TEXT, SPEECH AND LANGUAGE TECHNOLOGY

INDUCTIVE DEPENDENCY PARSING

By Joakim Nivre

Springer

Inductive Dependency Parsing

Text, Speech and Language Technology

VOLUME 34

Inductive Dependency Parsing

by

Joakim Nivre
Växjö University, Sweden

Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN-10 1-4020-4888-2 (HB)
ISBN-13 978-1-4020-4888-3 (HB)
ISBN-10 1-4020-4889-0 (e-book)
ISBN-13 978-1-4020-4889-0 (e-book)

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

www.springer.com

Printed on acid-free paper

Printed in the Netherlands

To Elisabeth and Fredrik

Preface

This book is based on work carried out over a period of roughly three years with the support of a number of people and organizations that deserve my heartfelt gratitude. In the first place, I want to thank my PhD students Johan Hall and Jens Nilsson, who have been involved in the project from the start. I also want to thank all the people who are part of the research group in computer science at Växjö University, for providing a stimulating environment to work in, and the Swedish Research Council for a grant that supported part of the work reported in this book (Vetenskapsrådet, 621-2002-4207).

Among the many people who have contributed, directly or indirectly, to the ideas and results presented in the book, I specifically want to mention Eckhard Bick, Sabine Buchholz, John Carroll, Atanas Chanev, Yuchang Cheng, Walter Daelemans, Ralph Debusmann, Denys Duchier, Gülsen Eryiğit, Jason Eisner, Kilian Foth, Kadri Hacioglu, Jan Hajič, Erhard Hinrichs, Tomáš Holan, Viggo Kann, Matthias Trautner Kromann, Geert-Jan Kruijff, Sandra Kübler, Marco Kuhlmann, Haitao Liu, Welf Löwe, Svetoslav Marinov, Erwin Marsi, Yuji Matsumoto, Ryan McDonald, Pierre Nugues, Tomasz Obrębski, Guy de Pauw, Aarne Ranta, Mario Scholz, Noah Smith, Antal van den Bosch, Hiroyasu Yamada, Anssi Yli-Jyrä, and Daniel Zeman. I also want to thank my editor, Jolanda Voogd, for practical assistance, and the series editors, Nancy Ide and Jean Véronis, for promoting the publication of the book in the first place.

I owe a special debt to John Carroll, Walter Daelemans and Welf Löwe, who scrutinized the first draft of the manuscript and suggested innumerable improvements, and to Viggo Kann, who spotted an error in one of the proofs. All remaining inadequacies are entirely my own responsibility.

Finally, I want to express my love and gratitude to my wife Elisabeth and my son Fredrik for making my life such a wonderful experience.

Växjö, February 2006 *Joakim Nivre*

Contents

1

Introduction

The automatic analysis of syntactic structure, or parsing, is a core component in many systems for natural language processing. This monograph explores the framework of *inductive dependency parsing* as an efficient method for syntactic parsing of unrestricted natural language text under the joint requirements of robustness and disambiguation. That is, given as input a natural language text, consisting of a sequence of sentences, we want the parser to assign to every sentence at least one analysis (*robustness*) and at most one analysis (*disambiguation*). Needless to say, we also want the single analysis assigned to a sentence to be correct as often as possible (*accuracy*). Finally, we want the computation for each sentence to take as little time and memory as possible (*efficiency*). Maximizing accuracy and efficiency while maintaining robustness and disambiguation is the problem that we have set ourselves. Finding out whether inductive dependency parsing can provide a solution to this problem is the topic of this book.

1.1 Inductive Dependency Parsing

In the framework of inductive dependency parsing, the syntactic analysis of a sentence amounts to the derivation of a dependency structure, using inductive machine learning to guide the parser at nondeterministic choice points. This methodology combines a number of themes that are prominent in the recent natural language processing literature, although the particular combination of ideas embodied in the resulting framework appears to be original. More precisely, inductive dependency parsing can be regarded as the simultaneous instantiation of two notions that have played a more or less central role in natural language parsing in recent years:

- Dependency-based parsing
- Data-driven parsing

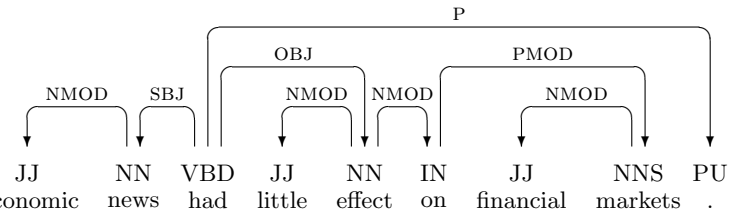


Fig. 1.1. Dependency structure for English sentence from the Penn Treebank

The fundamental idea in dependency-based parsing is that parsing crucially involves establishing binary relations between words in a sentence. This is illustrated in figure 1.1, which depicts the analysis of a short sentence taken from the Wall Street Journal section of the Penn Treebank (Marcus et al., 1993, 1994). In this example, the syntactic structure is built up by recognizing a subject relation (SBJ) from the finite verb *had* to the noun *news*, a nominal modifier relation (NMOD) from *news* to the adjective *Economic*, an object relation (OBJ) from *had* to the noun *effect*, and so on.

Dependency-based methods appear in many guises in the current literature on natural language parsing. On the one hand, we have what we may call dependency parsing in a narrow sense, where the goal of the parsing process is to build a dependency structure, i.e., a graph built from binary dependency relations as in figure 1.1, and where the analysis more or less closely adheres to the theoretical tradition of dependency grammar. Cases in point are Hellwig (1980), Maruyama (1990), Harper and Helzerman (1995), Tapanainen and Järvinen (1997), Menzel and Schröder (1998), and Duchier (1999). On the other hand, we have approaches that can be characterized as dependency-based parsing in a broader sense, where the syntactic analysis may not take the form of a dependency structure, but where the construction of the analysis nevertheless depends on finding syntactic relations between lexical heads. In this category, we find the widely used link grammar parser for English (Sleator and Temperley, 1993), as well as the influential probabilistic parsers of Collins (1997, 1999) and Charniak (2000), but also a variety of other lexicalized parsing models that can be subsumed under the general notion of bilexical grammars (Eisner, 2000). The use of bilexical relations for disambiguation has been a significant theme in research on natural language parsing during the last decade, although the results are not completely unambiguous (Collins, 1999; Gildea, 2001; Klein and Manning, 2003; Bikel, 2004).

The framework we develop in this book falls under the more narrow definition of dependency parsing, at least in the sense that it assumes dependency structures as the only form of syntactic representation. At the same time, we will focus more on formal methods for constructing dependency structures than on details of linguistic analysis, which means that the discussion

will remain rather agnostic with respect to different theoretical traditions of dependency grammar. More precisely, we will define the task of dependency parsing relative to a formal framework of dependency graphs, where only minimal assumptions are made concerning the linguistic analysis, but where the notions of robustness, disambiguation, accuracy and efficiency can be given precise definitions.

Although our formal characterization of dependency parsing is compatible with different parsing strategies, we will limit our attention in this study to deterministic methods, which means that we derive a single analysis for each input sentence in a monotonic fashion with no redundancy or backtracking. Historically, deterministic parsing of natural language has been investigated with a view to modeling human sentence processing in a psychologically plausible way, as in the work by Marcus (1980) and Shieber (1983), but it has also been explored as a way of improving the robustness and efficiency of natural language parsing, especially in various approaches to partial parsing using finite-state methods (Ejerhed, 1983; Koskenniemi, 1990; Abney, 1991, 1996).

In the present framework, we want to apply deterministic methods to full parsing, insofar as we want to derive a complete dependency structure for each input sentence. In this way, we hope to combine the gains in efficiency with a deeper analysis of syntactic structure. The parsing algorithm that we use was first presented in Nivre (2003), with a partial analysis of its complexity and robustness properties. It has also been shown that the algorithm favors incremental processing, something that may be desirable both for certain applications, such as language modeling for speech recognition, and for the kind of psycholinguistic modeling that inspired early research on deterministic parsing (Nivre, 2004a). In this book, we will for the first time provide a comprehensive analysis of the parsing algorithm with respect to robustness, disambiguation and complexity.

