UiO : Department of Informatics
University of Oslo

Use of Syntax in Question Answering Tasks

Marte Svalastoga

Master's Thesis Autumn 2014

Acknowledgements

First, I would like to express my deepest gratitude to my two supervisors Rebecca Dridan and Lilja Øvrelid for their patience, support, and brilliant feedback. I am greatly honoured to have been allowed to worked with them, their guidance has been invaluable.

I am also immensely grateful to my parents, who have offered support and motivation, especially when I was not able to see an end to this thesis.

Thanks to my boyfriend Marius, I have stayed mostly sane during the course of writing this thesis. He made sure I remembered to eat and sleep at appropriate intervals, and helped me keep a positive attitude throughout the process.

Finally, I would like to thank my fellow students and the staff of the Language Technology Group at the University of Oslo for creating an exceptionally good environment for me to work in, and also frequently sharing foodstuffs containing sugar.

Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Humans are born curious, and we learn by asking questions and exploring. With the explosive growth of the Internet and the availability of information online, learning how to ask questions the right way has become an art, first with boolean queries, and later special keywords to optimize use of a search engine. Often these queries do not reflect the way we would ordinarily use language. For example, if you want to know what the largest city in France is, not counting Paris, a likely search query would be `"largest city france -paris"`. While the form of this query makes sense from a technical perspective, as it includes the most relevant words and even shows what words to exclude from the search, it is not a natural utterance. A second inconvenience is that most search engines will then return a list of results where the user is required to manually examine the results in search of the answer.

If you were to ask another person the same question, the phrasing might be more like "What is the largest city in France, other than Paris?", and the human (if she knows the answer) will reply "Lyon". This format of interaction is the ultimate goal of **Question Answering**, allowing human users to interact with computers using natural language, and getting natural language answers.

Question answering is not about the technicalities of the input and output of the conversation, but going from a natural language question to a natural language answer. While the goal is to achieve this seamless form of communication, there is still a long way to go.

In this thesis, we will create a question answering system that is especially designed to answer multiple choice questions. The questions and the text accompanying it are supplied as part of a shared task, a joint effort within an academic community to further progress within specific parts of a field.

What we wish to investigate with this system is the effect of using syntax for a shared task, where our initial hypothesis is that using syntactic information will be beneficial. Syntax is "the study of the principles and processes by which sentences are constructed in particular languages", according to Noam Chomsky (Chomsky 1957). When we discuss use of syntax in this thesis, we refer to the information that can be gained about

the role a word has in a sentence. There are several forms of syntactic analysis, such as lexical analysis, which is used to determine whether a word is a noun, verb, etc., or grammatical analysis, where the word is assigned a grammatical function, such as subject or predicate of a sentence. In this thesis, words are analysed both lexically to get *part-of-speech tags* and grammatically to get *dependency relations*. More detail on this is given in the background chapter.

Without any syntactic analysis or external knowledge sources, methods of text processing are limited to counting words, either alone or in sets of multiple words in the order of appearance. With syntax, our goal is to abstract away from surface variations and thus creating a more robust system.

1.1 Motivation

The goal of this thesis is to investigate how syntactic information can be used to increase the performance of a question answering system, measured by its accuracy. Accuracy is defined as the number of correct answers divided by the total number of answers. While several systems in the given task did use syntax, there were other aspects in play that could affect the performance of the systems. For example, a system using syntax could score better than a system relying on shallow heuristics based on better document retrieval or a more refined ranking algorithm. We want to assess the usefulness of adding syntactic information seen in isolation, and to do this we create a system of our own to perform experiments on.

1.2 Results

After comparing results from the two configurations of our system, one baseline version using shallow heuristics and one using syntactic analysis on top of the baseline, we could not see any benefits from using syntax directly. When analysing the data, it became clear that the design of the task made this approach difficult. We came up with an alternative strategy using syntax to resolve some references in the text, and saw positive results.

1.3 Thesis outline

Chaper 2 — Background
In this chapter, the necessary fields, concepts and methods used in this thesis are introduced. First, an outline of Question Answering (QA) is given, including an example of a basic pipeline of a QA system. Two related fields within natural language processing are then briefly described, before we introduce the concept of shared tasks. Leading up to the presentation of the task this thesis will attempt to solve, we give a short history of the tasks preceding it. After presenting the task, a brief outline of a few selected systems that participated in the task is given, before we give an introduction to syntactic analysis.

Chapter 3 — Baseline system description
A baseline system is built and described, serving as a reference point for performance before syntactic features are added. After describing the process of preparing the background data, each of the modules in the system are introduced. The concept and application of normalisation is then described, before results of running the system are presented.

Chapter 4 — Experiments with syntax
This chapter details an error analysis performed on the results gained from the baseline system, before describing which syntactic features were added. A dependency parser is introduced, as well as the process of converting the data to a format readable by the parser. The results of this addition are briefly discussed at the end of the chapter.

Chapter 5 — Data analysis
With the results from adding syntactic features in mind, the data is examined more closely. In this chapter, the process of classifying the dataset is described, as are the different classes and their criteria. The results are then listed and discussed.

Chapter 6 — Experiments with anaphora resolution
In this chapter, syntactic information is being used in a different manner, namely to resolve anaphora. First, a naive algorithm for resolving anaphora using basic syntactic trees is described, before presenting a reinterpretation of this algorithm for use on dependency structures, created for this thesis. The results of running the algorithm on the text, and the impact this had on the complete system are then described.

Chapter 7 — Conclusion
We discuss our findings, and present ideas for potential improvements of the system, as well as some suggestions for future work.

Chapter 2

Background

In this chapter, we present the fields and concepts necessary to understand the main content of this thesis. The intended audience are fellow master's students in informatics, meaning that while no previous knowledge of language technology is required, we assume that the reader has a good understanding of computer programs and how they work.

We begin by introducing the task of Question Answering (QA) by presenting its purpose and describing a basic pipeline for a QA system. After the function of the different components are explained, we move on to introduce two of the sub-fields that are closely related to question answering. The concept of shared tasks is then presented, and the shared task this thesis focuses on is introduced. Details of the previous tasks are briefly summarised, before introducing the task from 2011. Three systems participating in the 2011 task are briefly outlined to give an idea of how these systems work. Finally, we introduce the basic concepts of syntactic analysis, describe the annotation scheme we use, and explain how this analysis can be used.

2.1 Question Answering

The task of Question Answering (QA) is a subtask of Natural Language Processing (NLP) which is concerned with answering questions posed in a natural language. NLP is also known as *human language technology* and *computational linguistics*, often depending on the focus of the person or group describing the field. The general focus of the field is on the use of natural languages in combination with computers. These uses include both permitting for the use of natural language in communication between humans and computers, and using computers to process spoken or written language. (Jurafsky and Martin 2009) The term *natural language* is simply a way to specify that the languages in question are ones that are or have been used by humans for communication, such as Norwegian or English, in contrast to *formal languages*. Formal languages are constructed languages, such as programming languages, that are clearly defined and unambiguous.

When presenting a QA system with a questions such as "When was the UN founded?", the system should provide a concise answer, for example

"24 October 1945" or "1945". The ability of a system to return only the answer itself is considered an improvement on traditional information retrieval systems that return a document or phrase likely containing the answer, but not necessarily phrased in a way to exactly answer the question.

QA is a useful task not only for its direct applications, but because in order to answer a question, the system must understand what is being asked for, know where to find the answer, and present the answer in a format that reflects the questions. All of these steps are helpful in bringing systems closer to a complete understanding of natural languages, which is an overarching goal of NLP.

Technology using question answering is already available on a multitude of platforms, such as the online search engine WolframAlpha[1] and Apple Software's personal assistant Siri, which is available on Apple's mobile devices. Both of these can take questions and return answers, Siri in spoken form and WolframAlpha in written form.

While there are many practical uses for question answering systems, this thesis concerns the academic interest in the field. Progress in the field is largely a result of shared tasks that aim to improve the general state of QA systems by focusing on different aspects of systems, working towards certain long-term goals. These tasks and goals will be introduced in section 2.3

In the next section, we will describe the layout of a basic QA system, the purpose of each component, and how they interact.

2.1.1 The basic QA pipeline

A QA system typically consists of a series of components, each performing an operation on some part of the given data, before sending results to the next components. We describe the pipeline of a system created to solve a generic QA task consisting of receiving a question, searching a given repository for texts containing the answer, and returning the answer. The focus on the components in the following sections is on the applicability for our system.

The components can vary, this layout is based on one described by Webb and Webber (Webber and Webb 2010). An illustration of the components from the same book is shown in figure 2.1. In this illustration, the arrows represent the data flow of the system.

Question typing

Question typing takes the question as input, and returns one or more labels describing what sort of entity the answer is likely to be. These labels describe the type of information the question asks for, for example PERSON or DEFINITION. This step is not a necessity for a system, but can be a useful tool to filter retrieved passages and rank answer candidates.

The list of labels is finite, and each label is given a set of defining features. These features can be syntactic, semantic and lexical. An example of a question and its label can be seen in figure 2.2.

1. `https://www.wolframalpha.com/`

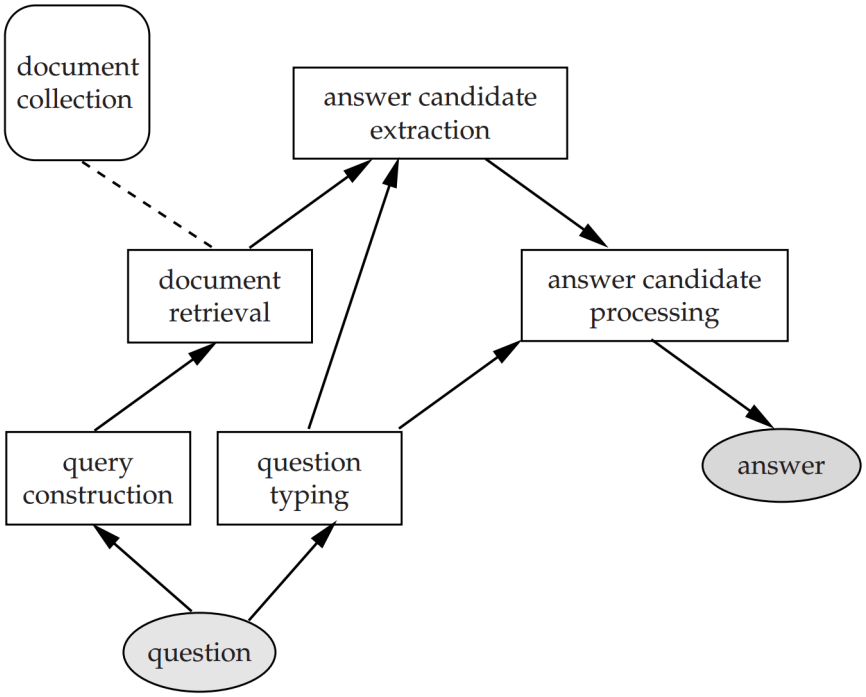

Figure 2.1: Basic QA system architecture

Q: Who was the founder of the Salvation Army
Label: PERSON

Figure 2.2: Question and assigned question type

In 2006, Li and Roth (Li and Roth 2006) described a classifier that achieved 92.6% accuracy when using 6 coarse-grained question types. These types were ABBREVIATION, DESCRIPTION, ENTITY, HUMAN, LOCATION, and NUMERIC. These were divided into 50 fine-grained types, and the classifier reached 89.3% accuracy on these. This classification was done by using a hierarchical classifier that extracted syntactic and semantic features, as well as utilised external knowledge.

A common alternative is to build a simpler classifier that works on hand-written rules for coarse classification. These rules can work by for example matching question words to labels. An few examples are shown in figure 2.3. These labels are then used in both query construction and answer candidate extraction, described in the next subsections.

Query construction and document retrieval

The aim of this part of the pipeline is to find the document(s) containing the answer to the question. In order to get the document, a **query** is constructed, which is then used as a basis to search through the document

Question word	Label(s)
Who	PERSON, ORGANIZATION, COUNTRY
Where	LOCATION
When	DATE

Figure 2.3: Examples of assignments of labels to question words

collection. Query construction and document retrieval are closely linked, as the type of retrieval used determines what sort of query should be created. In general, query construction is the component of the system concerned with constructing a query — a search string — to send to a search engine, in the hopes of retrieving a document containing the answer. A *document* is loosely defined as some body of text. It can be a large text consisting of multiple pages, or a sentence. This depends on both the dataset and how the system is designed, as some systems perform segmentation of the available document collection as part of the preprocessing.

We focus on two types of document retrieval. Both of these methods make use of a text search engine, but the strategy differs. The first of the two types of retrieval is **relevance-based retrieval**. Systems using these techniques aim to create queries that can retrieve texts *relevant* to a topic, by a chosen metric. To do this, the system must also include some measure to assess relevance. A common method to increase the chance of finding a relevant passage of text consists of breaking down the text to smaller segments and performing the search on these, assuming that segments that match the query are more relevant than those that do not match.

This method can be further refined by indexing types of *named entities* in addition to word positions, using *predictive annotation* (Prager et al. 2006). The named entities become useful only if question typing has been performed, as the system can then search for named entities with a label describing the form that the answer is likely to take. With these indexes, the system can perform more efficient searches for relevant documents. The entity types can be applied to the query construction as well, where the general idea is that the query should reflect the types of entities the answer might be.

