

Offline answer extraction using Dependency Relations

Afstudeerscriptie Informatiekunde

Jori Mur

Scriptiebegeleider en eerste lezer:

Dr. G. Bouma

Tweede lezer:

Dr. M. de Rijke

Informatiekunde

Rijks*universiteit* Groningen

Maart 2004

Contents

1	Introduction	1
2	Background QA research	5
2.1	General background	5
2.2	Historical background	6
2.2.1	Natural Language Interfaces to Databases	6
2.2.2	Text based QA-systems	8
2.3	TREC and CLEF Question Answering Track	9
2.4	A general architecture for a QA system	12
2.5	Scoring methods	14
2.5.1	Recall, Precision and the F-measure	14
2.5.2	Mean reciprocal rank	15
2.5.3	Confidence-weighted Score	15
2.6	Future Research in QA	16
3	Question Answering @ LIT	17
4	Offline extraction strategies for Question Answering	22
4.1	Fact Extraction using Regular Expressions	24
4.2	Beyond Regular Expressions	26
4.2.1	Dependency Parsing	27
4.2.2	Fact Extraction using Dependency Parsing	27
5	Experiments and Results	29
5.1	Experiments	29
5.2	Results	30
6	Discussion	37
7	Conclusion	41
	Bibliography	43

Chapter 1

Introduction

The quantity of information available online increases every day. It is up to the user to find in this huge amount of information a relevant part that answers the question he had in mind. Search engines, like Google and Altavista, so called Information Retrieval (IR) systems, can help the user in his search for an answer. Everyone who uses the internet is familiar with these kinds of tools. The user types in one or more query terms and a list of links to relevant documents is returned. Often one still has to read a large amount of text to find the answer.

Question Answering (QA) systems offer an alternative. Such systems take as input a natural language question (e.g. a question in English), they analyse the question and look for an answer in a large text collection. Then they return the answer immediately, instead of giving a list of links. The questions we are talking about here are usually factoid questions. Answers are typically named-entities. For example ‘Who was the first American in space?’ asks for a person, and ‘Where is the city Bazra located?’ asks for a location.

Today research is also focused on answering more complex types of questions, like definition questions (“Who is Colin Powell?”) and list questions (“Name 22 cities that have a subway system.”). Those questions are harder to answer than simple factoid questions, because information has to be extracted from several documents and combined to form one answer. The answer should not contain any overlap.

So the difficulty of the task depends on the kind of question. But it also depends a great deal on the text collection. If its vocabulary and structure matches the vocabulary and structure of the question, it is very easy to find the answer. But if the answer is formulated in other words and the structure does not match that of the question, then it is much harder to extract the answer.

Roughly you can say there are two different strategies for question answering. One is based on data redundancy. Redundancy based strategies rely on very high volumes of data (in many cases the Web). The assumption here is that the more data one has, the more likely it will be that the answer in the text collection is expressed in the same vocabulary as the question, so that shallow extraction

methods suffice to obtain an answer. An other strategy is the knowledge intensive approach. This approach is based on language understanding and sophisticated lexical and common sense reasoning, making use of various linguistic tools, e.g. named entity recognition¹, lexical relations in WordNet², part-of-speech tagging³, etc. Most systems have implemented both strategies in some way.

The LIT-group (The Language and Inference Technology group) at the University of Amsterdam, where I have been working for the last six months has also implemented a multiple strategy approach in their QA system, Quartz [11]. It turned out that different kind of questions are answered best by different kind of strategies.

One of the strategies the LIT-group has implemented is the *offline answer extraction* method. They call it the table stream. For many types of questions the answer occurs in a fixed pattern. Abbreviations, for example, occur most of the time between parenthesis, see 1(a). Information about persons has often the following structure: name, comma, followed by the position or role of that person, see 1(b).

1

(a) ... the office of the United Nations (UN) ...

(b) ... Ella Pamfilova , leader of the liberal Civil Dignity movement ...

The offline answer extraction method exploits this idea to transform unstructured data to semi-structured data. Information that could be an answer to a possible question is extracted from a text collection using regular expressions and stored in a table.

The tables also contain a lot of noise. Take for example the inhabitants table, where answers can be found for questions about how many inhabitants a country, state, city etc. has. This table contains the following “facts”:

California 10,000

California 4,000

California 7 million

California 32 million

The first three facts are extracted from the following sentences respectively:

(...) there are 10,000 people in California who use marijuana for medical reasons.

¹The automatic identification of names of persons, organisations, locations, etc. in plain text.

²WordNet is a large database of lexical relations (like synonymy) for English words. [14]

³Part-of-speech tagging is the process of assigning a part-of-speech label (noun, verb, pronoun, adverb, etc.) to each of a sequence of words. [14]

A major German corporation (...) , which (...) employs 4,000 people in California.

There are 7 million people in California without health insurance.

Those facts occurred only once. The correct fact, that California has 32 million inhabitants occurred more frequently and because of that this answer will be selected for the question “How many inhabitants has California?”

Another example is given by the manner of-death table, where answers can be found for questions about how someone died. It could contain an entry like “New York, shot”, extracted from the sentence “Here is a picture of New York, shot with my new camera.” We could improve precision by filtering out noise using a named entity tagger. But because a user normally would not ask such questions as how New York died, precision does not have a high priority.

It is much more important to increase recall. A question cannot be answered using this method of offline answer extraction if the answer could not be found in the table at all. One problem using regular expressions will always be that information that differs slightly from the predefined patterns will not be put in the table.

A solution for this problem could be to extract the information using a dependency parser. Dependency parsing is based on relations between words rather than between constituents. Using a dependency parser, useful information that differs from the usual surface patterns will still be found and extracted, because we are looking at grammatical relations between words. For example, in the following two lines, 2 (a) and (b), there is an apposition relation between Grover_Cleveland and President in (a) and a subject relation in (b).

2

(a) Grover_Cleveland, 22nd President ...

(b) Grover_Cleveland (March 18, 1837 - June 24, 1908) was the 22nd (1885 - 1889) and 24th (1893 - 1897) President ...

With dependency parsing the fact that Grover_Cleveland was president will be extracted from both sentences. But for pattern matching it could well be that we only defined a pattern for the first expression, because for the second expression there are too many words between the person name Grover_Cleveland and the role President. So if the fact is expressed in the way it is done in (b), then it will not be found and extracted. Recall will increase using a dependency parser and more questions will be answered correctly. In this thesis I have investigated to what extent the use of dependency relations for the offline answer extraction method could improve the results of question answering compared to using regular expressions for the offline answer extraction method.

The remainder of this thesis is organized as follows. In chapter 2 I will tell something about the background of QA research. In chapter 3 I will describe the

QA-system of the LIT-group of the University of Amsterdam. Then, in chapter 4 I will point out different offline strategies. In chapter 5 I will describe the experimental setting and the results I have obtained. In chapter 6 I will discuss the results. Finally, in chapter 7 I will formulate my conclusions and discuss future work.

Chapter 2

Background QA research

2.1 General background

An information retrieval system (IR-system) allows the user to type in some query terms. It then returns a ranked list of links to relevant documents. On the other hand, a question answering system (QA-system) lets the user type in a natural language question, and as a result the system returns an answer right away.

The task of question answering consists of a number of steps. First, the system analyses the question. What is the focus of the question, what should be the answer type? Then the system will look through the text collection to search for an answer. Possibly more candidate answers are found. The answer with the highest probability is selected from this set of candidate answers and returned to the user.

The first QA-systems were built on top of a database. This means that the data, in which the answer must be searched for, were structured. The user could pose a natural language question, but during the process the question was transformed to a query in some query language. The answer is found with this query. Such a system is more like an interface to a structured database. Furthermore, database oriented QA-systems are usually restricted to closed domains. The database contains, for instance, only data about baseball competitions. For another example see section 2.2.1.

Nowadays interest is particularly focused on open domain question answering. With the rise of the world wide web, people have access to more information than ever before. With this the need for tools that help the user to find the right answers to his questions becomes urgent. Since the Text Retrieval Conference (TREC) has started to evaluate QA-systems in 1999, research is focused on answering factoid questions by means of a large unstructured text collection.

Most QA-systems are built for English. Afterall, large amounts of data are available for this language. For other languages, like Dutch, less resources are available. But since 2003 the Cross Language Evaluation Forum (CLEF) has

also a QA-track. This forum evaluates question answering systems operating on different European languages, like Dutch. So now it is also possible to compare the results of QA-systems for languages other than English.

2.2 Historical background

At present much research is done in the field of question answering, both at national and international level. But this research area is certainly not new. Since the 1960s researchers have worked on QA-systems. Below I will describe in short some of those early systems. You can find a more detailed overview in Simmons (1965) [25]. In the eighties many of the problems in practical QA became apparent and system development efforts were going back. But since the rise of the World Wide Web, the interest for this subject has strongly increased.

2.2.1 Natural Language Interfaces to Databases

One of the first and best known systems was BASEBALL (Green et al. 1963) [6]. It answers questions about dates, locations, teams and scores of baseball games. You can ask only simple questions, without connectives such as ‘and’ or ‘or’ and without superlatives such as ‘most’ and ‘highest’. The data have been stored in a database in the form of lists of attributes with their values. Each month has a list with locations where games were played. Each location has a list with days, each day a list with games, and each game a list of teams with their score. See figure 2.1.

The questions were automatically put in the same format. First, they are partial parsed to distinguish the different phrases. Then using a dictionary, certain phrases are mapped to attribute-value pairs, *Yankees* is mapped to **team = Yankees**, for example. Question words indicate the attribute the user asks for, for example *Where* creates an entry of the form **Place = ?**.

When the list structure for the question is established, the system will search for an answer. In some cases it will be enough to compare the structure of the question with a structure in the database and to fill in the empty spots. In other

```
Month = July
  Place = Boston
    Day = 7
      Game Serial = 96
        Team = Red Sox, Score = 5
        Team = Yankees, Score = 3
```

Figure 2.1: Database of BASEBALL

cases, such as when a user asked for a number or if a quantifier is used (e.g. “Name *every* place where the Yankees played against the Red Sox”), searching for an answer will be harder and perhaps calculations will be necessary.