The second essential component of our methodology is a commitment to data-driven parsing, understood in a broad sense to include all approaches that make essential use of empirical data, in particular treebanks or parsed corpora (Abeillé, 2003b; Nivre, forthcoming), in the development of parsing systems for natural language. Research during the last ten to fifteen years has shown rather conclusively that an empirical approach is necessary in order to achieve accurate disambiguation as well as robustness in parsing unrestricted text, regardless of whether the parser uses a traditional grammar or a more radically data-driven model. In the former case, exemplified by broad-coverage deep parsers such as Riezler et al. (2002) and Toutanova et al. (2002), treebank data are used to tune and optimize the parser, in particular by constructing a statistical model for parse selection. In the latter case, represented by probabilistic parsers such as Collins (1997, 1999) and Charniak (2000), the grammar is replaced by a statistical model, the parameters of which are derived from treebank data using machine learning techniques.

An even more radical approach is to replace the statistical model by the treebank itself, and to reuse fragments of previously encountered syntactic

structures to construct new ones during parsing, as in the framework of Data-Oriented Parsing (DOP) (Bod, 1995, 1998; Bod, Scha and Sima'an, 2003). The DOP model can be seen as an instantiation of the paradigm of memory-based learning, or lazy learning, which is based on the idea that learning is the simple storage of experiences in memory and that new problems are solved by reusing solutions from similar old problems (Daelemans, 1999; Daelemans and Van den Bosch, 2005). Memory-based learning has been successfully applied to a wide variety of problems in natural language processing, such as grapheme-to-phoneme conversion, part-of-speech tagging, prepositional phrase attachment, and chunking (Daelemans et al., 2002). Memory-based approaches to syntactic parsing, in addition to the DOP framework, include Veenstra and Daelemans (2000), Buchholz (2002), De Pauw (2003) and Kübler (2004).

In this book, we will explore a memory-based approach to dependency parsing, using classifiers that predict the next parsing action based on the current state of the parser and a database of previously encountered parser states. Since the state of the parser results from a sequence of previous actions, this can also be seen as a form of history-based parsing (Black et al., 1992; Jelinek et al., 1994; Magerman, 1995), although we prefer the term *inductive dependency parsing* for the general idea of using inductive machine learning to predict the actions of a dependency parser.

An early version of this idea, with a simple probabilistic classifier, was reported in Nivre (2004b). The memory-based version was first presented in Nivre et al. (2004), with an evaluation on Swedish treebank data, and later in Nivre and Scholz (2004), with results from the Wall Street Journal section of the Penn Treebank. This book provides a comprehensive analysis of inductive dependency parsing, including a general characterization of the history-based model and a formal framework for the specification of model parameters. For the deterministic memory-based instantiation of this framework, we also present a detailed discussion of feature selection, and a thorough empirical evaluation of different models using treebank data from both Swedish and English that goes far beyond previously published results.

The framework developed in this book is implemented in a system called MaltParser, which is used in all the experiments reported below. MaltParser can be described as a language-independent parser-generator. When applied to a dependency-based treebank, the system generates a dependency parser for the language represented in the treebank. The memory-based version of this system uses the TiMBL software package (Daelemans et al., 2004) and supports a variety of options with respect to linguistic features as well as learning parameters. A version of MaltParser is freely available for research and educational purposes.[1]

[1] URL: http://www.msi.vxu.se/users/nivre/research/MaltParser.html

1.2 The Need for Robust Disambiguation

The usefulness of parsing in different language technology applications is a point of some controversy. For example, in speech recognition, syntax-based language models have had a hard time improving on the results obtained with probabilistic n-gram models (Rosenfeld, 2000). Similarly, attempts at improving accuracy in information retrieval by incorporating syntactic information have met with very limited success (Tzoukermann et al., 2003). If we move to applications that require some kind of semantic analysis of individual sentences, the role of parsing becomes more evident. For instance, information extraction normally involves at least partial parsing (Cowie and Wilks, 2000), and question answering systems often rely on semantic role labeling, for which full syntactic parsing has been shown to give a substantial improvement over partial parsing (Gildea and Palmer, 2002; Carreras and Màrquez, 2004). In machine translation, parsing has always been a core component of transfer-based systems, but syntax-based models are becoming more prominent also in statistical approaches (Yamada and Knight, 2001; Charniak et al., 2003).

At the end of the day, few researchers would question the relevance of syntactic analysis for the ultimate goal of building computer systems capable of full natural language understanding — however we define this — but there is still no consensus on what form the analysis should take and which methods should be used to derive it. In this book, we will focus on the development of a particular framework for natural language parsing, and even though we will sometimes draw on requirements from applications to motivate certain design choices, we will not be able to demonstrate that these choices actually improve applications, and the potential usefulness of parsing as such will simply have to be taken for granted.

The emphasis on robustness, disambiguation and efficiency in the context of natural language parsing may also need some further motivation. Starting with robustness, we see this as a fundamental requirement in any application of natural language parsing that deals with (more or less) unrestricted text, where the range of permissible inputs cannot be sharply delimited. Even if the likelihood of a correct analysis decreases as the input deviates more and more from our expectations, we want to have a system that degrades gracefully and always delivers some kind of analysis.

Disambiguation is a more controversial requirement, given that part of the information needed to choose between alternative analyses, such as word sense information and extra-sentential context, may be missing at parsing time. This observation leads naturally to the assumption that the parser should simply pass on all analyses that are compatible with the given input and leave the final decision to another component, typically a semantic or pragmatic interpreter. However, the same observation can be made about almost any kind of input analysis, from tokenization and sentence segmentation to semantic and pragmatic analysis. So, unless we adopt a completely holistic integration of all processing levels, there will be decisions at each level that are based on

incomplete information. Moreover, the requirement of robustness will often lead to a relaxation of syntactic constraints to the point where the number of analyses compatible with a given input becomes prohibitively large. This means that some degree of pruning is necessary in any case, even though the search space may only be reduced to the n best candidates rather than to a single analysis. Finally, the capacity for disambiguation can be very useful in applications where the parser is not used as part of a processing chain but rather is used to generate features for another kind of analysis. A typical case in point is the use of parse tree information in semantic role labeling referred to earlier. Thus, without wanting to claim that robust disambiguation is the solution to every syntactic analysis problem, we believe that it is useful in many situations, and it will be adopted as a basic requirement for the methods investigated in this book.

Efficiency, finally, is a non-functional requirement for parsers to be usable in practical applications, especially in systems working under hard time constraints, such as speech-based user interfaces, or dealing with large volumes of data, such as information retrieval and extraction systems. In many cases, there is a trade-off between efficiency and accuracy, and although we often give priority to accuracy over efficiency, it is nevertheless a joint optimization problem, since we cannot reduce efficiency to the point where parsers become unusable in practical applications.

The framework developed in this book is the result of a conscious strategy to adopt methods for parsing and disambiguation that are provably robust and efficient, in a sense yet to be made precise, and to work systematically towards higher accuracy while maintaining robustness, disambiguation and (as far as possible) efficiency. Needless to say, this is only one of many conceivable strategies, and it may not be the one that ultimately gives us the highest accuracy, although it should provide us with highly efficient methods with sufficient accuracy for certain applications.

From a scientific point of view, it is also interesting to see how far we can get by adopting an extreme approach and pushing it to its limits. At the very least, this may give a new perspective on results achieved in other frameworks using other strategies. More importantly, however, by concentrating on the systematic study of a few simple ideas and techniques, we may hope to gain a deeper understanding of the way in which they can contribute to improved methods for natural language parsing in general.

1.3 Outline of the Book

In this introductory chapter, we have tried to outline the aims of the study and to motivate the general research directions. The remainder of the book is structured as follows.

Chapter 2
Natural Language Parsing

Chapter 2 discusses the problem of parsing unrestricted natural language text, relating it to other notions of parsing, in particular the one associated with grammars in formal language theory. We compare different strategies for achieving robust disambiguation and define evaluation criteria for the key concepts of robustness, disambiguation, efficiency and accuracy.

Chapter 3
Dependency Parsing

Chapter 3 starts with a review of dependency grammar and its use in syntactic parsing. We then introduce a formal framework for dependency parsing, based on a general definition of labeled dependency graphs with a further characterization of properties such as connectedness, single-headedness, acyclicity, and projectivity. Finally, we present a deterministic parsing algorithm for projective dependency graphs, with proofs of complexity and properties related to robustness and disambiguation.