The other form of document retrieval is **pattern-based retrieval**. Methods used in this type of retrieval are based on rewriting the question to different surface variations of the same question, increasing the chances that the phrasing of the question better matches the phrasing used in the answer. One way of doing this consist of removing the question word, and creating permutations of the remaining words. Some possible permutations are shown in figure 2.4. In these examples, the determiner and the word following it was considered an entity, in order to preserve some meaning. However, as can be seen in the example, this alone is not enough to preserve noun phrases, “Salvation” and “Army” are separated. With the use of shallow heuristics, the suggestions can be kept somewhat grammatical without needing knowledge of syntax.

Who was the founder of the Salvation Army
was the founder of the Salvation Army
the founder of the Salvation Army was
the founder of was Army the Salvation
the Salvation Army was the founder of

Figure 2.4: Example of generated queries based on patterns

Q: Who was the founder of the Salvation Army?
A: <answer> was the founder of the Salvation Army

Figure 2.5: Example of adding an answer-slot

A second way of using patterns is by inserting an *answer slot* in the place of the question word. The assumption is that if the pattern is found, the answer is in the place of the answer slot. See figure 2.5 for an example. This method can be combined with the previous method, offering permutations of the question with the answer-slot in place.

Answer candidate extraction

The purpose of answer candidate extraction is to go from the documents containing the answer, to the answer itself. This means that the system must isolate the answer candidate from the surrounding text. Several candidates can be extracted, to be evaluated in the answer candidate evaluation step. If pattern-matching using answer slots was used in the previous step, the answer candidate is given directly, and can be sent to answer candidate evaluation. Likewise, if named entities were used with relevance-based retrieval, the matching entity can be evaluated directly. There are other methods for extracting answer candidates, but as this thesis is concerned with a task where there is a predefined set of answer candidates, we will not go into more detail on these.

Answer candidate evaluation

In this final component, the system chooses what answer to return. The system might make use of a ranked list of candidates, and return either the top candidate or the top n candidates, depending on the task it attempts to solve. Ranking answer candidates is the main task of this component, if the previous components have returned more than one candidate. The ranking should reflect the likelihood of the answer candidate being correct, with the strongest candidate appearing first. This part of the QA pipeline has a high amount of overlap with textual entailment (see section 2.2.1), as the task can be formulated as determining whether the text entails the answer candidate.

There are several options for how to rank the candidates, we will describe two of them here. First, the candidates can be ranked by their frequency in

the text, under the assumption that there is a relation between frequency and relevance. The dependency-based system (Babych et al. 2011) described in a later section uses this technique. The frequency count can be further refined by focusing on named entities in the candidate, which requires a named entity recognition step to be performed on the data. The usefulness of looking at counts depends on the data, as the correct candidate is more likely to only only a few times — or even just once — in a smaller dataset.

A second possibility described by Webb and Webber makes use of online resources, and uses these to estimate the probability of the correctness of an answer.

Some systems, like the retrieval-based system (Verberne 2011) we will introduce later in this chapters, will only give an answer if the confidence in the highest ranked answer candidate exceeds a set threshold. This confidence can be determined by looking at how close the top rated candidates are by whatever measure was used to rank them.

Once this component has chosen one or more answers to return, the system has completed its task, and can then be evaluated by some external measure on how many correct answers it achieved.

2.2 Related sub-fields

In this section, we briefly introduce some related fields that are especially relevant for this thesis. Question answering is one task under the larger field of NLP. Tasks in NLP focus on solving parts of a common goal, and have a lot in common. Some of the tasks that are closest related to QA are **information retrieval**, which can be used to find information in a collection of documents, and **textual entailment** to attempt to prove whether or not an answer is true given a source, or to choose the best answer given a list of candidates. In the next paragraphs, we describe the two tasks, and how they can contribute to question answering.

2.2.1 Textual Entailment

The focus in Textual Entailment (TE) is proving whether or not an hypothesis H is entailed by a text T. To be more exact, entailment is described in the first PASCAL Recognising Textual Entailment Challenge as "[...] T *entails* H if, typically, a human reading T would infer that H is most likely true." (Dagan, Glickman, and Magnini 2006) The most basic way of attempting to prove inference is through word overlap, simply assuming that high overlap equals entailment. A less common way is to translate sentences to logical statements, and use a logical prover to see if T entails H. The answer validation module of a QA system can be described as a TE problem in itself, where H is the answer candidate, and T is the text.

2.2.2 Information Retrieval

Information retrieval (IR) is the act of retrieving information from sources of data, either structured or unstructured. In this thesis, we choose to focus

on retrieval from unstructured data sources, i.e. texts written in human language, containing facts. The task of IR originated in the need to search in scientific publications and library records, but has expanded since (Manning, Raghavan, and Schütze 2008). In recent years, the World Wide Web has been a driving force in innovations within information retrieval, as users world wide use search engines to find relevant content on the web. For QA, IR methods are used for retrieving documents relevant to the question, and selecting documents likely containing the answer. Most QA systems use existing search engines, such as Lucene.[2]

2.3 Shared tasks

Shared tasks are tasks given within a specific field, in relation to a conference or lab, with training and test data offered by the organisers of the task. The goal of these tasks is to focus the community on further development of certain aspects of the field, and to be able to compare the performance of the submitted systems. The systems must be accompanied by a paper describing the approach used, sharing whatever new insights were gained while solving the task.

Before we introduce the task that is the main focus of this thesis, we briefly describe some of the shared tasks in related fields and what their focus is. Computational Natural Language Learning (CoNLL) is an annual conference that accepts contributions on a number of language learning topics, and hosts a shared task for every conference.[3] The topics of these tasks are varied, from the task in 2000 concerned with *chunking*, the process of splitting text into segments that are syntactically related, to the task from 2013 on grammatical error correction.

The Text REtrieval Conference (TREC) is co-sponsored by the National Institute of Standards and Technology (NIST) and U.S. Department of Defence.[4] As the name implies, the yearly conferences are focused on text retrieval, also known as information retrieval. From 1999 until 2007, TREC had a QA track with a strong focus on information retrieval. However, the data and tasks were only focused on English, leaving other languages behind. In order to bridge this gap, the CLEF initiative started its own multilingual QA track.

A shared task called Recognizing Textual Entailment (RTE) (Dagan, Glickman, and Magnini 2006) has been held anually since 2006, and has been the main task for textual entailment.

2.3.1 CLEF

The CLEF Initiative (Conference and Labs of the Evaluation Forum, formerly known as Cross-Language Evaluation Forum) describes themselves as a *"self-organized body whose main mission is to promote research,*

2. `http://lucene.apache.org/`
3. `http://ifarm.nl/signll/conll/`
4. `http://trec.nist.gov/overview.html`

innovation, and development of information access systems with an emphasis on multilingual and multi-modal information with various levels of structure." [5]

The initiative has hosted a series of tasks divided into different areas or tracks. These tracks cover a given area of computational natural language understanding, with a focus on multilinguality. One of these tracks is **QA@CLEF: Multilingual Question Answering Track at CLEF**, which has run from 2003 and is still ongoing at the time of writing. In the next section, we describe some of the tasks held previously in this track, leading up to an introduction of the task from 2011, which is the task we will focus on in this thesis.

2.3.2 Previous CLEF QA tasks

In this section, we briefly introduce the previous tasks for the QA-track at CLEF, and how each differed from the one the year before. Each year, there is a main task which consists of systems answering a series of question. This task can either be performed on only one of the languages supplied, or with a different source and target language.

2003 A monolingual main task was given where the systems could choose one of three languages. They were given 200 questions, and asked to return an answer alongside the ID of the document containing the answer. The answer could either be the exact answer, or a 50 byte long string. In addition to this, a cross-language task was given, asking the systems to find responses in an English corpus to queries posed in a different language. The best system had the correct answer ranked amongst its top three candidates in 99 of 200 questions. See Magnini et al. 2004.

2004 Nine source languages and seven target languages were part of the main task this year. For these, 200 questions were provided, and only exact answers were accepted. Systems needed to go from questions in a source language, to answers in a target language. With English as target language, all the systems combined managed to answer 65% of the questions, while individual systems ranged from 10.88% to 23.5%. There was no monolingual task for English this year. See Magnini et al. 2005.

2005 The task was mainly unchanged from 2004, except for the addition of a few languages on both the target and source side, giving participants a chance to improve their systems within the current restrictions. This year, the best monolingual systems (i.e. systems using the same source and target language) saw a drastic improvement, with the best system answering 64.5% of the questions correctly. See Vallin et al. 2006.

2006 Two tasks were given in addition to the main task from the previous

5. `http://www.clef-initiative.eu/web/clef-initiative/home`

years, WiQA (Jijkoun and De Rijke 2007) and Answer Validation Exercise (AVE) (Peñas et al. 2007). WiQA - short for Question Answering using Wikipedia was a task focused on finding new information given a topic. Both a monolingual and bilingual version was offered. In AVE, systems had to answer `YES` or `NO` to a question given a text-snippet, making it a textual entailment problem (see section 2.2.1). In the main task, the best monolingual system reached an all-time high with an accuracy of 68.95%. See Magnini et al. 2007.

2007 The same main and subtasks were given again, with the addition of a pilot task, Question Answering on Speech Transcripts (QAST) In this task, the systems had to look for answers to questions in transcripts of spontaneous speech.
In the main task, a new challenge was added by grouping the questions into topics, where coreferences — expressions referring to the same entity — could be kept between questions. A second change was made by adding data from Wikipedia as source material. For this task, results were somewhat lower than last year, with the best score at 54% accuracy, and the average down from 49% in 2006 to 42% in 2007. See Giampiccolo et al. 2008.

2008 The subtasks from the previous year were given again, in addition to the main task. The main task remained unchanged, allowing participants to get more experience with the changes from last year concerning coreferences in questions. Because of this, the monolingual scores increased, with the best system achieving 64% accuracy, and the average accuracy was 24%. See Forner et al. 2009.

2009 In a change from previous years, three separate tasks were held this year, none of them being the "main" task. QAST was continued, as well as GikiCLEF — a task on questions requiring geographic reasoning amongst other things — and ResPubliQA was added. The focus of ResPubliQA was retrieving the passage containing the answer to the given questions, using the European legislation as document collection. Combined, all the systems managed to retrieve the correct passages for 90% of the questions. The highest individual system managed it for 61% of the questions. See Peñas, Forner, Sutcliffe, et al. 2010.

2010 The broad strokes of the main task from 2008 remained the same, with two changes: systems were given the option to return either the paragraph containing the answer, or the answer itself. A second change was the addition of parts of the EUROPARL collection. There were separate evaluations for Paragraph Selection and Answer Selection. Due to a different scoring systems, scores could not directly be compared to those from previous tasks. See Peñas, Forner, Rodrigo, et al. 2010.

Throughout all of these tasks, the broad strokes of the main task have persevered, but some aspects have changed. In the first task, systems could submit a 50 byte string as an answer, and in 2009 they were to return the paragraph from the document collection containing the answer. The document collection itself has also changed, where it in 2003 had no particular theme, texts from Wikipedia were added in 2006, and different topics were introduced in 2007.

Several pilot tasks have been introduced, some of them focusing on specific parts of the QA pipeline. As the organizers saw an upper bound in the accuracy measured in percentage, systems struggled to push out of the 60s without succeeding (Peñas et al. 2011). The pilot tasks were added in the hopes of improving that number. However, even after including some of the changes from the pilot tasks into the main task, such as allowing answers to remain unanswered without affecting the accuracy by introducing the *c@1* measure, performance did not improve much.

With this in mind, the task for 2011 was created, called QA4MRE.

2.3.3 QA4MRE

Question Answering for Machine Reading Evaluation (QA4MRE) is a task that was introduced for the multilingual QA track of CLEF in 2011. The goal of this task is to make systems focus on reading comprehension, hoping it could change the way systems were designed, and ideally help them improve accuracy scores that had plateaued over the last few years. Up until this task, the questions had been simple, and rarely required any understanding of the text. When looking at the systems submitted for previous tasks, the organisers found two main areas that needed improvement: answer extraction and answer validation. For this task, answer candidates were given, meaning that answer extraction was not needed, answer validation became an even more significant step.

For this task, 3 topics with 4 tests each were given, with 10 questions and 5 answer candidates per test. Each reading test consisted of a transcribed TED talk on the given topic. These talks were chosen for the high quality crowdsourced translations available. The topics were **Aids**, **Climate Change** and **Music and Society**.

In addition to this, a background collection was supplied for each topic, with data obtained by webcrawling. The number of documents retrieved varied from around 25000 to 130000. While the texts that belong to each test were the same for all the languages, the background collection varied. The intention of the background collection was to supply data in the domain, which systems could use in whatever way they wanted.

This task is considered the first step towards QA systems that can both create an hypothesis based on the background collection and show supporting documents alongside the hypothesis.

As a big point in the CLEF tasks has always been multilinguality, this was no exception. The task was available in five languages, English, German, Italian, Romanian, and Spanish.

The questions were created by the organisers, and posed in such a way

Q: What is Annie Lennox's profession?