LUNAR (Woods 1973) [31] is another well-known system. It was developed at Bolt Beranek and Newman (BBN) to make it easier for lunar geologists to obtain, compare and evaluate chemical analysis data of lunar rock and soil. During a demonstration on a lunar science convention in 1971, it was able to answer 90% of the questions that were posed by lunar geologists without preceding instructions.

Questions in natural language are analysed by a parser and are translated in a database query language. To this end a dictionary, containing syntactic and morphological information of words, is consulted. For example question 3 (a) is translated to query 3 (b):

3

- (a) Does sample S10046 contain olivine?
- (b) (TEST (CONTAIN S10046 OLIV))

Depending on the question a query can become fairly complex. LUNAR can handle also successive questions such as:

4

- (a) How many breccias contain olivine?
- (b) What are they?

Although LUNAR showed that automatic question answering can be attractive for users, there were nevertheless many problems. Systems, such as LUNAR and BASEBALL, work only for very limited domains. Moreover, it appeared very cumbersome to transfer these systems to another domain. The complexity of human language was still a lot underestimated in the sixties and the seventies. Although it is nowadays fairly easy to build an interface, which can translate a limited number of questions to queries for a certain database, the question rises if this is useful. It is very difficult to develop a sub-language which is sufficiently expressive, but at the same time remains natural and avoids ambiguity. Moreover the restrictions of the language are not directly clear to the user, as a result of which it will still be hard to learn the language. Another point is that the data are structured and have been stored in a database, whereas researchers nowadays rather use a unstructured text collection.

However, the research that has been done in this area on syntactic and semantic analysis of questions and on pragmatics of the dialogue between people and computers is of value for the systems of today. For an overview of database-oriented QA-systems up to 1990, see Copestake and Sparck Jones (1989) [5].

2.2.2 Text based QA-systems

Text based systems do not assume that the data are structured. Both the question and the data must be analysed to find an answer. One of the first text-based systems was The Oracle system (Phillips 1960) [23].

This system carried out a syntactic analysis both on the question and on the text. The subject, object, verb and possible place and time determinations were identified. But the analysis works only on simple short sentences. As soon as a sentence becomes more complex, if it contained two objects for example, it goes wrong. This was the main restriction of Oracle.

The system first of all assigns each word a part-of-speech tag. The analysis of the text corpus can happen off-line, but the analysis of the question must be carried out within query-time. Next the question is transcribed into declarative form. The words will stand in a different order and auxiliary verbs must be combined with the head verb. For example question 5 (a) is transcribed to 5 (b):

5

- (a) Where did the teacher go?
- (b) The teacher went where?

The system then compares the transposed question with a possible answer sentence looking for identical words and the same word order. For the example above, the answer *to school* would be found if in the corpus the sentence ‘The teacher went to school’ appeared. But with this method many answers will be overlooked, for example when the sentence and the question use different words, which mean the same.

Nowadays researchers have become interested again in this type of system, because it offers a simple and effective approach to answering questions with a very large text corpus. The bigger the corpus, all the bigger the chance that there is an answer in the same vocabulary as the question. In that case, further syntactic or semantic analysis is not necessary.

Protosynthes (Simmons 1965) [25] is another text-based system. It tries to answer questions by means of an encyclopaedia. An index is made for each content word in the text. Words with the same stem are combined. Thus govern, governor, government etc. are all reduced to govern. Each index word gets a list of VAPS (Volume, Article, Paragraph, Sentence) numbers, which indicate where in the encyclopaedia the word occurs.

The question is extended with words with related meaning. To answer the question, all content words of the question are looked up in the index to select the associated VAPS numbers. This way a selection of relevant pieces of text from the encyclopaedia is made in advance.

Then the question and the selected text are parsed using a modification of the dependency-structure determination program developed by D. Hays (1962). [8]

Hays' program was an early form of dependency parsing, using rules to make a dependency graph. Only the candidate answers of which the structure of the dependency graph is similar to those of the question continue to the next step. Protosyntax has also a learning-component, in which the dependency parses are corrected by a human being. This is the only way the system can handle the problems which arise for sentences and questions that are open to several interpretations.

The largest difference of Protosyntax in comparison with other systems in the beginning period is the use of dependency graphs. Consequently this system can find, in a more flexible manner, resemblances in the question and the possible answer. For this reason dependency parsers are still used frequently today.

2.3 TREC and CLEF Question Answering Track

The Text REtrieval Conference

In the eighties much of the problems became clear and the interest for QA-systems decreased for some time. But with the rise of the World Wide Web in the late nineties intensive study into practical question answering was resumed. That is why TREC started evaluating QA-systems in 1999. Since then each year a track for question answering has been organized. The aim of this evaluation is to stimulate study into systems which give a direct answer to a question and not a list of documents. The emphasis lies on systems which function in an open domain.

The task, that systems should carry out was practically the same for TREC-8 (1999) [26] as for TREC-9 (2000) [27]. The corpus consisted of a large collection newspaper articles. The questions were easy in the sense that they asked for simple facts, such as "Who was the first American in space?". For each question it was guaranteed that there was at least one document in the collection, in which the answer appeared. Participants returned a list of five text fragments per question, which should contain the answer to the question plus the document which supported the answer. Answers could be 50 bytes long, or 250 bytes. A number of human judges assessed if an answer to a question was correct or not.

For the 50-byte run the two best systems of TREC-8 were able to find an answer for more than 66% of the questions. Furthermore, when the answer was found at all it was usually ranked first. For every organisation that submitted runs of both lengths, the 250-byte run had a higher mean reciprocal rank than the 50-byte run. The scores of TREC-9 were generally lower. For the 50-byte run of this track the best system found an answer for 66% of the questions, but this system did substantially better than the other systems. The second best system found an answer for only 43% of the questions.

Although the highest score at TREC-9 was the same as at TREC-8, nevertheless some progress had been made, because the track for TREC-9 was a little

harder. The reason was that for TREC-9 questions of users were used and not, like at TREC-8, questions, which were backformulations of the corpus.

There were more differences. Both the text corpus and the question set were larger for TREC-9. See table 2.1. Furthermore in TREC-9 a third judgement was added. If an answer was correct, but the associated document did not support this answer, then it was assessed as ‘unsupported’. Two scores were calculated: One under a strict judgement, where unsupported answers were considered as ‘incorrect’ and one under a less strict judgement, where unsupported answers were considered as ‘correct’.

	TREC-8	TREC-9	TREC-01	TREC-02	TREC-03
# documents	528000	979000	979000	1033000	1033000
# megabytes text	1904	3033	3033	3072	3072
# evaluated questions	198	682	500	500	500

Table 2.1: Data used for the TREC QA-tracks

In 2001, two new tasks, the *list task* and the *context task* were added. For the list task questions like “What are nine novels written by John Updike?” had to be answered. To find an correct answer for such a question, the system must be able to combine information from several documents to form one answer. For this task it was guaranteed that the document collection contained at least the target number of instances. For the context task series of questions were posed. It was assumed that the system could interpret a question at the end of the series by means of the meaning of questions at the beginning in the series. See figure 2.2.

- (a) How many species of spiders are there?
- (b) How many are poisonous to humans?
- (c) What percentage of spider bites in the U.S. are fatal?

Figure 2.2: Example of questions from the context task

Unfortunately to be able to answer questions later in the series did not prove to be correlated to answering questions in the beginning of the series. This task was therefore not repeated in TREC-2002 [29].

The *main task* was mainly the same as in TREC-8 and TREC-9. A difference was, that it was no longer guaranteed that for each question an answer in the corpus occurred. This year participants could only return strings of 50 bytes long, which should contain the answer. Afterwards a disproportionately large part of the questions (135 of the 500) proved to be definition questions. This type of questions is in general hard to answer.

At TREC-2002 there were only the main task and the list task. The corpus that was used this year was the AQUAINT corpus of English News Text. The collection contains approximately 1,033,000 documents or 3 gigabytes of text.

For the main task there were 500 questions, but this year it was decided to leave out definition questions. Because of this the test set became much easier than previous year, since definition questions belong to the most cumbersome types of questions. 46 questions had no answer in the set of documents. It turned out that the systems had a lot of trouble to ascertain whether there was an answer in the document collection or not.

The participants were obliged to give an exact answer to the question, nothing more and nothing less, in contrast to previous years, when one could return a text string that contained the exact answer. A fourth judgement was thus added: ‘not exact’, which implies that the string contains the correct answer and that the associated document supports the answer, but that the string contains more than only the answer or that parts of the answer are missing. Moreover participating systems could only return one answer exactly instead of a ranked list of five answers.

The list task consisted of 25 questions. Generally the scores for this task were very low. It appeared to be very hard to find the correct answers. Moreover not many organisations participated in this task. A large part of the participants indicated however that this was because of lack of time and not of interest.

The main task of TREC-2003 combined three question types, namely factoid questions, list questions and definition questions. Their aim was to encourage additional participation in other subtasks than the main task. Another task was added, the *passage task*. This task had been set up for research groups interested in QA systems that return text segments containing the answer.[30] The corpus that was used this year was the AQUAINT corpus of English News Text, the same corpus as the year before.

For the main task there were 413 factoid questions, 37 list questions and 50 definition questions. 30 factoid questions had no known answer in the document collection. The questions were drawn from search logs. The answer had to be exact.

The list questions asked for instances of a type where each instance was a factoid question answer. E.g. “What Chinese provinces have a McDonald’s restaurant?” The questions were created by the NIST assessors and in no question a target number to retrieve was specified. The response should be an unordered set of instances and an answer string was required to be exact.

The definition questions asked for a definition of a term or biographical data for a person. E.g. “What is pH in chemistry?”, “Who is Vlad the Impaler?” These questions were also drawn from search logs. The response should be an unordered set of strings, where each string represents a different facet of the definition. There was no limit on the length of the strings or on the number of strings.

The passage task uses the same 413 factoid questions as the main task. Systems had to return a single document extract that contained an answer to a factoid question or NIL. The response should be no longer than 250 characters.

