (n1 (past v) "on" n2 "in" n3 "in" NUMBER )

  (n2 n1 "from" NUMBER )

  (n2 "was" a NUMBER )

  (n1 a "around" n2 NUMBER )

  (n1 p NUMBER a "on" n2 )

 )


 (  ("How" "much" "did" (n1 ANY) "spend" (n2 ANY) "?" )

    (n1 "spent" MONETARY n2 ))



 (  ("How" "much" (n1 ANY) "should" (n2 ANY) (v Vsf) (n3 ANY) "?" )

    (n2 "should" v AMOUNT "of" n1 n3)

    (n2 "can" v AMOUNT )

    )
```

```
(  ("How" "much" "in" (n1 ANY) "is" (n2 ANY) "?" )
   (n2 "is" NUMBER n1))

(  ("How" "much" "does" (n1 ANY) "cost"  "?" )
   (n1 "costs" MONETARY) )

(  ("How" "much" "does" (x ANY) (v Vsf) ?)
  (x "in" MONETARY )
  (x (third-person v) MONETARY )
  (x MONETARY )
  (x (past v) MONETARY )
  (x v "on" MONETARY )
  (x "By" MONETARY )
  (x "to" v MONETARY )
  (x "per" MONETARY )
  (x "is" MONETARY )
  (x "would" v MONETARY )
  (x (past v) "in" MONETARY )
  (x "are" (Vpp v) "by" MONETARY )
 )

(  ("How" "much" "does" (n1 ANY) "cost" (n2 ANY) "?" )
   (n2 n1 "costs" MONETARY) )


(  ("How" "do" "you" (v Vsf) (n ANY) "?" )
   (n "is" (past v) CLAUSE))

(  ("How" "is"  (n ANY) (p Vpp) "?" )
   (n "is" p CLAUSE) )

(  ("How" "tall" "is" (n2 ANY) "?" )
   (n2 "is" DISTANCE "high")
   (n2 "is" DISTANCE "tall")
   (n2 "at" DISTANCE )
   (n2 "than" DISTANCE )
   (n2 "via" DISTANCE )
   (n2 "in" DISTANCE )
 )

(  ("How" "did" (n ANY) (v Vsf) "?" )
   ( n (past v) "of" CLAUSE )
   ( n (past v) "by" CLAUSE )
   ( n (past v) CLAUSE )
   ( n v CLAUSE )
   ( n v "for" CLAUSE )
   ( n (past v) "from" CLAUSE )
   ( n "is" (Vpp v) CLAUSE )
   ( n "was" (Vpp v) "with" CLAUSE )
   ( n "with" CLAUSE )
   ( n "was" (Vpp v) CLAUSE )
   ( n "was" (Vpp v) "to" v, "by" CLAUSE )
   )

(  ("How" "long" "did" (n ANY) (v Vsf) "?" )
   ( n (past v) DURATION )
   ( n "was"  DURATION )
   ( n "as" DURATION )
```

```
    ( n "is" DURATION )

    ( n (past v) "after" DURATION )

  )


 (  ("How" "far" "is" (x ANY) "from" (y ANY) "?" )

    ( x "is" DISTANCE "away" "from" y )

    ( x "and" y "are" DISTANCE )

    ( y "and" x "are" DISTANCE )

    (x y DISTANCE )

    (x DISTANCE "away" "of" y )

    (x "about" DISTANCE "away" "of" y )

    (x "about" DISTANCE y )

    (x "for" DISTANCE "along" y )

    (x "is" DISTANCE "from" y )

)


(  ("How" "far" "away" "is" (x ANY) "?" )

    ( x "is" DISTANCE "away" )

    (x "is" DISTANCE )

    (x "is" "around" DISTANCE )

    (x "was" DISTANCE )

    (x "were" DISTANCE )

)


(  ("How" "far" "is" (x ANY) "?" )

    ( x "is" DISTANCE "away" )

    ( x "is" DISTANCE )