1. *mother*
2. *nurse in a hospital*
3. *farmer*
4. **musician**
5. *dancer*

Where the sentence containing the answer is as follows:

> And so, subsequently, I participated in every single 46664 event that I could take part in and gave news conferences, interviews, talking and **using my platform as a musician**, with my commitment to Mandela, out of respect for the tremendous, unbelievable work that he had done.

Figure 2.6: Example of questions and answers, the correct answer and the corresponding sentence in the text is shown in bold.

$$\text{c@1}=\frac{1}{n}(n_R + n_U \frac{n_R}{n})$$

Where
n_R: number of questions correctly answered.
n_U: number of questions unanswered
n: total number of questions

Figure 2.7: Definition of the c@1 measure

that mere pattern matching would not suffice. An example is shown in figure 2.6.

Evaluation measure

The evaluation measure used in this task and several of the previous ones is *c@1*, as defined by Peñas and Rodrigo (Peñas, Rodrigo, and Rosal 2011), and is shown in figure 2.7.

The measure was created with the intention of rewarding systems for refraining from answering questions, rather than giving incorrect answers. This means that systems have something to gain from assessing the certainty with which it answers a question, a valuable trait in tasks where reading comprehension is measured.

2.4 Previous work

While the task description creates some constraints on the systems, a number of different solutions are chosen for the actual implementation. In the following sections, we describe a small selection of systems that participated in the QA4MRE task at CLEF 2011. The systems are chosen either for their results or chosen approach to the task. For each system, we describe some of their features, and list the suggested improvements from each accompanying paper.

2.4.1 The hybrid system

The first system had the highest score in the task, with its two best runs getting a c@1 of 0.57 and 0.47, while the second best system had its best run at 0.37. The paper accompanying this system was called "A Hybrid Question Answering System Based on Information Retrieval and Answer Validation" (Pakray et al. 2011).

The philosophy of the system can be described as a "more is more" approach, using a high number of modules combined. This approach proved to be very effective. In the next paragraphs, we highlight a few of the modules and describe their function.

Textual entailment

This module attempts to prove that a text fragment T proves the answer candidate H. The proving is done by checking for overlaps of unigrams, bigrams and skipgrams between T and H, including their synonyms from WordNet (University 2010).

Dependency parsing

Using the Stanford dependency parser for a dependency analysis, the system uses a set of comparisons between the relations in the retrieved sentence and in the hypothesis. WordNet was used in some of these comparisons, allowing for a match if the WordNet distance between a pair of verbs was below a given threshold, meaning the verbs could be considered of similar meaning.

Anaphora resolution

Anaphora are back-references to an entity that has already been introduced, and take the form of pairs of the antecedent — the entity itself — and the anaphor, the reference. In this instance, anaphora are pronouns. An example is shown in figure 2.8, where "Anne" is the antecedent to the anaphor, "she". When resolving the anaphora, the anaphor is replaced with its antecedent.

A few simple heuristics were applied to resolve anaphora, after a named entity tagger was run on the text to identify and mark Named Entities (NE). First person personal pronouns were replaced with the name of the author, with the exception of occurrences of these pronouns in direct speech,

Original text:
Anne bought a car. **She** did not regret it.

After anaphora resolution:
Anne bought a car. **Anne** did not regret it.

Figure 2.8: Example of anaphora resolution

in which the first NE of that sentence was used instead. Second person personal pronouns were resolved to the last NE of the previous sentence.

Suggested improvements

The authors did not make any suggestions for improvements in the paper detailing their system, but they participated in 2012 (Bhaskar et al. 2012) with some improvements to their system. The biggest change from 2011 and 2012 was the addition of a *knowledge base*, which contained a named entity list, abbreviation list and multi-word list for three of the four topics. The system had the lowest performance for the topic lacking a knowledge base, something the authors used to testify to its usefulness.

2.4.2 A retrieval-based system

A second system that performed well in the 2011 task was a retrieval-based system participating for the first time (Verberne 2011). Their approach was to rely on the strength of the text retrieval part of the pipeline.

Text expansion

About half the systems used the background collection in some way. This system used it to expand on the test document, making the assumption that any required facts that were not present in the test document were still related to information present in the test document. In this expansion process, text fragments of 3 sentences from the test document were used to search for similar sentences in the background collection, using a ranking function. The text fragments were then expanded with the ten highest-scoring sentences that were 4 words or longer.

Question expansion

Using "An English Hybrid Dependency Parser" (AEGIR) (Oostdijk, Verberne, and Koster 2010), the background collection was parsed, and facts were extracted. The form these facts take is shown in figure 2.9

The facts were then indexed both by subject and object, and if any sub-string of the questions was found as either subject or object in this fact collection, the fact was added to the question. These facts were used in the answer selection process, where answer candidates are scored on their similarity to facts and extracted target-fragments from the test document.

```
needle program | reduce | the spread |
Queen II Elizabeth | honor | Annie Lennox |
the future prime minister | write | romance novel | possibly
```

Figure 2.9: Example of extracted facts

Suggested improvements

In the fact expansion, facts containing pronouns were discarded. The authors mention that resolving the anaphora would be a better solution, but it was not a part of their system.

2.4.3 A dependency-based system

The final system we will consider from the QA4MRE at CLEF 2011 task is a system that attempts to solve the task for German (Babych et al. 2011), rather than English as the two previous systems did. The system uses the output of a dependency parser as a base, and attempts to transform this output to a semantic representation, abstracting away from surface variability.

Knowledge extraction

The system extracted hypernym-hyponym pairs and detected synonyms by using a combination of regular expressions and vector space models. These relations needed to score above a given threshold for similarity before being added to a database.

Inference

The main focus of the system was using the dependency relations as a step to translate the questions, answers and texts into propositional logic formulae that could then be used to prove the correctness of an answer candidate. This process takes advantage of the fact that answer candidates are given, and one is guaranteed to be correct, by changing parameters until only one candidate can be proved.

Suggested improvements

The authors were pleased with the performance of the system, but observed an inverse relation between coverage and accuracy. A suggested improvement was to move deeper into understanding rather than matching linguistic material, by mapping the text to a knowledge representation structure.

2.5 Syntactic analysis

We introduce the concept of dependency grammar and dependency analysis as we believe this form of syntactic analysis can be useful to a QA system. Syntactic analysis is "[...] the task of recognizing a sentence and assigning a syntactic structure to it." (Jurafsky and Martin 2009, 461). One form of syntactic analysis is *dependency analysis*, which will be the focus of this section.

2.5.1 Dependency grammar

Dependency grammar is a set of theories and formalisms centred around a core idea, described by Joakim Nivre (Nivre 2005) in the following way: "syntactic structure consists of lexical elements linked by binary asymmetrical relations called dependencies". The assymetrical relations mentioned are representations of the relationship between two words, where one is dominated by the other. While many names exist for the constituents of this relation, in this thesis the dominating word is called the **head** and the dominated word the **dependant**. The relations can be expressed in plain text as *abbreviated_relation_name*(head, dependant). The terms **governor** or **regent** and **modifier** are sometimes used in the literature. Every word in a sentence is dominated by another word except for one, and that word is called the *root* node.

2.5.2 Stanford Dependency scheme

The Stanford Dependency scheme (SD) is one way of representing dependencies. It was first described in a paper called "The Stanford typed dependencies representation" (De Marneffe and Manning 2008). A strong focus in the development of the scheme has been usability and readability for people without a background in computational linguistics. Design-wise, the scheme was built upon a series of principles that in summary focus on making it simple, semantically contentful, and making sure relations are between content words rather than indirectly through function words. As we consider all of these features useful for this thesis, we decided to use the SD for our analysis. Figure 2.10 shows a visual representation of a dependency structure, from the Stanford typed dependencies manual (Marneffe and Manning 2008). Several representations are available in the SD, in this example and the rest of the thesis, the **basic** style is used. With this scheme, the root node is marked as having a `root`-relation, with no head. From the example, we can see that the subject (`nsubj`) and object (`dobj`) are given from the root. The relation `nsubj(makes, Bell)` shows that *Bell* is the subject, and `dobj(makes, products)` that *products* is the object, the thing that is made. These two relations give a decent summary of the sentence, but some information is lost. The first part is less relevant to the meaning of the sentence, the part giving the location of the company. However, the apposition-relation `appos(Bell, company)` tells us that *Bell* is a company, a useful fact. In addition to this, the `nn(products, computer)` relation

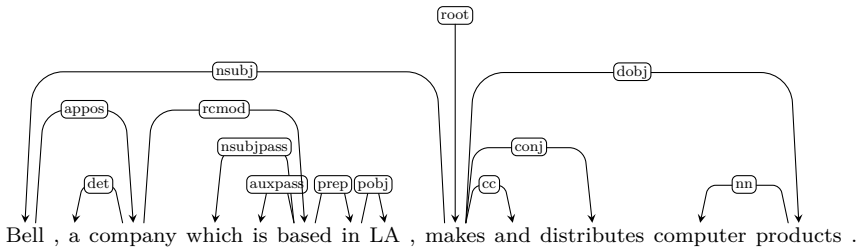

Figure 2.10: Visual representation of the dependency graph of a sentence

shows that *computer products* is a compound noun.

Formal Properties of Stanford Dependency Graphs

Three main conditions must be fulfilled for a dependency graph to be correctly formed in the basic version of the SD. The first, and most important, is that the graph **is a tree** with a single root-node forming the entry-point to the graph. Following from this, the graph must be **acyclic**, meaning that there must not be any cycles. A third condition follows, stating that the graph must be **connected**. All nodes must be connected to another node, except for the root node.

2.5.3 Dependency parsers

A parser is an implementation of an analysis scheme, automating the process of analysis. Parsers can be trained on annotated data, or work on manually created rules. Using a parser is a quick way of getting a dependency analysis, but the result is not necessarily correct. One of these parsers, MaltParser (Nivre, Hall, and Nilsson 2006), can get accuracies of 80-90% without any language-specific enhancements, according to the creators.

2.6 Summary

In this chapter, we have described what QA is, and shown an example of a basic pipeline. As QA is closely linked to other fields within natural language processing, we described two of these fields briefly, textual entailment and information retrieval. These two were chosen as they are especially close to the form of QA in the task. We then introduced the concept of shared tasks, before detailing the task that is the focus of this thesis. To get more insight in how the task can be solved, we have given a brief introduction to three systems that participated, focusing both on what features made them unique, and what improvements the authors of the papers accompanying the systems suggested. Finally, we introduced the concept of syntactic analysis and an annotation scheme for this that we will use in our system. We think that syntactic analysis can be a good way for a QA system to get a deeper understanding of text, and thus become more accurate.

Chapter 3

Baseline system description

In this chapter, we introduce the baseline QA system we built to solve the QA4MRE task at CLEF 2011. The goal of the system and this thesis is not to participate in a shared task, but investigate how syntax can be used in solving a QA task. Because of this, we will not focus on achieving accuracies that beat those achieved by others systems, but rather on creating a consistent system that can be used as a baseline for comparison after adding more advanced heuristics.

We describe the data preparation process that takes place before feeding data to the system, then present the different components of the system. The scores used to rank answer candidates are explained, before we present the results of running parts of the data through our system, and discuss the performance.

3.1 Motivation

In order to see changes in performance when utilizing syntax, we need a baseline system for comparison, which we describe in this chapter. Once we have a working baseline system, we can add heuristics using syntax, and examine changes in performance. We chose to use the task of QA4MRE at CLEF 2011 as a basis because its premise is relatively simple, and no answer candidate generation is involved. This means that getting a basic system up and running could be done quickly, and the focus would be on the answer candidate evaluation, limiting the potential sources of errors. A second reason for using a shared task is that the papers from the participating systems are all available, which gives some insight into what has been tried, and what the results were. With this in mind, we move on to describing the components of our system.

3.2 Data preparation

The English test data from QA4MRE at CLEF 2011 is given as a single structured XML file containing texts, questions and answers. The texts originally contained linebreaks and paragraphs, but these were removed as a part of the preformating done by the organizers of the task. This has led

```
1  <?xml version="1.0" encoding="UTF-8"?>
2    <test-set>
3      <topic t_id="1" t_name="AIDS">
4        <reading-test r_id="1">
5          <doc d_id="1">
6            Annie Lennox Why I am an HIVAIDS
7            activistI'm going to share with
8            you the story as to how I have
9            become an HIV/AIDS campaigner. And
10           this is the name of my campaign,
11           SING Campaign.
```

Figure 3.1: The first lines of the raw XML file

```
1  Annie Lennox: Why I am a HIVAIDS activist.
2  I'm going to share with you the story as to
3  how I have become an HIV/AIDS campaigner.
```

Figure 3.2: Edited version of the first two sentences of the first text

to two problems: first of all, since he title of the talk, including the author, is separated from the text with linebreaks, not with punctuation, the title of the talk has been merged into the first sentence, as seen on line 6 in figure 3.1. We solve this problem by manually editing the first sentence of each text before processing it further. As the formatting on title and speaker were not consistent, we decided on standardizing the format as shown in figure 3.2.

A second problem is that linebreaks are simply removed, not replaced with spaces. This means that some sentences are not separated by spaces, which can lead to issues with further parsing. We solve this by using a regular expression to split up the sentences.

Rebecca Dridan, one of the supervisors of this thesis, created a basic system for this task that reads the data, ranks the answer candidates based by their word-overlap with the question, and displays the results. These results are then compared with the gold standard answers, showing the ratio of correct answers for each reading test. We used this system as a skeleton for our system, allowing the focus to remain on building the answer validation component.

