Memory-Based Shallow Parsing of Spoken Dutch

Sander Canisius

Master's Thesis CS 04-01

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science of Knowledge Engineering
in the Faculty of General Sciences
of the Universiteit Maastricht

Thesis committee:
Prof. dr. H.J. van den Herik
Prof. dr. E.O. Postma
Dr. A.P.J. van den Bosch

Universiteit Maastricht
Institute for Knowledge and Agent Technology
Department of Computer Science
Maastricht, The Netherlands
January 2004

Contents

Preface

Bouma and Schuurman (1998) state that there have been three language-technological revolutions: 1) the invention of writing, 2) the invention of printing, and 3) the invention of the computer. Those languages that have not participated in the first two revolutions have eventually disappeared or have been marginalised. It can be expected that the same will happen to those languages that do not participate in the third. Being dependent on the availability of sufficient resources, research in natural-language processing has mostly been focused on the English language. If this situation sustained, languages with smaller communities of speakers, such as Dutch, would risk meeting the above expectation.

Fortunately, recent years have seen the development of resources enabling the application of language-technology research to Dutch. A notable example of these resources is the Spoken Dutch Corpus, which provides a large corpus of annotated Dutch texts. This thesis describes the development of a Dutch shallow parser based on the Spoken Dutch Corpus and memory-based learning techniques originating from the ILK research group at Tilburg University and the CNTS research group at the University of Antwerp. Hopefully, the research described in this thesis will be a valuable contribution to the field of language technology for Dutch.

Although the cover of this thesis mentions only my name, many people have, in one way or another, contributed to an environment that enabled me to write this thesis. In this preface I would like to take the opportunity to thank them. First of all, Antal van den Bosch has been my coach and guide into the world of computational linguistics. His advice has been invaluable throughout the entire course of my thesis research. Also, his enthusiasm for this research and for his field of study in general have been very motivating. I am looking forward to our cooperation in the years to come.

Jaap van den Herik and Eric Postma have been critical readers of this thesis in its final stages. They provided useful comments on both the global structure and the smaller subtleties of scientific writing. Moreover, Jaap van den Herik has especially supported my ambitions in computational linguistics by advising to do my graduate research in Tilburg. Also, both of them have been inspiring teachers during my study Knowledge Engineering.

Finally, I would like to thank my family, in particular my parents, for their support, and my friends for many hours outside university, making the past four years a happy period in my life.

Sander Canisius
Maastricht, January 2004

Chapter 1

Introduction

This thesis deals with the topic of parsing of spoken Dutch. Traditional parsing methods give rise to a knowledge-acquisition bottleneck. For the English language, data-driven techniques have been shown to be able to overcome this bottleneck. In particular, the memory-based shallow parsing method (Buchholz, Veenstra, and Daelemans, 1999) has been quite successful. The question is whether these techniques are also successfully applicable to languages other than English. Hence, this thesis explores to what extent the techniques can be applied to the Dutch language.

This chapter starts with presenting some background on the topic, in section 1.1. In section 1.2, the research objective is formulated. Next, section 1.3 gives a review of previous work, and as a consequence, the research question is formulated in section 1.4. Section 1.5, finally, presents the outline of the thesis.

1.1 Background

Traditionally, parsing of natural language has been grammar-based. Grammars are large formal descriptions of a language, describing how sentences are composed of simpler language units such as clauses and words. The form of such a description is specified by the specific grammar formalism used. Examples of such formalisms are dependency syntax, X-bar theory, and constraint-based grammars. Once a grammar has been constructed, a parsing algorithm, which is often tailored to the formalism used, can parse a sentence according to the rules defined in the grammar.

The necessity to hand-craft a grammar for each combination of language and formalism is a serious shortcoming of the grammar-based approach. To be successful, a grammar should be able to accept all possible sentences. This means that the grammar should not only describe the most common language constructs, but also those less frequently used and breaking the rules defined to describe the most common constructs. Moreover, grammar construction is complicated even more by the constantly evolving nature of language. These factors give rise to a knowledge-acquisition bottleneck, making grammar construction a difficult and slow process. In some cases this bottleneck can be overcome by constraining the language to be parsed, but when a comprehensive grammar is required, its development may take many years.

Alternative approaches claiming to overcome the knowledge-acquisition bottleneck use a variety of techniques to construct parsers that can learn the correct parse of a sentence from examples. Such approaches are collectively referred to as data-driven parsing methods. One such approach is based on memory-based learning. Memory-based parsers operate by drawing analogies between the sentence to be parsed and stored example parses, exploiting the fact that similar sentences often have similar parses. For such an approach to be effective, a large amount of training material should be available. To meet this need for training material, large corpora of typical language use are gathered and manually or semi-automatically parsed. The collections of syntactically annotated sentences that result from such a process are called treebanks.

The availability of treebanks of sufficient size is a necessary condition for research on data-driven parsing of a language to be conducted. English language treebanks, such as the Penn Treebank and the Susanne Corpus, have been available for a long time and consequently research on data-driven parsing of English has progressed steadily. The same cannot be said for languages with smaller communities of speakers. For those languages research is hindered by the lack of sufficient amounts of training material. For a long time this used to be the case for Dutch. In the absence of a large syntactically annotated corpus of Dutch language use, little research on data-driven parsing was performed. The Spoken Dutch Corpus (Dutch: "Corpus Gesproken Nederlands", or CGN) aims to overcome this deficiency by resolving the need for a large-scale treebank of commonly spoken Dutch. Another initiative is the recent Alpino Treebank project (Van der Beek et al., 2002). In this thesis, the CGN corpus is used.

The CGN in its final form will comprise approximately ten million words of contemporary spoken Dutch. One of the primary aims of the CGN project has been to provide resources for Dutch speech and language technology research. Therefore, large parts of the corpus have been enriched with additional information suited for computer processing: one million words will be syntactically annotated, thereby forming a useful source of training data for data-driven parsing techniques. At the time of writing the final corpus has yet to be completed, but the large part that has been finished already enables research to be performed.

1.2 Research objective

As indicated in the previous section, data-driven learning methods, for example memory-based learning, provide an alternative to grammar-based parser development that is not a priori affected by the knowledge-acquisition bottleneck. However, the performance of memory-based parsing is highly dependent on the availability of sufficient training data. For the Dutch language, the relatively large collection of syntactically annotated Dutch sentences provided by the CGN is intended to fulfil this condition. Therefore, the development of Dutch data-driven parsers has become possible.

The objective of this research is to construct and optimise a parser for Dutch using memory-based learning techniques. As much as possible, the memory-based approach that has been proven to be successful for English (Daelemans, Buchholz, and Veenstra, 1999) will be adopted. During both parser construction and parser optimisation the syntactically annotated part of the CGN will

be used. For constructing the parser, a selection of training sentences will be drawn from the corpus; for optimising it, a test set will be formed out of the remaining sentences. Achieving the research objective would have two important consequences. First, the development of natural-language processing applications targetting Dutch would no longer be impeded by the knowledge-acquisition bottleneck. Secondly, the claim to the language-independence of the memory-based approach to parsing would be supported by having a language other than English parsed by a memory-based parser.

In this research, the parsing task consists of two activities. First, the sentence is divided into its grammatical parts. The boundaries of the parts are identified and each part is labelled with its grammatical type. Next, the syntactic functions of the parts are determined. The syntactic function of a part is its relation to another part within the grammatical structure of the sentence. Importantly, rather than performing a full parse, the parsing task is restricted to what is often referred to as shallow parsing or chunking. During shallow parsing, only non-recursive constituents, or chunks are considered. The identification of grammatical relations also has a restricted definition. Only relations to verbal chunks will be considered. These include relation types such as subject and object, which are the most informative grammatical relations in a sentence.

The second part of the research objective requires that the parser be optimised. To perform the optimisation, some notion of optimality needs to be defined. This notion should be quantifiable in such a way that one parser configuration can be said to perform better than another. In this research, performance will be measured along two dimensions. The first dimension is the accuracy, that is, how well the parser is capable of correctly parsing previously unseen sentences. The other is the speed at which the parser operates. As will be demonstrated, it is not possible that both dimensions attain their highest levels at the same time. Increasing performance in one dimension will most likely harm it in the other. In such a situation, a trade-off between the two needs to be established.

1.3 Previous work

As mentioned in the introduction to this thesis, memory-based shallow parsing has already been successful when trained on English language corpora. Specifically, the research described in this section has generally been performed in combination with the Wall Street Journal corpus, which is a treebank of formally written English. In principle, however, memory-based shallow parsing is language-independent and therefore it should be possible to apply it to any language. For this reason, the research objective aims at adopting the memory-based shallow parsing method for constructing the Dutch memory-based parser.

The groundwork for the memory-based shallow parsing approach is laid by Daelemans (1996) who claims that all linguistic tasks can be reformulated as classification problems and that, as a result of this, it is possible to train memory-based learners for these tasks. To support his claim, he first shows that tasks in natural-language processing are context-sensitive mappings between representations and then argues that every linguistic problem can be described by one of two kinds of mappings: disambiguation and segmentation.

Disambiguation mappings assign one of a predefined set of categories to a context. This type of mappings includes part of speech tagging, where the

correct word class, for example noun, verb, or adjective, for a word is determined given its form and its place in the sentence. Segmentation mappings decide whether, given a target and its context, a boundary is associated with this target and, if so, which type of boundary. A typical segmentation task is the detection of chunk boundaries.

The cascaded Memory-Based Shallow Parser (Buchholz et al., 1999) implements the ideas described above. The parsing task is split up into a number of subtasks, each of which can be reformulated as either a disambiguation or a segmentation task. Starting with only the words forming a sentence, the available information is enriched by the results of each subtask. This way, the inputs to subtasks can be partly composed of information that is not directly accessible in the original data, but generated by other subtasks. For this reason, the memory-based classification modules trained to perform the subtasks are applied in sequence, so that each module precedes those dependent on its results. The Memory-Based Shallow Parser has modules for part of speech tagging, chunking, PNP[1] finding and grammatical-relation finding, in that order. Each of these tasks is individually introduced in the remainder of this section.

Part of speech tagging

Part of speech tagging is the process of associating each word in a sentence with its correct word class. This word class, also called the part of speech of the word, expresses how this word is used in the sentence. One word can be used in different ways and therefore the part of speech for a word is not always the same. A part of speech tagger should be able to assign the correct part of speech to a word, basing its decision on the word itself and the words surrounding it in the sentence.

Daelemans et al. (1996) introduce a memory-based approach to part of speech tagging. They show that a memory-based learner can be trained to perform part of speech tagging with good results. In addition, a tagger-generator is presented that automatically generates memory-based part of speech taggers, given tagged corpora as example data. In this thesis, part of speech tagging is not performed by a separate module. Rather, part of speech tagging and chunking are both performed by a single module, which is based on the techniques described by Daelemans et al. (1996).

Chunking

Chunking divides a sentence into non-recursive, non-overlapping constituents, referred to as chunks. Abney (1991) introduced the concept of a chunk, suggesting that dividing a sentence into chunks is a useful intermediate step towards full parsing. Nevertheless, chunking is a valuable process in its own right when the entire grammatical structure produced by a full parse is not required. Studies by Grishman (1995) and Appelt et al. (1993), for instance, indicate that the information obtained by a shallow parse is sufficient for information extraction to be performed. In terms of the two kinds of mappings proposed by Daelemans chunking is a segmentation task.

[1] A PNP chunk denotes a prepositional phrase that has a nominal complement; it is a higher-level kind of chunk than the ones identified in the chunking step.

The chunking module of the Memory-Based Shallow Parser (Veenstra, 1999) uses a classification scheme by Ramshaw and Marcus (1995): each word is assigned a chunk tag denoting whether the word is inside a chunk (I), inside a chunk, but not in the same chunk as the word directly preceding it (B), or outside any chunk (O). As there is more than one chunk type to distinguish, an additional chunk type symbol is added to the chunk tag. A chunk tag I-NP, for example, means the word is inside a noun phrase (NP). Buchholz (2002) additionally extends the chunk tag by attaching the part of speech tag, effectively eliminating the need for a separate part of speech tagging step.

PNP finding

Due to the fact that only non-recursive constituents are identified in the chunking step, prepositional phrases (PP) will not be recognised as such. Rather, PPs are divided into a PP chunk containing only a single preposition, and one or more NP chunks. In order to recover the original prepositional phrase, the PNP finding module of the parser joins PP chunks and the NP chunks related to it in PNP chunks. The chunks "[PP met] [NP de commissie] en [NP het consortium]", for example, would be joined in "{PNP [PP met] [NP de commissie] en [NP het consortium] }". PNP finding is implemented in a manner very similar to grammatical-relation finding.

Grammatical-relation finding

Finding grammatical relations is the final step in the parsing process. During this step grammatical relations between verbal chunks and chunks of other types are identified. Relations to verbs include the most important relations to capture the meaning of a sentence, such as the subject and object relations. Buchholz (2002) describes the design of a memory-based grammatical-relation finder. The technique described assumes that grammatical relations hold between the head words of chunks. The head word of a chunk is the most prominent word in that chunk.

Grammatical-relation finding is formulated as a disambiguation task: given the head word of a verbal chunk and the head word of another chunk it is predicted whether a grammatical relation holds between the two words and, if so, which type it has. Before relation finding can take place, the parts of speech and the chunks for the words in the sentence should have been determined. This information is used in the description of the instances fed to the relation finder. Apart from part of speech and chunk type, other features that were found to be good predictors for grammatical relations include spatial features, such as the number of intervening verb chunks and commas.

1.4 Research question

In the previous sections, it has been recognised that there is a need for techniques that can accelerate the development of natural-language processing applications for Dutch. Memory-based shallow parsing has been put forward as a technique satisfying this need. Although, for practical reasons, the emphasis of memory-based shallow parsing research has traditionally been on English, the approach is essentially language-independent and consequently does not exclude Dutch.

Moreover, as the CGN provides a large body of contemporary spoken Dutch language data, the enabling resources for the development of Dutch memory-based shallow parsers are available. Taking all of the above into consideration, the research objective has been formulated to be the development and optimisation of a Dutch memory-based shallow parser.

The approach to memory-based parsing adopted for this research has been proven to be successful on a corpus of formally written English language. Applying the same approach to a Dutch corpus can thus be considered a test of its language-independence. CGN, being a corpus of spoken language, however, adds another dimension in which the two corpora differ: the nature of the language use. Spoken language data, apart from being less formal, typically contain more noise than written language data as a result of various types of mistakes made by the speakers. For the above reasons, the CGN data form an interesting test case for the memory-based shallow parsing approach.

Given both the research objective and the remarks made above, the following research question is formulated:

> To what extent can the memory-based shallow parsing techniques be applied to a corpus that differs from the original training corpus in the sense that it is Dutch rather than English and that it contains spoken rather than written language?

The remainder of this thesis will be centred around answering this question. The process of constructing, optimising and evaluating the Dutch memory-based parser will be described in the chapters to come. A detailed outline of the thesis is given in the next section.