The Cross Language Evaluation Forum

CLEF has now also started a track for question answering for European languages. Their aim is to both stimulate study into monolingual question answering on languages other than English and to encourage the development of experimental systems for cross-language question answering.[22]

At CLEF 2003, there were two different tasks, a *monolingual* and a *cross-language task*. The participating languages for the monolingual task were Dutch, Italian and Spanish, for the cross-language task Dutch, French, German, Italian and Spanish. For the monolingual task the corpus, the questions and the answer were all in the same language. In the cross-language task there were non-English questions that searched for answers in an English text collection. The responses were also in English.

For the Dutch monolingual track newspaper articles from 1994 and 1995 were used as the text collection. The total corpus size was 540MB. There were 200 factoid questions. See table 2.2. For 20 questions there was no known answer in the collection.[9]

	CLEF-03
# documents	200000
# megabytes text	540
# evaluated questions	200

Table 2.2: Data used for the CLEF QA-track

Systems had to return three ranked answers for each question. Participants could return either strings of 50 bytes long that contained the answer or the exact answer, but they were not allowed to use both modalities within the same run. The track was divided into two subtracks with separated evaluations and results. Answers were judged to be ‘correct’ if the answer string contained the exact answer, nothing more and nothing less. Answers were judged to be ‘unsupported’ if the answer was correct but it was not supported by the document. A response was considered ‘non-exact’ if the answer was correct and supported by the document, but the answer string contained more than only the answer or that parts of the answer were missing. A response was considered to be ‘incorrect’ if it did not contain the answer. Two scores were calculated: One under a strict judgement, where only exact, supported answers were considered as ‘correct’ and one under a less strict judgement, where unsupported answers were also considered as ‘correct’.

2.4 A general architecture for a QA system

To get a clear picture of how question answering works, I describe in this paragraph a general architecture of a QA-system. See figure 2.3.

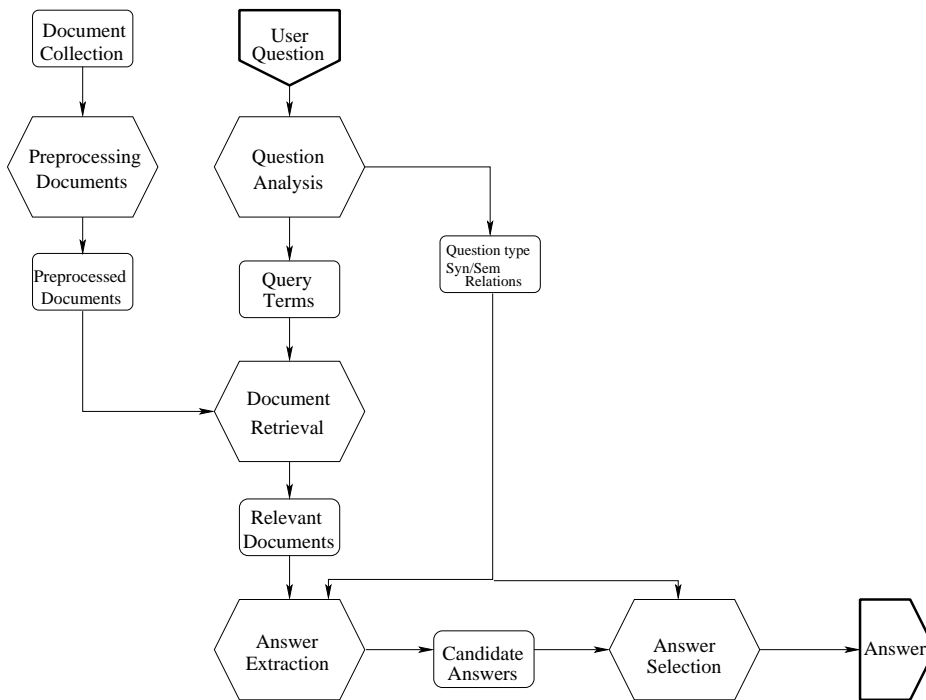


Figure 2.3: A general architecture for a QA system.

First of all the question is analysed. The input is the natural language question. Analysing the question means that the semantic type of the supposed answer is determined. For example a person, a date, a location etc. This generally happens by means of the question word. ‘Who’ ask for a person, ‘when’ for a data and so on. Furthermore the system looks if there are important words in the question that could be used to compare the question to sentences which are likely to contain the answer. Possibly it is determined which syntactic and/or semantic relations there must exist between the candidate answer and other words in the question. These restrictions are used again at the answer extraction and the answer selection component.

If questions need to be answered within a couple of seconds and the document collection in which the answer should be searched for, consists of some gigabytes, then it is important that these documents are prepared beforehand. Most of the systems use conventional index machines, but in principle one could already carry out all kinds of processes on the documents in advance, like for example named entity recognition or dependency parsing.

Then most of the systems use a conventional IR search engine to make a first selection of possible relevant documents. In this manner you only need to examine a small collection of documents in detail. This saves a lot of time and work.

Afterwards the selected documents can be analysed further. This is not neces-

sary if the system has already carried out an exhaustive analysis on the complete corpus during the preparation phase. But generally systems at this stage use a named entity identifier, part-of-speech-tagger and/or chunkparser.

Then the question is compared to the text of the candidate documents. From those documents a set of answer candidates is selected. This is done by looking if the semantic type determined by the question analysis component corresponds to the semantic type of the term which could be the answer. Furthermore it is determined if all remaining restrictions are satisfied. Eventually the answer candidates are ranked and the first ranked answer is returned to the user.

2.5 Scoring methods

There are several evaluation metrics for Information Extraction and Information Retrieval. General metrics are recall, precision and a combined measure, called the F-measure. CLEF used the mean reciprocal rank as did TREC in the beginning. TREC used the confidence-weighted score in 2002 and in 2003 they evaluated the system using precision, recall and the F-measure.

2.5.1 Recall, Precision and the F-measure

Recall is a measure to determine how well a system is able to extract relevant information from a text.

It is defined as follows:

$$Recall = \frac{A_c}{A_{tot}}$$

A_c : number of correct answers returned by the system

A_{tot} : total number of correct answer to be found in the text

In this way it can be calculated for the list task how well a system is able to find all the correct answers for a particular list question. Then A_c will be the number of correct, distinct responses returned by the system and A_{tot} will be the number of known answers in the document collection. If a system returns five responses for a list question while there are ten known answers in the document collection, then recall will be 0.50.

Precision (also known as accuracy) is a measure to determine how well a system is able to extract only relevant information.

It is defined as follows:

$$Precision = \frac{A_c}{A_r}$$

A_c : number of correct answers returned by the system

A_r : total number of answers returned by the system

In this way it can be calculated how well a system is able to find only correct answers. For a list question, for example, A_c will be the number of correct, distinct responses returned by the system and A_r will be the total number of responses returned by the system. If a system returns hundred responses for a list question of which twenty are distinct and correct, then precision will be 0.20.

A system that tries to increase recall, will lower its precision. And it is similar for the other way around. This has led to the use of a combined measure, called the F-measure.

The F-measure is defined as follows:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

When β is one, precision and recall are given equal weight. When β is greater than one, precision is favored, and when β is less than one, recall is favored.[14]

2.5.2 Mean reciprocal rank

Up to TREC-2001 the mean reciprocal rank (MRR) was used as an evaluation metric.[26, 17] Participants could give a ranked list of five answers to a question. Each question received a score equal to the inverse of the rank at which the first correct answer was found. That is called the reciprocal rank.

So if a correct answer is on the first rank, the score will be one over one, thus one. If it occurs on the second rank, then it is one over two, so 0,5 etc. If non of the five responses contained a correct answer, the score was zero. The mean reciprocal rank is the average score over all questions.

CLEF also used this evaluation measure. Except that participants were only allowed to return up to three responses per question.[9]

2.5.3 Confidence-weighted Score

From TREC-2002 onwards systems were limited to one response per question instead of five.[29] Thus the scoring metric had to change. The confidence-weighted score (CWS) was specifically chosen to test a system's ability to recognize when

it has found a correct answer. The questions were ordered from most confident response to least confident response. So the question ranked first was the question for which the system was most confident in its response and the last question was the question for which the system was least confident in its response. A system is rewarded more for a correct answer early in the ranking than for a correct answer later in the ranking.

More formally the confidence-weighted score is defined to be:

$$\frac{1}{Q} \sum_{i=1}^Q i_c/i$$

i_c : number of correct answers in the first i questions

Q : total number of questions

2.6 Future Research in QA

The number of participants to TREC for the question answering track has increased from 20 in 1999 to 33 in 2003. For European languages there is CLEF, since 2003. From this it appears that question answering is still an active research area. The systems improve every year, but there are still many aspects which can be examined.

Still relatively simple questions are used as a test set, but eventually a system also has to be able to answer more difficult types of questions such as ‘why’ and ‘how’ questions. In the future QA-systems should be able to search through documents in several languages and they also have to cope with other media such as pictures or speech. Furthermore research on the different types of users is necessary. Answers for questions of primary school students are not always good answers for the same questions of experts. These are only some examples of the challenges for question answering. But they make clear that there is still much to do.

Chapter 3

Question Answering @ LIT

The LIT group has been engaged in question answering since 2001. This year they have participated for the third time in the QA track of TREC.[11] They also joined in the first QA track of CLEF in 2003.

In the course of the time their system has been further developed and extended. Their first system, Tequesta, was a QA-system that used linguistic analysis to answer questions. [19] This approach was suitable for certain types of questions, but other types of questions seemed to benefit from a more shallow approach. All kinds of different approaches are possible to answer questions. For this reason it was decided to build a multi-stream system. This system consists of a question classifier component, six separate QA-streams, which each in itself forms a QA-system, and an answer selection component, which selects the final answer from the total set of answers of all streams. A practical advantage of this architecture is that you can evaluate each approach separately.

See for an overall overview of the system figure 3.1. Below I give a short description of the six streams: Table lookup, Pattern Matching on the web and on the corpus, Ngram mining on the web and on the corpus and Tequesta.