The answer ranking function in the original system used the length of the overlap between words in question and answer as score. We sought to improve upon this using other shallow heuristics, investigating what the system could and could not do with only these heuristics in place.

3.3 Document retrieval

Most QA systems use some form of document retrieval to collect the documents deemed most relevant by some measure to the question.

Input: "This is a sentence. It is followed by a second sentence."
Output: ['This', 'is', 'a', 'sentence.', 'It', 'is', 'followed', 'by', 'a', 'second', 'sentence', '.']

Figure 3.3: Input and output from word_tokenize

Input: "This is a sentence. It is followed by a second sentence."
Output: ['this', 'is', 'a', 'sentence', 'it', 'is', 'followed', 'by', 'a', 'second', 'sentence']

Figure 3.4: Final result of word tokenization

However, as the length of the texts in these task range from only 1234 to 3581 words, we decided to use the entire text, rather than extract segments.

With a large dataset, this approach would lead to very long run times, making development tedious. As this is not an issue, we decide to eliminate the potential risk of lack of accuracy introduced by the chosen methods for document retrieval, and examine the entire text instead.

We use the text in two ways: first as a single unit, one list of all the words in their original order, punctuation removed. A second representation is by sentences, using a sentence-tokeniser from NLTK (Bird, Klein, and Loper 2009). We will describe what unit of text is used for each of the heuristics described in the next sections.

The process of tokenisation means splitting something into smaller units. For our system, this means splitting a complete text into individual words, and a complete text into sentences.

The complete text is tokenised into words by using `word_tokenizer` from NLTK. As can be seen in figure 3.3, after performing the tokenisation, punctuation placed between words remains attached to the word before it. Punctuation at the end of the string is separate. Both types of occurrence of punctuation were removed, and all the words lower-cased. The result of this process can be seen in figure 3.4. While tokenising on words is good for treating the text as one unit, we also want to be able to use individual sentences as contained units.

Sentences are tokenized using the Punkt sentence tokeniser from NLTK, applying the supplied model for English. When sentences are used to rank answer candidates, the entity in question is compared to all the sentences in the text, and the highest score achieved for one single sentence is kept. Sentences are only seen in isolation.

3.4 Heuristics for answer ranking

When examining the system design of "The Hybrid system" (Pakray et al. 2011) mentioned in the previous chapter, one of the main features of the system was the sheer number of different heuristics that were combined to create a high accuracy system. We decided to use several simple heuristics

Heuristic	Text	Basis of score	Range
Overlap-based			
Q and A	none	overlap	0, 1, 2, ...
Q, A and sent.	Sentence	overlap×10	0, 10, 20, ...
A and text	Full	overlap	0, 1, 2, ...
Bigrams	Full	overlap×10	0, 10, 20, ...
Question words	Full	fixed	0, 50, 100, ...

Table 3.1: Scores for heuristics

in our system, but in contrast to "the Hybrid system" decided to limit our system to heuristics that only relied on the surface structure of the text, with no external sources, for simplicity.

3.4.1 Scoring

The score for each answer candidate is calculated by summing up the scores from each of the heuristics presented below. The candidate with the highest score is chosen as the system's suggested answer. If two or more answer candidates have the same score, the candidate appearing highest in the list, i.e. the candidate with the lowest index is chosen. The weight of the scores were initially based on intuition, then adjusted after manually analysing the results, giving the heuristics that seemed to best predict correct answers more weight than less reliable heuristics.

3.4.2 Overlap-based heuristics

The overlap-based heuristics measure the overlap between sets of words by taking the length of the intersection of the sets. With the use of sets, the ordering of words are disregarded, making this a bag-of-words model.

During development, a normalisation step was added where the overlap was divided by the length of the answer candidate, to make sure that longer candidates were not given an unfair advantage. However, this normalisation consistently lowered performance, and is not part of the final baseline system.

The idea behind these heuristics is that sentences sharing the same words should have largely the same meaning. In practice, this is not always true for a number of reasons. For example, negation words can completely change the meaning of a sentence, but occurrence of a word as "*not*" is not seen as any more significant than any other word. Another potential problem is that the use of synonyms between text and question/answer is not detected, only identical words are counted. As these types of heuristics are both fast and simple to implement and run, we decided to use them in our system despite their deficiencies.

Q: What is Nelson Mandela's country of origin?
A: South Africa
Overlap between question and answer: **0**

Figure 3.5: Overlap between question and answer

Q: What is Nelson Mandela's country of origin?
A: *South Africa*
*But do they all know about what has been taking place in **South Africa**, his country, the country that had one of the highest incidents of transmission of the virus?*
Overlap: $1 \times 2 = \mathbf{2}$

Figure 3.6: Overlap between answer and sentence

Overlap between question and answer

This heuristic measures overlap between the question and the answer candidate. The text is not taken into consideration, giving this heuristic limited credibility. An example is shown in figure 3.5. The concise nature of the answer candidates means that there will be a lot of cases with no overlap between question candidate and answer.

Overlap between answer and text

This heuristic measures the overlap between the answer candidate and sentences in the text. As seen in figure 3.6, this heuristic has clear potential to be helpful. An issue with this heuristic is that the question is not taken into account, the sentence containing the answer could might as well have been describing another person.

Combining overlap between question, answer and sentence

A score is given in this heuristic by multiplying the overlap between the question and the sentence, and the answer candidate and the sentence. The idea is that if a sentence overlaps with both question and answer, it is likely that the sentence entails the answer.

Q: What is Nelson Mandela's *country* of origin?
A: *South Africa*
*But do they all know about what has been taking place in **South Africa**, his **country**, the country that had one of the highest incidents of transmission of the virus?*
Overlap: **2**

Figure 3.7: Overlap between question, answer and sentence

Q: ***What is*** Nelson Mandela's country of origin?
A: South Africa
`query_pre:` south africa *is* nelson mandela's country of origin
`query_post:` nelson mandela's country of origin *is* south africa

Q: ***Who is*** the founder of the SING campaign?
A: Annie Lennox
`query_pre:` Annie Lennox *is* the founder of the SING campaign
`query_post:` the founder of the SING campaign *is* Annie Lennox

Figure 3.8: Creating queries based on answer-slots

An example of the outcome of this heuristic is shown in 3.7. With this heuristic, the sentence must have something in common with both the question and the answer candidate. We assume this heuristic will be a solid indicator for correct answers, and its scores are given more weight than the previous heuristics.

3.4.3 Bigram overlap

Bigrams are pairs of adjacent words. We created a generic bigram extractor, which takes a sentence, and returns a set of bigrams. Using this strategy, word order becomes highly relevant. This heuristic measures the overlap between the set of bigrams from the answer candidate and the bigrams from the text. The question is not considered in this heuristic.

3.4.4 Heuristics for question words

This heuristic constructs a phrase combining the question and answer in two different ways, `query_pre` and `query_post`. Both use a set of keywords as an indicator of where to split the query. The keywords are inflected forms of *is* and *do*. The first combination places the answer candidate first, followed by the keyword, then the rest of the question sentence. The second combination places the question sentence excluding the keyword first, followed by the keyword, then the answer candidate.

If either one of these sentences are found in the text, it is considered strong evidence of a correct answer candidate, and they are given more weight than the other heuristics in the total score. Some examples are shown in 3.8.

3.5 Normalisation

We added an option to the system which performs normalisation of all text. The normalisation works by removing stopwords from a predefined list, and performing lemmatizing of the remaining words. Stopwords are function

words such as “the”, “is” etc., that are not meaningful in themselves. The lemmatizing is performed by using NLTK’s `WordNetLemmatizer`, which seeks to restore words to their dictionary form by removing inflectional endings.[1] In the results listed in the next section, we show the accuracy achieved with and without normalisation.

3.6 Baseline results

Reading test	Accuracy	w/norm.
AIDS		
1	0.3	0.4
2	0.2	0.2
3	0.2	0.2
Average	**0.233**	**0.267**
Climate Change		
5	0.2	0.3
6	0.2	0.3
7	0.4	0.2
Average	**0.267**	**0.267**
Music and Society		
9	0.4	0.5
10	0.5	0.3
11	0.3	0.4
Average	**0.4**	**0.4**
Total average	**0.3**	**3.111**

Table 3.2: Accuracy of the baseline system, first without and then with normalisation

The system is tested a development set consisting of nine reading tests, three from each of the three topics, leaving one reading test from each topic for unseen testing. As there are ten questions in each reading test, an accuracy of 0.2 means that two answers were answered correctly. The results are listed in table 3.2. As can be seen, the accuracy on each topic varies, with *Music and Society* having the highest average accuracy at 0.4. Between all the tests, individual accuracies range from 0.2 to 0.5. The accuracies when using normalisation are slightly higher than without, but as the net difference is a change of 1 question in 90, it’s not necessarily significant.

As there are five answer candidates where one is always correct, the random baseline accuracy is 0.2. 19 of 43 runs submitted to QA4MRE at CLEF 2011 for English were below this random baseline, so the fact that our system consistently gets accuracies of 0.2 and above for all reading tests is promising.

1. `https://wordnet.princeton.edu/wordnet/man/morphy.7WN.html`

3.7 Summary

In this chapter, we described how the baseline system was built, including preparations made to the data, and the heuristics used in ranking the answer candidates. With the implementation of this baseline system, we can measure changes in performance when adding more linguistically motivated heuristics. The performance of the baseline system is not especially strong on the development data, but it consistently performs at or above the random baseline. In the next chapter, the strengths and weaknesses of this system are described, and new heuristics implemented, with the hopes that our system can answer more questions than it does in its current configuration.

Chapter 4

Experiments with syntax

In this chapter, we carry out an error analysis on the results from running the baseline system, in an attempt to find out what the weaknesses of the system are, and where adding information about syntax might help increase the accuracy. We then introduce the heuristics that use syntax, and present the new version of our system. Finally, we run the system, and describe how the accuracy changes.

4.1 Error analysis on baseline system

While the number of questions our system could and could not answer tells us something, it is far from the full story. To get more insight into what worked, we ran the system in a configuration that detailed the score each heuristic gives to each answer candidate, allowing us to see which heuristics were the most and least useful. This analysis was done manually, and we tried to detect patterns in either the questions or answers to understand the underlying problem.

4.1.1 What didn't work

One of the most surprising discoveries is that the heuristic that constructed queries based on answer-slots did not get any matches. The queries generated by the system seemed well-formed, but did not match any part of the text. The organizers of the task stated that they were moving away from simple questions and answers, which is a likely explanation why this heuristic didn't trigger.

A second surprising finding is that the heuristic that measured the overlap between each sentence and the answer candidate on several occasions sabotaged the results, and did not help once.

Table 4.1 contains the accuracy of the different question types. Question types are classified manually by looking at the question words used in the sentence. Further changes could be made to this classification scheme, e.g. placing questions beginning with "In what year" in `when` rather than `what`. As the goal of the categorisation was to gain a quick overview of the question types present, these rough categories were sufficient. As can

Category	Q in category	# correct	% correct
What	45	20	44.44
Why	14	4	28.57
How	10	4	40.00
Which	6	1	16.67
Name	5	1	20.00
Where	4	0	0.00
Who	3	2	66.67
When	2	1	50.00
Do	1	0	0.00

Table 4.1: Performance on question types

Q: What kind of energy would help to reduce CO2 emissions?
1: nuclear
2: geothermal
3: hydraulic
4: electric
5: kinetic

Q: What planet in the solar system has a size similar to our planet?
1: Earth
2: Venus
3: Mercury
4: Saturn
5: Mars

Figure 4.1: Example of what-question with answer candidates.

be seen in the table, there are 45 questions in the `what` category, but the pattern-based heuristic did not trigger once. These questions are mostly very straight-forward, and should intuitively not be too difficult to answer. Further analysis is needed to understand why our system did not handle these questions as well as we expected. Examples of two different what-questions are shown in figure 4.1, the correct answers are emphasized with bold.

The first question is one the system clearly solved, with three heuristics agreeing on the first candidate, giving it a total score of 12, and all other candidates 0. The second example was less successful. The first three answer candidates were found in the text, giving each of them a score of 1, and no other heuristics were triggered. As the scores were tied, the first of the tied candidates were chosen, candidate 1, which is not the correct answer.

4.1.2 What worked

Some heuristics proved to be especially useful, and were good indicators of correct answers.The heuristic that measured combined overlap of question, answer and sentence proved to be the most useful heuristic, confirming our initial intuition of giving it a significant weight. What detracts from the performance of this heuristic is its tendency to give the same score to several alternatives, leaving our system to choose the answer candidate appearing first in the list.

Close behind combined overlap was word-bigram-overlap, which seemed to offer some help in those cases where the combined overlap gave the same scores. The word-bigram-overlap heuristic tended to agree with the combined overlap, but there were some instances of it sabotaging what would otherwise have been a correct answer.

4.1.3 Tied scores

As mentioned earlier, if two or more answers share the same highest score, the one appearing first in the list is chosen. Of the 90 questions in the test set, the highest score was shared in 13 of them. Of these 13, we saw the following numbers:

- The correct answer was amongst the candidates with the highest scores, but was not chosen: **4 questions**.
- The correct answer was chosen: **4 questions**.
- The correct answer was NOT amongst the candidates with the highest scores: **5 questions**.