1.5 Thesis outline

The outline of the remainder of this thesis is as follows. Chapter 2 introduces the learning tasks and the training data. In the first part of the chapter, the different stages of the memory-based shallow parser are represented as memory-based classification tasks. The survey collects work described in several different publications that makes up the actual memory-based shallow parsing framework. The second part of the chapter presents the CGN training data. First a general overview of the annotation philosophy of the corpus is given. Next, the transformation of the corpus data to a format suited for the learning tasks, is described.

In chapter 3, the core of the research is described. The central issue in this chapter is formed by the experiments conducted to optimise the Dutch memory-based shallow parser. The results of these experiments will shed light on the central research question of this thesis. In the first part of the chapter the experimental setup is addressed; an overview of the various algorithmic parameters that can be varied, is given and the general optimisation method is described. The second part of the chapter presents the results of the experiments.

Evaluation of the experimental results in the light of the research question takes place in chapter 4. It is discussed whether the differences in language and nature of the CGN cause notable difficulties for the memory-based shallow parsing approach. Chapter 5 concludes this thesis by summarising the results and proposing an answer to the research question. In addition, recommendations for future research are presented.

Chapter 2

Learning tasks and data

In chapter 1, the research and its objective have been introduced. Two important elements of the research are the learning tasks and the training data. Both have been mentioned shortly in the previous chapter. This chapter gives a more detailed description of their role in the research and lays the theoretical groundwork for the experiments. First, the learning tasks to be performed by the modules that make up the memory-based parser, are presented. Next, the CGN and the conversion of its contents to a format suited for a memory-based learner, are dealt with.

2.1 Learning tasks

As explained in section 1.3, the parsing process is split up into 1) a combined part of speech tagging/chunking step, 2) PNP finding, and 3) grammatical-relation finding. Analogously to this subdivision, the actual memory-based parser has been designed as a modular system, consisting of separate memory-based classifiers for each distinct subtask. These classifier modules can be approached from two different angles. On the one hand, they can be dealt with as independent units that can be described, trained and optimised individually. Indeed, all the parsing steps described in this section have individually been the central topic of one or more publications. These publications proposed a translation of the tasks to the memory-based learning domain and gave an overview of its performance when tested in isolation.

On the other hand, the modules are interdependent components of a global parsing framework, as is illustrated in figure 2.1. The grammatical-relation finder, for instance, uses results of both the chunker and the PNP finder modules. Without these two modules, the grammatical-relation finder can only be applied in a test environment where the necessary information is extracted from a corpus. This interdependency, then, has two important consequences. First, actual parsing of a sentence requires that all modules have been implemented and joined together in the parsing framework. Secondly, the accuracy of the output of a module not only depends on its own classification performance, but also on the accuracy of its input data, which are partly generated by other modules.

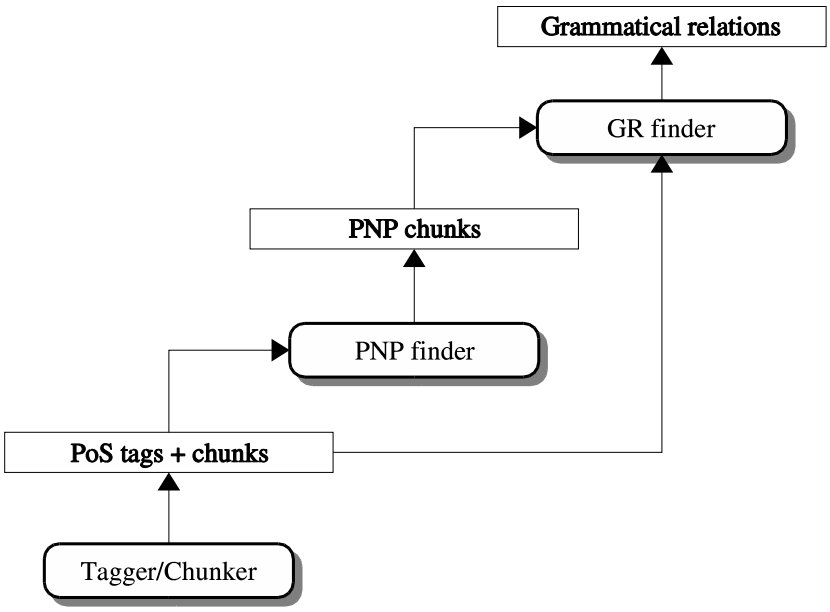

Figure 2.1: Architecture of the memory-based shallow parsing framework. The rounded rectangles represent the three parser modules; the normal rectangles and the arrows depict the information flow between the modules.

These two faces of the classifier modules will return in chapter 3, which describes the experimental optimisation and evaluation of the memory-based parser. In this section, however, the subtasks performed by the modules are only considered in isolation. In order to train a memory-based learner for any of the three parsing subtasks, the subtasks first need to be formulated in a form suited for the memory-based classification algorithm. The three algorithms used in this research are IB1, IGTree and TRIBL as provided by the TiMBL software (Daelemans et al., 2002). These algorithms expect their instance base to consist of instances described by a fixed number of attribute-value pairs and a category label indicating the correct classification for the instance. The exact details of the learning tasks and instance formats for tagging/chunking, PNP finding, and grammatical-relation finding respectively, are described in the following subsections.

2.1.1 Tagging/chunking

The first module in the parsing cascade combines the part of speech tagging and chunking steps into a single learning task. Early implementations of the memory-based chunker, such as those described by Veenstra (1999) and Veenstra and Van den Bosch (2000), used feature vectors that include features for the parts of speech of words. These parts of speech, then, had to be predicted by a separate part of speech tagger that was run before the chunking step. As an alternative to this two-module approach, Buchholz (2002) introduced a module that predicted parts of speech and chunks at the same time. This combined

tagging/chunking approach is adopted for the memory-based shallow parser in this thesis.

Chunking consists of identifying the syntactic parts of a sentence. Finding syntactic parts can be performed with differing degrees of detail. Many grammar-based parsers, for example, reconstruct the entire phrase structure tree as a result of parsing a sentence. In such a tree, the root node represents the complete sentence and its children denote the immediate constituents entering into the sentence construction. These immediate constituents may be recursively further reducible to even smaller grammatical units until each such a unit corresponds to a single word.

In contrast, chunking ignores most of the higher-level constituents of a sentence and only identifies the non-recursive, non-overlapping constructions. Abney (1991), who proposed this shallow approach to parsing, refers to these specific constructions as chunks. Central to the notion of a chunk is its head, which can informally be defined as the most prominent word of its chunk. The relevance of the head is that it establishes the phrasal category of a chunk. For example, a chunk of type NP is generally headed by a substantive; likewise, the type of a chunk headed by any form of a verb is one of the verbal categories.

As an example, the sentence "ze waren gewoon vergeten hoe laat de bus kwam." is chunked as follows:

> [NP ze] [SMAIN waren] [AP gewoon] [PPART vergeten] [ADVP hoe laat] [NP de bus] [SSUB kwam] .

In order to translate the chunking task to the domain of memory-based classification, two issues need to be decided upon. First, it should be specified how to map sentences to machine-learning instances. Then the reverse of this mapping, that is, how to extract the chunk properties from the classification results, has to be established. Both decisions are based on the approach to chunking as an instance of a class of segmentation tasks, referred to in section 1.3. The essence of such tasks is to decide for each word in a sentence, whether that word corresponds to some kind of boundary and, if so, to which kind of boundary.

The first mapping, sentences to machine-learning instances, requires that the variable-length sentences are mapped to fixed-size instances. Windowing is the generally accepted technique to meet this requirement. A window consisting of a fixed number of slots is placed on the sentence; each slot captures a single word. The word in the centre of the window is often taken to be the focus word, that is, the word to be classified. The surrounding words, then, encode information about the local context of the focus word. Starting with the first word, the window is slid over every word in the sentence, each time making an instance out of the words shown in the slots. Thereby, one sentence causes as much instances to be generated as it contains words. The size of the instances, however, remains fixed, as was required.

After a sentence has been converted into machine-learning instances and these, in turn, have been classified by a memory-based classifier, the classification results of all instances should again be rejoined in a chunked version of the original sentence. Closely related to this reverse mapping of instances to chunks is the design of the set of classification categories. In fact, these categories should be able to encode the entire shallow parse of the sentence. The definition of segmentation tasks suggests classification categories that either de-

note a type of boundary or indicate that no boundary is associated with the focus word.

A naive application of this principle would devise category labels for the boundaries of each type of chunk and another one to signify the absence of any boundary. The instances corresponding to the phrase "een mooi huis", for example, would be classified as "een$_{\text{L-NP}}$ mooi$_{\#}$ huis$_{\text{R-NP}}$", which means that "een" and "huis" are the left and right boundaries of an NP chunk respectively, and that "mooi" does not correspond to any boundary. While simple, the naive encoding is not sufficiently robust to be used for machine learning, which involves the presence of classification errors. Incorrect classification of "een" as #, for example, would prevent the entire chunk to be found. Therefore, a more robust encoding should be used.

Tjong Kim Sang and Veenstra (1999) report on an empirical comparison of various encodings of chunk structures. The Memory-Based Shallow Parser uses an encoding proposed by Ramshaw and Marcus (1995) that performs better in the presence of classification errors than the naive encoding. Their technique assigns each word a tag indicating that the word is either part of a chunk with a specified type or not contained in any chunk. The same sentence as used earlier would be classified as: "een$_{\text{I-NP}}$ mooi$_{\text{I-NP}}$ huis$_{\text{I-NP}}$", which means that all three words are inside an NP chunk. Using this encoding, incorrect classification of "een" as "O", indicating that the word is outside any chunk, does not prevent "mooi huis" to be identified as an NP chunk. Most of the time, this will be preferable over not finding a chunk at all.

Apart from the already mentioned "I" and "O" tags, the encoding additionally includes a B-*XP* tag. This tag signifies that the focus word is inside a chunk of type *XP*, but in a chunk different from the one of the previous word; in short: the focus word is the first word of a new chunk. The "B" tag is necessary to distinguish two directly adjacent chunks from one single chunk in case both are of the same type.

For the purpose of parsing CGN data, the chunk tags are extended to indicate whether the focus word is the head word of its chunk as well. Chunks for the English language memory-based shallow parser were defined to end after their head word. With this definition, head words are easily designated once the chunks of a sentence have been identified. The chunks that are extracted from the CGN, however, do not necessarily end after their head word. Therefore, in order to identify the head words of chunks, their target class label is extended with an "HD" postfix.

In the presence of classification errors, the use of special chunk tags for head words may lead to chunks that have no head, or more than one head. However, each chunk should have one unique head word. For this reason, given the words making up a chunk, the rightmost of them that has been assigned a head tag is selected as the head word for that chunk. If no word has been assigned a head tag, the rightmost word overall is selected as head.

Typically, chunking is preceded by a part of speech tagging step. The part of speech tags of the words are then available when chunker instances are generated and thus can be used as instance features. Buchholz (2002), however, shows that adding part of speech information to the input of the chunker does not lead to significantly better results. In fact, the chunker can be extended to predict part of speech and chunk tags in one classification step, without loss of

performance. This is achieved by letting the classifier predict target classes that are a concatenation of the part of speech tag and the chunk tag for the focus word.

With these target classes, the sentence of the start of this section is tagged as follows:

> ze/VNW3-I-NP-HD waren/WW2-I-SMAIN-HD gewoon/ADJ9-I-AP-HD vergeten/WW7-I-PPART-HD hoe/BW-I-ADVP laat/WW1-I-ADVP-HD de/LID-I-NP bus/N1-I-NP-HD kwam/WW1-I-SSUB-HD ./LET-O

The tagging/chunking module is constructed using the Memory-based tagger (Mbt) (Daelemans et al., 1996). Although the most obvious use of Mbt is for generating part of speech taggers, it can be trained to perform any tagging task, in which a tag is to be assigned to a focus word given a local context consisting of a number of surrounding words. The task performed by the tagger/chunker is essentially a tagging task, where the chunk tags described above are to be assigned to the words in a sentence. Mbt enables a simplified specification of instances for memory-based tagging tasks. Based on this simplified specification, Mbt generates feature vectors that are fed to standard memory-based learning algorithms.

Mbt generates its feature vectors by using the windowing technique on the words in a sentence. Two types of features are used to describe the words in a window. One of them is the word form itself. The other is either the tag of the word or an ambiguous tag, describing all possible tags for the word. If a word precedes the focus word in the sentence, then Mbt has already predicted the correct tag for this word; hence, this tag can be used in the instance description. In contrast, if a word follows the focus word in the sentence, then the correct tag for this word has yet to be predicted. As an alternative to the fully disambiguated tag for the word, then, a symbol encoding all possible tags for this word according to the training data, is used. A more detailed overview of the issues involved in constructing memory-based classifiers by Mbt is given in Daelemans et al. (1996).

2.1.2 PNP finding

During the chunking step, prepositional phrases are not recognised as such. A prepositional phrase consists of a head word, always a preposition, that is followed by one or more complements, which are generally noun phrases. As sentences are only parsed into non-recursive, non-overlapping constituents, prepositional phrases are split up into a PP chunk and one or more NP chunks. Nevertheless, finding prepositional phrases in sentences may be very useful, since they can contain information valuable to higher-level applications, for example information about times and locations. The PNP finding task consists of reconstructing the original prepositional phrases out of separate PP and NP chunks. These reconstructed prepositional phrases will be denoted as PNP chunks.

Although the structure of prepositional phrases is lost when a sentence is divided into chunks, the information to reconstruct them can be derived from the grammatical relations between the head preposition and the noun phrases making up its complement. The technique used by the PNP finder is therefore very similar to the one used by the grammatical-relation finder, which is

described in the next section. However, while the grammatical-relation finder identifies relations between verbal chunks and other chunks, the PNP finder aims at those between PP chunks and NP chunks instead. Having found these relations, PNP chunks are constructed from a PP chunk and the NP chunks grammatically related to it.

Instances for the PNP finding task, as for the grammatical-relation-finding task, are centred around pairs of chunks. In PNP finding instances, such a pair corresponds to a PP chunk and an NP chunk, which is henceforth referred to as the focus chunk. For a given sentence, instances are generated for each pair of a PP chunk and an NP chunk following it within a certain distance. To encode the variable-length chunks by fixed-size feature vectors, the chunks are represented by their head words only. The heads are the most prominent words of chunks and therefore, grammatical relations could equally well be considered to hold between head words of chunks, rather than between chunks themselves. Apart from the word form, the features to describe the chunks are the part of speech of the head word and the chunk type.

In addition to the PP and focus chunks, instances include features to encode the local context of the focus chunk. This local context consists of a number of chunks directly preceding and a number of chunks directly following the focus chunk. Again, these chunks are represented by their head words, part of speech tags and chunk types. In short, the complete feature vector of PNP finding instances consists of the following features:

- The distance between the preposition and the focus chunk, counted in number of chunks or words outside chunks.
- The number of PP chunks between the preposition and the focus chunk.
- The head word of the PP chunk.
- The focus chunk, which is described by two features.
  - The head word of the focus chunk.
  - The part of speech of the head word.
- The context chunks, each described by three features.
  - The head word of the context chunk.
  - The part of speech of the head word.
  - The type of the context chunk.