Table lookup

The table lookup stream uses tables with specialised information. These tables have been filled with information which is extracted from the corpus before the actual process of question answering has started. The question type indicates whether the answer could appear in one of the tables. When that is the case, and in the concerning table an answer is found, a high certainty is granted to this answer

Certain information, that people request, occurs frequently in fixed patterns (such as abbreviations, capitals and names of political leaders). The LIT-group has created manually a number of regular expressions to extract this kind of information. Several tables for different categories were made this way. See table 3.1.

The table with the category ‘Leaders’, for example, contains information of

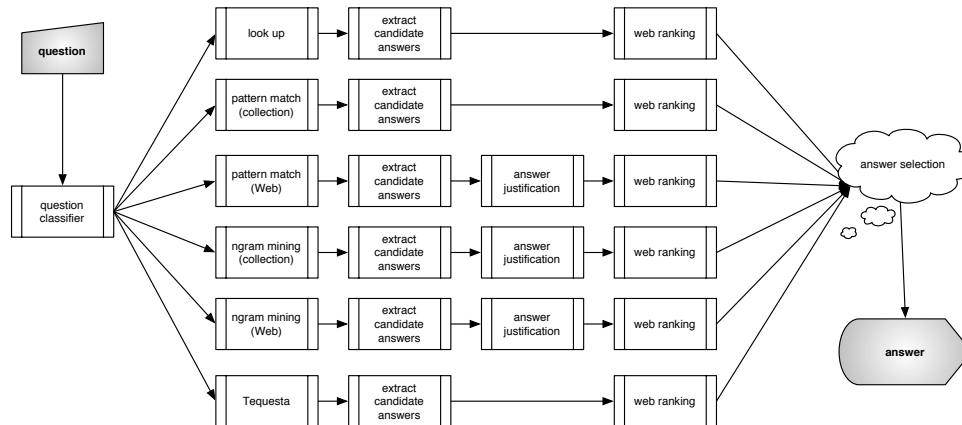


Figure 3.1: Quartz-e System Overview.

political leaders in the following manner: “Dutch Finance minister Gerrit Zalm XIE20000714.0034”. The last number is the documentid. It indicates from which document the information has been obtained.

The table ‘Roles’ is similar to the ‘Leaders’ table, with the exception that it concerns not only leaders, but all kinds of occupations which people can have, like basketball player, biologist or bridesmaid, etc. This stream uses a number of external resources such as WordNet to compose a roleslist. Further explanation about the table categories follows in chapter 4.

When a question is assigned a certain type for which perhaps an answer can be found in one of the tables, the system first identifies the focus words of the question. Then the system searches for a line in the concerning table, in which those same words occur in the same order. If no line is found, it searches again for a line which contains all the focus words, but now in random order. If still no line is found, it will cross out words of the list of focus words. The order in which words are removed from the list is determined by the frequency of the words in the corpus and the part-of-speech tag. Words, which frequently occur

Table 3.1: Facts extracted from the AQUAINT corpus.

Category	# Facts	Category	# Facts
Abbreviations	31737	Birthdates	9156
Capitals	1273	Currencies	231
Dates	9331	Deathdates	1510
Geograpy	70363	Height	15603
Inhabitants	2025	Languages	853
Leaders	18073	Locations	1348
Manners of death	857	Organizations	98758
Roles	396558		

are removed first and superlatives are removed last from the list. This continues until a certain threshold has been reached. When a line is found, the text, which appears in the right column, is considered as an answer and passed on to the answer selection component .

Pattern Matching

Sometimes the structure of an answer to a question can be inferred from the question itself. For example, for the question: “What is the richest country in the world?” its answer will possibly be found according to the pattern: “<Capitalized Words> (, |is) the richest country in the world.” This way the position of the answer in the context is also immediately known. In the example above the answer would be the word or the words with a capital letter.

The LIT-group has implemented two alternatives of this ‘stream’, web pattern matching and collection pattern matching. For the first alternative the World Wide Web is used as a text corpus, for the second alternative the Aquaint corpus, provided by TREC.

The Pattern Matching stream consists of three steps. First of all possible answer patterns are generated by means of the analysis of the question. For simple questions such as “Where lies Mount Olympos?” it is enough to know the type and the focus of the question. These are both provided by the question analysis component. For more complex questions such as ”What date did Thomas Jefferson die?” it is necessary to combine the auxiliary verb and the head verb to transform the question into a declarative sentence (Thomas Jefferson (died|dies) (on|in) <answer>). For this manually created rules are used which are based on part-of-speech tags of the words in the question and a dictionary which contains the different forms of words.

In the next step for each pattern a query of the words in the pattern is created. With those queries the system obtains relevant documents from the corpus. For this retrieval the LIT-group used their own search engine FlexIR. For the Web alternative they used in addition the documents which were retrieved by Google. In the last step the patterns were compared with the retrieved documents and when the text matched, the answer was extracted.

To find a document id of a document in the local corpus (the Aquaint corpus) that supports the answer that was found by the Web alternative, a query was constructed with keywords from the question and the answer and the top-ranking document found by this query was considered to be the supporting document.

Ngram Mining

For this stream the LIT-group has also implemented a web version and a Aquaint version, where respectively the World Wide Web and the Aquaint corpus are used as the text collection. For each question a weighted list of queries is constructed. That is done in a similar manner as the queries in the ‘Pattern Matching stream’ were created. Then the queries are sent to the text collection. For the web

alternative Google is used as the search engine and for the Aquaint alternative FlexIR.

Then the stream looked at word ngrams in the relevant retrieved text snippets. For the web alternative the text snippets provided by Google were used and for the Aquaint alternative the text snippets formed by the 20 bytes around the queryterms in the retrieved document. The ngrams were put in order of relevance. The relevance of each ngram was determined by the weight of the query that had found the text snippet containing the ngram, the NE type of the ngram, the frequency of the ngram in the text snippet, the distance of the ngram to the query words and some more parameters. The ngrams which were on top of the list, were considered as a candidate answer.

Finally, finding a document id of a supporting document was done in the same way as for Pattern Matching.

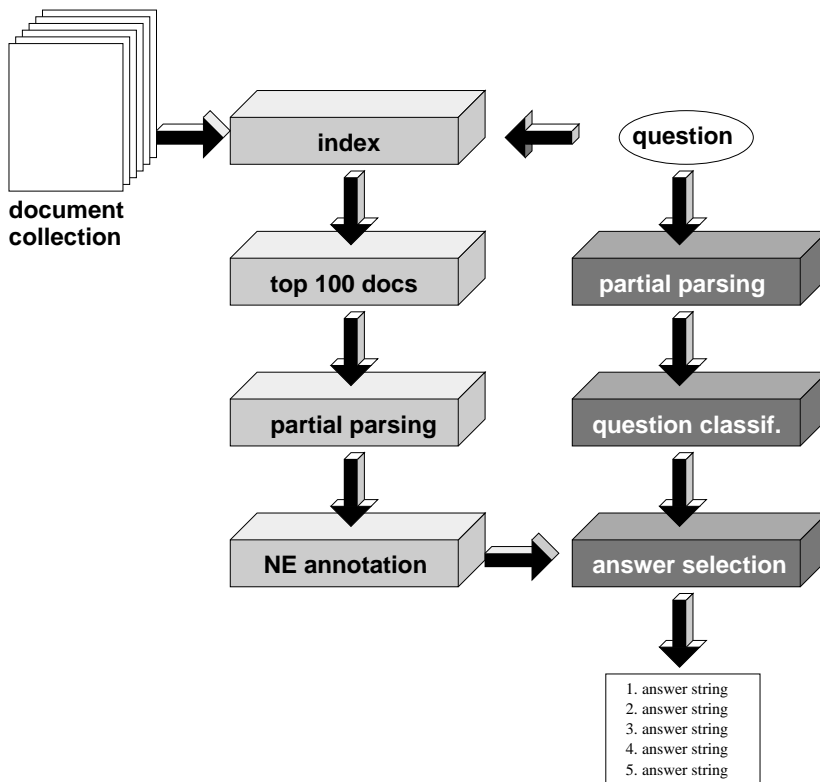


Figure 3.2: Tequesta System Overview.

Tequesta

The last stream, Tequesta, is in fact the original QA-system of the LIT-group. You can see an overall overview of the architecture in figure 3.2.

Tequesta was built on top of a search engine. First of all an index is made for the text corpus. The LIT-group transposes the question to a query, which is send to their own search engine, FlexIR. FlexIR produces a list of relevant

documents. Only the first hundred were examined more closely. For each of these documents a fragment was extracted, which was further analysed. It was ensured that each fragment started with a sentence and finished with a sentence, so that the fragment could be parsed. For parsing Dekang Lin's dependency parser MINIPAR was used [15].

Depending on the question type a named entity recognizer was used to identify phrases of the same semantic type as the expected answer. If some user asks for a numerical expression, for example he or she poses the question "How high is the Empire State Building?", then only phrases of that type (numeric) were annotated.

The questions were also parsed by MINIPAR. The LIT group had the questions subdivided in 33 categories, among other things date, location, name and number. To find out what was asked for, pattern matching was applied again. When a question corresponded to several patterns, it was also subdivided in several categories.

Finally, by means of the parsed and annotated document fragments and the parsed and categorized questions, an answer is selected. WordNet was used to find words with the same meaning.

These were the six streams of the QA-system of the LIT-group. A number of components of the system is used by all streams, among which the following:

Question Classifier

A question is analysed first to find out its type (e.g. date-of-birth), to determine what the type of the expected answer should be (e.g. date) and to determine the focus. The system distinguishes 33 question types. The question analysis has been based on manual formed surface and part-of-speech patterns. Just like for Tequesta the LIT-group uses here hierarchic relations in WordNet to identify semantic classes of question focus words.

Answer Selection

The six streams all produce candidate answers with confidence scores. Each score is adjusted with a factor which is based on how well the stream has performed in the past on questions of the same type. In this collection of candidate answers the system searches for identical or similar answers. These are merged and their scores are added up. Eventually the candidate with the highest score is given as the final answer.