These numbers indicate that our system performs poorly when it comes to discriminating between candidates. Half the time the system assigns the highest score to multiple candidates including the correct answer, the correct answer is chosen.

4.1.4 Summary

We feel this is a reasonable system to use as a baseline, where only simple shallow heuristics are used. Our error analysis has highlighted a few problems, but these seem to be because of the data rather than the system itself. It seems that the data is not well suited to these shallow heuristics, as questions are possibly phrased in different ways on purpose, to discourage this type of system. Based on this error analysis, using heuristics motivated by deeper linguistics seems like a natural next step. As several of the systems mentioned in the background section used heuristics based on output from a dependency parser, we will try this approach next.

4.2 Adding dependency heuristics

What we hope to achieve by adding dependency heuristics is to make our system more robust and able to handle more variation in surface form of questions and answers. Previously, all the information we had about the

content of a sentence pertained to information about the words themselves, and their position in the sentence. What we lacked was information about how the words relate to each other, how the sentence is structured.

Several of the systems participating in the task used dependency parsers in different ways, using the information about relations between words to improve their systems. While the retrieval-based system (Verberne 2011) used the relations to create fact-expansions, the hybrid system (Pakray et al. 2011) used the relations more directly, comparing the words with similar relations between question, answer and text. These relations will give us some insight into the structure of the sentence, and allow the system to see connections between words beyond their order.

4.2.1 Dependency parser

We need to use a dependency parser to get the dependency relations. There are several parsers available online, of these we choose to use MaltParser (Nivre, Hall, and Nilsson 2006), a language-independent data-driven dependency parser, for multiple reasons. One strong reason is that it has a pretrained model for English supplied. This model has been trained on 4000 questions from QuestionBank (Judge, Cahill, and Van Genabith 2006) and sections of the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993). Having a parser trained on questions is an advantage, as parsers perform best on text resembling that on which they have been trained. A second reason for using MaltParser is that it is freely available for research purposes

The pretrained model we use is called `engmalt.poly-1.7.mco`,[1] and the output when using this model is in the form of Stanford typed dependencies, the dependency scheme introduced in chapter 2. Before the data can be sent to MaltParser it must be converted to a given format, the process of which will be described in the next section.

4.2.2 Converting the data

MaltParser supports multiple data formats out of the box, of these we chose to use the CoNLL-format,[2] as it is simple to both create and parse, and contains sufficient information for our system. In this format, words are on separate lines, and sentences are separated by empty newlines. Each line consists of 10 columns separated by a single tab character, an underscore is used where no information is available.

We created a converter program that takes the XML-file as input, and returns two CoNLL-files for each reading test - one with the text, and one with the questions and answers. To adhere to the CoNLL-format, multiple processes are applied to the text. First, to get the data from the XML file, the Python library `xml.etree.ElementTree` was used. The text then had to be split in to sentences, which was done using NLTK's `sent_tokenize` in the same way as described in the previous chapter. The sentence must

1. `http://www.maltparser.org/mco/english_parser/engmalt.html`
2. `http://nextens.uvt.nl/depparse-wiki/DataFormat`

then be split into words, and part-of-speech-tags must be added. We used `word_tokenize` and `pos_tag`, both from NLTK, for this part of the process. Finally, lemmas were added, again using `WordNetLemmatizer` from NLTK. With all the information in place, the data was written to files, adhering to the CoNLL-format.

The columns in the CoNLL-format contain the following information:

1 Contains the position of the word within the sentence. The numbering starts at 1, as 0 is reserved as the id of the *root node*, used to identify the head of the sentence.

2 Holds the word form after tokenisation of the sentence.

3 Contains the lemma.

4 Holds the coarse-grained part-of-speech tag.

5 Holds the fine-grained part-of-speech tag. In our dataset, it is the same as the coarse-grained tag.

6 Holds features, not relevant for our purposes.

7 The first column where the parser makes changes. This column holds the ID of the head of the current token, or 0 if the token has no head.

8 Contains the dependency relation to the head, or `null` if the token is the root of the sentence.

There are two more columns, but these are not used by MaltParser and thus not relevant for this thesis.

These generated files are then sent to MaltParser, which adds dependency relations. An example output of the converter program, and the input to MaltParser is shown in 4.2. MaltParser's output for the same sentence is shown in 4.3.

A visual representation of the graph is shown in figure 4.4. The part-of-speech tags are shown beneath the words, in capital letters.

4.2.3 Dependency heuristics

In the following sections, we list the different heuristics and how they impact the scoring. The types of comparisons were inspired by the previously mentioned Hybrid system (Pakray et al. 2011). These heuristics can be used as an addition to the heuristics in the baseline, or by themselves, depending on the chosen configuration.

For ease of access of the information gained from the dependency parse, we use the package `DependencyGraph` from NLTK (Bird, Klein, and Loper 2009). This package takes a parsed sentence in CoNLL-format as input, and returns an object representation of the graph. We expanded the package to support lemmas as well as words. Lemmas are the dictionary form of a

```
1       Bell        Bell        NNP     NNP     _
2       ,           ,           ,       ,       _
3       a           a           DT      DT      _
4       company     company     NN      NN      _
5       which       which       WDT     WDT     _
6       is          is          VBZ     VBZ     _
7       based       based       VBN     VBN     _
8       in          in          IN      IN      _
9       LA          LA          NNP     NNP     _
10      ,           ,           ,       ,       _
11      makes       make        VBZ     VBZ     _
12      and         and         CC      CC      _
13      distributes distributes VBZ     VBZ     _
14      computer    computer    NN      NN      _
15      products    product     NNS     NNS     _
16      .           .           .       .       _
```

Figure 4.2: Example of CoNLL-file, input to MaltParser

```
1       Bell        Bell        NNP     NNP     _   11   nsubj      _   _
2       ,           ,           ,       ,       _   1    punct      _   _
3       a           a           DT      DT      _   4    det        _   _
4       company     company     NN      NN      _   1    appos      _   _
5       which       which       WDT     WDT     _   7    nsubjpass  _   _
6       is          is          VBZ     VBZ     _   7    auxpass    _   _
7       based       based       VBN     VBN     _   4    rcmod      _   _
8       in          in          IN      IN      _   7    prep       _   _
9       LA          LA          NNP     NNP     _   8    pobj       _   _
10      ,           ,           ,       ,       _   1    punct      _   _
11      makes       make        VBZ     VBZ     _   0    root       _   _
12      and         and         CC      CC      _   11   cc         _   _
13      distributes distributes VBZ     VBZ     _   11   conj       _   _
14      computer    computer    NN      NN      _   15   nn         _   _
15      products    product     NNS     NNS     _   11   dobj       _   _
16      .           .           .       .       _   11   punct      _   _
```

Figure 4.3: Output from MaltParser

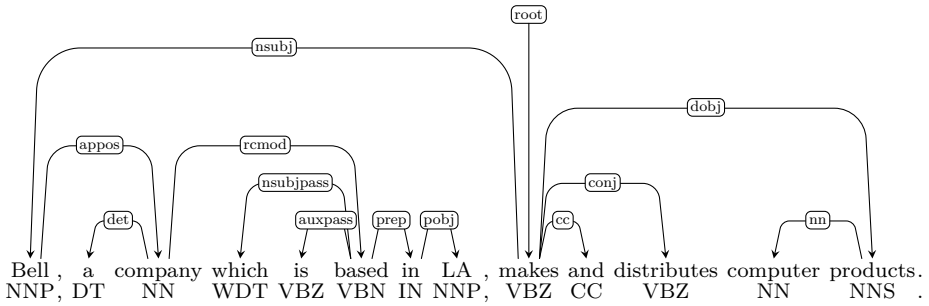

Figure 4.4: Dependency structure visualized as a graph

word, stripped of any inflections.

All the heuristics are focused on comparing the answer candidate with sentences from the text. Each answer candidate is compared to each sentence in the text in turn, and the comparison with the highest score is kept for the answer candidate.

Subject-verb comparison

This heuristic is the one we think will be most useful — it compares the lemmas of the subject-verb pairs, identified by the `nsubj` (nominal subject) and relations for subjects and part-of-speech tags beginning with `VB` for the verbs. Because of the focus on using content words in the Stanford typed dependencies representation, we assume this heuristic will be especially useful. Looking at the example graph in figure 4.4, this heuristic would highlight two words, **Bell** and **makes**. If an answer contains these same words, with the same relation, it is likely that the content is similar, and the answer candidate will be given a score of 40 points.

Object-verb comparison

Similarly to the previous heuristic, this compares the object-verb pairs. Objects are identified by relations containing `obj`, eg. `dobj` (direct object), `iobj` (indirect object). This relation corresponds to that between **makes** and **products** in figure 4.4. This heuristic is assumed to be slightly less informative than the subject-verb comparison, and matches give a lower score, 20 points.

Subject-object-verb comparison

While we do not expect this heuristic to trigger, it awards a high score in case it does. It compares the subject, object and verb of each sentence with the same triplet in the answer candidate. Looking at the example sentence again, **Bell makes products** would be the words that were compared to those of the answer candidate. In the case of a match, this heuristic awards 60 points.

4.2.4 Results

As seen in table 4.2, the overall accuracy drops with the addition of the new heuristics. In total, the system is able to answer two less questions than without the dependency heuristics.

Looking more closely at the results, we see that the heuristic comparing subject-verb pairs in the text and answer candidates gave a score in 3 cases, while the object-verb comparison scored 8 answer candidates. The last heuristic did not trigger once. Of the two heuristics that did work, both gave points to the right candidate once, and gave points to the wrong candidate the rest of the time. Both dependency heuristics helped on the same answer candidate, but this candidate was already chosen as the highest ranked using the baseline heuristics. Twice the heuristics turned a correct answer from

Reading test	Baseline Accuracy	Accuracy
AIDS		
1	0.4	0.3
2	0.2	0.2
3	0.2	0.2
Average	**0.267**	**0.233**
Climate Change		
5	0.3	0.3
6	0.3	0.3
7	0.2	0.2
Average	**0.267**	**0.267**
Music and Society		
9	0.5	0.5
10	0.3	0.2
11	0.4	0.4
Average	**0.4**	**0.367**
Total average	**0.311**	**0.289**

Table 4.2: Results after adding dependency-based heuristics

the baseline scores to an incorrect one, by giving points to the wrong answer candidate, changing the outcome. In figure 4.5, an example is shown of a question where the points from a dependency heuristic changed an answer from correct to incorrect. Answer candidate 3 is correct, but candidate 1 was chosen because of the 40 points gathered from having a matching subject-verb-pair with a sentence in the text. While the subject-verb pair occurs both in a sentence in the text, the sentence does not contain the answer, and the combination of subject and verb is not very meaningful or unique. As can be seen from the other subject-verb-pairs that were extracted from the sentence, they are overall less informative than expected.

While we did not expect the heuristics to support the correct answer candidate every time, we had hopes that their effect would be overall positive, and make a bigger impact than they did. Allowing the heuristic to award as many as 40 points seems excessive, and fine-tuning of the scores might have given different results. However, a total of 450 answer candidates were evaluated (90 questions with 5 candidates each, and ff these, only 11 had matches in one of the dependency heuristic, totalling 2.44%. As the heuristics affected such a low number of answer candidates, we decided to spend our time examining the data rather than adjusting the scores.

It seems like using syntax directly is not useful on this dataset, leading us to wonder what our system must be able to do in order to see bigger improvements in accuracy. To answer this question, we look more closely at the supplied data in the next chapter.

Q: Why is it dangerous to use a dirty needle?
Excerpt from text: *"And if you put your public health nerd glass on, you'll see that if we give people the information that they need about what's good for them and what's bad for them, if you give them the service that they can use to act on that information, and a little bit of motivation, people will make rational decision and live long and healthy life."*
Subject-verb-pairs: (you, put), (we, give), (they, use), (you, give), (you, see), (they, need)

Answer candidate 1: because you can be put in jail
Subject-verb-pairs: (you, put)
Points from baseline: 20.25
Points from dependency heuristics 40
Total points: 60.25

Answer candidate 3: because you can get infected by hiv
Points from baseline: 20.43
Total points: 20.43

Figure 4.5: Dependency heuristic sabotaging results

Chapter 5

Data analysis

In this chapter, we take a closer look at the dataset supplied for the task our system attempts to solve. The goal is to gain an understanding of what the challenges of the dataset are, in the hopes of discovering why our attempts at adding heuristics based on syntax were not successful. We begin by describing the basic numbers of the data, before introducing a manual classification scheme, detailing the criteria for each of the classes. The result of this classification is then presented, alongside a discussion of the results and a plan for improving the system.