The classification categories for the PNP finding task are kept simple. Instances are classified to decide whether a given preposition and focus chunk are to be joined in a PNP chunk. This is the case if the two are grammatically related. In contrast with the grammatical-relation-finding task, for PNP finding, the type of the grammatical relation is unimportant and consequently need not be predicted. PNP finding instances can be classified as one of either "+" or "-", thereby only indicating whether, not how, the preposition and focus chunk are related.

2.1.3 Grammatical-relation finding

The final step in the memory-based parsing cascade is grammatical-relation finding. Having found the chunks of a sentence in the two preceding parsing steps, in this step, the parser determines their syntactic functions by identifying grammatical relations between pairs of chunks. Important grammatical relations include the subject and object relations. By adding grammatical relations to the parse of a sentence, it is transformed from a set of separate chunks to a coherent structure of interrelated constituents. Additional processing modules could, for example, extract the intended message from this structure.

As was already mentioned in the research objective and following Buchholz (2002), grammatical-relation finding in this thesis is restricted to relations to verbal chunks. Although this restriction will exclude potentially useful relations, the central meaning of a sentence can often be retrieved quite well by only considering relations to verbs. Moreover, Buchholz (2002) speculates that finding relations to nouns or adjectives might require information different from that useful for finding relations to verbs and would therefore be better performed in separate tasks.

The essence of the grammatical-relation-finding task has already been reviewed in the previous section. As PNP finding and grammatical-relation finding share many similarities, the learning tasks are very similar too. With grammatical-relation finding, however, the relations to be identified are not those between PP chunks and NP chunks, but consist of relations between verbal chunks and any other chunks. Furthermore, while the PNP finder only needs to determine whether two chunks are related, the grammatical-relation finder should predict the type of the relation as well.

Machine-learning instances for the grammatical-relation-finding task are centred around the head words of two chunks. They represent the verbal chunk and a focus chunk, which may be of any type. Additionally, instances contain features to encode the local context of the focus chunk. This local context consists of a number of chunks directly preceding and a number of chunks directly following the focus chunk in the sentence. To find the grammatical relations in a sentence, instances are generated for each verbal chunk and focus chunk within a certain distance to the left or to the right. The classifications of these instances indicate whether and, if so, how the focus chunk is related to the verbal chunk.

More features than those mentioned above can be added to the instance description to improve classification performance. Buchholz (2002) specifically intends to identify features that are useful for memory-based grammatical-relation finding. A notable result is the highly informative value that features coding sequences of part of speech tags or chunks, appear to have. In this thesis, however, the simpler instance format used by Buchholz et al. (1999) is used. The features of this format are presented below.

- The distance between the verb and the focus chunk, counted in number of chunks or words outside chunks. A negative distance indicates that the focus chunk is to the left of the verb, a positive distance means the focus chunk is to the right of the verb.

- The number of verbal chunks between the verb and the focus chunk.

- The verb, that is, the head word of the verbal chunk.
- The part of speech of the verb.
- The focus chunk, which is described by four features.
  - The prepositional head word, if the focus chunk is a PNP chunk, or "-" otherwise. This is the head word of the PP chunk within the PNP.
  - The head word of the focus chunk. If the focus chunk is a PNP chunk, this means the head word of the NP chunk within it.
  - The part of speech of the head word.
  - The type of the focus chunk.
- The context chunks, each described by three features.
  - The head word of the context chunk.
  - The part of speech of the head word.
  - The type of the context chunk.

The classification categories to be assigned to these instances correspond to the types of relations to be predicted and "-", to indicate that the given focus chunk is not related to the given verb.

2.2 Data

The previous section described the instance formats for the three learning tasks that make up for the memory-based shallow parsing framework. In this section, the extraction of the information required for generating these instances is presented. First, the format of the syntactic annotations of the CGN is reviewed. Then, the conversion of CGN annotations to the chunks and grammatical relations required for the learning tasks, is explained.

2.2.1 The Spoken Dutch Corpus

The CGN was the first corpus to provide a large-scale treebank for Dutch. It has been developed, and actually still is being developed at the time of writing, in the context of a project aiming to compose a corpus consisting of 1,000 hours of spoken standard Dutch. The data for the corpus are collected both in The Netherlands and in Belgium. As the corpus is intended to be a useful resource for various different research interests, among which is speech and language technology, it includes a wide variety of transcriptions and annotations. The design of the corpus aims at a high degree of theory-independence, while preserving the possibility to adapt the corpus data to specific theories.

In the final release of the CGN, a selection of one million words will have been assigned an extensive, syntactic annotation. At the time of writing, approximately half of this material is publicly available. This particular part of the CGN forms the basis for the experiments in this thesis. For this reason, the annotations that are relevant for the experiments are reviewed in this section. A

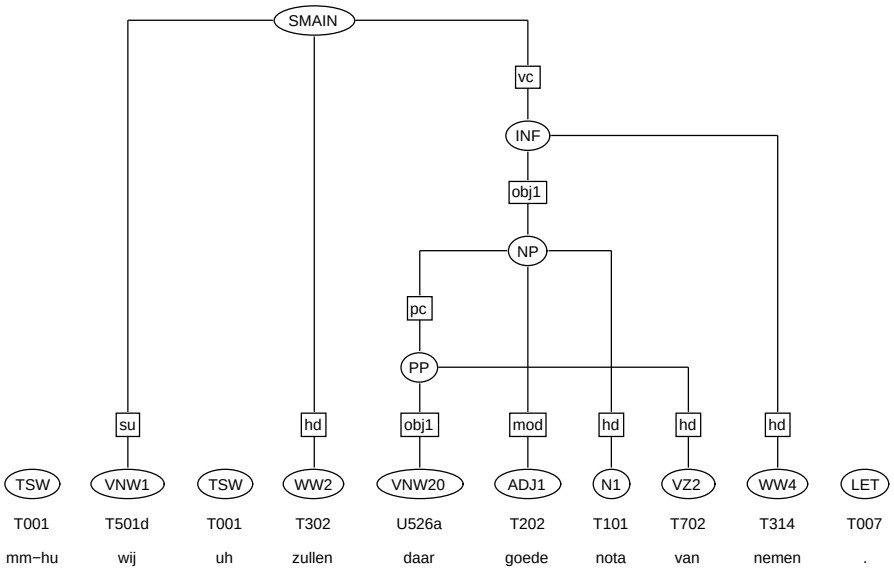

Figure 2.2: Graphical representation of a CGN annotation structure.

general overview of the CGN is provided by the various publications that have originated from the project (Oostdijk, 2000; Oostdijk et al., 2002).

The syntactically annotated part of the CGN is structured very similarly to the NEGRA corpus (Skut et al., 1997). The annotation format used by both corpora structures its information as a directed acyclic graph (DAG), an example of which can be seen in figure 2.2. In this DAG, three kinds of information are stored[1]. The leaf nodes of the DAG correspond to the individual words in a sentence and are labelled with part of speech tags. The internal nodes represent non-terminal constituents and are labelled according to their phrasal types. Finally, the edges of the DAG encode information about the semantic relations, or dependencies, between constituents.

Words appear at the lowest level of the annotation structure as terminal nodes and are assigned a part of speech label. The complete tag set comprises 316 different part of speech categories (Van Eynde, 2001); the labels starting with T or U in figure 2.2 denote part of speech categories. For the syntactic annotation, the tag set is reduced to a set consisting of 72 different labels relevant for the annotation. In addition to part of speech tags, some words may be assigned additional annotations. For example, slips of the tongue and interrupted or foreign words are marked as such.

Although most terminal nodes in the annotation structure correspond to transcriptions of words, terminal nodes are also created for other events during the conversation, such as unintelligible words or laughter. Such phenomena are transcribed using specific labels and are assigned special part of speech tags. Furthermore, the annotation structures also include punctuation marks, for which nodes are created as well. Only three such punctuation marks appear

[1] An overview of all labels used in the CGN syntactic annotation is given in appendix A.

in the CGN transcriptions: the full stop (“.”), the question mark (“?”) and the omission mark (“...”). No commas are transcribed, since their position in spoken sentences cannot be objectively determined.

The non-terminal nodes in the annotation structure correspond to higher-level syntactic constituents, dependency domains in CGN terminology. They are assigned one of 25 phrasal category labels (Moortgat, Schuurman, and Van der Wouden, 2001). The daughter nodes of a dependency domain are either words or lower-level dependency domains. Generally, a dependency domain has exactly one daughter that is referred to as the head of that domain. The other daughters of the domain complement or modify the head.

If a domain would only have a head daughter and no complements or modifiers, no node is created for this domain. Instead, the head represents its mother domain in the annotation structure. For example, the word “wij” in figure 2.2 is directly attached under the SMAIN domain, although syntactically, its directly enclosing constituent is an NP. However, as there are no complements or modifiers for “wij”, the intermediate NP domain has been left out and “wij” is made a direct descendant of the SMAIN domain. This approach leads to flat annotation structures.

The final element of the CGN annotation is the dependency relation, which is stored on the edges of the annotation graph. A dependency relation between a daughter and a mother domain either identifies that daughter as the head of its mother domain or expresses its syntactic function to that head. The CGN annotation uses 34 dependency labels. The semantic meaning of these labels is context dependent, however. For example, in a nominal domain MOD denotes an adjectival adjunct, while in a verbal domain, it indicates an adverbial adjunct.

As a result of the choice for a DAG structure, a node can be dependent on zero, one or even several mother domains. A node that has no mother domain is either at the highest level of an annotation structure or represents a disfluency in the sentence, in which case it is just a single terminal node. In figure 2.2, for example, “mm-hu” and “uh” are disfluencies. A daughter can have multiple dependency relations to express the fact that it has a syntactic function in several dependency domains. In this case, one of those dependencies is denoted as the primary relation, thereby expressing that this is the function of the node in the main argument structure. All remaining dependencies are referred to as the secondary relations.

Another property of the CGN annotation format is its use of crossing branches to simplify the notation of discontinuous dependencies. For example, the annotation structure in figure 2.2 uses a crossing branch to express that “daar” en “van” are part of the same PP domain, even though, in the sentence, they are separated by “goede nota”. Annotation formats that use a tree structure rather than a DAG, require special features to deal with both multiple dependency roles and crossing branches. The high frequency with which these phenomena appear in spoken Dutch justifies the DAG structure of the NEGRA/CGN format.

2.2.2 Data conversion

The syntactically annotated CGN data described in the previous section form the basis for the experiments that are reported on in this thesis. The parsed

fn000573.syn	1	0	I-NP	VNW3	ze	SU	waren	1
fn000573.syn	1	1	I-SMAIN	WW2	waren	None	_	-1
fn000573.syn	1	2	I-AP	ADJ9	gewoon	MOD	waren	1
fn000573.syn	1	3	I-PPART	WW7	vergeten	VC	waren	1
fn000573.syn	1	4	I-ADVP	BW	hoe	NOFUNC	laat	5
fn000573.syn	1	5	I-ADVP	WW1	laat	OBJ1	vergeten	3
fn000573.syn	1	6	I-NP	LID	de	NOFUNC	bus	7
fn000573.syn	1	7	I-NP	N1	bus	SU	kwam	8
fn000573.syn	1	8	I-SSUB	WW1	kwam	BODY	laat	5
fn000573.syn	1	9	O	LET	.	NOFUNC	_	-1

Table 2.1: Example of a CGN sentence in intermediate format. The first three columns of the intermediate format contain administrative information: a file identifier, a sentence number and a word number. The remaining columns encode syntactic information: an IOB-style chunk tag, a part of speech tag, a word, the grammatical relation of the word, the target word of the grammatical relation and the word number of this target word.

sentences are used as training and test data for all learning tasks in the parser. However, while these learning tasks are all formulated in terms of chunks and relations between chunks, the CGN data are fully parsed. Thus, in order to create machine-learning instances from the corpus data, the chunks and the relations between them have to be extracted from the annotation structures.

All of the learning tasks require that chunks have been explicitly annotated in the learning data and two out of three, PNP and grammatical-relation finding, additionally require that relations between chunks have been annotated, as well. Therefore, extracting these chunks and relations is a process that has to be repeated for each type of machine-learning instance that is created from the dependency annotation. Buchholz (2002) has performed the similar task of converting the Penn Treebank data to machine learning instances. She splits up this conversion into two subtasks. The original treebank data is first converted into an intermediate format in which chunks and grammatical relations are explicitly marked. Then, machine-learning instances can quite easily be generated from this intermediate format data.

In this thesis, the same intermediate format is used. An example of the intermediate representation of a sentence taken from the CGN is displayed in table 2.1. The steps required to convert CGN dependency annotations to this intermediate format are described in this section. The conversion process mainly consists of extracting chunks and relations from a CGN dependency structure. Buchholz (2002) describes this process for converting the Penn Treebank format to the intermediate format. In comparison with the Penn Treebank format, the CGN format has the advantage that, in general, heads of constituents are marked explicitly. Also, grammatical relations, although not between chunks, are an elementary part of the CGN annotation structure, while the Penn Treebank format indirectly indicates them by appending a function tag to the syntactic labels of constituents.

The conversion process described in the remainder of this section is restricted in the sense that it only considers primary dependency relations. The decision

to ignore secondary relations is based on several reasons. First, it is more difficult to identify the chunks of a sentence unambiguously when secondary relations are taken into consideration. Secondary relations generally break the main syntactic structure of a sentence. One of the consequences is that a single word might function as head for several dependency domains. Chunks, however, were defined to be non-overlapping and to have a unique head word. Secondly, grammatical-relation finding on data where chunks can perform multiple functions has not received as much attention in the literature as the case in which chunks perform only one function. Finally, the research objective of this thesis is to apply existing memory-based parsing methods to data of a different nature. It does not aim to develop a new learning task.

Heads and chunks

The first step in converting the CGN data to the intermediate format is the extraction of chunks from the dependency structures. The chunks in a sentence can be found by identifying their head words. From these head words, the other words in the chunks can be inferred by applying the following definition from Abney (1991): the root node R of a chunk with head h is the highest node in the parse tree T that has h as its head. Given this root node, the syntactic structure of a chunk is the largest continuous subgraph of T rooted in R and not containing the root of any other chunk. The words of a chunk, then, correspond to the leaf nodes of its chunk structure and the chunk label is defined as the phrasal type of its root node. With Abney's definition, finding chunks comes down to identifying the head words in a sentence and determining the chunk structures they give rise to.

As mentioned in the previous section, the CGN annotation format explicitly indicates heads of dependency domains by means of dependency labels. Some domains, however, have not been assigned a head. Such is the case if an annotation error has been made, but also for some types of domain that have simply been defined to be headless. Whatever the cause, for these domains, a head is determined using a head table. A head table specifies for each type of domain, which are the types of dependency the head may have to this domain. Using such a head table, determining the head of a domain comes down to the following.