Chapter 4

Offline extraction strategies for Question Answering

QA systems always use large text corpora like the World Wide Web. After all, the more information is available, the more questions can be answered. But as the corpora increase every time, it becomes more cumbersome to find an answer, because there is also much unusable information in a corpus, especially on the World Wide Web [7]. The advantage of offline answer extraction is that useful information can be extracted and classified into different categories before the actual QA process. This way the search for an answer becomes more targeted.

Because extraction takes place offline, it is possible to pass through the whole corpus. No IR techniques are necessary. This is a second advantage, because although standard IR techniques work quite well for regular ad hoc retrieval, they are not optimal for question answering. [18]

Zipf's law appears to apply to question distribution.[17] That means that a few question types account for a large portion of all question instances. Information that could be an answer to those types of questions should thus be extracted. Table 3.1 shows which categories of useful information the LIT-group found.

In the following paragraphs I will explain the different categories for which a table was created. All the entries in all the tables are followed by a documentid and a frequency number. The documentid indicates from which document the current information has been obtained and the frequency number indicates how many times it occurred.

The table for abbreviations contains abbreviations and their expansions in the following way:

AA Airport Authority

First comes the abbreviation itself and then the expansion. In the table 'Capitals' you can find entries of the following form: Country, Capital of that country. For instance:

Afghanistan Kabul

The idea for the tables 'Inhabitants', 'Languages' and 'Currencies' is the same.

But instead of the capital there are the number of inhabitants, the spoken language and the currency respectively:

Argentina 37 million
 Angola Portuguese
 Andorra euro

There are three tables for dates, ‘Birthdates’, ‘Deathdates’ and ‘Dates’. In the table for ‘Birthdates’ is listed who was born when. First comes the person name, then the year in which that person was born.

Adolf Hitler 1889

If the day and month are also known, then that information will be stored in the table too. The table ‘Deathdates’ is almost the same, except that it contains information about who died when. The table ‘Dates’ is a bit different from the other two date tables. Each entry consists of a date and an event that happened on that day. E.g.:

April 12, 1861 the American Civil War began

The event could also be a birth or a death.

The table ‘Heights’ contains information about heights of persons, buildings, mountains etc.:

Albert Memorial 200-foot-high
 Angel Falls 978 meters
 Anthony Clement 6-foot-7
 Avalon Towers 16-story

Everything that has a height could be stored in this table.

The tables ‘Location’ and ‘Geography’ both contain information about where places are located. In each table an entry consists of a location A that is located in location B. But in the ‘Location’ table it is also defined what kind of location location A is. For example:

Alexandria second-biggest city Egypt

In the table ‘Geography’ you will only find:

Alexandria Egypt

Because it turned out that people very often asked about how someone died, the LIT-group also created a manner-of-death table. In this table names of persons are stored and after the names the causes of death, like this:

Anna Karenina committed suicide

Linda McCartney breast cancer

In the table ‘Organisations’ information is stored about organisations, institutions, companies etc. This can be all kinds of information. For instance, what a company is:

American Express credit card company

What it does or did:

American Federation of Teachers which represents a million teachers
 ABC has signed the agreement

So not only entities are stored in this table, but also whole parts of sentences.

The same holds for the table ‘Roles’. This table contains information about persons and the ‘roles’ that they have. These roles are professions, occupations, positions etc. But also things that people did or what they are famous for are stored in this table. Examples are:

A. Claude Monet founder of Impressionism
 Andy Warhol discovered rock band the Velvet Underground
 Angelina Jolie first-time nominee who played a mental patient

The ‘Leaders’ table also contains information about persons, but for this table the information is restricted to leader roles, like president, minister, chancellor etc. First is listed the country, company or organisation to which the role is related, then the leader role and after that the person name, like this:

Afghanistan President Burhannudin Rabbani
 Coca-Cola president J. Farrell

In this table only entities are stored, not whole part of sentences like in the Organisation and Roles table.

For this thesis I have only looked at the roles and the leaders table. Information about people occurs a lot and usually in less fixed patterns than abbreviations, for instance. Moreover for person questions a extensive answer is often correct. The question ‘Who was Columbus?’ could be answered by sentences telling when he lived and where he came from and that he was the one who discovered America. A short biography so to say. For this type of questions it is important to collect as much information as possible. On the other hand for questions concerning capitals, currencies, inhabitants etc. only one answer is correct. Higher recall is less important for these tables than for the roles and leaders table. Therefore I have chosen for these last two categories.

My aim was to compare offline answer extraction strategies, based on surface patterns vs. syntactic patterns. Clearly, the performance of any answer extraction strategy depends on the set of the covered linguistic phenomena, so I tried to keep the two strategies parallel in terms of the covered phenomena.

4.1 Fact Extraction using Regular Expressions

The LIT-group uses regular expressions for offline answer extraction.[11] It works quite well, because for many types of questions the answer occurs in a fixed pattern. A number of regular expressions had to be created, able to extract information about all categories. This can be done manually or with machine learning techniques.

The LIT-group first tagged the whole text collection with a Named Entity tagger based on TnT [2] and trained on CoNLL data [24]. Although the tagging itself was not perfect and produced many mistakes, it was useful for restricting the patterns.

To extract information about roles the LIT-group used the following manually created set of patterns:

PAT1 = (...) ROLE , PERSON ,
The British actress, Emma Thompson,
 PAT2 = (...) \b([a-z]+\west|first|last|most)\b (...) , PERSON
The first man to set foot on the moon, Armstrong
 PAT3 = PERSON , (...) ROLE (...) (;|.|.\ <)
Audrey Hepburn, goodwill ambassador for UNICEF.
 PAT4 = PERSON , (...) /\b([a-z]+\west|first|last|most)\b/ (...) (;|.|.\ <)
Brown, Democrats' first black chairman.
 PAT5 = PERSON , (...) /\b(INTERESTING VERB)\b/ (...) (;|.|.\ <)
Christopher Columbus, who discovered America
 PAT6 = (...) ROLE PERSON , (...)
The British actress Emma Thompson,
 PAT7 = ROLE (...) PERSON (...)
The captain of the Titanic Edward John Smith

And to extract information about leaders the LIT-group created manually the following two patterns:

PAT1 = PERSON, (...) LEADERSROLE COUNTRY ,
Tony Blair, the prime minister of England,
 PAT2 = COUNTRY (...) LEADERSROLE , PERSON
The british foreign secretary , Jack Straw

To identify roles in the text collection they created a list of possible roles, extracting all hyponyms of the word 'person' (15703 entries) from the WordNet Lexical database. An interesting verb (PAT5) is an entry from the manually created list of 'interesting' verbs (see table 4.1). For the leadersroles the LIT-group made a small hand-built list of phrases identifying leadership (see table 4.2).

Then, for each category the system made a single pass through the text collection to identify useful information using the patterns. Upon finding something the system extracted it and stored it in a table.[13] The format of the tables was plain text, where every line contained a few columns of unstructured text.[1]

Table 4.1: Interesting verbs.

assassinate	discovered	manufactures	signed
assassinated	fired	owned	stab
coined	founded	owns	stabbed
compose	held	paint	starred
composed	hold	painted	start
create	holds	receive	started
created	invent	received	succeed
defeat	invented	release	succeeded
defeated	kill	released	win
direct	killed	reports	won
directed	lived	shot	write
discover	manufactured	sign	wrote

Table 4.2: Leaders.

chancellor	co-leader	secretary	princess
ex-leader	head of state	undersecretary	attorney
ex-minister	king	minister	dictator
ex-secretary	queen	premier	lord
ex-premier	leader	president	
ex-president	minister	prince	

Problem

As I have mentioned before on page 3, the main problem of using regular expressions for offline answer extraction is the recall problem. Information that is slightly different from the predefined patterns will not be stored in the tables. For example, from the string “the gun Jack Ruby used to kill Lee Harvey Oswald” the fact that Jack Ruby killed Lee Harvey Oswald was not extracted by the table stream of the LIT group, because “Jack Ruby used to kill Lee Harvey Oswald” is an unusual pattern.

4.2 Beyond Regular Expressions

A possible solution for the recall problem might be to use extracting methods that are linguistically more sophisticated than simple pattern matching. Specifically, a dependency parser might recognize more facts, because the structure of the described fact does not have to be so strict as with pattern matching.

For example, if the following sentence would occur in the text collection: “London, which has one of the busiest airports in the world, lies on the bank of the river Thames.” we would not be able to extract the location of London if we used regular expressions, because there are too many terms between London and

its location, the bank of the river Thames. But using a good dependency parser, one should be able to extract this fact.

4.2.1 Dependency Parsing

Dependency parsing is based on relations between words rather than on constituents. Each word in a sentence points to a single parent. This parent is called the head of the relation. The word that modifies the parent is called the dependent or the modifier of the relation. A word in a sentence may have several modifiers, but each word may modify at most one word. The root of the dependency tree does not modify any word and is called the head of the sentence.[16]

See figure 4.1 for an example. The head of the sentence is the verb ‘ate’. There are six dependency relations, depicted by six arcs from the heads to the modifiers. Each arc is labeled with a name that indicates the grammatical relation between the words. An arc between a noun and a determiner always gets the label ‘Det’. ‘Subj’ en ‘Obj’ indicate respectively which noun is the subject and which the object of the verb. And a relation between a noun and an adjective gets the label ‘Mod’, that stands for modifier.

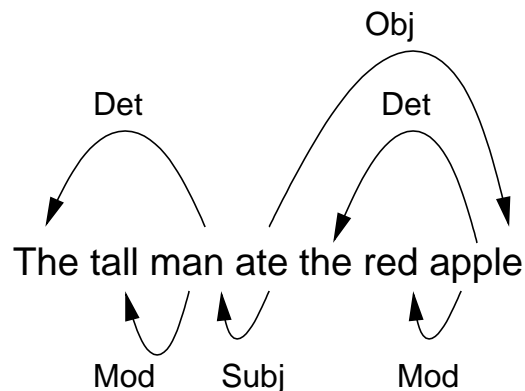


Figure 4.1: An example of a dependency parse.