5.1 Introduction to the data

We use the English dataset supplied for QA4MRE at CLEF 2011. A breakdown of the numbers of texts, questions and answers can be seen in table 5.1. This means we have a total of 12 texts to work with. Of these, a test set consisting of one text from each of the topics is set aside (reading tests 4, 8 and 12), and the remaining texts from each category are used for development. This is done to ensure that we are not overfitting for the data, but still expose our system to phenomena that might be specific for each topic. The texts are transcripts of TED-talks, which can add a layer of complexity, as spoken language often has a less strict sense of grammar and structure than written language. The questions can stand alone, not requiring any information from previous questions. The answer candidates are all succinct, containing only the words required to answer the question, and seldom form complete sentences. An example of one of the

Topics	3
Texts per topic	4
Questions per text	10
Answer candidates per question	5
Average text length in words	2292

Table 5.1: Numbers of topics, questions and answers in the test data

Excerpt from reading test 9: *"Now, for those of you who don't know, a Rube Goldberg machine is a complicated contraption, an incredibly over-engineered piece of machinery that accomplishes a relatively simple task."*
Question 5: What is a Rube Goldberg machine?
Answer candidates:
1: a simple system for performing a complicated operation
2: a song which picks up complex emotion
3: a complicated device for performing a simple operation
4: any machine containing gold
5: something watched on YouTube

Figure 5.1: Example of text, question and answer

more straightforward questions is shown in figure 5.1, the correct answer candidate is marked with bold. While the connection between the given excerpt from the text, the question and the answer candidate is very clear to a human reader, an automated system faces a number of challenges. First, there is the use of synonyms. Note for example that the text uses both the words *contraption* and *machinery* to describe the Rube Goldberg machine, while the answer candidate uses the word *device*. Second, the first answer candidate contains almost the same words as the third and correct candidate, only with a slight change in order. The order in which the words appear is important to be able to distinguish between the answer candidates. A third problem is actually finding this relevant sentence in the text. Earlier in the text, the topic of the second answer candidate is discussed, meaning heuristics that disregard the question will likely give this candidate a high score.

This way of analysing questions can be of help to refine our heuristics, and in the next section we will apply a classification scheme to formalise the analysis of all the questions.

5.2 Classifying the questions

The questions are classified by the type of information needed to select the correct answer candidate based on the supplied text. Questions can belong to multiple classes. The focus while classifying has been to uncover what is needed for complete understanding of both questions and answers. As five answer candidates are given, and one is guaranteed to be correct, complete understanding is not actually needed to solve this task. However, it serves as a useful baseline to ensure consistency for the classification. The analysis is performed on the 9 texts in the development set, totalling a number of 90 questions.

In the following sections, we describe the different classes and the requirements for each class.

Q: What African country did Bono Vox visit?
A: Abyssinia
*"We lived in **Ethiopia** for a month, working at an orphanage"*

Figure 5.2: Question requiring background knowledge

Q: Why is Avelile suffering from AIDS?
A: because her mother transmitted the virus to her
"Avelile's mother had HIV virus. She died from HIV related illness. Avelile had the virus. She was born with the virus."

Figure 5.3: Question requiring textual entailment

5.2.1 Background knowledge

A question is defined as belonging to this class if it requires knowledge that is more extensive than syntactic alternations, and that information is not present in the text. An example can be seen in figure 5.2, where the fact that Ethiopia was historically known as Abyssinia is not given in the text, but is a fact that can be extracted from a knowledge source like an encyclopaedia. In some cases, the distinction between whether a question background knowledge or synonymy/hyponymy was unclear. A general rule was established that questions relating to named entities that required further information were classified as needing background knowledge.

5.2.2 Inference

If a question requires a combination of facts from multiple sentences, it is classified as requiring inference. These facts build upon each other and must be registered together for the answer to become clear. In the example in figure 5.3, the combination of facts from three of the four sentences in the excerpt is required to correctly answer the question. The second sentence is not necessary to get to the answer, but is kept in to keep the context of the excerpt. Sentences containing the needed facts are usually consecutive, but as the example shows there are exceptions.

5.2.3 Anaphora

Anaphora are expressions referring to previously introduced entities, often by using personal pronouns. Questions belonging to this category requires the system to make the connection between the anaphora and the antecedent it refers to. We have limited this category to anaphora that are personal pronouns.

This category contains the subcategory **first person**, as there are relatively simple algorithms to resolve these for our data. In figure 5.4, two cases of the third-person plural personal pronoun *they* can be seen (emphasis added), which must be resolved to *OK Go* for the system to answer the question.

Q: What did OK Go do previously in a video?
A: they danced on treadmills
"Now, when we first started talking to OK Go – the name of the song is 'This Too Shall Pass' – we were really excited because ***they*** *expressed interest in building a machine that they could dance with. And we were very excited about this because, of course,* ***they*** *have a history of dancing with machines."*

Figure 5.4: Questions requiring resolving anaphora

Q: What is Paul David Hewson's occupation?
A: rock star
"And though ***I****'m a rock star, I just want to assure you that none of my wishes will include a hot tub."*

Figure 5.5: First person pronoun anaphora

First person

This subcategory of anaphora consists of questions requiring first person personal pronouns to be resolved. For most of the tests in the dataset, first person personal pronouns can be replaced with the name of the author of the speech / text. A typical example can be seen in figure 5.5, where Paul David Hewson – under the alias of Bono Vox – is the person giving the speech.

5.2.4 Lexical semantics

Lexical semantics is an area of linguistics concerned with the mapping between the meaning and the form of a word (Miller 1986). Question are placed in this class if recognizing how the meaning of two words relate to each other is required to answer the question. We have focused on two categories within lexical semantics, namely synonymy and hyponymy, which we describe in the next subsections.

Synonyms

Synonyms are defined as different words with the same or very similar meanings. For a question to fit in this category, one or more of the words in the combination of question, answer and supporting sentence in the text required to understand the content must be synonyms. A system can use this knowledge to match synonyms as it would match identical words. An example of a synonym is shown in figure 5.6, where "contraption" and "device" mean the same thing.

Q: What is a Rube Goldberg machine?
A: a complicated **device** for performing a simple operation
"Now, for those of you who don't know, a Rube Goldberg machine is a complicated ***contraption****, an incredibly over-engineered piece of machinery that accomplishes a relatively simple task. "*

Figure 5.6: Synonyms in answer candidate and supporting text

Q: What did **Nelson Mandela** say at the press conference?
A: thousands of people were being wiped out by AIDS
"In that moment in time, ***Mandela*** *told the world's press that there was a virtual genocide taking place in his country, that post-apartheid Rainbow Nation, a thousand were dying on a daily basis, and that the front line victims, the most vulnerable of all, were women and children."*

Figure 5.7: Co-reference in the dataset.

Hyponyms

Hyponyms describe an "is-a" relation between pairs of words. Questions in this category have at least one such pair present between question, answer candidate or text. A simple example of this can be shown by looking at types of trees — a **birch** *is a* **tree**. This information can be gained from lexical resources, such as WordNet.

Co-reference

The term co-reference describes a situation where two or more different expressions are used to refer to the same entity. Questions are classified as belonging to this class if they require resolving of co-reference in order to be answered. For the purposes of this classification, anaphora are in a separate class, rather than grouped with co-reference, as there are simpler algorithms available to resolve anaphora. In figure 5.7, a typical example is shown, where both **Nelson Mandela** and **Mandela** refer to the same person.

5.2.5 "Impossible" questions

Questions are classified as impossible if the type of knowledge needed to answer them is near impossible to gain from the text or simple knowledge sources. An example of this is shown in figure 5.8, where the problem is that to answer the question, the system must be aware of the presence of an audience, and understand their act of standing up based on the utterance by the speaker. As human readers, we can understand this sort of inference, but it still feels like a stretch to interpret "people" in the sentence as referring to only the audience, as well as understanding the act that causes the response,

and what it means. In figure 5.9 there is one issue that can be solved through background knowledge, namely the problem of the different representations of the name of Sir Bob Geldof. The main problem is linking him as a founder of "Band Aid", as this is mainly referenced by the phrase "issued a challenge to 'feed the world"', an excerpt from the lyrics of one of the songs mentioned in the text. As human reader without previous knowledge of these initiatives would struggle with making the connection, and it seems improbable that our system would be able to correctly answer this question based on its understanding.

5.2.6 Comments on the classes

For inspiration on what classes to use, we looked at a paper classifying the Text-Hypothesis pairs for the PASCAL Recognizing Textual Entailment Challenge (Vanderwende and Dolan 2006). The paper focuses on classifying which pairs can be solved using a syntactic parser, and which phenomena that parser must handle. The authors describe the classifying as performed by two skilled linguists, and list a large number of classes. As our dataset is smaller, and the classification is performed by one person, we chose to use only one of the classes suggested, and added some of our own, motivated by the issues we saw in the data. The criteria for classification were not listed in the paper, only the names of the phenomena, but seeing what phenomena they focused on did help us narrow down what to look for when examining the data. Of the classes listed in the paper *anaphora* were added to our scheme. This was chosen because we had seen several examples of the phenomena in our data, and we had a good idea of how to classify them. *Named Entity Recognition* was considered as well, but a combination of problems with consistency in classifying as well as the early decision not to use external resources lead to the exclusion of the class.

5.3 Applying the classification

The classification was done manually, looking at each question and the corresponding passage in the text. Only the correct answer candidates were considered, to limit the amount of time spent on classification. This means that any benefits from eliminating of incorrect answer candidates has not been considered in this classification.

5.3.1 Results of classification

The result of the classification is shown in table 5.2. A total of 43 questions are probably out of scope for our system to fully understand, although it still be able to answer them correctly. This number is surprisingly high, but is likely a direct consequence of the concious effort of the organizers to make questions more difficult to answer by pattern-matching.

The class with the most questions is inference, with 29 questions. Questions requiring anaphora resolution come in second, with 26 questions. It is worth noting that most questions belong to more than one class, so

Q: Do people agree that governments should be committed to fighting AIDS?
1: definitely yes
2: definitely no
3: unknown
4: sometimes
5: only one person agrees
"So, I would like to say to you, each one in the audience, if you feel that every mother and every child in the world has the right to have access to good nutrition and good medical care, and you believe that the Millennium Development Goals, specifically five and six, should be absolutely committed to by all governments around the world – especially in sub-Saharan Africa – could you please stand up. I think that's fair to say, it's almost everyone in the hall.Thank you very much."

Figure 5.8: A question marked as "impossible" due to required world knowledge

Q: Name a founder of Band Aid.
A: Robert Frederick Zenon Geldof
"You may remember that song, 'We Are the World,' or, 'Do They Know It's Christmas?' Band Aid, Live Aid. Another very tall, grizzled rock star, my friend Sir Bob Geldof, issued a challenge to 'feed the world."'

Figure 5.9: A question marked as "impossible" due to required intricate inference

Total number of questions analysed	90
Deemed impossible	43
Questions in class	
Inference	29
Anaphora resolution	26
Background knowledge	15
Hyponyms	14
Co-reference resolution	10
1st person anaphora resolution	9
Synonyms	9
Questions belonging to multiple classes	
Only one class	7
Two classes	14
Three classes	15
Four classes	8
Five classes	2
Six classes	1

Table 5.2: Classification of the questions from QA4MRE at CLEF 2011

resolving any one of these problems might not be enough. Two classes were deemed especially hard to solve, namely background knowledge and inference. Only 11 questions did not belong to either of these classes.

5.3.2 Using the results

With these results, we have an idea of what phenomena our system should handle in order to answer more questions correctly. Inference makes up the class with the most questions, but solving inference is a challenging task that is outside the scope of this thesis. Instead, we settle for the second largest class, questions requiring anaphora resolution.

Chapter 6

Experiments with anaphora resolution

The problem of ambiguity is present in many aspects of natural language processing, from the ambiguity of words, to the ambiguity of references. The word *anaphora* denotes the combination of the antecedent and the anaphor. The problem of anaphora resolution is finding the antecedent to the anaphor. As highlighted by Hobbs, humour and confusion can arise when there is ambiguity of antecedents: "There's a pile of inflammable trash next to your car. You'll have to get rid of it.", where *it* can refer both to *the car* and *pile of inflammable trash*. While this distinction is simple to make for a human being, it's not quite as simple to create rules to automatically choose the right interpretation.

In this chapter, we will describe an approach for resolving anaphora, as the phenomenon is very common in the dataset used in this thesis. Of 90 questions, 29 can benefit from anaphora resolution. We begin by introducing one fairly simple algorithm for resolving anaphora using phrase structure trees, before briefly presenting some alternative strategies. The approach used in this thesis is then described, including detailing a modification of the first algorithm described, for use with dependency strategies. The results of running the system on the data after anaphora resolution are then discussed, before testing on the held-out data is performed, and finally these results are commented on.

6.1 The Hobbs algorithm

In his famous paper from 1978, Hobbs presents two approaches to the problem of resolving pronoun references (Hobbs 1978). The first is a naive algorithm using phrase structure trees, while the second is geared towards systems for semantic analysis. As we focus on syntax in our system, we will make use of the first algorithm. See algorithm 1 for the full algorithm.