1. If the domain has a daughter linked by an HD dependency, this daughter is the head of the domain.
2. If the domain has no HD daughter, the head table is consulted for this domain type:
   (a) If there are terminal daughter nodes linked by one of the dependencies listed in the head table, the right-most of those is the head of the domain.
   (b) If only non-terminal daughter nodes are linked by a dependency listed in the head table, all of those are considered head of the domain.
3. Finally, if none of the previous steps led to a head, the domain is kept headless. This step would only be reached if the annotation is erroneous and the head table incomplete.

The conversion described by Buchholz (2002) uses such a head table for all constituents, to compensate the fact that heads are never explicitly marked in the Penn Treebank. In contrast, the CGN does explicitly mark most heads; hence, the head table is only needed in a few exceptional cases.

The above procedure identifies the head words of multi-word chunks, but fails to recognise head words of single-word chunks. Single-word chunk head words are not explicitly marked in the CGN due to its flat annotation approach. A consequence of this flat annotation is that no dependency domain is created for a constituent if there are no complements or modifiers for its head. Such is the case for constituents comprising only one word. This means that no non-terminal node is created for a constituent consisting of only a single word and therefore this word is not marked as a head word, unless it is the head word of a higher-level domain as well. The above procedure, however, presupposes that head words are annotated as such and consequently will not identify head words of single-word chunks.

In this thesis, head words of single-word chunks are determined by assuming that terminal daughters of certain types of domains, in particular verbal and prepositional domains, are all head words of single-word chunks. Their chunk structures, then, equal the single terminal node corresponding to the head word. In brief, a word is marked as a head word if it is identified as the head of its mother domain or if this mother domain is verbal or prepositional.

Each head word found by this procedure represents a chunk in the sentence. Unless such a head word corresponds to a single-word chunk, its chunk is further made up by those words that satisfy all of the following requirements.

1. It has the same mother domain as the head word.
2. All words between this word and its head word are also part of the chunk.
3. It does not correspond to a single-word chunk.

Not all words will be part of a chunk. The requirement that chunks are continuous, as implied by requirement 2, might cause some words to fall outside a chunk, even though they are related to the same dependency domain as a head word. Such a situation arises when words in a domain are intersected by crossing branches, disfluencies, or single-word chunks. Words preceding or following such an intersection do not end up in the chunk. In the intermediate format, such words are annotated as being outside any chunk, unless they correspond to a single-word chunk.

Having identified the chunks in a sentence, their types can be determined. In general, the type of a chunk equals the phrasal type of its root node. According to Abney's definition this is the node corresponding to the highest constituent for which the head of the chunk is the head word. For single-word chunks, however, this method does not produce the correct chunk type. The chunk structure for such chunks consists of only the node for the head word and, consequently, this node is the root of the chunk structure. The label of such a node denotes a part of speech type, rather than a phrasal type that can serve as chunk type.

In the CGN, the actual root node of such a single-word chunk has been left out to achieve a flat annotation. Nevertheless, the type of this missing node can be inferred from the available information. In the CGN annotation, the label of a dependency domain is projected by its head. Thus, nouns give rise to an NP

domain, while prepositions give rise to PP domains. This property of the CGN annotation is used to create a table that associates each part of speech with the chunk type it projects. Then, the type of a single-word chunk is determined by consulting the table entry matching the part of speech of the head word.

Grammatical relations

Having found the heads and chunks, grammatical relations can be extracted from the CGN data. In contrast with the Penn Treebank format, the CGN annotation explicitly represents these relations. A dependency link between a daughter node and its mother domain corresponds to a grammatical relation between the head word of the daughter node and that of the mother domain. For the purpose of converting the CGN data to intermediate format, these correspondences have to be recorded explicitly.

In the intermediate format, grammatical relations are considered to be directed. The origin of a relation matches the head word of the daughter node and its target corresponds to that of the mother domain. As all the head words in a sentence have already been identified, the task that remains is, given a head word as origin, to find its grammatical relation as well as the target of this relation.

Two useful concepts for extracting grammatical relations from the CGN annotation are those of a projection and projection line of a node. A projection of a terminal node is a dependency domain of which the word corresponding to that terminal node is the head word. The maximal projection of a node, then, is the highest-level dependency domain that is a projection of that node. Finally, the projection line of a node is the path starting at that node containing all of its projections.

With these two concepts, the process of finding the grammatical relation originating at a given head word can be explained as follows. The projection line of the head word is followed upwards until its maximal projection is reached. Then, the grammatical relation to be found equals the dependency relation between this maximal projection and its mother domain. The target of this relation corresponds to the head word of that mother domain. It is found by following the projection line downwards from the mother domain, until a terminal node is reached.

As noted earlier, the procedure used to determine the head of a dependency domain sometimes designates several heads for one domain. As a result, following the projection line of a dependency domain downwards may end up in several words, rather than one. In such a case, all of these words are considered target of the grammatical relation. In the intermediate format, then, only one relation is recorded, but at the time instances are generated for the grammatical-relation-finding task, the relation causes as much instances to be generated as there are target words.

Chapter 3

Optimisation and evaluation

The previous chapter described the learning tasks making up the memory-based shallow parser framework, as well as the conversion of CGN data to a format that is suited for training the parser modules. These two procedures form the groundwork for the experiments that have been performed to optimise the parser framework and to evaluate its performance. A report of these experiments is given in this chapter. In section 3.1, the general approach to the experiments is presented. Next, the optimisation of the individual modules and the evaluation of the entire parsing cascade are described in sections 3.2 and 3.3, respectively.

3.1 Experimental setup

The central issue in this chapter is the experimental optimisation and evaluation of the Dutch memory-based shallow parser. The three main ingredients of the experiments are introduced in this section. Subsection 3.1.1 describes the machine-learning algorithms that are used and their parameters. The procedure followed to find an optimal parameter setting is discussed in subsection 3.1.2. Finally, subsection 3.1.3 presents the evaluation techniques used to measure the performance of the parser.

3.1.1 Parameter setting

The central goal of the experiments reported in this chapter is finding parameter settings for the various parser modules that yield optimal performance on their learning tasks. The performance of different parameter settings is evaluated by systematically varying the values of each parameter of a module. In general, two types of parameters can be distinguished. The first type relates to the properties of the learning task itself. Rather than fixing these in advance and only optimising parameters of the learning algorithm, part of the optimisation is aimed at improving the instance format in order to boost performance.

In this research, the optimisation of this type of parameters is given a restricted definition. Given the instance formats as described in section 2.1, the only parameters to be optimised are those describing the size of the local context of

focus words or chunks. The optimal settings of this parameter are expected to be highly language-specific and therefore optimal context sizes reported in the literature are likely to be suboptimal for spoken Dutch.

The second type of parameters comprises the algorithmic parameters of the machine learner that is used to train the parser module. The most important one of those is the choice for the machine-learning algorithm itself. All the remaining parameters are dictated by this choice. In this research, the algorithms that are tested are IB1, IGTree and TRIBL as implemented by the TiMBL software. A short introduction to these algorithms and their parameters is given in this section. For a thorough description as well as for implementation details, the TiMBL reference guide (Daelemans et al., 2002) should be consulted.

All three algorithms have in common that their instances are described by fixed-length vectors of feature-value pairs and a label describing the class the instance belongs to. Having been trained on a large number of example instances, the learning algorithm should be able to predict the class of a test instance that is consistent with the information in the training data. The technique that is used to determine this class is specific to each of the algorithms and, as such, gives rise to different parameters.

IB1

IB1 (Aha, Kibler, and Albert, 1991) is a variant of the classic memory-based k-Nearest Neighbour algorithm (Cover and Hart, 1967). Of the three algorithms used in this research, it is the purest representative of the memory-based learning method: every training instance is kept in memory and test instances are classified by drawing analogies with similar training instances. The conditions according to which two instances are to be considered similar and the method to derive correct classes from a number of similar instances are both configurable by setting the parameters of the algorithm.

Similarity in IB1 is defined by a distance between the test instance and an instance stored in memory. Equation 3.1 shows the function used by IB1 to compute this distance between instances X and Y, both described by a feature vector of length n.

$$\Delta(X, Y) = \sum_{i=1}^{n} w_i \delta_i(x_i, y_i) \tag{3.1}$$

In this equation, distance is defined as the weighted sum of the distances per feature. w_i signifies a weight factor for feature i and δ_i denotes a function for feature i that computes the distance per feature. In TiMBL, feature weights can either be turned off, that is w_i equals 1, or configured to be one of the following.

- Gain Ratio (Quinlan, 1993)
- Information Gain (Quinlan, 1986)
- Chi-squared (White and Liu, 1994)
- Shared Variance (White and Liu, 1994)

Possible functions for the distance per feature parameter in TiMBL are listed below.

- Overlap (Aha et al., 1991)
- Modified value difference (Stanfill and Waltz, 1986; Cost and Salzberg, 1993)
- Numeric difference

Using this distance metric, the classification of a test instance starts by finding the k nearest instances in the instance base. Next, the correct class for the test instance is derived by applying a voting mechanism to these nearest neighbours. The classic voting mechanism, called majority voting, assigns the class most frequently found in the neighbouring instances, to the test instance. With this mechanism, each instance has equal importance, even though some may be more similar to the test instance than others. For this reason, other voting mechanism have been proposed in which the vote of an instance is weighted by a function of its distance to the test instance. In TiMBL, the following voting mechanisms have been implemented.

- Majority voting
- Inverse distance (Dudani, 1976)
- Inverse linear (Dudani, 1976)
- Exponential decay (Shepard, 1987)

IGTree

The IB1 algorithm has two important drawbacks, both caused by the fact that it defers most of its work until a test instance is classified. First, storing every training instance in memory causes the memory requirements for the algorithm to be very high. Additionally, having to search through the entire instance base for instances that are similar to a given test instance results in higher processing times than for eager learning algorithms. Although both of these drawbacks are dealt with by optimised implementations, IB1 remains an algorithm with high resource requirements, which, in some situations, might not be desirable.

IGTree (Daelemans, Van den Bosch, and Weijters, 1997) is a heuristic approximation of IB1 aiming to overcome these drawbacks, while still retaining acceptable performance. The algorithm stores its training instances in a decision tree structure, in which each level corresponds to a test on one of the features and the branches of nodes on a level are labelled with the possible feature values. A training instance, then, is encoded by a path from the root node to one of the leaf nodes. This path, however, contains only as much nodes, or feature tests, as necessary to distinguish uniquely this instance from all the other instances in the training set. As a result, the instance base of the IGTree algorithm consumes less space if features with high informative value are stored early in the tree. In order to benefit from this property and achieve high compression, IGTree uses the feature weights that were introduced with IB1 to sort instance features according to their informative value.

Classification in IGTree comes down to simply following the path starting at the root node, of which the edge labels correspond to the feature values of the test instance. In contrast, IB1, theoretically, has to search the entire instance base for instances similar to the test instance. The faster classification procedure of IGTree, however, does come at the expense of generalisation accuracy. Nevertheless, IGTree shows adequate performance while keeping memory requirements and processing time low.

TRIBL

As made clear by the previous two sections, IB1 achieves good generalisation accuracy at the expense of high memory requirements and long classification times, while IGTree manages to limit both the memory requirements and classification times, though shows lesser accuracy. The choice for either IB1 or IGTree, therefore is a choice between the quality of classification results and resource efficiency; no trade-off between the two is possible.

The TRIBL algorithm combines IB1 and IGTree and thereby the advantages of both algorithms. A configurable number of features that have the highest informative value according to a chosen feature weight is stored in an IGTree. Unlike in standard IGTree, however, in TRIBL the leaf nodes do not denote a class but correspond to an IB1 instance base comprising all the training instances that are consistent with the feature tests on the path to this leaf node. For the remaining features, IB1 is applied to this instance base, which is now considerably smaller than a standard IB1 instance base.

3.1.2 Parameter-optimisation method

In this section, two choices with respect to the optimisation are discussed. First, the procedure that is used for finding optimal parameter settings is introduced. Secondly, the choice between global optimisation of the parsing cascade and local optimisation of individual parser modules is presented. Arguments for deciding upon the latter are given.

To determine optimal parameter settings for the parser modules, a number of experiments are conducted in which the various parameters of both the learning tasks and the learning algorithms are systematically varied. An important difficulty in experiments in which multiple parameters have to be optimised, is the possibility of interaction between parameters. Therefore, in order to find the optimal parameter setting, each possible combination of parameter values should be tested. However, given the size of the CGN data set, the cost of evaluating only one parameter setting makes such an exhaustive search infeasible for the experiments in this research.

The alternative approach taken in this thesis is best described as a heuristic hill-climbing search through the parameter space. Each parameter is, in turn, tested with all possible values, while the value of the other parameters remains fixed. The parameter value showing best performance in these tests is selected and kept fixed for the rest of the optimisation. As is a well-known property of hill-climbing search, this will most likely not end up in a global optimum. Instead, it enables a trade-off between quality of the results and cost of the optimisation procedure.

The second choice discussed in this section, is about whether to optimise the global performance of the entire parsing cascade or only the local performance of the individual parser modules. In section 2.1, it was already mentioned that the memory-based shallow parsing framework can be approached from two different points of view. These two approaches reflect upon the optimisation procedure.

On the one hand, the parser is composed of three separate modules that can each be considered individual units. As a result of this modularity, each module can be optimised and tested in isolation, in which case the data on which the module is trained and tested, consist of gold-standard, that is, perfect corpus data. The advantage of optimising each module in isolation is that errors in the output of the module are known to be caused by the module itself, not by any side-effect as a result of the module being part of a larger whole.

On the other hand, the parser modules are interdependent parts of the global framework and their performance may well be highly dependent upon each other. For example, both the PNP finder and the grammatical-relation finder use information about chunks, which, unless gold-standard corpus data are used as input, is generated by the chunking module. Consequently, errors made by the chunker will affect the performance of the other two modules.

The above suggests that global performance of the entire parsing cascade is not a straightforward result of the performance of each of its modules. Interdependencies between modules are an important factor in the actual performance of the parser. Therefore, in order to optimise global parsing performance, the modules should be optimised as part of the framework from the start, rather than optimising each of them separately and only then joining them. For example, the PNP finder module could be optimised on input that is generated by the chunker. In this case, it would "learn" which kinds of errors the chunker makes and, thereby become more robust to these errors. Modules optimised in this way, would probably be suboptimal when tested in isolation, but would perform better in the parsing cascade. Van den Bosch (1997) shows that such an optimisation strategy, in which modules are trained on the output of preceding modules, is an effective countermeasure against cascading errors, that is, errors that can be traced back to faulty output of preceding modules.