    ( x "is" "about" DISTANCE )

    ( x "in" DISTANCE )
    ( x "than" DISTANCE )
  )

 (  ("How" "long" "is" (x ANY) "?" )
    ( x "lasts" DURATION )
    (x  "was" DURATION )
    (x "is" DURATION )
 )

 (("How" "long" "is" (n1 ANY) "in"  (n2 ANY) ?)
  (n1 DURATION n2 )
  (n1 "was" "for" DURATION )
 )

 (("How" "long" "do" (x ANY) (p Vpp) ?)
  (x p DURATION )
  (x p  "in" DURATION )
  (x "for" DURATION )
  (x "are" DURATION )
  (x "was" DURATION )
  (x "for" "over" DURATION )
  (x "can" "be" "for" "over" DURATION )
  (x "will" "be" "up" "to" DURATION )
)

(("How" "big" "is" (n1 ANY) "in"  (n2 ANY) ?)
  (n1 "is" DISTANCE )
  (n1 "with" DISTANCE )
  (n1 DISTANCE "in" n2 )
```

```
   (n1 "is" DISTANCE "in" n2 )
   (n2 "is" DISTANCE n1 )
   (n2 "is" DISTANCE "from" n1 )
   (n1 "bigger" "than" DISTANCE "in" n2 )
   (n1 "was" DISTANCE "in" n2 )
  )

 (  ("How" (a ADJ) "is" (x ANY) "?" )
    ( x "is" CLAUSE a ) )

 (  ("How" "wide" "is" (x ANY) ?)
  (x "is" DISTANCE )
  (x DISTANCE )
  (x "about" DISTANCE )
  (x "was" DISTANCE )
  (x "with" DISTANCE )
  (x "within" DISTANCE )
 )

 (  ("How" "fast" "is" (x ANY) (a Vpp) ?)
  (x "per" CLAUSE )
  (x CLAUSE )
  (x "in" CLAUSE )
  (x "at" CLAUSE )
  (x "is" a "of" CLAUSE )
  (x "during" CLAUSE )
  (x "is" a "from" CLAUSE )
  (x "is" a "by" CLAUSE )
 )
```

```
(  ("How" "often" "does" (n1 ANY) (v Vsf) "at"  (n2 ANY) ?)
 (n2 (past v) CLAUSE )
 (n2 "is" (Vpp v) CLAUSE "between" x )
)


 (  ("How" "cold"  "should"  (x ANY) (v Vsf) ?)
  (x CLAUSE )
  (x "at" CLAUSE )
  (x "was" CLAUSE )
  (x "is" CLAUSE )
  (x "were" CLAUSE )
  (x "in" CLAUSE )
  (x "at" CLAUSE )
  (x "was" (Vpp v) "to" "be" "to" CLAUSE )
  (x "on" CLAUSE )
  (x "with" CLAUSE )
  (x "to" CLAUSE )
  (x "were" CLAUSE )
  (x "below" CLAUSE )
  (x "should" v CLAUSE )
  (x "can" "be" (past v) "at" CLAUSE )
  (x "is" "between" CLAUSE )
  (x "by" "With" CLAUSE )
  (x "can" v "that" CLAUSE )
  (x "should" v "between" CLAUSE )
  (x "should" v "to" CLAUSE )
  (x "must" v "between" CLAUSE )
 )
```

```
(   ("How" "big" "is" (x ANY) ?)
 (x "is" CLAUSE )
 (x "in" CLAUSE )
 (x "with" CLAUSE )
 (x "is" "up" "of" NUMBER )
 (x "are" CLAUSE )
 (x "beyond" CLAUSE )
)

(  ("How" "fast"  "can"  (x ANY) (v Vsf) ?)
 (x v CLAUSE )
 (x "at" CLAUSE )
 (x "will" v "fast" CLAUSE )
 (x "will" v CLAUSE )
  (x (past v) CLAUSE )
)
```