Chapter 4
Inductive Dependency Parsing

Chapter 4 extends the framework of dependency parsing to incorporate the use of inductive machine learning to guide the parser at nondeterministic choice points. We derive a history-based model of dependency parsing and show how this can be combined with the deterministic parsing algorithm presented in chapter 3 and with discriminative learning methods that induce classifiers from treebank data. We define a formal method for the specification of feature models, we introduce memory-based learning and classification as a method for solving the inductive learning problem defined by the parsing method, and we briefly describe the implemented MaltParser system.

Chapter 5
Treebank Parsing

Chapter 5 contains an empirical evaluation of the parsing methodology with respect to accuracy and efficiency, based on data from Talbanken, a small Swedish treebank, and the Penn Treebank of American English. The chapter starts with a general discussion of treebanks and their use in syntactic parsing, moves on to a description of the evaluation framework and the experimental setup, and concludes with a discussion of the results in relation to previous work on treebank parsing, in particular dependency-based parsing.

Chapter 6
Conclusion

Chapter 6 summarizes the main results of the study and points to promising directions for future research, such as the extension to non-projective dependency structures, which may be needed for languages with more flexible word order; the introduction of mild forms of nondeterminism and stochastic disambiguation; the exploration of alternative learning methods, including an integration of inductive and deductive learning; and the use of more refined evaluation methods.

2

Natural Language Parsing

Research on natural language parsing has over a period of several decades produced a wealth of knowledge concerning different methods for automatic syntactic analysis. Most of the results, however, concern formal grammars and algorithms that are only indirectly related to the more practical problem of analyzing syntactic structure in naturally occurring texts. This has led to a somewhat paradoxical situation where, despite the increase in knowledge about the complexity of problems and algorithms for formal grammars, we know relatively little about the formal properties of text parsing. In fact, it is still not clear that there is a well-defined parsing problem for natural language text that is computable in the strict sense.

In this chapter, we will begin by contrasting the two notions of parsing, the well-defined parsing problem for formal grammars, familiar from both computer science and computational linguistics, and the more open-ended problem of parsing unrestricted text in natural language, which is the focus of the investigations in this book. We will then review different strategies for text parsing, including both grammar-driven and data-driven approaches, and discuss the different kinds of problems that arise with different methods. On the basis of this discussion, we will then define the basic requirements of robustness, disambiguation, accuracy and efficiency, which are central to the investigations of text parsing in this book, and discuss evaluation criteria for each of the requirements.

The primary goal of this chapter is to set the scene for the exploration of inductive dependency parsing in later chapters, by defining the basic problems and evaluation criteria, but in doing so we will also have reason to review some of the more important trends in recent research on natural language parsing. First of all, however, we need to say a few words about the desired output of the parsing process, i.e., about syntactic representations for natural language sentences.

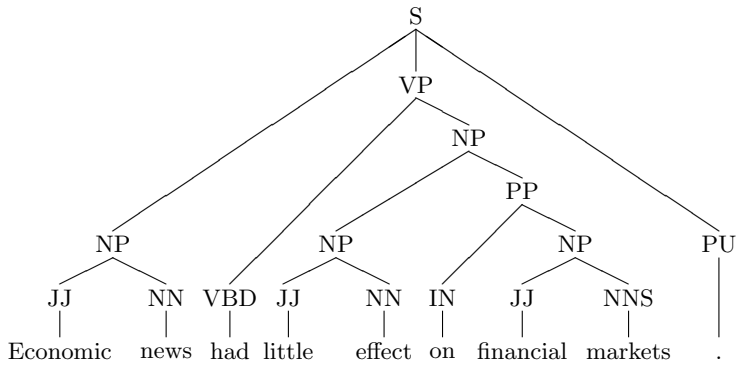


Fig. 2.1. Constituent structure for English sentence from the Penn Treebank

2.1 Syntactic Representations

The type of syntactic representation that has been dominant during the last fifty years, both in theoretical linguistics and in natural language processing, is based on the notion of *constituency*. In this representation, a sentence is recursively decomposed into smaller segments, called *constituents* or *phrases*, which are typically categorized according to their internal structure into *noun phrases*, *verb phrases*, etc. Constituency analysis comes from the structuralist tradition represented by Bloomfield (1933) and was formalized in the 1950s in the model of phrase structure grammar, or context-free grammar (Chomsky, 1956). Figure 2.1 shows a typical constituency representation of an English sentence, taken from the Wall Street Journal section of the Penn Treebank (Marcus et al., 1993, 1994).[1]

A wide range of different theories about natural language syntax are based on constituency representations. In addition to the theoretical tradition of Chomsky (1957, 1965, 1981, 1995), this includes frameworks that are prominent in computational linguistics, such as Lexical Functional Grammar (LFG) (Kaplan and Bresnan, 1982; Bresnan, 2000), Generalized Phrase Structure Grammar (GPSG) (Gazdar et al., 1985), Tree Adjoining Grammar (TAG) (Joshi, 1985, 1997), and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1987, 1994).

Another type of syntactic representation, which has a long tradition in descriptive linguistics especially in Europe, is instead based on the notion of *dependency*. In this representation, a sentence is analyzed by connecting its words by binary asymmetrical relations, called *dependencies*, which are

[1] The representation is equivalent to the treebank annotation except that the part-of-speech category '.' has been replaced by PU (for *punctuation*) to avoid a name clash with the terminal '.'. This will simplify exposition later on.

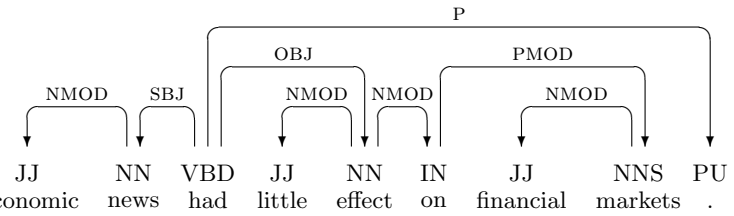


Fig. 2.2. Dependency structure for English sentence (same as figure 1.1)

typically categorized according to their functional role into *subject*, *object*, etc. Figure 1.1, repeated here for convenience as figure 2.2, shows a typical dependency representation of the same sentence as in figure 2.1.

According to some scholars, dependency analysis can be traced back to Antiquity (Kruijff, 2002), but the start of the modern tradition is usually taken to be the work of Tesnière (1959). Linguistic theories that are based on dependency representations include Word Grammar (Hudson, 1984, 1990), Functional Generative Description (Sgall et al., 1986), Lexicase (Starosta, 1988), and Meaning-Text Theory (Mel'čuk, 1988).

A third kind of syntactic representation is found in *categorial grammar*, which connects syntactic (and semantic) analysis to inference in a logical calculus. The syntactic representations used in categorial grammar are essentially proof trees, which cannot be reduced to constituency or dependency representations, although they have affinities with both. The categorial grammar tradition goes back to Ajdukiewicz (1935), was taken up in the 1950s by Bar-Hillel (1953) and Lambek (1958), and was used by Montague (1970, 1973) in his influential work on model-theoretic semantics. More recent frameworks that are prominent in the literature on syntactic parsing includes Combinatory Categorial Grammar (CCG) (Steedman, 2000), Type-Logical Grammar (Morrill, 1994, 2000), and Grammatical Framework (Ranta, 2004; Ljunglöf, 2004).

One final type of representation that is widely used in modern syntactic theories is the notion of a *feature structure*, or *attribute-value* representation (Johnson, 1988; Carpenter, 1992), which is usually formalized as a directed acyclic graph. Theories that make use of feature structures are often said to be *unification-based* (Shieber, 1986), since *unification* is the major operation used to combine information from different structures. Feature structures are often combined with constituency representations, either by decorating tree nodes with feature structures, as in GPSG (Gazdar et al., 1985), or by adding new layers of representation over and above the constituency representation, as in LFG (Kaplan and Bresnan, 1982), or by using feature structure representations to encode all aspects of linguistic structure, including constituency,

as in HPSG (Pollard and Sag, 1994). However, feature structures are also found in dependency-based theories, such as Dependency Unification Grammar (DUG) (Hellwig, 2003), and in categorial grammar frameworks, such as Categorial Unification Grammar (CUG) (Uszkoreit, 1986) and Unification Categorial Grammar (UCG) (Zeevat et al., 1991).