4.2.2 Fact Extraction using Dependency Parsing

The text collection was parsed with Minipar [16], a broad coverage dependency parser for the English language. I chose this parser because of the rich output it produces, e.g. explicitly distinguishing between subjects, objects, adjuncts etc., unlike state-of-the-art statistical phrase structure parsers (Collins [4], Charniak [3]). Minipar is reported to achieve 89% precision and 79% recall with respect to dependency relations when evaluated in the SUSANNE corpus. I also found that it performed pretty well on the newspaper and newswire text of my text collection

and that it was fairly robust to fragmented and not well-formed sentences frequent in this domain.

The following relations were used to extract information about persons:

- Apposition $person \xrightarrow{\text{appo}} role$;
a major developer, Joseph Beard
- Apposition $person \xleftarrow{\text{appo}} role$;
Jerry Lewis, a Republican congressman
- Clause $person \xrightarrow{\text{subj}} verb \xleftarrow{\text{obj}}$;
Bell invented the telephone
- Person $person \xrightarrow{\text{person}} role$;
Vice President Al Gore
- Noun-noun modifier $person \xleftarrow{\text{nn}} role$;
businessman Bill Shockley
- Subject $person \xrightarrow{\text{subj}}$ *role*;
Alvarado was chancellor from 1983 to 1984
- Conjunction, (frequently occurring parse error) $person \xleftarrow{\text{conj}} role$;
Fu Wanzhong, director of the Provincial Department of Foreign Trade

Minipar only labels a relation between a person and a role with ‘apposition’ if there is a comma between the two entities. If there is no comma, then the label will be ‘person’ if the role is the head of the relation and ‘noun-noun modifier’ if the person name is the head of the relation.

For all patterns, *role* is one of the nouns in the list of roles that was also used for the regular expressions, extended by the leader roles in the list of leaders. So I combined the tables for leader roles and the other roles. A *verb* is one of the words in the list of “interesting” verbs, the same list that was used for the regular expressions. To prevent that pronouns would appear in the table as person names (*I* or *He*), it was also required that *person* was a proper noun.

When a pattern matched a part of a sentence parse, the pair (person, info-bit) was extracted, where *info-bit* is a sequence of all words below *role* or *verb* in the dependency graph (i.e. all dependents along with their dependents etc.), except for *person*. For example, for the sentence “Jane Goodall, an expert on chimps, says that evidence for sophisticated mental performances by apes has become ever more convincing.” the extracted information was *Jane Goodall, an expert on chimps*.

Chapter 5

Experiments and Results

5.1 Experiments

The aim is to examine if the syntactic patterns extraction method, using dependency relations outperforms the surface patterns extraction method that uses regular expressions. The patterns for the surface pattern module had already been created. I only had to define which dependency relations were useful. I did this by parsing a small corpus and by looking which relations there existed between roles and person names. I wanted at least all the information, which appeared in the table of the surface pattern module also in my syntactic pattern table. When the two scripts were finished I created tables for the whole corpus.

I could not simply compare the size of the tables with each other, since a part will also be noise and I do not know which part that is. If a table is larger it does not have to mean, that it contains more useful information, it could contain much more noise just as well. We could evaluate it manually, but it would also be very informative to measure the performance of this block in context, that is, working as a sub-part of a larger system. Therefore I decided to evaluate the two extraction modules within the context of Quartz, the QA system of the LIT-group, assuming that the best extraction method will answer the most questions correctly.

I took the person questions and text collections from TREC-8 up to TREC-2003. In the first three years (from TREC-8 until TREC-2001) a different corpus was used as in the last two years. See section 2.3 and table 2.1. I have stucked to this partition for my experiments and I have made two separate tables for the two texts collecties and questions sets per module.

The questions set of TREC also contained questions for which there was no answer to retrieve in the text collection. I have left these questions aside. I was interested in extracting information, not in determining if there was an answer in the corpus at all. The questions have been classified as person questions by the question analyzer component of Quartz. Afterwards I manually removed the

incorrectly classified questions. Eventually I kept 239 questions concerning the old corpus and 97 questions concerning the Aquaint corpus.

The error analysis at this stage showed that many of the incorrect answers were due to the table lookup process (see page 18) rather than the information extraction method itself: correct answers were stored in the tables, but the lookup mechanism failed to find them or returned other, non-relevant pieces of information. But I wanted to evaluate the two extraction methods, not the lookup mechanism. Another problem was that it took too much time to parse the whole corpus. So due to lack of time and an ill-performing lookup mechanism I decided to perform an other experiment.

I reduced the sizes of the collection to only those documents, that actually contained an answer, to simplify automatic lookup. For each TREC question, NIST provides a list of documents that are known to contain an answer to this question. I have put together the document lists for the 239 old questions and the document lists for the 97 aquaint questions, which left me with much smaller collections (16.4 and 3.2 megabytes respectively). Then I ran the two extraction modules on these small collections and evaluated the performance of the QA system on the resulting tables.

This discription of the experiment could create the impression that there is too much emphasis on recall and that the system with the highest recall will be favoured. It is true that in this way, a module which retrieves a lot is likely to find an answer and the chance to find a wrong answer is relatively small, because the documents have been selected on relevance. But it should also work this way if there was much noise in the table and the lookup mechanism worked perfectly. Because the lookup mechanism does not work perfectly, we do the experiment in a kind of ‘artificial’ way by selecting all the relevant documents.

5.2 Results

To give an idea how many facts we extracted for each extraction run we show the numbers in table 5.1. These numbers do not indicate which module works better, because it is not known which part of each table is noise. However, it is clear that the syntactic module has extracted more than the surface pattern module. But in theory it is still possible that the surface pattern module has a higher recall. For this reason we now look at the results of the question answering runs.

Corpus	Syntactic patterns	Surface patterns
old	9830	6028
aquaint	1959	1336

Table 5.2 gives the results of the runs for both the syntactic pattern extraction module and the surface pattern module on the older TREC collection with 239 questions. The results of the runs on the Aquaint collection with 97 questions are shown in table 5.3. Because the lookup mechanism does not work perfectly, it is possible that a correct answer appears in the table but is not selected as the first answer. For this reason we looked at the number of times at least one correct answer occurred in the top three. But for most of the questions the answer occurred at the first rank. For the Aquaint-corpus it even turned out that each correct answer was at the first rank.

It is striking that the scores are rather low in comparison with the scores of the QA-systems that participate in TREC. The best systems that participate in TREC have in general more than half of the questions correct. The reason for this could be that person questions are harder to answer than other types of questions. There are for example definition questions, such as ‘Who was Claude Monet?’ Those kind of questions are difficult. There are also questions like ‘Who was chosen to be the first black chairman of the military joint Chiefs of Staff?’ In the roles table you can find that Colin Powell was chairman of the military joint Chiefs of Staff, but not that he was the first black one. Another problem is for example when a question asks for someone’s spouse: ‘Who is Tom Cruise married to?’ You can find in the table who Tom Cruise’s wife is, but the system was not able to link ‘married’ with ‘wife’.

These kinds of problems one does not encounter for many other types of questions. The question ‘What is the currency of the Netherlands?’ is much easier, for example. For this reason it was to be expected that the score for only person questions ended up lower than for all types of questions.

Another reason could be, that we have imposed on ourselves the requirement to keep the syntactic pattern module parallel to the surface pattern module. If we would ignore this requirement, then we might improve the system still further, as a result of which the score would also be higher.

The runs on the old corpus score better than the runs on the Aquaint corpus. That was to be expected, because the questions from the first tracks of TREC are easier than the questions from the last track. For example, the questions from TREC-8 were backformulations of the corpus, developed specifically for that track by either the participants or the NIST-accessors.

Rank	Syntactic patterns	Surface patterns
1-3	79 (33,1%)	59 (24,7%)
1	73 (30,5%)	54 (22,6%)

In theory the relatively simple facts, which were extracted by the surface pattern module should at least all be extracted by the syntactic pattern module.

Rank	Syntactic patterns	Surface patterns
1-3	14 (14,4%)	6 (6,2%)
1	14 (14,4%)	6 (6,2%)

This last module should extract even more facts, namely those of which the structure deviates from the patterns predefined in the surface pattern module. To investigate how many questions answered by the surface pattern module were also answered by the syntactic pattern module, we counted the number of questions that were correctly and incorrectly answered by one module vs. the questions that were correctly and incorrectly answered by the other module. See table 5.4 and 5.5.

<i>Surface patterns</i>	<i>Syntactic patterns</i>	
	correct	incorrect
correct	47	12
incorrect	32	148

Table 5.4: Performance analysis old collection.

<i>Surface patterns</i>	<i>Syntactic patterns</i>	
	correct	incorrect
correct	4	2
incorrect	10	81

Table 5.5: Performance analysis Aquaint collection.

As the two tables indicate, not all questions answered by the surface pattern module were also answered by the syntactic pattern module. In total there were 14 such questions, 12 for the old corpus and 2 for the Aquaint corpus. So, I took a closer look at the questions for which the two modules performed differently. Table 5.6 gives a breakdown of errors that caused an incorrect answer by the syntactic pattern module for questions that were answered correctly by the surface pattern module.

There were three types of errors. First of all parse errors. Information which was extracted by the surface pattern method, was not extracted by the syntactic pattern method, because the information was not parsed well. For example, the following sentence was parsed incorrectly: “Voyager_2 will glance at Neptune, said Voyager project manager Norm_Haynes.” See figure 5.1 on page 33. For the sake of clarity I only show the arcs of that part of the sentence that matters for this example. There is an arc between ‘Norm Haynes’ and ‘manager’, so the

Parse error	6
Lookup error	4
Surface better than Syntactic	2

fact that Norm Haynes was a manager was extracted. It was even extracted that he was a ‘project manager’, because ‘project’ is a dependent of ‘manager’. But it is not know that Norm Haynes was the manager of the ‘Voyager project’, because instead of making ‘Voyager’ dependent of ‘project’, which would have been correct, Minipar decided that it was a dependent of the term ‘Norm_Haynes’.

Question 102¹: “Who is the Voyager project manager?” could thus not be answered. However, the right information was extracted by the surface pattern module, and for this reason the question was answered correctly by this module.