The advantages of this global optimisation do come at the expense of transparency. Observed errors are not as easily attributed to a single module as for local optimisation, which makes analysis of the separate learning tasks more difficult. For this reason, in this thesis, optimisation is only performed on the level of modules. The central research goal is testing the effectiveness of a memory-based approach to parsing on a corpus of spoken Dutch. This goal will benefit more from transparency than from better performance.

3.1.3 Evaluation method

During the experiments, evaluation is used for two different purposes. First, while optimising the parser modules, test results are evaluated in order to compare the performance of two parameter settings. Secondly, after optimal parameter settings have been found for the individual parser modules, the performance of the entire parsing cascade has to be measured. Such an evaluation of the performance of the memory-based shallow parser on spoken Dutch will lay the groundwork for answering the central research question of this thesis.

In this section, three performance measures that are often used in combination with linguistic data, are described. These measures are used to quantify the parsing performance. To actually evaluate the effectiveness of an approach, some kind of reference score should be available. For this purpose, baseline experiments are performed; they are also introduced in this section. Finally, at the end of this section, some tools are presented with which the statistical significance of results is tested.

Performance measures

For evaluating a parser module or the parsing cascade, measures are needed that can quantify their performance. In combination with linguistic data, the performance measures generally in use are precision and recall, which are well-known measures from information retrieval. In this research, they are applied to the results of all three parsing tasks, that is tagging/chunking, PNP finding, and grammatical-relation finding. Precision measures the percentage of syntactic units found by a module that are correct. Recall corresponds to the percentage of syntactic units present in the corpus that are found by a module. In these definitions, a syntactic unit corresponds to a chunk, a PNP and a grammatical relation, respectively. A chunk is considered to be correct if 1) both of its boundaries match those in the corpus, 2) it has the same type as the chunk in the corpus and 3) the head word is the same as in the corpus.

In addition to precision and recall, another performance measure that is often used is the F-score (Van Rijsbergen, 1979), which equals the harmonic mean of the precision and recall scores. The function to compute this score is shown in equation 3.2.

$$F_\beta = \frac{(\beta + 1) \cdot precision \cdot recall}{\beta^2 \cdot precision + recall} \tag{3.2}$$

β denotes the relative importance attached to the precision and recall scores. In this research, it equals 1, which means that both scores are considered equally important. The F-score measure is useful in optimisation experiments, in which different parameter settings are to be compared. Rather than basing this comparison on two measures, only one figure has to be taken into consideration.

Baseline performance

Using the precision, recall and F-score measures, the performance of different parameter settings can be quantified. However, quantified results only are not very informative with respect to the effectiveness of a certain approach. They become meaningful when related to an estimate of the difficulty of the learning task that has been performed. Therefore, experimental results of the system to be evaluated are compared to those of another system performing the same task.

Theoretically, this other system could be another research result of which performance measures have been published. In this case, however, the learning task on which it has been tested should be exactly the same as the one used to test the system that is evaluated. Often, such research results are not available. For this reason, baseline experiments are performed. Such a baseline corresponds to the simplest classifier that is able to perform the given learning task.

Its performance is used as a reference for the difficulty of the task. A machine learner, then, is considered to be effective if it performs at least significantly better than the baseline score for the learning task.

For all three learning tasks that are evaluated in this research, baseline experiments have been proposed in parsing literature. They will be used to assess the quality of the memory-based shallow parser modules for spoken Dutch. A description of each baseline is given in the sections that report on the optimisation and evaluation of the corresponding parser modules.

Statistical significance

In the experiments, instances extracted from the CGN corpus serve two different purposes. On the one hand, they are used as example data when training the parser modules. On the other hand, instances are also needed as test cases for measuring the generalisation accuracy of a trained module. Those instances used for training and those for testing must not overlap, since classifying an instance that is a member of the training set does not require generalisation, but merely reproduction of training experience. This is certainly the case with memory-based learning, which stores every training instance in memory. Therefore, techniques for evaluating machine-learning applications divide the data set into disjoint training and test sets.

The technique used in this research is k-fold cross-validation. In this technique, the data set is divided into k disjoint subsets of approximately equal size. Then, k experiments are performed, each time using a different subset as test set and the other $k-1$ subsets as training set. The measurements of these runs are averaged into one score for each one of precision, recall and F-score.

The results of two k-fold cross-validation experiments, for example two different parameter settings or a memory-based parser module and a baseline score, are tested by means of a one-tailed paired t-test. Rather than directly comparing the scores of two different experiments, a t-test decides whether two performance measures are significantly different. Given that one parameter setting is found to be significantly better than another, one can be fairly confident that it will also perform better on new, unseen test instances.

3.2 Optimisation of the parser modules

In the previous section, the general approach to optimising the memory-based shallow parser has been described. As part of this approach, it has been decided to optimise each parser module in isolation, rather than optimise the parsing cascade as a whole. With respect to the central research question of this thesis, a modular system in which the role of each learning task can be easily recognised, is preferable to one that is more accurate but tightly coupled. Therefore, in this section, each of the three parser modules is individually optimised for performance on gold-standard corpus input. In section 3.3, they are put together to form the actual memory-based shallow parser.

Each of the following subsections reports the systematic experimental optimisation of one of the parser modules. The starting point of these experiments is the default parameter setting of TiMBL, which is depicted in table 3.1. Each of these parameters is, in turn, tested with every possible value, while using the

Parameter	Setting
Learning algorithm	IB1
Feature weighting	Gain Ratio
Distance metric	Overlap
Number of nearest neighbours	1
Voting mechanism	Majority voting

Table 3.1: Default parameter setting.

current setting for the remaining parameters. The best performing setting for the parameter under consideration is then selected for future use.

As the experimental optimisation aims at finding a parameter setting that is optimal with respect to generalisation performance, the role of the learning algorithm parameter in it is a different one from that of the other parameters. As a consequence of their design, both TRIBL and IGTree will most likely be outperformed by IB1. They have been devised as alternatives to IB1 that run faster at the expense of generalisation accuracy. Therefore, initially, all optimisation experiments are performed using IB1 as the learning algorithm, thereby determining the potential of the Dutch memory-based shallow parser with respect to generalisation accuracy. In this phase, the focus is on generalisation performance; no attention is paid to run-time performance.

However, different uses for a parser might have to meet different requirements and, consequently the trade-off between generalisation performance and run-time performance is a different one for each case. For this reason, after an optimal parameter setting has been determined for the IB1 algorithm, TRIBL and IGTree are used to measure both the generalisation performance and the run-time performance of various trade-offs between the two.

3.2.1 Optimisation of the tagger/chunker

In this section, various parameter settings for the tagger/chunker module are tested in order to determine an optimal setting for the tagging/chunking task. Having found such a parameter setting, the quality of the module is evaluated. For this evaluation, it is important to have a reference score to which to compare the obtained results. For this reference a baseline score is computed. A baseline often used for tagging tasks, labels a word with the tag that is most frequently associated with it in the training data. If the word is not present in the training data, the tag most frequently encountered overall is assigned. When applied to the tagging/chunking task on the CGN data, this results in a precision of 64.62, a recall of 62.14 and an F_β of 63.36.

What follows is a review of the results obtained by performing the optimisation procedure described in subsection 3.1.2 on the tagging/chunking task. The main goal of this optimisation is selecting a configuration that makes optimal use of the available information. Finding out why certain parameter settings perform better than others is not part of this goal. Therefore, the scores of different parameter settings will only be shortly commented upon.

Size of the local context Feature vectors for the tagging/chunking task consist solely of features describing the local context of the focus word, that is the

left	right	precision	recall	F_β
6	4	77.98	79.05	78.51
5	3	78.45	79.64	79.04
4	2	79.07	80.45	79.75
3	2	79.16	80.60	79.87
3	1	79.25	81.12	80.18

Table 3.2: Performance on the tagging/chunking task using different local context sizes.

	precision	recall	F_β
No weighting	68.46	70.70	69.56
Gain ratio	79.25	81.12	80.18
Information gain	79.69	81.42	80.54
Chi-square	78.52	80.69	79.59
Shared variance	79.03	81.32	80.16

Table 3.3: Performance on the tagging/chunking task using different feature weighting schemes.

words directly preceding and those directly following it in the sentence. The size of this context is therefore an important factor in the performance of the tagger/chunker: if it is too small, the context may not be sufficiently informative for a good classification of the focus word, but if it is too large, performance may be harmed by the presence of irrelevant features.

Daelemans et al. (1999) report good results on a corpus of written English with a context of five words to the left and three words to the right. This context size is adopted as a starting point for the experiments. In addition, one larger context size and three smaller ones are also tested. The scores for these context sizes are presented in table 3.2.

Remarkably, all of the smaller context sizes perform better than the starting point. Furthermore, the larger context scores worse than all the other. The context of three words to the left and one word to the right is selected for the following experiments.

Feature weighting The feature weighting parameter selects a weighting scheme that computes the weight of a feature in the distance calculation. The default parameter setting uses Gain-ratio weighting. Table 3.3 shows the performance changes for different weighting schemes. Information gain performs better than the default, Gain ratio, which is second best. Assigning equal weight to all features performs considerably worse than all the other options. Information-gain weighting will be used in the following experiments.

Distance metric The distance metric calculates the distance in a single feature between two instances. The default distance metric for IB1 is Overlap. This metric defines the distance between two feature values to be 1 if they are equal, or 0 otherwise. In contrast, the Modified value difference metric (MVDM) distinguishes degrees of similarity, based on the co-occurrence of values with

	precision	recall	F_β
Overlap	79.69	81.42	80.54
All MVDM	82.27	83.17	82.72
Words MVDM	80.76	81.95	81.35
Tags MVDM	80.87	82.40	81.63

Table 3.4: Performance on the tagging/chunking task using different distance metric configurations.

	precision	recall	F_β
1	82.27	83.17	82.72
3	83.59	85.53	84.55
5	83.62	85.86	84.72
7	83.55	85.92	84.71
9	83.39	85.85	84.60
11	83.32	85.85	84.56
13	83.16	85.71	84.42

Table 3.5: Performance on the tagging/chunking task using different numbers of nearest neighbours.

target classes. MVDM is particularly suited for linguistic data, in which, for example, two nouns should obviously be considered more similar than a noun and a verb.

Distance metrics are configured on a per-feature basis. Table 3.4 gives an overview of the scores of four different configurations: all features are assigned the Overlap metric, all features are assigned the MVDM metric and finally two configurations in which either the word features or the tag features are assigned the MVDM metric and the remaining features the Overlap metric.

The results reveal that the configuration in which all features are assigned the MVDM metric outperforms those that only assign it to either words or tags. In turn, those configurations score visibly better than the Overlap metric assigned to all features. The following experiments will use the MVDM metric for all features.

The number of nearest neighbours Thus far, the target class assigned to an instance is the class of the most similar instance in the instance base. In a series of experiments, the number of nearest neighbours on which the classification of a test instance is based, is gradually augmented. In table 3.5 the performance of various numbers of neighbours is presented. Taking the five nearest neighbours into consideration when determining the class of test instance scores best, and therefore, this setting is selected for the remaining experiments.

Voting mechanism The voting mechanism specifies how the target class for a test instance is derived from the multiple nearest neighbours that have been configured in the previous optimisation step. Majority voting, the default mechanism, chooses the class that is most frequently present among the nearest neighbours. Other mechanisms attach different importance to the classes of the neighbours, depending upon their distance to the test instance.

	precision	recall	F_β
Majority	83.62	85.86	84.72
Inverse distance	83.91	85.91	84.89
Inverse linear	83.27	84.59	83.92
Exponential decay, $\alpha = 10$	82.82	83.98	83.39
Exponential decay, $\alpha = 20$	82.38	83.37	82.87
Exponential decay, $\alpha = 30$	82.22	83.11	82.66
Exponential decay, $\alpha = 40$	82.17	82.99	82.58

Table 3.6: Performance on the tagging/chunking task using different voting mechanisms. The α parameter for the Exponential decay mechanism is a constant determining the slope of the decay function.

Algorithm	precision	recall	F_β	words/sec
IB1	83.91	85.91	84.89	29
TRIBL, $q = 1$	82.75	84.80	83.77	125
TRIBL, $q = 2$	82.41	84.53	83.46	174
TRIBL, $q = 3$	78.00	79.30	78.64	615
TRIBL, $q = 4$	77.34	78.72	78.03	470
TRIBL, $q = 5$	77.26	78.66	77.95	471
IGTree	77.03	78.47	77.74	2923

Table 3.7: Performance on the tagging/chunking task of different trade-offs between classification speed and accuracy, established by using the IB1, IGTree and TRIBL algorithms.

Table 3.6 shows test results for the various voting mechanisms. Remarkably, only Inverse-distance voting performs better than majority voting. The other voting mechanisms score considerably worse. For this reason, Inverse-distance voting is selected for future use.

Learning algorithm The parameter setting obtained in the preceding optimisation steps has been tuned for optimal generalisation accuracy. With respect to the trade-off between generalisation accuracy and run time, all focus has been directed at accuracy. This might, however, not be the suitable trade-off for every application of the parser. To shed some light on the consequences of various different trade-offs, the TRIBL algorithm has been applied to the tagging/chunking task using the parameter setting found above and a number of threshold values q to switch from IGTree to IB1 behaviour. In addition to the three performance measures used earlier, the classification speed in terms of words per second is also measured.

As can be seen in table 3.7, the differences in speed between the three algorithms are considerable. Nevertheless, this increase in speed does not always negatively affect generalisation performance. TRIBL with $q = 1$ runs considerably faster than IB1, being only slightly less accurate. The same is true for IGTree in comparison with TRIBL and $q = 5$.

left	right	precision	recall	F_β
1	1	97.09	97.83	97.45
2	1	97.19	98.00	97.59
2	2	97.07	97.86	97.46

Table 3.8: Performance on the PNP finding task using different local context sizes.

	precision	recall	F_β
No weighting	95.78	98.20	96.97
Gain ratio	97.19	98.00	97.59
Information gain	97.26	97.94	97.60
Chi-square	97.25	98.02	97.63
Shared variance	97.25	98.02	97.63

Table 3.9: Performance on the PNP finding task using different feature weighting schemes.

3.2.2 Optimisation of the PNP finder

The optimisation of the PNP finding module is performed in the same way as the optimisation of the tagger/chunker module described in the previous section. First, a baseline score for the PNP finding task is determined. This score will be used as a reference point for assessing the quality of the optimal memory-based PNP finder. The baseline experiment for PNP finding is straightforward but already quite good. Predicting a PNP chunk if and only if the NP directly follows the PP chunk, results in a precision of 98.23, a recall of 92.44 and an F_β of 95.24.

Next, an optimal parameter setting for performing the PNP finding task is searched for. Again, this search is performed by systematically varying the algorithmic parameters and selecting the best performing value for each parameter. The intermediate steps in this procedure are shortly commented upon in this section.

Size of the local context Three different context sizes have been tested. The performance for each context size is listed in table 3.8. The context of two words to the left and one to the right scores best, but the other two are only slightly worse. The following experiments will all use a context of two words to the left and one word to the right.