Throughout this book, we will mainly be concerned with dependency representations, which will be discussed in more depth in chapter 3. However, in this chapter we will try to abstract away from the particular representations used and concentrate on issues in natural language parsing that cut across different frameworks. Thus, when we speak about syntactic parsing as the problem of assigning an *analysis* to an input string, it will be understood that the analysis is a syntactic representation as defined by the relevant framework. To some extent, this means that we will be comparing apples and oranges, since the problems involved in parsing are not independent of the nature of syntactic representations. Still, we feel that different frameworks have enough in common to make a general discussion fruitful, although we will also make reference to different syntactic representations when this is relevant.

2.2 Two Notions of Parsing

The term *parsing*, derived from the Latin *pars orationis* (parts of speech), was originally used to denote the grammatical explication of sentences, as practiced in elementary schools. The term was then borrowed by linguistics and computer science, where it has acquired a specialized sense in connection with the theory of formal languages and grammars. However, in practical applications of natural language processing, the term is also used to denote the syntactic analysis of sentences in text, without reference to any particular formal grammar, a sense which is in many ways quite close to the original grammar school sense.

In other words, there are at least two distinct notions of parsing that can be found in the current literature on natural language processing, notions that are not always clearly distinguished. Although we are certainly not the first to notice this ambiguity, we feel that it has not been given the attention that it deserves. While it is true that there are intimate connections between the two notions, they are nevertheless independent notions with quite different properties in some respects. In order to highlight these differences, we will now proceed to a contrastive examination of the notions of *grammar parsing* and *text parsing*.[2]

[2] The term *text* in *text parsing* is not meant to exclude spoken language, but rather to emphasize the relation to naturally occurring language use. Although we will have nothing to say about the parsing of spoken utterances in this book, we want the notion of text parsing to encompass both written texts and spoken dialogues. An alternative term would be *discourse parsing*, but this would give rise to misleading associations of a different kind.

S → NP VP PU	JJ → Economic
VP → VP PP	JJ → little
VP → VBD NP	JJ → financial
NP → NP PP	NN → news
NP → JJ NN	NN → effect
NP → JJ NNS	NNS → markets
PP → IN NP	VBD → had
PU → .	IN → on

Fig. 2.3. Context-free grammar for a fragment of English

2.2.1 Grammar Parsing

The notion of grammar parsing is intimately tied to the notion of a formal grammar G defining a formal language $L(G)$ over some (terminal) alphabet Σ. Formally, an alphabet Σ is a set of elementary symbols, and a formal language over Σ is a subset of the set Σ^* of all strings formed from symbols in Σ. A grammar G is a formal system for deriving strings over some alphabet Σ, and the language $L(G)$ defined by G is the set of all strings x derivable in G. Moreover, each (canonical) derivation of a string x corresponds to a syntactic analysis of x according to G. The *parsing problem* can then be defined as follows:

> Given a grammar G and an input string $x \in \Sigma^*$, derive some or all of the analyses assigned to x by G.

This is sometimes called the *universal* parsing problem for grammars of a certain type. It is also possible to define the parsing problem relative to a *particular* grammar G of some type, although this is usually considered less interesting (Barton et al., 1987). The analysis of formal grammars and their parsing problems goes back to the pioneering work of Noam Chomsky and others in the 1950s and continues to be a very active area of research. A classic introduction to the field is Hopcroft and Ullman (1979), which recently appeared in a new and revised edition (Hopcroft et al., 2001).

The most widely used type of formal grammar, in computer science as well as computational linguistics, is the *context-free grammar* (CFG) of Chomsky (1956), which is equivalent to the independently defined *Backus-Naur Form* (BNF) (Backus, 1959). Figure 2.3 shows a context-free grammar defining a fragment of English including the sentence analyzed in figure 2.1. One of the analyses assigned to this sentence by the grammar is the *parse tree* depicted in figure 2.1, which corresponds to a canonical derivation of the word string.[3]

[3] Without the restriction to some canonical form of derivation (e.g., a leftmost or a rightmost derivation), there is generally more than one derivation of the same parse tree.

Over the years, a variety of formal grammars have been introduced, many of which are more expressive than context-free grammar and motivated by the desire to provide a more adequate analysis of natural language syntax. This development started with the transformational grammars of Chomsky (1957, 1965) and has continued with many of the theoretical frameworks mentioned in the previous section, such as LFG and HPSG. In recent years, there has been a special interest in so-called mildly context-sensitive grammars (Joshi, 1985), exemplified by TAG and CCG, which appear to strike a good balance between linguistic adequacy and computational complexity.

Solving the universal parsing problem for a particular type of grammar requires a *parsing algorithm*, i.e., an algorithm that computes analyses for a string x relative to a grammar G. Throughout the years a number of parsing algorithms for different classes of grammars have been proposed and analyzed. For context-free grammar, some of the more well-known algorithms are the Cocke-Kasami-Younger (CKY) algorithm (Kasami, 1965; Younger, 1967), Earley's algorithm (Earley, 1970), and the left corner algorithm (Rosenkrantz and Lewis, 1970). These algorithms all make use of tabulation to store partial results, which potentially allows exponential reductions of the search space and thereby provides a way of coping with ambiguity. In the computational linguistics literature, this technique is best known as *chart parsing* (Kay, 1980; Thompson, 1981). This type of method, which constitutes a form of *dynamic programming* (Cormen et al., 1990), can also be generalized to more expressive grammar formalisms.

Deterministic parsing algorithms, such as LR parsing (Knuth, 1965), can only handle restricted subsets of context-free grammars and have their main use in compilers for programming languages. However, deterministic parsing techniques based on shift-reduce parsing (Aho et al., 1986) have also been applied to natural language parsing, often with the ambition to model human sentence processing (Marcus, 1980; Shieber, 1983). In addition, the generalized LR (GLR) parsing algorithm proposed by Tomita (1987) can handle arbitrary context-free grammars and avoids exponential search by using a graph-structured stack and tabulation of partial results.

Traditional methods for parsing can be described as *constructive* in the sense that they analyze sentences by constructing syntactic representations in accordance with the rules of a given grammar. An alternative is to use an *eliminative* parsing method, which treats the grammar as a set of constraints and views parsing as a constraint satisfaction problem, which can be solved by successively eliminating analyses that violate constraints until only valid analyses remain. This strategy presupposes a compact representation of the space of possible analyses and has therefore mainly been used with syntactic representations that are reducible to an assignment of categories or structural attachments to word tokens, as in Constraint Grammar (CG) (Karlsson, 1990; Karlsson et al., 1995), Constraint Dependency Grammar (CDG) (Maruyama, 1990; Harper and Helzerman, 1995; Menzel and Schröder, 1998), and Topological Dependency Grammar (TDG) (Duchier, 1999). In special cases, this

parsing strategy can also be implemented using finite state techniques, as in Parallel Constraint Grammar (Koskenniemi, 1990, 1997). Besides constructive and eliminative parsing methods, we may also distinguish a *transformational* approach, where parsing starts from some kind of default representation and applies grammatical rules that transform this to an output representation (Brill, 1993; Foth et al., 2004).

We will make no attempt to review the vast literature on grammar parsing here but will limit ourselves to a few observations concerning the properties of the parsing problem and the methods used to solve it.[4] First of all, it is worth noting that the parsing problem for a class of grammars is a well-defined *abstract problem* in the sense of algorithm theory (Cormen et al., 1990), i.e., a relation between a set I of inputs, which in this case are pairs consisting of a grammar G and a string x, and a set O of outputs, which are syntactic representations of strings in $L(G)$. A parsing algorithm provides a solution to this problem by computing the mapping from arbitrary inputs to outputs.

Secondly, the parsing problem for formal grammars is intimately tied to the corresponding *recognition problem*, i.e., the problem of deciding whether the string x is in $L(G)$. It is only strings in $L(G)$ that receive an analysis in the parsing process, and most parsing algorithms in fact solve the recognition problem simultaneously.

Thirdly, we note that the analyses to be assigned to a particular input string x are completely defined by the grammar G itself. For example, if G is a context-free grammar, we may be interested in the set of distinct parse trees that result from derivations of x from the start symbol S of G. In principle, this means that the correctness of a parsing algorithm can be established without considering any particular input strings, since the set of all input-output pairs are given implicitly by the grammar G itself.