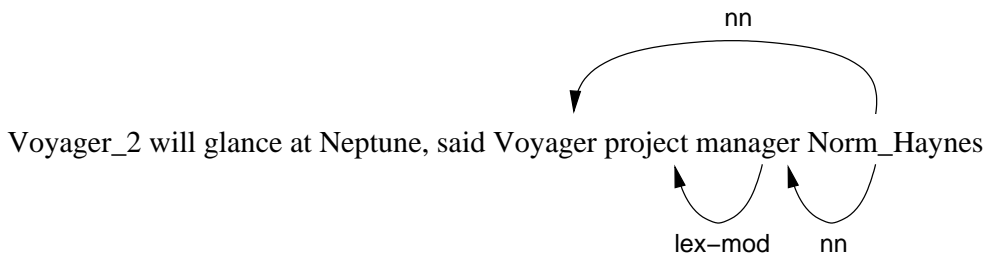


Figure 5.1: Wrongly parsed sentence.

Another type of error was the lookup error. Here the information was extracted and put in the table, but because the lookup did not work perfectly, the correct answer was not selected. For example for question 323: “Who is the richest woman in the world?” the syntactic pattern module returned ‘Queen’ as the answer. However, the correct answer ‘Queen Elizabeth II’ appeared more often. See table 5.7. The lookup mechanism was not able to select the answer with the highest frequency.

And finally there was an error that was more cumbersome to define. In these cases it concerned a formulation of a fact where the surface pattern method appeared more robust by means of wildcards. By way of illustration: from the sentence “(. . .) aviator Charles Lindbergh married Anne Spencer Morrow (. . .).” the syntactic pattern method extracted

‘Charles Lindbergh aviator’

nothing more, because ‘aviator’ has no dependents. However, the surface pattern method extracted

¹This is the numbering of TREC

‘Anne Spencer Morrow aviator Charles Lindbergh married’

It finds the person name ‘Anne Spencer Morrow’ and searches for a role left of this name . Then it finds ‘aviator’. Actually this is not even correct, because the role ‘aviator’ belongs to Charles Lindbergh. It extracts the role and the words between the role and the person name. This way the surface pattern module was able to answer question 646 “Who was Charles Lindbergh’s wife?”, but the syntactic pattern module was not.

Errors which are caused by the lookup mechanism indicate nothing concerning the quality of the extraction. To prevent these errors the lookup mechanism should be improved, not the extraction module. After all, the answer could be found in the table. On the other hand, for both methods the same lookup mechanism is used and there were no lookup errors for the surface pattern method. Possibly higher recall caused problems by making the lookup process harder.

So, eight out of twelve errors, caused by parse errors and by the wildcard functioning of the surface pattern module, are significant errors. But discussion is possible for the four lookup errors.

We also look at the inverse. If the syntactic pattern module is able to extract more information than the surface pattern module, more questions should be answered correctly. Table 5.8 gives a breakdown of errors that caused an incorrect answer from the surface pattern module for questions that were answered correctly by the syntactic pattern module.

For the surface pattern module there were also three types of errors. It appeared that some patterns were not yet added to the script. Such as the following pattern:

PERSON /\b(INTERESTING VERB)\b)/ (...) (;|,|\ <)

We have, however, a pattern which is almost the same, see page 25, PAT5, but that requires a comma between PERSON and INTERESTING VERB. The missing pattern should have been added, because it is more common than the pattern with the comma. For example, the fact that King John signed the Magna

Table 5.7: Entries in the syntactic pattern table about Queen Elizabeth II

Queen	The the richest woman in the world with investments alone estimated at Pounds 50m
Queen Elizabeth II	Britain ’s is worth an estimated \$ 10.9 billion compared with \$ 8.7 billion last year Fortune reported
Queen Elizabeth II	Britain ’s the richest woman with \$ 10.7 billion
Queen Elizabeth II	Ranked fourth after the Mars family the world ’s richest woman Britain ’s is worth an estimated \$ 10.9 billion compared with \$ 8.7 billion last year Fortune reported
Queen Elizabeth II	believed to be the world richest woman

Pattern not added	16
NE error	10
Non local	6

Carta could not be extracted from the sentence “One in five of the teachers surveyed did not know King John signed the Magna Carta (...)”, although it is formulated in a very simple way.

The reason why it was not added in the first place, was because such a pattern gives also much noise in the table. With dependency parsing that is not an issue because it is possible to define that PERSON should be the subject of VERB. So in this case the syntactic pattern module has a clear advantage.

A second type of error was caused by the NE tagger. If an entity gets an incorrect tag, it is not recognised by the appropriate pattern. Abraham Lincoln always got the tag <LOCATION>, for instance. Therefore, there was never a sentence with his name which matched one of the predefined patterns, because it was not recognised as a person name. Questions concerning Abraham Lincoln could thus not be answered.

This was not a problem for the syntactic pattern module, because I only used the NE tagger to cluster names and surnames of persons. I removed all the tags before I started parsing.

And finally information could be found by the syntactic pattern module which could not be found by the surface pattern module, because it was formulated in such a way that there was no pattern for it. For example, the surface pattern module did not extract that Lee Harvey Oswald killed President John F. Kennedy from the sentence “ (...) when Lee Harvey Oswald allegedly shot and killed President John F. Kennedy (...)”, because none of the patterns matched. There is another interested verb between ‘Lee Harvey Oswald’ and ‘killed’, namely ‘shot’ and furthermore, there is a stop word, ‘and’. So question 1274 “Who killed John F. Kennedy?” could not be answered. But when this sentence is parsed, it is very clear that Lee Harvey Oswald killed John F. Kennedy.

In what case is decided that an error occurred because the pattern had not yet been added and in what case because the information was too hard to extract with patterns? Well, when it turned out that a pattern was less complicated than an existing pattern, for example in the case of the pattern mentioned above compared to PAT5 on page 25, the new pattern should have been added. But if the pattern is very complicated, such as the pattern of the sentence about Lee Harvey Oswald, then the error was classified as a non local error. Non local means that the information that is useful for our tables is too far apart in the sentence or other significant words are in between.

So, half of the questions would be answered correctly, if we just had added

the patterns. On the other hand, this makes clear that the syntactic pattern module needs less patterns than the surface pattern module, to extract the same information. Thus, for those questions discussion is also possible. In the next chapter we will come back to that.

Chapter 6

Discussion

I wanted to keep the two extracting strategies, the surface pattern method and the syntactic pattern method, parallel in terms of the covered linguistic phenomena. However, I did not entirely managed to do this.

For instance, the syntactic pattern method had a relation ‘subject’ for sentences with a linking verb between the person name and the role, such as “Alvarado was chancellor’. For this example Minipar finds a ‘subject’ relation between the person name ‘Alvarado’ and the role ‘chancellor’. However, there was no surface pattern defined, analogous to this syntactic pattern. But it became clear from the erroranalysis, that if I had added a surface pattern ‘PERSON was|is ROLE’ then this method would have answered only one question more. The number of questions that were answered correctly by the syntactic pattern method and not by the surface pattern method would have been decreased with only one.

From the erroranalysis (see table 5.8) became also clear that most questions could have been answered correctly if the surface pattern for the sentence in which the answer occurred had simply been added. The dependency relations, which I defined in my syntactic pattern script, were revealed by parsing the information, which was extracted by the surface patterns. It turned out that there did not exist a one to one relation between the syntactic patterns and the surface patterns. The dependency relations which I found after parsing extracted more information than the surface patterns, of which they had been derived.

By way of illustration: In the surface pattern script the following pattern was defined (PAT5 on page 25):

PERSON , (...) /\b(INTERESTING VERB)\b/ (...) (;|,|\ <)

The corresponding syntactic pattern proved to be:

Clause *person* $\xrightarrow{\text{subj}}$ *verb* $\xleftarrow{\text{obj}}$

(last pattern on page 28) But with this syntactic pattern more information was extracted for which no surface pattern existed. To keep the two methods parallel,

I should have done it the other way around and derive surface patterns from the extracted parsed information. On the other hand, this shows a clear advantage of syntactic patterns. There are several surface patterns needed, where only one syntactic pattern is sufficient.

A another advantage of the syntactic pattern method is that you get more relevant information in the table, because everything what is dependent on the role is extracted. The following information is stored in the table of the syntactic pattern method, for example

Bob Brier a professor of philosophy Egyptology at the C.W. Post campus of Long Island University in New York

From the same sentence the surface pattern method extracted only this:

Bob Brier professor of philosophy

The question “Name a professor from the C.W. Post campus of Long Island University” can only be answered by the syntactic pattern method, not by the surface pattern method.

For both corpora the syntactic pattern method gives better results than the surface pattern method. If we consider the answer correct if it appears in the top three, then the syntactic pattern method answers almost a third of the questions for the old corpus correctly. On the other hand only a quarter of the questions is answered correctly with the surface patterns. For the Aquaint corpus the differences are smaller, but still the syntactic pattern method gives better results. (See table 5.2 and table 5.3.)

But what we really want to know is if this difference in performance of both modules is significant or if it is just mere coincidence. If you have two systems that perform just as well, it is possible, by chance, that for one run the first system gets better results than the second. To be sure that this is not the case with our two modules and that the syntactic pattern module really performs better than the surface pattern module, we have to do a significance test.

We assume that there is no difference in the performance of both the modules. This will be our null hypothesis. Then we calculate the chance that we get the results that we got. If this chance is below a certain critical significance level α we can say that our results are significant at level α . And then we reject the null hypothesis.

I looked at the results and erroranalysis of the two modules for the questions of the old corpus (see table 5.4, 5.6 and 5.8), because for the Aquaint corpus there were too few questions to do a significance test (see table 5.5).

There are twelve questions for which the syntactic pattern module did not return a correct answer, but the surface pattern did. Initially, we do not take the

lookup errors in account, because the correct answer had been put in the table properly. Therefore the extraction had succeeded. Out of the twelve questions, eight are left. There are 32 questions for which the surface pattern module did not return a correct answer, but the syntactic pattern did. Sixteen questions could be answered by the surface pattern method if we had expanded it with more patterns, including the question that would have been answered if there was a pattern defined analogous to the 'subject' relation in the syntactic pattern script. So we also do not take these errors in account. Sixteen questions out of the 32 are left.