To summarise, it looks for antecedents starting with the children of the nearest NP- or S-NODE, to the left of the path leading from the pronoun to the node. If no antecedent is found, it looks for the next NP- or S-NODE, and repeats the process. Thus it searches in sub-trees right-to-left from the

1. Begin at the NP-NODE immediately dominating the pronoun.
2. Go up the tree to the first NP- or S-NODE encountered. Call this node X, and call the path used to reach it p.
3. Traverse all branches below node X to the left of path p in a left-to-right, breadth-first fashion. Propose as the antecedent any NP-NODE that is encountered which has an NP- or S-NODE between it and X.
4. If node X is the highest S-NODE in the sentence, traverse the phrase structure trees of previous sentences in the text in order of recency, the most recent first; each tree is traversed in a left-to-right, breadth-first manner, and when an NP-NODE is encountered, it is proposed as an antecedent. If X is not the highest S-NODE in the sentence, continue to **step 5**.
5. From node X go up the tree to the first NP- or S-NODE encountered. Call this new node X, and call the path traversed to reach it p.
6. If X is an NP-NODE and if the path p to X did not pass through the $\bar{n}$ NODE that X immediately dominates, propose X as the antecedent.
7. Traverse all branches below node X to the *left* of path p in a left-to-right, breadth-first manner. Propose any NP-NODE encountered as the antecedent.
8. If X is an S-NODE, traverse all branches of node X to the *right* of path p in a left-to-right, breadth-first manner, but do not go below any NP- or S-NODE encountered. Propose any NP-NODE encountered as antecedent.
9. Go to **step 4**.

Algorithm 1: Hobbs' naive algorithm

pronoun, but searches the trees themselves left-to-right.

If no antecedent is found in the same sentence, it moves on to the previous sentence, left-to-right, breadth first, proposing the first NP-NODE encountered as antecedent. Again, if no alternative is found, it moves to the previous sentence, repeating until an antecedent is found.

Note that further restrictions can be put on the candidate antecedents, deeming them eligible or not.

6.2 Alternative approaches

There are multiple alternatives to Hobbs algorithm exist, and several of them rely on finding possible anaphora, before filtering based on certain requirements. One of these is an algorithm by Lappin and Leass (Lappin and Leass 1994) for third person pronouns, known as RAP (Resolution of Anaphora Procedure). It handles the problem of pleonastic (semantically empty) pronouns, such as the *it* in *it is likely* and *it is thought* etc. It also considers agreement between anaphor and antecedent, and scores NPs on several salience parameters, forming a ranking of candidates.

A similar approach was used by a team from Stanford for the CoNLL-2011 shared task (Lee et al. 2011). The system uses a series of sieves, building upon each other, mainly placing high-recall components near the top, followed by high-precision recall components. First, the system gathers potential mentions, using a set of handwritten rules to exclude certain mentions, such as numeric entities, pleonastic *it* pronouns, and certain stop words. 12 additional sieves follow, resulting in a recall of 87.9% (with predicted annotations), leading the team to victory in the task. Precision is lower, but the team argues that it is expected, and compensated for in post-processing, when many of those mentions are discarded.

While these approaches to anaphora resolution are strong candidates, they require more information than the Hobbs algorithm in order to consider agreement and in the case of the latter requires handwritten rules. Based on this, we choose to focus on using the Hobbs algorithm.

6.3 Our strategy

For our system, we limit ourselves to resolving pronouns that refer to proper nouns. The anaphora resolution in our system is based on Hobbs algorithm, and is performed as a pre-processing step before sending the text to the QA system, replacing the anaphors with their antecedents.

6.3.1 First person pronouns

In a special case for this task, first person personal pronuns are all resolved to the author of the text. The name of author is assumed to be the first set proper nouns consecutively together in the first sentence. This change was made as texts all have a certain format, where the speaker does not change throughout the text.

6.3.2 Using the Hobbs algorithm with dependency structures

In this section, we will detail how we implemented the algorithm. For our purposes, the anaphora resolution consists of replacing the anaphor with the antecedent. If no suitable antecedent is found, the anaphor is kept as is. To keep the selection process simple, we focus on resolving *he*, *she*, *him*, *her* and *his* first, and only resolve if we find a *proper noun* candidate.

While the Hobbs algorithm originally requires the use of phrase structure trees, we found that with certain changes, an implementation using dependency structures instead is possible. Little work seems to have been done on using dependency structures for anaphora resolution at the time of writing this thesis.

A phrase structure tree consists of a set of internal nodes and terminal nodes. The internal nodes are grammatical categories or roles, and the terminal nodes are words. Phrase structure trees are generated from a Phrase Structure Grammar (PSG), which is hierarchical. A PSG contains rules for how each grammatical category can expand to other categories or words.[1]

When describing the changes made to the algorithm, we will briefly introduce the phrasal categories used, to explain how we found the equivalent in the dependency relations.

In the altered algorithm, we use part-of-speech tags to identify the various constituents, and the directional relations between the words to navigate through the tree. Moving *up* in the tree is done by moving from dependant to head, and moving down by moving from head to dependant.

The first step of the algorithm requires finding the NP-NODE immediately dominating the pronoun to have a starting point. NP stands for *Noun Phrase*, and in this context means a phrase headed by a noun. Dependency structures have no concept of phrases, our version of the algorithm skips this step, and begins at the pronoun itself.

The next step consists of moving up the tree until an NP- or S-NODE is encountered, denoting the path used p and the node X. An S-NODE is the category used for a complete sentence, and would thus form the starting point of a tree. We equate this with finite verbs in our interpretation, identified by `VB` (verb, base form), `VBD` (verb, past tense) and `VBZ` (verb, 3rd person singular present) part-of-speech tags. While the portion of a tree below a finite verb is not necessarily a full sentence, it will form the root of a clause. This step of the algorithm also asks that we keep track of the path used to get there. Rather than store the path, our version stores the WORD-ID of the previously visited node, i.e. the position of that word in the sentence. The path will be used as a constraint on what nodes to investigate, this constraint can be kept by comparing WORD-ID of possible node and p.

The rest of the steps mirror that of the original algorithm. The full algorithm is shown in algorithm 2.

1. For more information, see Jurafsky and Martin 2009

Filtering antecedents

While the algorithm has steps that include proposing antecedents as candidates, it does not describe how to determine which antecedents are good candidates and not. When resolving the listed pronouns, we determine that phrases headed proper nouns, tagged by `NNP`, should be chosen. We do not keep the entire tree below that word, only the word in question and any words with an `nn`-relation to that word, indicating a compound noun. The algorithm stops as soon as a valid antecedent is found.

Definitions:
Nominal - nouns, denoted by a part-of-speech tag containing `NN`.
Finite verb - finite verb tokens, denoted by `VB`, `VBD`, and `VBZ` part-of-speech tags.
p - position of previously visited node.

1. From the pronoun, go up the tree to the first NOMINAL or FINITE VERB encountered. Call this node *X*.
2. Traverse all branches below node *X* with a position to the left of *p* in a left-to-right, breadth first fashion. Propose as antecedent any NOMINAL node that is encountered which has a NOMINAL or FINITE VERB node between it and *X*.
3. If node *X* is the root of the sentence, traverse the dependency structures of the previous sentences in the text in order of recency, the most recent first; each structure is traversed in a left-to-right, breadth-first manner, and when a **nominal** node is encountered, it is proposed as an antecedent. If *X* is not the root of the sentence, continue to **step 4**.
4. The new node *p* is node *X*. From node *X* go up the tree to the first NOMINAL or FINITE VERB node encountered. Call this new node *X*.
5. If *X* is NOMINAL and the path did not pass through a NOMINAL node that *X* immediately dominates, propose *X* as a candidate.
6. Traverse all branches below node *X* with words to the left of *p* in a left-to-right breadth-first manner. Propose any NOMINAL node encountered as the antecedent.
7. Go to **step 3**.

Algorithm 2: The Hobbs algorithm for use on dependency structures

6.3.3 Results

When running the program resolving anaphora on the development set, 104 instances of the pronouns *he*, *she*, *his*, *him*, and *her*, were found, of which a total of 64 were resolved. Looking at the output, the program seems to do a reasonably good job of resolving pronouns, but the total count of these pronouns is lower than we expected.

RT	he/she		it		% resolved
	Count	Resolved	Count	Resolved	combined
1	31	17	13	12	65.91
2	8	7	67	57	85.33
3	33	20	47	42	77.50
5	4	4	15	13	89.47
6	11	2	29	26	70.00
7	6	4	95	92	95.05
9	2	2	34	32	94.29
10	5	5	4	4	100.00
11	11	3	50	41	82.00
Sum	111	64	354	319	Avg: 82.37

Table 6.1: Results on running the anaphora resolver per reading test

Using the heuristic for first-person personal pronouns resolving to the author name, all 399 instances of *I*, *me*, *my* and *mine* were resolved.

We examine the data to determine what other pronouns to attempt to resolve next. In table 6.2, the number of occurrences of pronouns throughout all 9 documents are listed.

As can be seen in the table, *we* is very commonly used, but upon inspection it is mostly used in the context of *"we as people"* or *"we as Americans"*, rather than referring to a previously explicitly introduced group. Similarly, *you* mostly refers to *"the listener"* or *"the audience"*, and can be left as it is.

Based on the ranking, the next anaphora to be included should be *it*. This brings an added challenge in restricting potential antecedents, as the system has so far been able to rely on proper nouns. For this pronoun, we decide to change the restriction to nouns, and include the part of the tree below the noun as part of the answer.

When adding adding *it* to the list pronouns we are already looking at, it brings us to a total of 868 occurrences that can potentially be changed, out of the 2303 pronouns in total. The result of adding *it* is shown in table 6.1. The column header RT is short for *Reading Test*, and the headers he/she also include numbers for *his*, *her* and *him*.

After looking at these numbers and implementing support for *it*, we got the counts shown in table 6.3. In total, 91% of the pronouns we looked for were resolved. When not counting the first person personal pronouns, all of which were resolved because of the approach used, the number of resolved pronouns drops to 83%. A few examples are shown in fig 6.1 of sentences before and after anaphora resolution. As can be seen, the replacements work in some cases, but not all. The results are not good sentences in English, but easier to understand for computers. In the third example, we see a problem with the algorithm. The issue likely stems from the fact that while pronouns

Pronoun	Count	Pronoun	Count
we	393	she	29
it	354	their	25
i	304	her	17
you	299	his	16
they	144	its	8
our	63	him	7
my	53	itself	4
us	50	themselves	4
he	42	myself	4
me	42	yourself	1
your	42	himself	1
them	31		
		Sum	2303

Table 6.2: Pronouns and their frequency in development set

Pronoun(s)	Count	Resolved	% Resolved
i/me/my/mine	399	399	100.00
it	354	319	90.11
he	42	26	61.90
she	29	11	37.93
her	17	6	35.29
his	16	15	93.75
him	7	6	85.71
Sum	864	782	90.51

Table 6.3: Number of resolved pronouns

Before: *I*'m going to share with you the story as to how *I* have become an HIV/AIDS campaigner.
After: *Annie Lennox* 'm going to share with you the story as to how *Annie Lennox* have become an HIV/AIDS campaigner.

Before: And 46664 is the number that Mandela had when *he* was imprisoned in Robben Island.
After: And 46664 is the number that Mandela had when *Mandela* was imprisoned in Robben Island.

Before: *It* all works perfect.
After: *the there, I would say, the instruments, the intricate rhythms, the way it's played, the setting, the context, it's all perfect* all works perfect.

Figure 6.1: Example of resolved anaphora

are limited to being resolved to proper nouns, *it* is currently resolved to be the entire part of the graph below the nominal the algorithm chooses. The intention was to be able to capture phrases like "the piano Mozart used", but we have not yet figured out a way to narrow down these results properly.

6.4 Running the system after anaphora resolution

Having verified that the anaphora resolution worked, we rerun our system to see if the changes in the text make a difference in performance. The results are shown in table 6.4, showing results from running the original baseline system, the baseline system with dependency-based heuristics added (denoted as "Syntax") and the baseline system on the data after anaphora resolution.

The results are positive, increasing the overall average of both configurations of the system. The number of correct answers did not change for the reading tests on the topic of climate change, likely due to the style of the texts. The texts on AIDS and music and society are more about personal experiences and people, making anaphora highly relevant, whereas the texts on climate change is less personal. While the size of the dataset makes it difficult to make claims with much certainty, it seems that using syntax to resolve anaphora can be of use to a question answering system.

6.5 Testing on held out data

Three reading tests have been held out from development, for the purposes of testing the finalised system. We ran the system in the same four configurations as in the previous section, to observe the overall performance, as well as the effect of adding dependency heuristics and anaphora resolution. As can be seen in figure 6.5, the results seem to reflect our experiences from the development data, where the addition of dependency heuristics does not

Reading test	Baseline	Syntax	Anaphora
AIDS			
1	0.4	0.3	0.4
2	0.2	0.2	0.3
3	0.2	0.2	0.3
Average	**0.267**	**0.233**	**0.367**
Climate Change			
5	0.3	0.3	0.3
6	0.3	0.3	0.3
7	0.2	0.2	0.2
Average	**0.267**	**0.267**	**0.267**
Music and Society			
9	0.5	0.5	0.5
10	0.3	0.2	0.5
11	0.4	0.4	0.5
Average	**0.4**	**0.367**	**0.5**
Total average	**0.311**	**0.289**	**0.367**

Table 6.4: Results after running anaphora resolution on text

make an impact, but there are slight improvements when adding anaphora resolution on the text.