The abstract nature of the grammar parsing problem is reflected in the evaluation criteria that are usually applied to parsing methods in this context. For example, a parsing algorithm is said to be *consistent* if, for any grammar G and input string x, it only derives analyses for x that are licensed by G; it is said to be *complete* if, for any G and x, it derives *all* analyses for x that are licensed by G. For example, the grammar in figure 2.3 is ambiguous and assigns to our example sentence not only the analysis in figure 2.1 but also the analysis in figure 2.4. Thus, a complete parsing algorithm must compute both these analyses, while a consistent algorithm must not compute any other analysis. However, both consistency and completeness of an algorithm can be proven without considering any particular grammar G or input string x, given the formal definition of the class of grammars and the relevant notions of derivation and representation.

The same goes for considerations of efficiency, where proofs of complexity, either for particular parsing algorithms or for classes of grammars, provide the

[4] For general overviews of grammar parsing techniques, with special reference to natural language parsing, see Samuelsson and Wirén (2000) and Carroll (2003).

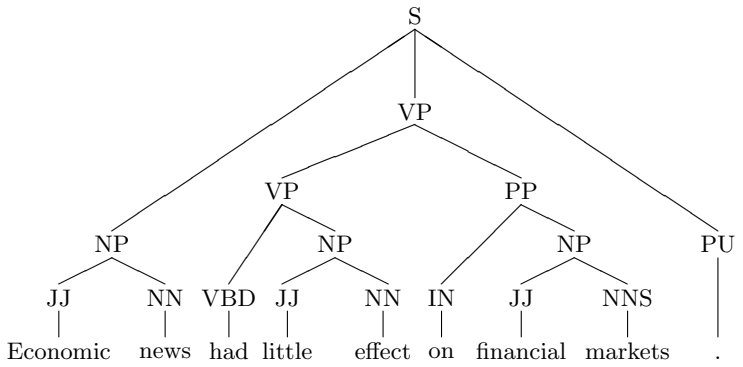


Fig. 2.4. Alternative constituent structure for English sentence (cf. figure 2.1)

most relevant tools for evaluation. Research on the complexity of linguistically motivated classes of grammars was pioneered by Barton et al. (1987) and has been followed by a large number of subsequent studies. For a context-free grammar G, parsing can be performed in $O(n^3)$ time, where n is the length of the input string x, using one of the dynamic programming algorithms mentioned earlier. For mildly context-sensitive grammars, parsing complexity is still polynomial — typically $O(n^6)$ — while for more expressive formalisms running time becomes exponential in the worst case.

Although complexity results often need to be supplemented by practical running time experiments, as shown for example by Carroll (1994), the role of empirical evaluation is rather limited in grammar parsing, especially as far as correctness is concerned. This follows from the fact that grammar parsing, as defined here, is an abstract and mathematically well-defined problem, which can be studied using formal methods only. One setting where such studies have been performed very fruitfully is within the framework known as *deductive parsing* (Shieber et al., 1995).

2.2.2 Text Parsing

The notion of text parsing applies to concrete manifestations of a language L, where we do not necessarily assume that L is a formal language. In particular, we are interested in the case where L is a natural language, or possibly a restricted subset of a natural language. We assume that a *text* in a language L is a sequence $T = (x_1, \ldots, x_n)$ of sentences x_i, and we characterize the *text parsing* problem as follows:

> Given a text $T = (x_1, \ldots, x_n)$ in the language L, derive the correct analysis for every sentence $x_i \in T$.

The term *sentence* should be understood here in the sense of *text sentence* rather than *system sentence* (Lyons, 1977), i.e., it refers to a segment of text without any specific assumptions about syntactic completeness or other structural properties. What constitutes a sentence in this sense may differ from one language to the other and may not always be completely clear-cut. For the time being we will simply disregard this problem, although it is well-known that the problem of sentence segmentation in text processing is far from trivial (Palmer, 2000). It is also worth pointing out that the concept of a text as a sequence of sentences is an abstraction in the sense that it ignores all aspects of non-sequential structure that resides, e.g., in the graphical layout of printed texts. This seems like an appropriate abstraction for written text, which is what we are concerned with in this study, but we will probably need a different abstraction for spoken dialogue, where different principles of structural organization are at play and where the notion of sentence has a more unclear status.

To exemplify the notion of text parsing, let us return again to the example sentence from figure 2.1. In its original context, which is a text taken from the Wall Street Journal and included in the Penn Treebank, this sentence has an interpretation that corresponds to the analysis in figure 2.1 — rather than the alternative analysis in figure 2.4. Therefore, the former analysis is the one and only correct analysis in the context of text parsing.

Let us now return to the observations made concerning grammar parsing in the previous section and see in what respects text parsing is different. First of all, it is not clear that text parsing is a well-defined abstract problem in the same sense as grammar parsing, especially not when we consider texts in a natural language. It is true that text parsing has the structure of a mapping problem, but in the absence of a formal definition for the language L, there is no precise delimitation of the input set. Moreover, even if we can agree on the formal properties of output representations, there is no formal grammar defining the correct mapping from inputs to outputs. For example, the syntactic representation in figure 2.1 is clearly of the kind that can be defined by a context-free grammar. But according to our conception of the text parsing problem, there is no specific instance of this formal grammar that defines the mapping from an input string to a specific representation. In other words, despite a superficial similarity to the definition of grammar parsing, our characterization of the text parsing problem is not a formal definition at all.

One way of looking at the problem is instead to say that it is an empirical *approximation problem*, where we try to approximate the correct mapping given increasingly large but finite samples of the mapping relation. Needless to say, this is a view that fits very well with the data-driven approach to text parsing, which will be discussed in a later section. However, the main point right now is simply that, unlike grammar parsing, the problem of text parsing lacks a precise definition in formal terms.

Secondly, text parsing lacks the connection between parsing and recognition that we observed for grammar parsing. This is a direct consequence of the fact that the input language is not formally defined, which means that recognition is not a well-defined problem. Therefore, we can no longer *require* that an input string be part of the language to be analyzed. In most cases, we instead have to *assume* that any text sentence is a valid input string. And if we want to be able to reject some input strings as ill-formed, then we cannot refer to a formal language definition but must appeal to some other criterion.

Thirdly, while there is no reference to a grammar in the definition of text parsing, there is instead a reference to the sequence of sentences that provide the textual context for each sentence to be analyzed. This is based on the assumption that text parsing deals with language use, and that the analysis assigned to a sentence is sensitive to the context in which it occurs. In particular, we assume that each text sentence has a single correct analysis, even if the string of words realizing the sentence may be found with other interpretations in other contexts. In other words, text parsing entails disambiguation.

However, the absence of a formal grammar also means that we need some external criterion for deciding what is the correct analysis for a given sentence in context. For natural languages, the obvious criterion to use is human performance, meaning that an analysis is correct if it coincides with the interpretation of competent users of the language in question. This leads to the notion of an *empirical gold standard*, i.e., a reference corpus of texts, where each relevant text segment has been assigned its correct analysis by a human expert. In the case of syntactic parsing, the relevant segments are sentences and the corpus will normally be a *treebank* (Abeillé, 2003b; Nivre, forthcoming). Thus, our reason for saying that the analysis given in figure 2.1 is correct is simply that this is the analysis found in the Penn Treebank.

The use of treebank data to establish a gold standard for text parsing is problematic in many ways, and we will return to this problem in chapter 5. For the time being, we will simply note that even if a gold standard treebank can be established, it will only provide us with a finite sample of input-output pairs, which means that any generalization to an infinite language will have to rely on statistical inference. This is in marked contrast to the case of grammar parsing, where the consistency and completeness of parsing algorithms, for any grammar and any input, can be established using formal proofs.