Feature weighting As can be seen in table 3.9, all feature-weighting schemes perform better than no feature weighting. Gain ratio, the default waiting scheme, scores second worse. Chi-square and Shared-variance weighting are the best performing schemes. From those two schemes, Chi-square weighting is arbitrarily selected for future use.

Distance metric The same distance metric assignments that were tested for tagging/chunking have also been tested for PNP finding, that is: Overlap for all features, MVDM for all features and two in which MVDM is assigned to words only or tags only, respectively, and Overlap to the remaining features.

	precision	recall	F_β
Overlap	97.25	98.02	97.63
Words MVDM	97.03	96.96	96.99
Tags MVDM	97.17	97.58	97.37
Words+tags MVDM	96.12	95.17	95.64
Distance numeric	97.00	97.61	97.30
Intervening numeric	97.15	97.55	97.35
Distance+intervening numeric	96.86	97.56	97.21
All MVDM	95.94	95.15	95.54

Table 3.10: Performance on the PNP finding task using different distance metric configurations.

	precision	recall	F_β
1	97.25	98.02	97.63
3	97.45	98.45	97.95
5	97.37	98.45	97.90
7	97.11	98.37	97.74
9	97.00	98.28	97.64
11	97.01	98.11	97.55
13	96.96	97.98	97.47

Table 3.11: Performance on the PNP finding task using different numbers of nearest neighbours.

Additionally, another configuration, in which the default metric is MVDM, but the distance and intervening PPs features are treated as numeric features, is tested. For numeric features, the distance is defined as a normalised difference between the two numbers.

Table 3.10 shows that, surprisingly, the best performing configuration is the one that assigns the Overlap metric to all features, which will therefore continue to be used in the following experiments. Also remarkable is the fact that the global MVDM metric performs worse of all. Specifically treating the distance and intervening PPs features as numeric does not improve performance.

The number of nearest neighbours The scores listed in table 3.11 reveal that considering the class of three nearest neighbours in classification performs best and that, as neighbours are added or removed, the performance deteriorates. For this reason, all remaining experiments will use three nearest neighbours.

Voting mechanism The voting mechanisms and their corresponding performance are presented in table 3.12. Inverse-distance voting is the best performing mechanism. Simple majority voting outperforms all remaining options, of which the exponential-decay variants are the four worse performing. The following experiments will use Inverse-distance voting.

Learning algorithms As was done for chunking, the trade-off between speed and accuracy has been explored by applying IGTree and TRIBL to the PNP finding task. The results of these tests, presented in table 3.13 show that the

	precision	recall	F_β
Majority	97.45	98.45	97.95
Inverse distance	97.48	98.48	97.98
Inverse linear	97.26	98.21	97.73
Exponential decay, $\alpha = 10$	92.50	99.57	95.90
Exponential decay, $\alpha = 20$	92.50	99.57	95.90
Exponential decay, $\alpha = 30$	92.50	99.57	95.90
Exponential decay, $\alpha = 40$	92.50	99.57	95.90

Table 3.12: Performance on the PNP finding task using different voting mechanisms. The α parameter for the Exponential decay mechanism is a constant determining the slope of the decay function.

Algorithm	precision	recall	F_β	instances/sec
IB1	97.48	98.48	97.98	247
TRIBL, $q = 1$	97.50	98.48	97.99	233
TRIBL, $q = 2$	97.61	97.30	97.45	1008
TRIBL, $q = 3$	97.63	97.26	97.45	1028
TRIBL, $q = 4$	97.57	97.30	97.43	988
TRIBL, $q = 5$	97.73	96.61	97.16	2057
IGTree	98.23	92.44	95.24	4114

Table 3.13: Performance on the PNP finding task of different trade-offs between classification speed and accuracy, established by using the IB1, IGTree and TRIBL algorithms.

baseline performance is only slightly outperformed by the best scoring machine learning implementations. As for precision, the baseline score is even better than six out of seven learning algorithms; only the precision of IGTree is comparable to the baseline performance. However, the recall rate of IGTree also matches that of the baseline, while the other algorithms perform considerably better on this measure, which makes them the better performing implementations overall. Among these configurations, the two best performing are IB1 and TRIBL with a threshold value of 1, which perform equally well; as do the TRIBL variants with thresholds from 2 to 4.

3.2.3 Optimisation of the grammatical-relation finder

In this subsection, the grammatical-relation finder is trained and optimised to find three types of relations: subject, direct object, and indirect object. In the CGN annotation these relations correspond to dependencies of types SU, OBJ1 and OBJ2, respectively. The optimisation procedure performed for the grammatical-relation finder is the same as for the tagger/chunker and the PNP finder.

Before the optimisation is described, a baseline score is established. Buchholz (2002) proposes a baseline experiment that always predicts the class most frequently associated in the training data with the value of the distance feature

left	right	precision	recall	F_β
2	1	82.81	85.28	84.03
1	1	83.44	84.44	83.94
2	2	83.77	86.12	84.93
3	3	83.38	86.03	84.68

Table 3.14: Performance on the grammatical-relation-finding task using different local context sizes.

	precision	recall	F_β
No weighting	78.91	85.97	82.28
Gain ratio	83.77	86.12	84.93
Information gain	79.15	85.85	82.37
Chi-square	76.33	85.18	80.51
Shared variance	79.34	85.88	82.48

Table 3.15: Performance on the grammatical-relation-finding task using different feature weighting schemes.

of the test instance. When applied to the CGN data this strategy results in predicting SU if the distance is 1 or -1 and no relation for all other distances. The precision rate obtained by this baseline is 63.88; its recall is 57.84 and its F_β, 60.71.

The remainder of this section contains a survey of the intermediate steps in the optimisation of the grammatical-relation finder.

Size of the local context Buchholz (2002) describes the optimisation of the memory-based grammatical-relation finder for written English and reports an optimal context size of two chunks to the left and one chunk to the right of the focus chunk. This context size, together with one smaller and two larger context sizes have been tested with the CGN data.

Table 3.14 shows that the two larger context sizes yield better performance than the one from Buchholz (2002), which is, itself, more accurate than a context of only one chunk both to the left and to the right of the focus chunk. Hence, a context of two words to the left and two words to the right is selected for the following experiments.

Feature weighting Gain-ratio weighting is by far the best performing feature weighting scheme on the grammatical-relation-finding task, as can be seen in table 3.15. In contrast, the Chi-square weighting scheme shows the worse accuracy; the other schemes are in between. Gain-ratio weighting will continue to be used in the remaining experiments.

Distance metric The same distance metric assignments as were tested for PNP finding have been applied to grammatical-relation finding. The results of these settings are presented in table 3.16. It can be seen that, unlike the results for PNP finding, the global Overlap assignment is not among the best performing options for grammatical-relation finding. As for the numeric features, explicitly marking the intervening verbs feature does result in good performance, while

	precision	recall	F_β
Overlap	83.77	86.12	84.93
All MVDM	86.66	84.84	85.74
Tags MVDM	85.07	85.15	85.11
Words MVDM	84.43	84.55	84.49
Distance numeric	84.50	85.23	84.86
Intervening numeric	86.66	84.84	85.74
Distance+intervening numeric	84.50	85.23	84.86

Table 3.16: Performance on the grammatical-relation-finding task using different distance metric configurations.

	precision	recall	F_β
1	86.66	84.84	85.74
3	89.18	85.87	87.49
5	89.84	85.99	87.87
7	90.23	85.90	88.01
9	90.41	85.84	88.06
11	90.49	85.70	88.03
13	90.54	85.55	87.97

Table 3.17: Performance on the grammatical-relation-finding task using different numbers of nearest neighbours.

this is not the case for the distance feature. Overall, the best performance is obtained by the global MVDM metric, which will therefore be used in all following experiments.

The number of nearest neighbours Table 3.17 reveals that examining nine nearest neighbours in order to classify a test instance leads to the best performance, which is far better than that for one neighbour classification. However, the other options listed in the table are not drastically worse. In future experiments, nine nearest neighbours will be used.

Voting mechanism As demonstrated by table 3.18, all weighted voting mechanisms except Inverse-linear voting improve upon the performance of the default setting, that is, majority voting. The best performing parameter setting uses Inverse-distance voting. Furthermore, all exponential-decay variants score approximately the same. Inverse-distance voting is selected as best performing voting mechanism.

Learning algorithm The trade-off between speed and accuracy implemented by the IB1 and IGTree algorithms, is illustrated by table 3.19. Surprisingly, TRIBL with a threshold of 4 is the best performing setting, thereby outperforming both IB1 and TRIBL with $q = 1$, 2, and 3. The results of the other settings show the usual situation: accuracy deteriorates and speed rises as the threshold parameter of TRIBL increases.

	precision	recall	F_β
Majority	90.41	85.84	88.06
Inverse distance	90.56	85.94	88.19
Inverse linear	89.73	86.25	87.96
Exponential decay, $\alpha = 10$	90.40	85.96	88.12
Exponential decay, $\alpha = 20$	90.40	85.96	88.12
Exponential decay, $\alpha = 30$	90.40	85.97	88.13
Exponential decay, $\alpha = 40$	90.42	85.99	88.15

Table 3.18: Performance on the grammatical-relation-finding task using different voting mechanisms. The α parameter for the Exponential decay mechanism is a constant determining the slope of the decay function.

Algorithm	precision	recall	F_β	instances/sec
IB1	90.56	85.94	88.19	58
TRIBL, $q = 1$	90.56	85.94	88.19	60
TRIBL, $q = 2$	90.56	85.94	88.19	63
TRIBL, $q = 3$	90.56	85.95	88.19	81
TRIBL, $q = 4$	90.85	86.25	88.49	272
TRIBL, $q = 5$	90.47	85.47	87.90	497
IGTree	77.96	71.82	74.76	4089

Table 3.19: Performance on the grammatical-relation-finding task of different trade-offs between classification speed and accuracy, established by using the IB1, IGTree and TRIBL algorithms.

3.3 Evaluation of the parsing cascade

Thus far, the three parser modules have been treated as individual units, each of them being independent of the other two. Section 2.1 has described the learning tasks devised for the parser modules in terms of certain presupposed input information. In the previous section, memory-based implementations of these learning tasks have been trained and optimised using gold-standard corpus data to provide this information. However, in practical parsing applications, the only information actually available are the raw words, grouped in sentences. The PNP finder and the grammatical-relation finder, which require chunk information for their input, are therefore unable to operate in isolation.

To resolve the input requirements of the parser modules, they are linked together in a cascade in which output of one module is used to construct input vectors for the following modules. As part of such a parsing framework, the modules are no longer the independent units as they have been presented throughout the preceding chapters. Rather, the performance of one module may be tightly related to that of the modules responsible for providing its input. For example, in the memory-based parsing cascade, the quality of the output of the grammatical-relation finder is as much affected by the accuracy of the tagger/chunker and the PNP finder, as it is by the accuracy of the grammatical-relation finder itself.

As made clear by the above, the individually optimised parser modules from the previous section alone are not sufficient to answer the research question. An evaluation of the effectiveness of the memory-based shallow parsing method on spoken Dutch should be based on the performance of the entire parsing framework, rather than on the scores collected in the previous section. For this reason, the experiments in this thesis are concluded by evaluating the performance of the final Dutch memory-based shallow parser.

This section reports on the evaluation of three different variants of the memory-based shallow parser. The first variant corresponds to the standard parsing cascade consisting of the tagger/chunker, PNP finder and grammatical-relation finder. The other two try to improve the performance of the parser by filtering disfluencies, which are phenomena typical of spoken language, from the input sentences. The performance of the standard parsing cascade is reviewed in subsection 3.3.1. After that, the value of adding disfluency filtering to the parsing framework is evaluated in subsection 3.3.2.

3.3.1 Evaluating the standard parsing cascade

The memory-based parsing cascade consists of the three parser modules optimised earlier in this chapter, linked in sequence. Sequential execution of the modules is important, since both the PNP and grammatical-relation finder require information that is only available after the tagger/chunker is finished; in addition, the grammatical-relation finder depends on the PNP finder for information about PNP chunks.

For these experiments three different versions of the parsing cascade have been tested, based on different parameter settings. One parsing cascade is formed by using the optimal parameter settings found for the parser modules; this parser produces the most accurate results. Another cascade combines the three modules, this time implemented by the IGTree algorithm, which produces

	tagger/chunker			PNP finder			relation finder		
	prec.	recall	F_β	prec.	recall	F_β	prec.	recall	F_β
gold-standard	83.91	85.91	84.89	97.48	98.48	97.98	90.56	85.94	88.19
optimal	83.91	85.91	84.89	86.15	90.71	88.37	83.76	79.11	81.37
fastest	77.03	78.47	77.74	81.89	78.95	80.40	68.07	62.29	65.05
trade-off	82.41	84.53	83.46	84.88	88.22	86.52	83.07	77.94	80.42

Table 3.20: Performance of the parser modules as part of the memory-based shallow parsing cascade. For comparison, the performance of the modules tested in isolation is also listed.

the fastest running version, and, finally, a reasonable trade-off between speed and accuracy is built using TRIBL-based modules.

The performance scores for all three versions have been collected in table 3.20, which shows the performance of the individual modules tested on gold-standard input data, too. The results clearly show that the performance of the tagger/chunker has an important impact on the performance of the other two modules. With optimal settings, both score almost ten and seven percent worse in comparison with their performance in isolation.

3.3.2 Improving performance by disfluency filtering

The memory-based shallow parser that has been constructed and evaluated in the previous section consists of the same three modules as are used by Buchholz et al. (1999) for parsing written English. Although using the exact same framework for spoken Dutch does illustrate its generality and robustness, it is expected that performance can be improved if the default framework is adapted to the specific properties of the target data. Such a property of the CGN data that is the central issue in this section, is its spoken nature. An important property of spoken language is the existence of a variety of phenomena that are considered to be noise, that is, data that do not contain any informative value with respect to the correct classification. In contrast, written language hardly ever contains such noise.

More specifically, the type of noise the optimisations in this section try to counteract are disfluencies. In the CGN annotation disfluencies are represented as single nodes that are not connected to the rest of the syntactic tree. They include such phenomena as laughter and fragmented or repeated words. In the default parsing cascade, disfluencies are treated as any other word and therefore may end up in the local context of instances for all of the learning tasks. However, considering that they most likely do not add any informative value to the local context of a chunk, it is expected that filtering them out of the data could improve the performance of the individual modules and the entire parser in general. Two approaches to disfluency filtering have been evaluated, one of which can easily be added to the existing parser, whereas the other would require an additional classification module.