One interesting observation was that these results did not follow the same patterns as the other reading tests, where the climate change category was the weakest, it is the strongest in this sample. A second observation is that the system performed worse on reading test 4, under the topic of AIDS, than it had previously done on any other reading tests. Out of curiosity, we ran the system in the other available configurations to see what accuracies were achieved on this reading test, and found a significant improvement when skipping the normalising step. Without normalising, the system had an accuracy for 0.5, as opposed to 0.1 with normalising. After closely examining the scores from the different heuristics, it seems that the normalising sabotages the otherwise good performance of the baseline heuristics for this reading tests. Three of the four correct answers it could only solve without normalising were clear-cut when it came to score, while the fourth was a lucky tie. This confirms our suspicion that there is large variability in the dataset, making fine-tuning more difficult.

6.6 Summary

After performing anaphora resolution on our dataset, we saw a slight increase in accuracy. This tendency was confirmed when running the system on the held out data. However, the held out data showed some trends that were at odds with what was seen on the development data. Performance on the

Reading test	Baseline	Syntax	Anaphora
AIDS			
4	0.1	0.1	0.1
Climate Change			
8	0.5	0.5	0.6
Music and Society			
12	0.4	0.4	0.4
Average	**0.333**	**0.333**	**0.367**

Table 6.5: Results of running system on held-out data

different topics was one of these observations, where climate change was the weakest topic overall with the development set, but is the one with the highest accuracy of the held out data. In general, it seems that the dataset is too small and variable to base any firm conclusions on, even though some tendencies could be observed.

Chapter 7

Conclusion

In this thesis, we set out to study the effects of using syntactic information to solve a given question answering task. This idea came from reading papers describing systems participating in the aforementioned task, where several authors listed lack of more linguistically motivated heuristics as a potential cause for their low performance. We were curious to investigate in what ways syntax could be used, and what the effects would be.

7.1 What we learned

When adding the dependency-based heuristics to the baseline system we created, we saw negative results. Scores of accuracy either went down or remained unchanged. Surprised by this finding, we analysed the data by performing a manual classification and found that it did not lend itself well to using dependency structures directly. The data analysis showed that there were some large classes of problems that — if resolved — were likely to boost the performance. One of these was anaphora resolution, so we modified a well-known algorithm for anaphora resolution to work on dependency structures, and saw positive results.

In the next section, we go into more detail on what we could have done differently in this thesis, in the light of hindsight.

7.2 Reflections

While we eventually saw some increase in performance when adding anaphora resolution, there are some points that should be addressed regarding the design of the system.

7.2.1 Using dependencies

We performed a dependency analysis of answer candidates in the hopes of matching the structures with those found in the text. However, the assumption this would be useful is likely to be flawed. Because of the format of the candidates, many answer candidates will simply be names, locations, or snippets of text that do not form complete sentences, and the likelihood

of erroneous parses is high. It is still possible that using the structures from the questions and sentences could be useful, perhaps by looking for a matches of a verb between the question and a sentence, and subject between the answer and the same sentence.

When looking at the generated dependency structures, they are in some cases very generic. While we did get useful information like `dobj(emission, reduce)`, we also got instances of `nsubj(it, is)` and `nsubj(you, put)`, which are less salient. To filter out these generic matches, some form of word weighting could be applied, to either one or both of the words in the relation, giving words which occur very frequently less weight.

7.2.2 Scoring

The size of the scores the different heuristics could give answer candidates was determined by the author's intuition, with slight adjustments made after looking at some of the results. This process could have been made more accurate, but the size of the dataset was a limitation. In the fear of overfitting for the data, we decided to not linger too long on the scores. The scores for the dependency heuristics were clearly too high, but due to the low number of questions affected, tuning these scores were not a priority. A second problem with the scores is related to how they are calculated. Most of the heuristics go through each sentence in the text for each answer candidate, and calculate a score for the current combination of sentence, question and answer candidate. The highest score is kept, and whether only one sentence got any matches or ten makes no difference. We think that some form of weighting of redundancy could be beneficial in this case, perhaps using inverse document frequency.

7.2.3 Using the background collection

In the original task, a background collection was supplied, containing a large set of material related to each of the topics. We did not have access to this data, but it is possible that by parsing this data, we could have identified the terms that were especially relevant for each topic, and perform some domain adaptation for the heuristics by giving higher scores to matches containing these terms, for example.

Finally, sentences are considered individually, with no other context. Some simple chunking heuristics, like looking at sentences in sets of threes, could be an improvement.

7.3 Conclusion

One of the main problems when attempting to use syntax to solve this task has been the dataset, in our opinion. This task was designed to move away from pattern matching, and has in some ways moved very close to requiring full understanding. While full understanding is a future goal for question answering systems, it is not a level of complexity that is achievable to approach within the scope of a master's thesis. The amount of data has

also caused some problems, especially the lack of training data. We worked around this by dividing the already small dataset into a development set consisting of 75% of the data and a test set with the remaining 25%. When running the system on the test set, we saw that while the results adhered to the same pattern we had seen previously, the results were different from those we got on the development set. The data is very variable, with each reading test having a different tone, and the difficulty of the questions varying from very simple to difficult for some humans.

7.4 Future work

There are many opportunities for future work based on the work done in this thesis. While there were problems with the dataset supplied, we are curious to see how our system would perform on another dataset, and perhaps do some work on what types of data it takes for the use of syntax to be beneficial.

A second possibility is to investigate whether or not the proposed solution for anaphora resolution is applicable to other system. The solution could be expanded, to work on more anaphors, and have more refined filtering methods for possible antecedents.

Based on the analysis done on the data, it seems likely that the use of external resources will be beneficial, both for background knowledge and lexical semantics. In the future, we would like to determine what changes we would have to make to our system in order to make use of these.

Bibliography

Babych, Svitlana, Alexander Henn, Jan Pawellek, and Sebastian Padó. 2011. "Dependency-Based Answer Validation for German." In *CLEF 2011 Labs and Workshop, Notebook Papers.* Amsterdam, The Netherlands.

Bhaskar, Pinaki, Partha Pakray, Somnath Banerjee, Samadrita Banerjee, Sivaji Bandyopadhyay, and Alexander F Gelbukh. 2012. "Question Answering System for QA4MRE@ CLEF 2012." In *CLEF 2011 Labs and Workshop, Notebook Papers.* Amsterdam, The Netherlands.

Bird, Steven, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python.* O'Reilly.

Chomsky, Noam. 1957. *Syntactic structures.* Mouton & co.

Dagan, Ido, Oren Glickman, and Bernardo Magnini. 2006. "The pascal Recognising Textual Entailment Challenge." In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment,* 177–190. Springer.

De Marneffe, Marie-Catherine, and Christopher D Manning. 2008. "The Stanford typed dependencies representation." In *COLING Workshop on Cross-framework and Cross-domain Parser Evaluation.*

Forner, Pamela, Anselmo Peñas, Eneko Agirre, Iñaki Alegria, Corina Forăscu, Nicolas Moreau, Petya Osenova, Prokopis Prokopidis, Paulo Rocha, Bogdan Sacaleanu, et al. 2009. "Overview of the CLEF 2008 multilingual question answering track." In *Evaluating Systems for Multilingual and Multimodal Information Access,* 262–295. Springer.

Giampiccolo, Danilo, Pamela Forner, Jesús Herrera, Anselmo Peñas, Christelle Ayache, Corina Forascu, Valentin Jijkoun, Petya Osenova, Paulo Rocha, Bogdan Sacaleanu, et al. 2008. "Overview of the CLEF 2007 multilingual question answering track." In *Advances in Multilingual and Multimodal Information Retrieval,* 200–236. Springer.

Hobbs, Jerry R. 1978. "Resolving pronoun references." *Lingua* 44 (4): 311–338.

Jijkoun, Valentin, and Maarten De Rijke. 2007. "Overview of the WiQA task at CLEF 2006." In *Evaluation of Multilingual and Multi-modal Information Retrieval,* 265–274. Springer.

Judge, John, Aoife Cahill, and Josef Van Genabith. 2006. "Questionbank: Creating a corpus of parse-annotated questions." In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics,* 497–504. Association for Computational Linguistics.

Jurafsky, Daniel, and James H Martin. 2009. *Speech and language processing.* Pearson.

Lappin, Shalom, and Herbert J Leass. 1994. "An algorithm for pronominal anaphora resolution." *Computational linguistics* 20 (4): 535–561.

Lee, Heeyoung, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2011. "Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task." In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task,* 28–34. Portland, OR.

Li, Xin, and Dan Roth. 2006. "Learning question classifiers: the role of semantic information." *Natural Language Engineering* 12 (03): 229–249.

Magnini, Bernardo, Danilo Giampiccolo, Pamela Forner, Christelle Ayache, Valentin Jijkoun, Petya Osenova, Anselmo Peñas, Paulo Rocha, Bogdan Sacaleanu, and Richard Sutcliffe. 2007. "Overview of the CLEF 2006 multilingual question answering track." In *Evaluation of Multilingual and Multi-modal Information Retrieval,* 223–256. Springer.

Magnini, Bernardo, Simone Romagnoli, Alessandro Vallin, Jesús Herrera, Anselmo Penas, Víctor Peinado, Felisa Verdejo, and Maarten de Rijke. 2004. "The multiple language question answering track at CLEF 2003." In *Comparative Evaluation of Multilingual Information Access Systems,* 471–486. Springer.

Magnini, Bernardo, Alessandro Vallin, Christelle Ayache, Gregor Erbach, Anselmo Peñas, Maarten De Rijke, Paulo Rocha, Kiril Simov, and Richard Sutcliffe. 2005. "Overview of the CLEF 2004 multilingual question answering track." In *Multilingual Information Access for Text, Speech and Images,* 371–391. Springer.

Manning, Christopher D, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval.* Vol. 1. Cambridge university press Cambridge.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. "Building a Large Annotated Corpus of English: The Penn Treebank." *Computational Linguistics* 19 (2): 313–330.

Marneffe, Marie-catherine De, and Christopher D. Manning. 2008. *Stanford typed dependencies manual.*

Miller, George A. 1986. "Dictionaries in the Mind." *Language and Cognitive Processes* 1 (3): 171–185.

Nivre, Joakim. 2005. "Dependency grammar and dependency parsing." *MSI report* 5133 (1959): 1–32.

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2006. "MaltParser: A data-driven parser-generator for dependency parsing." In *Proceedings of LREC,* 2216–2219. Vol. 6.

Oostdijk, Nelleke, Suzan Verberne, and Cornelis HA Koster. 2010. "Constructing a Broad-coverage Lexicon for Text Mining in the Patent Domain." In *LREC.*

Pakray, Partha, Pinaki Bhaskar, Somnath Banerjee, Bidhan Chandra Pal, Sivaji Bandyopadhyay, and Alexander Gelbukh. 2011. "A hybrid question answering system based on information retrieval and answer validation." In *CLEF 2011 Labs and Workshop, Notebook Papers.* Amsterdam, The Netherlands.

Peñas, Anselmo, Pamela Forner, Álvaro Rodrigo, Richard F. E. Sutcliffe, Corina Forăscu, and Cristina Mota. 2010. "Overview of ResPubliQA 2010: Question Answering Evaluation over European Legislation." In *Multilingual Information Access I. Text Retrieval Experiments.*

Peñas, Anselmo, Pamela Forner, Richard Sutcliffe, Álvaro Rodrigo, Corina Forăscu, Iñaki Alegria, Danilo Giampiccolo, Nicolas Moreau, and Petya Osenova. 2010. "Overview of ResPubliQA 2009: question answering evaluation over European legislation." In *Multilingual Information Access Evaluation I. Text Retrieval Experiments,* 174–196. Springer.

Peñas, Anselmo, Eduard Hovy, Pamela Forner, Álvaro Rodrigo, Richard Sutcliffe, Corina Forascu, and Caroline Sporleder. 2011. "Overview of QA4MRE at CLEF 2011: Question answering for machine reading evaluation." *CLEF 2011 Labs and Workshops, Notebook Papers* (Trento, Italy).

Peñas, Anselmo, Alvaro Rodrigo, and Juan del Rosal. 2011. "A simple measure to assess non-response." In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies,* 1415–1424. Vol. 1.

Peñas, Anselmo, Álvaro Rodrigo, Valentín Sama, and Felisa Verdejo. 2007. "Overview of the Answer Validation Exercise 2006." In *Evaluation of Multilingual and Multi-modal Information Retrieval,* 257–264. Springer.

Prager, John, Jennifer Chu-Carroll, Eric W Brown, and Krzysztof Czuba. 2006. "Question answering by predictive annotation." In *Advances in Open Domain Question Answering,* 307–347. Springer.

University, Princeton. 2010. "About WordNet." Princeton University. `http://wordnet.princeton.edu`.

Vallin, Alessandro, Bernardo Magnini, Danilo Giampiccolo, Lili Aunimo, Christelle Ayache, Petya Osenova, Anselmo Peñas, Maarten De Rijke, Bogdan Sacaleanu, Diana Santos, et al. 2006. "Overview of the CLEF 2005 multilingual question answering track." In *Accessing multilingual information repositories,* 307–331. Springer.

Vanderwende, Lucy, and William B Dolan. 2006. "What syntax can contribute in the entailment task." In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment,* 205–216. Springer.

Verberne, Suzan. 2011. "Retrieval-based Question Answering for Machine Reading Evaluation." In *CLEF 2011 Labs and Workshop, Notebook Papers.*

Webber, Bonnie, and Nick Webb. 2010. "Question answering." In *The handbook of computational linguistics and natural language processing,* 630–654. Vol. 57. John Wiley & Sons.