The empirical nature of the text parsing problem is reflected also in the evaluation criteria that are applied to parsing methods in this context. Since notions of consistency and completeness are meaningless in the absence of a formal grammar, the central evaluation criterion is instead the empirical notion of *accuracy*, which is standardly operationalized as agreement with gold standard data and which will be discussed in detail later on. However, it is important to remember that, even though it is often difficult to apply formal methods to the text parsing problem itself given its open-ended nature, the parsing methods we develop to deal with this problem can of course be subjected to the same rigorous analysis as algorithms for grammar parsing. Thus,

if we are interested in the efficiency of different methods, we may use results about theoretical complexity of algorithms as well as empirical running time experiments. Formal methods can also be used to study aspects of robustness and disambiguation, as we shall see later on. However, for the central notion of accuracy, there seems to be no alternative but to rely on empirical evaluation methods, at least not given the current state of our knowledge.

2.2.3 Competence and Performance

The discussion of grammar parsing and text parsing leads naturally to a consideration of the well-known distinction between *competence* and *performance* in linguistic theory (Chomsky, 1965).[5] It may be tempting to assume that grammar parsing belongs to the realm of competence, while text parsing is concerned with performance. After all, the whole tradition of generative grammar in linguistics is built on the idea of using formal grammars to model linguistic competence, starting with Chomsky (1957, 1965). The idea that natural languages can be modeled as formal languages unites theorists as different as Chomsky and Montague (1970), although it is much less prominent in recent formulations of Chomsky's theory (Chomsky, 1981, 1995). Within this tradition, it might be natural to view the study of grammar parsing, when applied to natural language, as the study of idealized human sentence processing.

The traditional notion of linguistic competence has recently been called into question, and it has been suggested that many of the properties typically associated with linguistic performance, such as frequency effects and probabilistic category structure, also belong to our linguistic competence (Bod, Hay and Jannedy, 2003). We will not pursue these complex and controversial issues here, and the nature of linguistic competence, fascinating as it is, falls outside the scope of this study.

On the other hand, it seems quite clear that text parsing is concerned with linguistic performance, at least if we want to use text parsing methods to build practical systems that can handle naturally occurring texts. This means that a model of linguistic competence is of use to us only if it can be coupled with an appropriate model of performance. So, regardless of whether grammar parsing is a good model of linguistic competence or not, it is still an open question what role it has to play in text parsing (cf. Jensen, 1988; Chanod, 2001).

This question will be discussed in detail in the next section, but before we leave the topic of grammar parsing as such, it should be emphasized that this is in any case a very fruitful area of research. Investigations of the complexity of formal grammars and their parsing algorithms have applications in many

[5] Before Chomsky, similar distinctions had been proposed by De Saussure (1916), between *langue* and *parole*, and by Hjelmslev (1943), between *system* and *process*, among others.

areas of computer science, and work on the complexity of linguistically motivated formalisms has had a profound influence on the development of natural language parsing, including the approach investigated in this book.

2.3 Methods for Text Parsing

The main conclusion from the preceding section is that grammar parsing and text parsing are in many ways radically different problems and therefore require different methods. In particular, grammar parsing is an abstract problem, which can be studied using formal methods and internal evaluation criteria, while text parsing is an empirical problem, where formal methods need to be combined with experimental methods and external evaluation criteria. In this section, we will go on to discuss methods that have been proposed for text parsing, which is the problem that concerns us here. Some of these methods crucially involve grammar parsing; others do not.

Following Carroll (2000), we distinguish two broad types of strategy: the *grammar-driven approach* and the *data-driven approach*. However, we want to emphasize at the outset that these types are in a sense stereotypes, representing extreme strategies, which means that many existing approaches actually combine elements of both. In fact, at the end of this section we will even go so far as to suggest that recent developments in the field can be seen as signs of convergence between these apparently opposite strategies. Nevertheless, we believe that it is instructive to start out by considering them as alternative methods, since this will highlight the way they tackle the different problems that arise in parsing unrestricted natural language text.

By necessity, any method for text parsing must rely on an approximation, where a well-defined abstract problem is used as a model of the real practical text parsing problem. By computing a solution to the abstract model problem, we can approximate a solution to the real problem. But the adequacy of this approximation can only be assessed through empirical evaluation. The main difference between the grammar-driven and the data-driven approach lies in the type of abstract problem that is chosen to model the text parsing problem.

2.3.1 Grammar-Driven Text Parsing

In the grammar-driven approach, text parsing is modeled by the abstract problem of grammar parsing. Hence, a formal grammar G is used to define the language $L(G)$ that can be parsed and the class of analyses to be returned for each string in the language. A grammar parsing algorithm is then used to compute the analyses of a given input string, as described in section 2.2.1. The grammar may be hand-crafted or it may be wholly or partially induced from corpus data. It may be coupled with a statistical model for parse selection, and it may allow partial analyses to achieve robustness. But the essence of

the grammar-driven approach, as understood here, is that it is based on a grammar G defining a formal language $L(G)$.

Given our characterization of text parsing in section 2.2.2, it is clear that the grammar-driven approach is based on a crucial assumption, namely that the formal language $L(G)$ is a reasonable approximation of the language L that we want to process. In practice, it is arguably the case that most if not all of the formal grammars that have been developed for natural languages to date fail to meet this assumption, and the formal language $L(G)$ is at best a restricted subset of the natural language L. This does not undermine the grammar-driven approach in principle, since it is always possible to argue that advances in linguistic theory will lead to successively better approximations, but it does create important problems for practical applications of grammar-driven text parsing in the meantime. Many of the research directions in natural language parsing during the last two decades can be seen as motivated by the desire to overcome these problems.

An analogy with syntactic parsing in compilers for programming languages may be illuminating at this point. In a way, parsing computer programs is also a kind of text parsing, but there is a crucial difference in that we know from the start that the approximation $L(G) = L$ can be made perfect, simply because the programming language L is itself a formal language with a precise syntax definition. Moreover, this definition is usually expressible in a carefully restricted subclass of context-free grammar. This does not necessarily mean that compilers always use grammar-driven parsers in practice, or that they use a grammar G such that $L(G)$ exactly coincides with L, but it means that it is always possible to find out whether the approximation is perfect or not. For natural languages, this is unfortunately not the case.

One of the hardest problems for the grammar-driven approach has traditionally been to achieve *robustness*, where robustness can be defined as the capacity of a system to analyze any input sentence. Alternatively, we may view robustness as a matter of degree and say that a system is more robust if it can adequately handle a larger proportion of the input data. In either case, the shortcomings of grammar-driven systems in this respect can be traced back to the fact that some input sentences x_i in a text T are not in the language $L(G)$ defined by the formal grammar G.

Theoretically speaking, it is possible to distinguish two problematic cases where $x_i \notin L(G)$. In the first case, x_i is a perfectly well-formed sentence of the language L and should therefore also be in $L(G)$ but is not. This is sometimes referred to as the problem of *coverage*, since it should be eliminated by increasing the coverage of the grammar. In the second case, x_i is considered not to be part of L, and should therefore not be in $L(G)$ either, but nevertheless has a reasonable syntactic analysis. This can then be called the problem of robustness proper. Examples of the latter type include sentences where some word is misspelled or even omitted, but where it is nevertheless possible to analyze the syntactic structure of (the rest of) the sentence.

For example, if our example sentence from figure 2.1 had contained the token *impact* or the token *effact*, instead of the token *effect*, then the sentence would not have been included in the language defined by the grammar in figure 2.3. In the first case, this would be a problem of coverage, since *impact* is a real English word, which can occur in the same structural position as *effect*. In the second case, it would be a problem of robustness, since *effact* is not a word of English.

However, even though there are many clear-cut examples like these, there are also many cases where it is difficult to decide whether a sentence that is not in $L(G)$ is in L, at least without making appeal to a prescriptive grammar for the natural language L. For certain practical applications, such as grammar checking, it is obviously both relevant and necessary to use this kind of information, but it can be problematic in the general case. Moreover, since it is difficult to apply the distinction between coverage and robustness to approaches that are not grammar-driven, we will not try to maintain this distinction in general but simply treat all failures to analyze input sentences as problems of robustness.[6]

As pointed out by Samuelsson and Wirén (2000), there are essentially two methods that have been proposed to overcome the robustness problem for grammar-driven systems. The first is to relax the grammatical constraints of G in such a way that a sentence outside $L(G)$ can be assigned a complete analysis (Jensen and Heidorn, 1983; Mellish, 1989). This method has the potential drawback that the number of relaxation alternatives that are compatible with analyses of the complete input may become extremely large, and this in turn will aggravate the problem of disambiguation to be discussed below.