The first approach to filtering of disfluencies uses information generated by the tagger/chunker to filter interjections, a restricted subclass of disfluencies, from the input to the PNP finder and the grammatical-relation finder. In the

	tagger/chunker			PNP finder			relation finder		
	prec.	recall	F_β	prec.	recall	F_β	prec.	recall	F_β
gold-standard	83.91	85.91	84.89	97.48	98.48	97.98	90.56	85.94	88.19
standard	83.91	85.91	84.89	86.15	90.71	88.37	83.76	79.11	81.37
interjection filter	83.91	85.91	84.89	86.17	90.70	88.37	83.81	79.28	81.48
disfluency filter	85.50	86.61	86.05	88.04	92.39	90.17	85.66	80.48	82.99

Table 3.21: Comparison of the performance scores for the standard memory-based shallow parsing cascade and two parsing cascades to which some form of disfluency filtering is added.

CGN annotation, interjections are assigned the TSW part of speech tag. Consequently, they are detected by the tagger/chunker, the same way nouns or verbs are detected. This information can be exploited by inserting an interjection filter between the tagger/chunker and the PNP finder that removes all TSW-tagged words that are not part of a chunk.

The second approach does not use information that is already generated by one of the existing modules, but instead, requires an additional classifier to be added to the parsing cascade. Lendvai, Van den Bosch, and Krahmer (2003) describe a memory-based classifier that detects any type of disfluency present in the CGN. An important property of this classifier is that it does not require any other information than what can directly be extracted from the raw input text. For this reason, it can be inserted as the very first step in the parsing cascade, thereby not only improving the input to the PNP- and grammatical-relation-finder modules, but also that to the tagger/chunker. As was shown in the previous section, improving the performance of the tagger/chunker might also have an important effect on the performance of the other two modules.

For the experiments reported in this section, the classifier described by Lendvai et al. (2003) has not actually been implemented. Rather, the input data to the tagger/chunker is preprocessed based on the information available in the CGN. This has the effect of an infallible disfluency detector. Undeniably, this decision causes the results to turn out too positive with respect to those that would result from using a real disfluency classifier. However, the scores obtained by this experiment do provide an approximate upper-bound for the actual performance.

The effect that both filters have on the performance of the parsing cascade is presented in table 3.21. For comparison the scores of the standard parsing cascade and those of the individual modules tested in isolation on gold-standard input data, are also listed. The interjection filter does not significantly improve the accuracy of the PNP finder. The improvement of the grammatical-relation finder is significant, but only small. In contrast, the cascade with disfluency filter does show a significant improvement for all three modules with respect to the unfiltered parser. It should be kept in mind, however, that these scores have been obtained by applying a perfect disfluency filter to the input data.

Chapter 4

Discussion

Chapter 3 reported on the experimental optimisation of the three memory-based parser modules. Additionally, the performance of the complete memory-based shallow parser, obtained by applying the three modules in sequence, has been reviewed. In this chapter these findings are related to the central research question of this thesis, that is, to what extent can the memory-based shallow parsing method be employed to parse spoken Dutch.

The answer to this research question is divided into two parts. First, it is demonstrated that the Dutch memory-based parser of the previous chapter has successfully learned to parse spoken Dutch. Successful learning, in this sense, is assessed by comparing the performance measures that have been collected in the previous chapter with the minimum requirements that could be expected from a trained system.

Secondly, the performance of the Dutch memory-based shallow parser is compared with that of its English equivalent. Rather than verifying successful learning this comparison relates the performance of the Dutch parser to the potential of memory-based parsing that has been illustrated on written English. As a result of this, speculations with respect to the possible cause of observed differences are given and improvements to the Dutch parser are proposed.

4.1 Assessing parser performance

To decide whether a memory-based parser module has successfully learned its task, the conditions a successfully learned module should fulfil, have to be defined. For this purpose the baseline experiments that have been introduced in chapter 3 are used. These baselines have straightforward classification procedures, based on naive task representations. The tagging/chunking baseline assigns a tag to a word solely based on the word form itself. Likewise, the baselines for the PNP finding and grammatical-relation finding tasks use nothing but the distance between PP or verbal chunk and the focus chunk to predict the correct target relation.

In contrast to these baselines, the memory-based parsing task representations used in this thesis are much more complex. Their design has been based on linguistic knowledge to select properties of the input data that are expected to be good predictors for the target class and include these as features in the

	precision	recall	F_β
tagging/chunking baseline	64.62	62.14	63.36
tagging/chunking mbsp	83.91	85.91	84.89
pnp finding baseline	98.23	92.44	95.24
pnp finding mbsp	97.48	98.48	97.98
gr finding baseline	63.88	57.84	60.71
gr finding mbsp	90.56	85.94	88.19

Table 4.1: Comparison of the performance scores of the baseline experiments and the best performing memory-based classifiers for the three shallow parsing subtasks.

instance representation. As for all three learning tasks in this thesis, the most important contribution from linguistics is the finding that the local context of a word or chunk is an important predictor for its syntactic type or function. Therefore, an encoding of this local context makes up the largest part of the feature vectors.

Deciding whether a certain learning task representation enables a task to be learned, then, is a question of whether the extra information leads to an improvement in performance with respect to the baseline score. Such an improvement ascertains that the features used in the instance representation are useful for performing the task. From that, it can be concluded that the task has been successfully learned.

In table 4.1, the baselines scores for the three parsing subtasks are compared with the optimised memory-based classifiers that have been trained. It can be seen that all three memory-based classifiers improve upon the baseline score. As could be expected, the improvement achieved by the tagger/chunker and by the grammatical-relation finder is much greater than the improvement observed for the memory-based PNP finder. Detecting PNP chunks is a relatively easy task that can already be performed well with only minimal information, as is shown by the baseline score.

In fact, the precision rate of the baseline PNP finder is higher than that of the memory-based PNP finder. This can be explained by the fact that almost all PP and directly following NP chunks are PNP chunks. As the baseline always predicts a PNP if the NP chunk directly follows the PP chunk, its precision is very high. The memory-based PNP finder also tries to predict PNP chunks if the NP chunk is further away and thereby its precision drops slightly. However, it does result in a considerably higher recall rate, which makes that, overall, the memory-based PNP finder outperforms the baseline.

Conclusion

Considering that all memory-based parser modules perform better than the baseline scores, it can be concluded that they have successfully learned to parse spoken Dutch using the memory-based shallow parsing techniques. In accordance with expectations, it can therefore be confirmed that the memory-based shallow parsing method is language-independent.

	CGN			WSJ		
	prec.	recall	F_β	prec.	recall	F_β
tagging/chunking baseline	64.62	62.14	63.36	72.58	82.14	77.07
tagging/chunking mbsp	83.91	85.91	84.89	91.05	92.03	91.54
pnp finding baseline	98.23	92.44	95.24	98.21	95.38	96.77
pnp finding mbsp	97.48	98.48	97.98	98.50	97.40	97.90
gr finding baseline	63.88	57.84	60.71	49.43	37.30	42.51
gr finding mbsp	90.56	85.94	88.19	85.41	75.39	80.09

Table 4.2: Comparison of the performance of the parser modules trained on CGN data and on WSJ data. The performance measures for the WSJ chunker, PNP finder and grammatical relation finder are obtained from Veenstra and Van den Bosch (2000), Buchholz et al. (1999) and Buchholz (2002), respectively.

4.2 Restrictions of the current approach

By comparing the performance of the Dutch memory-based parser modules with baseline scores, it has been shown that the three parsing subtasks have been successfully learned. A partial answer to the research question, then, is that the approach proven to be effective for written English is also applicable to spoken Dutch. However, given the supposed language-independence of memory-based parsing, this finding only confirms what was already expected. For this reason, the research question specifically mentions the extent to which the techniques are useful for parsing spoken Dutch.

This aspect of the research question is discussed by reviewing whether the memory-based shallow parsing method is as effective for spoken Dutch as it is for written English. The fact that techniques developed for written English can successfully be applied to spoken Dutch demonstrates that there are many similarities between the two and that the dissimilarities do not require a fundamentally different approach to parsing. Nevertheless, it is expected that some of these dissimilarities do have a negative effect on the parsing performance. To verify this, the performance of the Dutch and English memory-based shallow parsers are compared.

This comparison is presented in table 4.2, in which the performance scores of the three memory-based parser modules trained on CGN data or Wall Street Journal (WSJ) data, are listed, together with baselines scores for all tasks on both corpora. The Wall Street Journal corpus is a treebank of written English. In contrast, the CGN treebank consists of spoken Dutch sentences. Consequently, performance differences are likely to be caused by the differing properties of the data set.

Tagging/chunking As for the tagging/chunking task, it can be seen that the baseline score on the WSJ data is considerably higher than on the CGN data. This suggests that the word form alone, is a better predictor for the correct syntactic type on the WSJ data than it is on the CGN data. Therefore, tagging/chunking on the CGN can be considered a more difficult task. This is reflected in the performance difference between the two memory-based tagger/chunkers, which indicates that written English is better parsed than spoken Dutch.

PNP finding The PNP finding baseline scores are very similar on both corpora. It has been mentioned earlier that PNP finding is the simplest of the three parsing tasks and that heuristics such as the one used by the baseline PNP finder perform almost as good as the best memory-based classifiers. This is confirmed by the scores in table 4.2, which show that the memory-based PNP finders score a few percents better than the baselines. The performance on both corpora can be considered to be equally good.

Grammatical-relation finder For the grammatical-relation finder, the scores in table 4.2 cannot be used to make a fair comparison. The WSJ grammatical-relation finder was trained to find all relations to verbs, while the CGN grammatical-relation finder only finds subject and direct or indirect object relations. This difference is expressed by the baseline scores, which suggest that the task performed by the WSJ relation finder is more difficult than the one performed by the CGN relation finder. However, to make a sound comparison between the two grammatical-relation finders, a carefully designed experimental setup, in which all factors are equal, will be necessary.

The above shows that, in comparison with the original English memory-based parser, the tagger/chunker module performs notably less accurate when applied to CGN data. The PNP finders perform approximately equally well. Based on the available information, the grammatical-relation-finding modules cannot be compared soundly. However, in the parsing cascade, the performance of the tagger/chunker module has already an important impact on the accuracy of the PNP finder and the grammatical-relation finder. Consequently, these two modules will nevertheless perform worse than their English counterparts in practical applications, where the input information has to be provided by the tagger/chunker. For this reason, in order to improve the overall performance of the Dutch memory-based parser, the factors that hinder tagger/chunker performance should be identified, after which the learning task can be adapted to overcome them.

It can reasonably be expected that most of the factors hindering tagger/chunker performance on the CGN data but not on the WSJ data are caused by the differences between these two corpora. The two most obvious properties in which the CGN differs from the WSJ corpus are the language and the nature of the language use, that is, the CGN is Dutch rather than English, and it contains spoken rather than written language. Therefore, factors hindering the performance of the CGN tagger/chunker are most likely to be related to properties of the Dutch language or to phenomena specific to spoken language. In the present research, there have been no findings that point in either direction. Moreover, from an analytical point of view, the CGN corpus is not particularly suited to relating performance issues to one of these causes, since the CGN, at the same time, differs in two dimensions from the WSJ corpus. For this reason, it would be worthwhile to apply the memory-based parsing techniques also to the Alpino Treebank, which is a treebank of written Dutch. Then, it would be possible to vary either language or nature of language use, while keeping the other dimension fixed.

Among factors that are specific to spoken language are constructs that break common grammatical practice and noise such as disfluencies. As for the first, memory-based parsing does not have an inherent notion of grammaticality, but

instead, learns to parse whatever grammatical constructs are frequently used in the training data. Consequently, it is to a certain extent robust to grammatical errors, provided these errors are often committed. At present, it is unclear, however, whether less predictable grammatical errors have an important impact on parsing accuracy.

With respect to disfluencies, it has been shown in chapter 3 that perfect filtering of disfluencies from the input to the tagger/chunker improves performance. For this reason, it would be interesting to test whether this improvement remains significant when disfluencies are automatically predicted by a memory-based classifier. Still, the performance increase caused by disfluency filtering is quite small, which suggests that the tagger/chunker is already largely robust to noisy input.

A property of the Dutch language that might hinder parsing accuracy is the fact that it is less constrained in its word order than English. In languages that have a relatively free word order, immediately neighbouring words may be poorer predictors of the target class of the word than is generally the case for constrained-order languages. Parsing of less constrained-order languages such as Dutch might benefit from larger amounts of training data. This hypothesis can be tested when the final release of the CGN arrives, since this release will contain a million syntactically annotated words as opposed to half a million words that are available in release 6, which has been used in this research.

Chapter 5

Conclusions

In the introduction of this thesis, it was stated that traditional methods of parser development give rise to a knowledge-acquisition bottleneck. Data-driven parsing techniques, and in particular memory-based shallow parsing, were identified as alternatives that overcome this bottleneck and that have been proven to be successful in parsing written English. With the aim of transferring the benefits of memory-based shallow parsing to the development of a Dutch parser, the research objective formulated in this thesis was to apply these techniques to the CGN, a corpus of spoken Dutch. This led to the research question: to what extent can the memory-based shallow parsing techniques be applied to a corpus that is Dutch rather than English and that contains spoken rather than written language?

This final chapter discusses the results of the research; it is subdivided into two parts. First, the answer to the research question is presented, based on the findings in chapters 3 and 4. Then, recommendations for future research in the context of Dutch memory-based parsing, are given.

The main conclusion of this thesis is that parsing of spoken Dutch can be successfully learned using the memory-based shallow parsing techniques. All three parser modules yield a good performance when trained and tested on the CGN data, which shows that most dissimilarities between spoken Dutch and written English do not require a fundamentally different learning-task definition. When applied in sequence, these modules form a complete shallow parser that can divide sentences into syntactic chunks and identify the most important grammatical relations.

By comparing the performance scores of the memory-based parsing modules applied on spoken Dutch and those applied on written English, it is expected that, of all three parser modules, the tagger/chunker is affected most by the differences between the two target languages. Therefore, the performance of the tagger/chunker and the parsing cascade overall, would benefit most from additions to the learning-task definition that are specifically directed at properties unique to spoken language or to Dutch. This is supported by the results showing that filtering disfluencies from the input of the tagger/chunker causes a small but significant performance increase.

Future research

Extensions to the current research that aim to improve parsing performance should be based on a systematic analysis of the causes of parsing errors committed by the current parser. As the CGN differs in two dimensions from the original written English training data, it is difficult to predict whether frequently committed errors are caused either by properties of Dutch or by properties of spoken language. Therefore, it would be worthwhile to apply the same memory-based parsing techniques also to the Alpino Treebank, a corpus of written Dutch. Then, comparisons could be made between parsers for spoken Dutch and written Dutch, as well as between parsers for written Dutch and written English.

If the cause of certain types of errors has been determined, solutions can range from extensions of the current learning task to additional modules that preprocess the input data. For example, it has been shown that perfect filtering of disfluencies from the input to the tagger/chunker improves performance. Whether this improvement remains significant if disfluencies are automatically predicted, however, should be established by further empirical tests.

Furthermore, in chapter 4, it has been explained that the availability of the final release of the CGN, which will approximately double the number of syntactically annotated words available as training data, may have a positive effect on the performance of the parser. While it is generally the case with all parsing tasks that increasing the amount of training data improves performance, it is hypothesised that increasing the amount of training data will particularly benefit parsers for languages with a relatively free word order, such as Dutch. This hypothesis can be tested as soon as the final release of the CGN is available.