Then there are $8 + 16 = 24$ questions for which the two modules performed differently. For 16 out of those 24 the syntactic pattern module gave a correct answer, for which the surface pattern module did not. Our null hypothesis was:

H_0 : There is no difference in the performance of both modules.

To calculate the chance that we get 16 successes out of 24 trials given H_0 , we use the following formula for binomial chance [21]:

$$P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

So $n = 24$, $k = 16$ and $p = 0.5$, because we assume H_0 . The chance of 16 successes out of 24 trials given H_0 is thus:

$$P(X = 16) = \binom{24}{16} 0.5^{16} (1 - 0.5)^{24-16} = 0.0758.$$

Using these numbers we get $p = 0.0758$. This means that the results are not significant at the traditional significance levels, 0.01, 0.005 and 0.001. We can not reject the null hypothesis.

But we could take those questions, for which no pattern was added yet, in account nevertheless. Because it indicates that the syntactic pattern module with less patterns was more robust than the surface pattern module with more patterns. The exception will be that particular question that was not answered correctly, because there was no pattern defined analogous to the 'subject' relation. In this case the syntactic pattern module was not more robust, I simply forgot to define a pattern for that linguistic structure.

So $n = 39$, $k = 31$ and $p = 0.5$. The chance of 31 successes out of 39 trials given H_0 is thus:

$$P(X = 31) = \binom{39}{31} 0.5^{31} (1 - 0.5)^{39-31} = 0.000147.$$

Using these numbers we get $p = 0.000147$. If we choose $\alpha = 0.001$ then still $p < \alpha$, because $0.000147 < 0.001$, so now our results are significant at level $\alpha =$

0.001.

However, one could argue that it is not entirely correct to leave out the lookup errors. After all, we decided to evaluate the modules within the context of a QA system and the table lookup is a part of it. Both methods have used the same lookup mechanism. Therefore, we decided to take these errors in account as well.

Then $n = 43$, $k = 31$ and $p = 0.5$. The chance of 31 successen out of 43 trials given H_0 is thus:

$$P(X = 31) = \binom{43}{31} 0.5^{31} (1 - 0.5)^{43-31} = 0.00270.$$

Using these numbers we get $p = 0.00270$. If we choose $\alpha = 0.001$ then $p > \alpha$, because $0.00270 > 0.001$, so now our results are not significant at level $\alpha = 0.001$. But they are at level $\alpha = 0.005$. We can conclude that our results show that we are for 99% sure, that the syntactic pattern module works better than the surface pattern module.

Chapter 7

Conclusion

The aim of this thesis was to investigate to what extent the use of dependency relations for the offline answer extraction method could improve the results of question answering compared to using regular expressions for the offline answer extraction method.

One problem using regular expressions will always be that information that differs slightly from the predefined patterns will not be extracted. A solution for this problem could be to extract the information using a dependency parser. With syntactic patterns, useful information that differs from the usual surface patterns will still be found and extracted, because we are looking at grammatical relations between words. So, we expected that recall would increase using a dependency parser and more questions would be answered correctly.

In this thesis we compared two offline answer extraction modules for question answering. The first module was developed by the LIT-group of the University of Amsterdam. This module used regular expressions to extract useful information. The other module that we developed used a dependency parser. We only looked at person information. To evaluate the two modules within the context of the QA system of the LIT-group, the text collections and question sets of TREC were used.

We didn't manage to keep the two extracting modules parallel in terms of the covered linguistic phenomena. But that was partly due to the fact that the syntactic pattern module only needed one pattern where the surface pattern method needed two or more patterns. This is actually an advantage of the syntactic pattern method. An other advantage of that method is that it is easier to extract relevant information than with surface patterns.

The results of our experiments confirm our hypothesis that recall would increase using a dependency parser and that more questions would be answered correctly. Evaluating the results for the question set of the old text collection where an answer was considered correct if it was ranked in the top three, we found that the syntactic pattern module performed significantly better than the surface pattern module.

Future work

Future plans could be focused on extensions of the categories. I have only looked at tables for person information, but it could also work for other categories like locations. Some categories, like abbreviations probably benefit more from the surface pattern method. This suggests that both strategies should be combined.

From the results of my experiments becomes clear that the syntactic pattern method answered a larger percentage of questions correctly than the surface pattern method. However, I have not examined if the syntactic pattern method had a larger percentage of correct answers. With other words, I have looked at the recall (number of correct answers returned by the system divided by the total number of correct answer to be found in the text) , not at precision (number of correct answers found by the system divided by the total number of answers returned by the system).

In this research an incorrect answer is just as wrong as returning no answer. However, a user, who poses a question trusts the system. When the system gives an answer to his question, the user will think that this is the correct answer. For this reason it is better to return no answer then to give a wrong answer. In future work this could be taken into account.

Bibliography

- [1] R. Bernardi, V. Jijkoun, G. Mishne and M. de Rijke. *Selectively Using Linguistic Resources Throughout the Question Answering Pipeline*. In: Proceedings 2nd CoLogNET-ElsNET Symposium, 2003.
- [2] T. Brants. *TnT, a statistical part-of-speech tagger*. In: Proceedings 6th Applied NLP Conference, 2000.
- [3] E. Charniak. *Statistical parsing with a context-free grammar and word statistics* In: Proceedings of the 14th National Conference on Artificial Intelligence AAAI, 1997.
- [4] M. Collins. *A new statistical parser based on bigram lexical dependencies*. In: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, 1996.
- [5] A. Copestake and K. Sparck Jones. *Natural Language Interfaces to Databases*. In: The Knowledge Engineering Review 5(4), 1990.
- [6] B.F. Green, A.K. Wolf, C. Chomsky and K. Laughery. *BASEBALL: An automatic question answerer*. In: Proceedings of the Western Joint Computer Conference 19, 1961.
- [7] M. Fleischman, E. Hovy and A. Echihabi. *Offline Strategies for Online Question Answering: Answering Questions Before They Are Asked*. In: Proceedings 41st Annual Meeting of the Association for Computational Linguistics, 2003.
- [8] D.G. Hays. *Automatic language data processing*. In: Computer Applications in the Behavioral Sciences, 1962.
- [9] J. Herrera, B. Magnini et al. *The Multiple Language Question Answering Track at CLEF 2003*. In: Working Notes for the CLEF 2003 Workshop, 2003.
- [10] L. Hirschman and R. Gaizauskas. *Natural Language Question Answering: The View from Here*. In: Natural Language Engineering 7(4), 2001.

- [11] V. Jijkoun, J. Kamps, G. Mishne, C. Monz, M. de Rijke, S. Schlobach, and O. Tsur. *The University of Amsterdam at TREC 2003*. In: TREC 2003 Notebook Papers, 2003.
- [12] V. Jijkoun, G. Mishne, and M. de Rijke. *The University of Amsterdam at QA@CLEF 2003* In: Working Notes for the CLEF 2003 Workshop, 2003.
- [13] V. Jijkoun, G. Mishne, and M. de Rijke. *Preprocessing Documents to Answer Dutch Questions*. In: Proceedings BNAIC'03, 2003.
- [14] D. Jurafsky and J. H. Martin. *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice-Hall, New Jersey, 2000.
- [15] D. Lin. *PRINCIPAR - an efficient, broad-coverage, principle-based parser*. In: Proceedings of the 15th International Conference on Computational Linguistics (COLING-94), 1994.
- [16] D. Lin. *Dependency-Based Evaluation of MINIPAR*. In: Proceedings of the Workshop on the Evaluation of Parsing Systems, First International Conference on Language Resources and Evaluation, 1998.
- [17] J. Lin and B. Katz. *Question Answering Techniques for the World Wide Web*. Tutorial presentation at The 11th Conference of the European Chapter of the Association of Computational Linguistics (EACL-2003), 2003.
- [18] C. Monz. *From Document Retrieval to Question Answering*. PhD Thesis, ILLC, University of Amsterdam, 2003.
- [19] C. Monz and M. de Rijke. *Tequesta: The University of Amsterdam's Textual Question-Answering System*. In: Notebook Papers TREC-10, 2001.
- [20] C. Monz, J. Kamps, and M. de Rijke. *The University of Amsterdam at TREC 2002*. In: The Eleventh Text REtrieval Conference (TREC 2002). National Institute of Standards and Technology, 2002.
- [21] D.S. Moore and G.P. McCabe. *Statistiek in de Praktijk*, 2e herziene uitgave., Academic Service, Schoonhoven, 1994.
- [22] C. Peters. *Introduction to the CLEF 2003 working notes*. URL: http://clef.iei.pi.cnr.it/2003/WN_web/00.2-intro.pdf, Last visited january 23rd 2004.
- [23] A.V. Phillips. *A question-answering routine*. Memo. 16, Artificial Intelligence Project, 1960.

- [24] E.F. Sang and F. de Meulder. *Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition*. In: Proceedings of CoNLL-2003, 2003.
- [25] R. F. Simmons. *Answering English Questions by Computer: A Survey*. In: Communications of the AMC 8(1), 1965.
- [26] E.M. Voorhees and D. K. Harman, editors. *Proceedings of the Eighth Text Retrieval Conference (TREC-8)*. NIST Special Publication, 2000.
- [27] E.M. Voorhees *Overview of the TREC-9 Question Answering Track* In: The Ninth Text REtrieval Conference (TREC 9) (5), 2001.
- [28] E.M. Voorhees. *Overview of the TREC 2001 Question Answering Track*. In: The Tenth Text REtrieval Conference (TREC 2001), 2002.
- [29] E.M. Voorhees. *Overview of the TREC 2002 Question Answering Track*. In: The Eleventh Text REtrieval Conference (TREC 2002), 2003.
- [30] E.M. Voorhees. *Overview of the TREC 2003 Question Answering Track*. In: The Twelfth Text REtrieval Conference (TREC 2003), 2004.
- [31] W. Woods. *Progress in natural language understanding - an application to lunar geology*. In: AFIPS Conference Proceedings 42, 1973.