The second method is to maintain the constraints of G but to recover as much structure as possible from well-formed fragments of the sentence. This leads to the notion of *partial parsing*, which has been explored within many different frameworks such as deterministic parsing (Hindle, 1989, 1994), chart parsing (Lang, 1988), finite state parsing (Ejerhed, 1983; Koskenniemi, 1990, 1997; Abney, 1991, 1996; Oflazer, 2003), and Constraint Grammar (CG) parsing (Karlsson, 1990; Karlsson et al., 1995), and which nowadays includes not only recovery techniques but also approaches that never attempt to build a complete syntactic structure. Compared to constraint relaxation, partial parsing has the advantage of hiding ambiguity instead of increasing it, since a partial syntactic analysis can be viewed as an underspecified representation of a complete analysis and therefore fails to exhibit some of the ambiguities that distinguish non-equivalent complete analyses. For example, a chunking analysis can be viewed as a partial, underspecified constituency analysis, and a CG analysis as a partial dependency analysis.

In this way, partial parsing can be seen as a way to sacrifice completeness and depth of analysis to improve robustness and efficiency (Abney, 1997).

[6] For a more extensive discussion of robustness in natural language parsing, see Menzel (1995), Junqua and Van Noord (2001) and Basili and Zanzotto (2002).

This may be useful especially in applications that do not require a complete syntactic analysis. However, it is also possible to view partial parsing as a way to break down the complex parsing problem into subproblems that may be easier to manage. This leads to the notion of *cascaded partial parsing* (Abney, 1996), where full parsing is achieved through a sequence of partial parsers, where each parser takes as input the output of the preceding one. A variant of this approach is the use of *supertagging*, pioneered by Bangalore and Joshi (1999), where the words of a sentence are annotated with rich structural or functional categories in order to facilitate the derivation of a syntactic structure in a second step (Joshi and Sarkar, 2003). In a similar vein, the framework of Functional Dependency Grammar (FDG) (Tapanainen and Järvinen, 1997; Järvinen and Tapanainen, 1998) uses CG parsing as a form of supertagging for the construction of dependency structures.

Although we have made a distinction between constraint relaxation and partial parsing as two strategies for achieving robustness in grammar-based text parsing, it should be pointed out that practical implementations often combine the two techniques. For example, in a CG parser that constructs a partial analysis by applying constraints in an eliminative fashion, there is usually a mechanism to prevent the elimination of the last analysis, which is in fact a general mechanism for constraint relaxation that guarantees that the system will output an analysis even if there is no analysis that satisfies all the constraints of the grammar.

Another major problem for grammar-driven text parsing is the problem of *disambiguation*, which is caused by the fact that the number of analyses assigned to a sentence x_i by the grammar G can be very large, while text parsing requires that a small number of analyses (preferably a single one) are selected as appropriate in the context of the text T. For example, the grammar in figure 2.3 assigns two different analyses to our example sentence (cf. figure 2.1 and figure 2.4), which means that a text parsing system using this grammar must incorporate a mechanism for selecting one of them as correct in the given context.[7]

Again, we can make a theoretical distinction between two reasons that the grammar parser outputs more than one analysis for a given string. On the one hand, we have cases of true ambiguity, i.e., where x_i admits of more than one syntactic analysis in the language L, even though only one of them is appropriate in the textual context, and where the grammar G captures this by assigning several analyses to x_i. On the other hand, it may be the case that the grammar G contains rules that license analyses for x_i that are never encountered in L. The latter problem is sometimes called the *leakage* problem, in allusion to Sapir's famous statement that '[a]ll grammars leak' (Sapir, 1921, 39), or simply *overgeneration*. Although one might argue that it

[7] For this particular sentence, it is not clear that the syntactic ambiguity makes a difference in meaning. In fact, it could even be argued that *have little effect* is a light verb construction and that the analysis in figure 2.4 is more appropriate.

is only the former problem that relates to disambiguation proper, it is again very difficult in practice to draw a sharp distinction between problems of leakage and problems of disambiguation, and we will therefore use the term *disambiguation* for the process of reducing the number of analyses assigned to a string, whether the analyses should be licensed by an underlying grammar or not.

Early work related to the ambiguity problem used specialized grammars for different domains of text. Even though this will not lead to complete disambiguation in all cases, it can drastically reduce the number of analyses assigned to a given string, compared to broad-coverage domain-independent grammars. A more sophisticated variant of this approach, which also allows grammar resources to be reused across domains, is to use machine learning techniques to specialize a grammar to a new domain (Grishman et al., 1984; Samuelsson and Rayner, 1991).

A different approach to disambiguation in grammar-driven text parsing is to use deterministic processing and try to ensure that, as far as possible, a correct decision is made at each nondeterministic choice point corresponding to an ambiguity (Marcus, 1980; Shieber, 1983). As mentioned earlier, this line of research has often been motivated by a desire to model human sentence processing, which is assumed to use deterministic disambiguation in combination with backtracking if necessary. An early version of the framework investigated in this book used a simple grammar together with hand-crafted heuristics to achieve deterministic dependency parsing (Nivre, 2003).

Disambiguation is not independent of the basic parsing methodology, and it could be argued that eliminative and transformational methods are more geared towards disambiguation than traditional constructive methods. In an eliminative framework, parsing and disambiguation can be said to coincide, since parsing is performed by successively eliminating candidate analyses, as in the CG framework or its descendant FDG. On the other hand, this kind of eliminative parsing presupposes that we construct a compact representation of the space of possible analyses, which means that it can also be viewed as a very simple and efficient form of constructive parsing followed by eliminative disambiguation. In the transformational approach, parsing is instead the successive transformation of a single representation, which means that there is only one analysis available at any point in time. This technique has been used for disambiguation in transformation-based parsing (Brill, 1993) and Weighted Constraint Dependency Grammar (WCDG) (Foth et al., 2004).

However, the most common approach to disambiguation in recent years has been the use of statistical information about the text language L to rank multiple competing analyses (n-best parsing) or to select a single preferred analysis. There are several ways in which statistical information can be integrated into the grammar-driven approach, but the most straightforward approach is to use a stochastic extension of a formal grammar, the most well-known example being *probabilistic context-free grammar* (PCFG). In a PCFG, every context-free production is associated with a probability p in such a way

S → NP VP PU	1.0	JJ → Economic	0.3
VP → VP PP	0.3	JJ → little	0.5
VP → VBD NP	0.7	JJ → financial	0.2
NP → NP PP	0.2	NN → news	0.4
NP → JJ NN	0.5	NN → effect	0.6
NP → JJ NNS	0.3	NNS → markets	1.0
PP → IN NP	1.0	VBD → had	1.0
PU → .	1.0	IN → on	1.0

Fig. 2.5. Probabilistic context-free grammar for a fragment of English

that the probabilities of all productions with the same left-hand side sum to 1. Thus, the probability of a production $A \to \omega$ is the conditional probability $P(\omega \mid A)$ of the right-hand side ω given the left-hand side A. The probability of a parse tree is then the product of probabilities of all the productions used to construct it (Booth and Thompson, 1973), which amounts to assuming that all production probabilities are mutually independent.

Figure 2.5 shows a PCFG based on the CFG in figure 2.3. Although the actual probabilities assigned to the different rules are completely unrealistic because of the very limited coverage of the grammar, it nevertheless serves to illustrate the notion of a probabilistic grammar. According to this grammar, the probability of the parse tree in figure 2.1 is 0.0000756, while the probability of the parse tree in figure 2.4 is 0.0001134. In other words, using this PCFG for disambiguation, we would prefer the second analysis, which attaches the PP *on financial markets* to the verb *had*, rather than to the noun *effect*. According to the annotation in the Penn Treebank, this would not be the correct choice.

Early work on PCFG parsing used unsupervised machine learning, in particular the Inside-Outside algorithm (Baker, 1979), applied to text corpora to estimate the probabilistic parameters of hand-crafted context-free grammars (Fusijaki et al., 1989; Pereira and Schabes, 1992). But given a treebank with context-free syntactic representations, it is also possible to extract productions directly from the analyses in the treebank and to use frequency counts to estimate probabilistic parameters in a supervised fashion, which results in a so-called *treebank grammar* (Charniak, 1996).