References

Abney S. (1991). Parsing By Chunks. In *Principle-Based Parsing*, pp. 257–278. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Aha D.W., Kibler D., and Albert M. (1991). Instance-based Learning Algorithms. *Machine Learning*, Vol. 6, pp. 37–66.

Appelt D., Hobbs J., Bear J., Israel D., and Tyson M. (1993). FASTUS: A Finite-State Processor for Information Extraction From Real-World Text. In *Proceedings International Joint Conference on Artificial Intelligence*, pp. 1172–1178.

Bouma G. and Schuurman I. (1998). De positie van het Nederlands in Taal- en Spraaktechnologie. Rapport in opdracht van de Nederlandse Taalunie.

Buchholz S. (2002). *Memory-Based Grammatical Relation Finding*. Ph.D. thesis, Tilburg University.

Buchholz S., Veenstra J., and Daelemans W. (1999). Cascaded Grammatical Relation Assignment. In *EMNLP-VLC'99, the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pp. 239–246.

Cost S. and Salzberg S. (1993). A weighted nearest neighbour algorithm for learning with symbolic features. *Machine Learning*, Vol. 10, pp. 57–78.

Cover T.M. and Hart P.E. (1967). Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, Vol. 13, pp. 21–27.

Daelemans W. (1996). Abstraction considered harmful: lazy learning of language processing. In Van den Herik H.J. and Weijters A., editors, *Proceedings of the Sixth Belgian–Dutch Conference on Machine Learning*, pp. 3–12. MATRIKS, Maastricht, The Netherlands.

Daelemans W., Buchholz S., and Veenstra J. (1999). Memory-Based Shallow Parsing. In *Proceedings of CoNLL*, pp. 53–60. Bergen, Norway.

Daelemans W., Van den Bosch A., and Weijters A. (1997). IGTree: using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, Vol. 11, pp. 407–423.

Daelemans W., Zavrel J., Berck P., and Gillis S. (1996). MBT: A Memory-Based Part of Speech Tagger Generator. In Ejerhed E. and Dagan I., editors, *Proceedings of Fourth Workshop on Very Large Corpora*, pp. 14–27. ACL SIGDAT.

Daelemans W., Zavrel J., Van der Sloot K., and Van den Bosch A. (2002). TiMBL: Tilburg Memory Based Learner, version 4.3, Reference Guide. ILK Technical Report 02-10, Tilburg University. Available from http://ilk.uvt.nl/downloads/pub/papers/ilk.0210.ps.

Dudani S. (1976). The Distance-Weighted k-Nearest Neighbor Rule. In *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-6, pp. 325–327.

Grishman R. (1995). The NYU system for MUC-6 or where's syntax? In Sundheim B., editor, *Proceedings of the Sixth Message Understanding Conference*, pp. 167–175. Morgan Kaufmann, San Mateo, CA.

Lendvai P., Van den Bosch A., and Krahmer E. (2003). Memory-based Disfluency Chunking. In *DISS 2003 : Disfluency in Spontaneous Speech Workshop*, pp. 63–66.

Moortgat M., Schuurman I., and Van der Wouden T. (2001). CGN Syntactische Annotatie. Technical report, http://lands.let.kun.nl/cgn/publicat.htm.

Oostdijk N. (2000). The Spoken Dutch Corpus. Overview and first evaluation. In *LREC 2000 : Second Int. Conference on Language Resources and Evaluation*, Vol. II, pp. 887–894.

Oostdijk N., Goedertier W., Van Eynde F., Boves L., Martens J.P., Moortgat M., and Baayen H. (2002). Experiences from the Spoken Dutch Corpus Project. In *LREC 2002 : Third Int. Conference on Language Resources and Evaluation*, Vol. I, pp. 340–347.

Quinlan J. (1986). Induction of Decision Trees. *Machine Learning*, Vol. 1, pp. 81–206.

Quinlan J. (1993). C4.5*: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

Ramshaw L. and Marcus M. (1995). Text Chunking using Transformation-Based Learning. In *Proceedings of the 3rd ACL/SIGDAT Workshop on Very Large Corpora*, pp. 82–94. Cambridge, MA.

Shepard R. (1987). Toward a universal law of generalization for psychological science. *Science*, Vol. 237, pp. 1317–1228.

Skut W., Krenn B., Brants T., and Uszkoreit H. (1997). An Annotation Scheme for Free Word Order Languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing ANLP-97*, pp. 88–95. Washington, DC.

Stanfill C. and Waltz D. (1986). Toward Memory-Based Reasoning. *Communications of the* ACM, Vol. 29, No. 12, pp. 1213–1228.

Tjong Kim Sang E. and Veenstra J. (1999). Representing Text Chunks. In *Proceedings of the EACL*, pp. 173–179. Bergen, Norway.

Van den Bosch A. (1997). *Learning to pronounce written words: A study in inductive language learning*. Ph.D. thesis, Universiteit Maastricht.

Van der Beek L., Bouma G., Malouf R., and Van Noord G. (2002). The Alpino Dependency Treebank. In *Computational Linguistics in the Netherlands (CLIN) 2001*, pp. 8–22. Twente University.

Van Eynde F. (2001). Part of Speech Tagging en Lemmatisering. Technical report, Centrum voor Computerlinguistiek, K.U.Leuven, http://lands.let.kun.nl/cgn/publicat.htm.

Van Rijsbergen C. (1979). *Information Retrieval*. Buttersworth, London, UK.

Veenstra J. (1999). Memory-Based Text Chunking. In *Proceedings of ACAI'99*. Chania, Greece.

Veenstra J. and Van den Bosch A. (2000). Single-Classifier Memory-Based Phrase Chunking. In Cardie C., Daelemans W., Nedellec C., and Tjong Kim Sang E., editors, *Proceedings of CoNLL-2000 and LLL-2000*, pp. 157–159. Lisbon, Portugal.

White A. and Liu W. (1994). Bias in Information-Based Measures in Decision Tree Induction. *Machine Learning*, Vol. 15, No. 3, pp. 321–329.

Summary

Traditional methods of parser development have mostly been grammar-based. When developing grammar-based parsers, much effort needs to be invested in constructing comprehensive grammars of the target language. This eventually leads to a knowledge-acquisition bottleneck. Data-driven parsing methods, and in particular those based on memory-based learning, are alternative approaches to parser development that are not a priori affected by this bottleneck. While the English language has seen steady progress in memory-based parsing, languages with smaller communities of speakers, such as Dutch, lagged behind due to a lack of the necessary resources. For Dutch, this situation has changed with the arrival of two large-scale syntactically-annotated corpora: the Spoken Dutch Corpus (CGN) and the Alpino Treebank, which are both enabling resources for data-driven parser development.

With the above in mind, the research objective of this thesis has been formulated to target the development of a Dutch shallow parser based on the memory-based shallow parsing framework by Buchholz et al. (1999) and on the CGN as training data. This objective has led to the research question: "To what extent can the memory-based shallow parsing techniques be applied to a corpus that differs from the original training corpus in the sense that it is Dutch rather than English and that it contains spoken rather than written language?" The remainder of the thesis is centred around answering this question.

The memory-based shallow parsing framework splits up the parsing task into three subtasks: 1) part of speech tagging/chunking, 2) PNP finding, and 3) grammatical-relation finding. The modules performing these subtasks are applied in sequence to the input data, so that results produced by one module can be used as input by another. In order to train memory-based classifiers for these tasks, they have to be formulated in a form suited to memory-based learning algorithms, that is, into fixed-length vectors of feature-value pairs that have a unique class label. The task representation of all three subtasks is essentially based on the windowing technique: an instance contains one focus word or chunk and a number of words or chunks directly preceding or following it. In addition, instances encode other information useful for the task. For example, instances for the tagging/chunking task include the classifications of words preceding the focus word in the sentence. Instances for the PNP-finding task and for the grammatical-relation-finding task contain features describing spatial properties, such as the distance and the number of verbal chunks between the focus chunk and the target verb.

The CGN stores its annotations in dependency structures, which are directed acyclic graphs of which the nodes and edges have been labelled with syntactic information. To convert the CGN dependency structures to the instance format

of the learning tasks, a conversion procedure has been described. The main task of this procedure is to extract chunks and grammatical relations from the CGN data, which only implicitly contain these concepts.

To optimise the performance of the parser modules, a systematic optimisation procedure is performed on all three modules in isolation, that is, the input of the modules is composed of gold-standard corpus data, rather than of the outputs of the preceding modules in the parsing cascade. The optimisation procedure leads to F_β scores of 84.89 for the tagger/chunker, 97.98 for the PNP finder, and 88.19 for the grammatical-relation finder.

Combining the three modules into a complete memory-based shallow parsing cascade reveals that the performance of the modules drops significantly due to errors committed by the preceding modules. In particular the performance of the tagger/chunker appears to have an important effect on that of the entire parsing cascade. The F_β scores for the PNP finder and the grammatical-relation finder drop to 88.37, and 81.37, respectively. In addition to experiments with the standard memory-based shallow parsing cascade, some other experiments have been performed in which either all disfluencies or only interjections are filtered from the input sentences. The results of these experiments show that this intermediary filtering step improves the performance of the parser slightly, but significantly. The F_β scores for the tagger/chunker, the PNP finder and the grammatical-relation finder with perfectly filtered disfluencies are 86.08, 90.17, and 82.99, in that order.

Based on the experimental results, the answer to the research question is centred around two main considerations. One the one hand, the memory-based parser modules appear to have successfully learned to parse spoken Dutch sentences. On the other hand, a comparison of the scores of the Dutch memory-based shallow parser with the memory-based shallow parser for written English, brings to light that parsing spoken Dutch is a more difficult task than parsing written English. Future research should therefore focus on extending the learning tasks to deal better with the typical properties of spoken Dutch that hinder the performance of the parser. In order to identify such properties, more analytical experiments will have to be performed, for example experiments comparing the performance of the memory-based shallow parser for spoken Dutch with one trained for written Dutch.

Appendix A

Syntactic labels in the CGN

A.1 Part of speech tags

ADJ1	Prenominal adjective, base form
ADJ2	Prenominal adjective, comparative form
ADJ3	Prenominal adjective, superlative form
ADJ4	Nominalised adjective, base form
ADJ5	Nominalised adjective, comparative form
ADJ6	Nominalised adjective, superlative form
ADJ7	Postnominal adjective, base form
ADJ8	Postnominal adjective, comparative form
ADJ9	Adjective used predicatively, base form
ADJ10	Adjective used predicatively, comparative form
ADJ11	Adjective used predicatively, superlative form
ADJ12	Adjective, dialectal word
BW	Adverb
LET	Interpunction
LID	Determiner
N1	Common noun, singular
N2	Common noun, singular, genitive case
N3	Common noun, plural
N4	Common noun, dialectal word
N5	Proper noun, singular
N6	Proper noun, singular, genitive case
N7	Proper noun, plural
N8	Proper noun, dialectal word
SPEC	Rest category
TSW	Interjection
TW1	Ordinal number
TW2	Cardinal number
VG1	Coordinating element
VG2	Subordinating element
VNW1	Personal pronoun, nominative case
VNW2	Personal pronoun, oblique case
VNW3	Personal pronoun, nominative or oblique case

VNW4	Personal pronoun, genitive case
VNW5	Personal pronoun, dialectal word
VNW6	Personal or reflexive pronoun
VNW7	Reflexive pronoun
VNW8	Reciprocal pronoun, oblique case
VNW9	Reciprocal pronoun, genitive case
VNW10	Reciprocal pronoun, dialectal word
VNW11	Possessive pronoun
VNW12	Question word
VNW13	Relative pronoun
VNW14	Question word or relative pronoun
VNW15	Question word or relative pronoun, used adverbially
VNW16	Exclamative pronoun
VNW17	Question word or relative pronoun, used as determiner
VNW18	Exclamative pronoun, used as determiner
VNW19	Demonstrative pronoun
VNW20	Demonstrative pronoun, used adverbially
VNW21	Demonstrative pronoun, used as determiner
VNW22	Indefinite pronoun
VNW23	Indefinite pronoun, used adverbially
VNW24	Prenominal indefinite pronoun, used as determiner
VNW25	Nominalised indefinite pronoun, used as determiner
VNW26	Indefinite pronoun used predicatively as determiner
VNW27	Indefinite pronoun, dialectal word
VZ1	Preposition preceding its complement
VZ2	Preposition following its complement
VZ3	Fused preposition + article
WW1	Inflected verb form, singular
WW2	Inflected verb form, plural
WW3	Inflected verb form with -t
WW4	Infinitive used predicatively
WW5	Prenominal infinitive
WW6	Nominalised infinitive
WW7	Past particle used predicatively
WW8	Prenominal past particle
WW9	Nominalised past particle
WW10	Present particle used predicatively
WW11	Prenominal present particle
WW12	Nominalised present particle
WW13	Dialectal verb

A.2 Domain types

AHI	Long infinitive group headed by *aan het*
AP	Adjectival group
COMPP	Various comparative constructions
CONJ	Conjunction
CP	Clause headed by any kind of complementiser
DU	Discourse unit

INF	Short infinitive group
LIST	Asyndetic conjunction
MWU	Merged-word unit
NP	Nominal group
OTI	Long infinitive group headed by *om*
PP	Prepositional group
PPART	Past/passive particle group
PPRES	Present particle group
REL	Relative clause
SMAIN	Main clause, verb in second position
SSUB	Subordinate clause, verb in final position
SV1	Sentence with a sentence-initial inflected verb
SVAN	Subordinate clauses headed by *van*
TI	Long infinitive group
UI	Long infinitive group headed by *uit*
WHQ	WH-question, verb in second position
WHREL	Headless relative
WHSUB	Embedded WH-question

A.3 Dependency types

APPOS	Apposition
BODY	Body of subordinate clause
CMP	Grammatical complementiser
CNJ	Member of conjunction
CRD	Coordinator
DET	Determiner
DLINK	Discourse particle joining discourse fragments
DP	Discourse part (member of a DU domain)
HD	Head
HDF	Second part of a circumposition
LD	Locational or directional complement
LP	List part (member of a LIST domain)
ME	Measure complement
MOD	Modifier
MWP	Multi-word part (member of a MWU)
NUCL	Nuclear clause (member of a DU domain)
OBCOMP	Comparative complement
OBJ1	Direct or first object
OBJ2	Indirect or secondary object
PART	Partitive
PC	Prepositional complement
POBJ1	Provisional direct or first object
PREDC	Predicative complement
PREDM	Secondary predicate
PRT	Part of particle group
RHD	Complementiser heading (headless) relative
SAT	Satellite (member of a DU domain)
SE	Obligatory reflexive object

SU	Subject
SUP	Provisional subject
SVP	Verbal particle
TAG	Tag (member of a DU domain)
VC	Verbal complement
WHD	Complementiser heading a WH question