The probabilistic model associated with PCFGs has turned out to be a rather blunt tool for disambiguation, because its independence assumptions are such that it misses dependencies that seem to be important for correct disambiguation. For example, bilexical dependencies, i.e., dependencies between word pairs, are outside the reach of the basic model. This has led people to explore various kinds of lexicalized stochastic grammars, either based on the PCFG model or on other formal grammars such as Lexical Tree Adjoining Grammar (LTAG) (Schabes et al., 1988), for which stochastic versions have been defined by Schabes (1992) and Resnik (1992). In fact, most of the advances in text parsing during the last decade can be traced to the development

of better statistical methods for disambiguation. This often results in a combination of the grammar-driven and data-driven approaches, and we will postpone our discussion of these methods until the next section, where they will be treated together with other data-driven methods for text parsing.

The problems of robustness and disambiguation cannot be studied in isolation from the problem of *accuracy*. If robustness and disambiguation have traditionally been considered the stumbling blocks for grammar-driven text parsing, it is often assumed that this approach has an advantage with respect to accuracy, since the grammar G is meant to guarantee that the analysis assigned to a sentence x_i in a text T is linguistically adequate. However, even if we disregard the leakage problem, this argument is only tenable as long as we do not require robustness and disambiguation. As we have seen above, robustness may require the analysis of strings that are not in the language $L(G)$ defined by the grammar. And disambiguation normally entails discarding most of the analyses assigned to a string by the grammar. All other things being equal, these requirements will decrease the likelihood that a given string $x_i \in T$ is assigned the contextually correct analysis by the parsing system. This means that we need to consider the joint optimization of robustness, disambiguation and accuracy, even if we can decide to prioritize them differently.

The need for joint optimization also includes the final problem that we will consider, namely *efficiency*, which can be a more or less serious problem for the grammar-driven approach depending on the expressivity and complexity of the formal grammars used. For many linguistically motivated frameworks, such as LFG and GPSG, the parsing problem has been shown to be computationally intractable (Barton et al., 1987).[8] It should be remembered, though, that these results concern the theoretically worst case, which for many frameworks occurs only under very special circumstances. Therefore, exponential time algorithms can often be used in practical parsing systems, possibly in combination with special mechanisms to limit the computational effort when the worst-case exponential behavior is encountered (Kaplan et al., 2004). Another approach, which is less common in practice, is to make use of context-free approximations of the full-fledged grammar (Torisawa et al., 2000).

But even for grammars that allow parsing in polynomial time and space, efficiency can be a problem in practical applications. This is the case for many of the mildly context-sensitive grammar formalisms that have been proposed for natural language syntax, such as Tree Adjoining Grammar (TAG) (Joshi, 1985, 1997) and Combinatory Categorial Grammar (CCG) (Steedman, 2000), where the time complexity for parsing is $O(n^6)$ relative to the length n of the input string. Moreover, the requirements of robustness and disambiguation can easily compromise efficiency. Enforcing robustness by relaxing constraints may lead to a combinatorial explosion in the number of possible analyses,

[8] LFG parsing is NP hard, which means that it is probably not computable in polynomial time; GPSG parsing is EXP-POLY hard, which means that it is certainly not computable in polynomial time (Barton et al., 1987).

and disambiguation may require the enumeration of an exponential number of analyses from the compact tabular representation computed during parsing. In addition, both time and space complexity are normally dependent on the size of the grammar, as well as the degree of lexical ambiguity, factors that may in fact dominate the time and space consumption as grammar size grows with the increased coverage required by robustness. However, important progress has been made in recent years to speed up parsing for highly expressive grammar frameworks, as reported, e.g., in Malouf et al. (2000); Oepen and Carroll (2000); Riezler et al. (2002); Miyao et al. (2003); Kaplan et al. (2004); Clark and Curran (2004); Curran and Clark (2004).

Finally, it is worth noting that the most efficient methods for grammar-driven text parsing are those that are based on deterministic algorithms, whether they apply to context-free grammars (Hindle, 1989, 1994), to finite state models (Ejerhed, 1983; Koskenniemi, 1990, 1997; Abney, 1991, 1996; Roche, 1997; Oflazer, 2003), or to automatically induced finite state approximations of context-free grammars (Pereira and Wright, 1997; Nederhof, 1998, 2000; Mohri and Nederhof, 2001).

2.3.2 Data-Driven Text Parsing

In the data-driven approach to text parsing, a formal grammar is no longer a necessary component of the parsing system. Instead, the mapping from input strings to output analyses is defined by an inductive mechanism applying to a text sample $T_t = (x_1, \ldots, x_n)$ from the language L to be analyzed. Hence, the abstract problem used to approximate text parsing in this case is a problem of inductive inference, which may or may not be constrained by a formal grammar. In general, we can distinguish three essential components in a data-driven text parser:

1. A formal model M defining permissible analyses for sentences in L.
2. A sample of text $T_t = (x_1, \ldots, x_n)$ from L, with or without the correct analyses $A_t = (y_1, \ldots, y_n)$.
3. An inductive inference scheme I defining actual analyses for the sentences of any text $T = (x_1, \ldots, x_n)$ in L, relative to M and T_t (and possibly A_t).

This may not be the most familiar way to describe data-driven text parsing, but we believe that this characterization provides a useful abstraction over many existing approaches and furthermore allows a fruitful comparison to the grammar-driven approach.[9]

The first thing to note is that the formal model M may in fact be a formal grammar G, in which case permissible representations will be restricted to strings of the formal language $L(G)$. For example, in the standard PCFG model the permissible analyses are defined by a context-free grammar G. But

[9] For a somewhat different but largely compatible view, see Collins (1999); cf. also Manning and Schütze (2000).

it can also be a model that provides constraints on representations without defining a string language in the process. A simple example would be a model that allows any context-free parse tree whose nonterminal nodes are labeled with symbols from a given set N but whose terminal nodes are labeled with arbitrary word tokens occurring in text sentences of L. As pointed out by Bod (1998), such a model is not a grammar in the formal sense but a dynamic system, since the set of accepted strings cannot be defined independently of the input to the system.

The sample of text T_t, which will normally be called the *training data*, may or may not be annotated with representations satisfying the constraints of M, i.e., it may or may not be extracted from a *treebank* of the language L. If T_t is a treebank sample, then there also exists a corresponding sequence of analyses $A_t = (y_1, \ldots, y_n)$, where y_i is the correct analysis of x_i according to the treebank annotation. Then the inductive inference scheme I will typically be based on a form of *supervised machine learning* (Mitchell, 1997; Hastie et al., 2001). A typical example is the induction of a PCFG by extracting all context-free productions encountered in the treebank and using the relative frequency of each production to estimate its probability, a so-called *treebank grammar* (Charniak, 1996). If T_t is a raw text sample, there is no sequence of analyses given, but *unsupervised learning* may be used. Thus, early work on PCFG parsing applied the Inside-Outside algorithm to raw text corpora in order to estimate the probabilistic parameters of a hand-crafted context-free grammar (Fusijaki et al., 1989; Pereira and Schabes, 1992). However, since the accuracy obtained with unsupervised methods remains inferior to supervised approaches, the latter have dominated the field in recent years.

The inductive inference scheme I, which defines the actual analyses of a given string x, relative to the model M and the sample T_t, can often be decomposed into three distinct components:

1. A parameterized *stochastic model* M_Θ assigning a score $S(x, y)$ to each permissible analysis y of a sentence x, relative to a set of parameters Θ.
2. A *parsing method*, i.e., a method for computing the best analysis y for a sentence x according to $S(x, y)$ (given an instantiation of Θ).
3. A *learning method*, i.e., a method for instantiating Θ based on inductive inference from the training sample T_t.

In the PCFG model, the parameters in Θ are the probabilities associated with the rules of the context-free grammar, while the score $S(x, y)$ is the joint probability $P(x, y)$, which can be computed by multiplying rule probabilities according to the normal independence assumptions. As we have already seen, this score can be used to rank alternative analyses and select the optimal analysis according to the model. As parsing method, we can use one of the standard context-free parsing algorithms extended to PCFG parsing, such as the CKY algorithm (Ney, 1991) or Earley's algorithm (Stolcke, 1995). As learning method, it is common to use some form of maximum likelihood estimation, either based on relative frequencies from a treebank, or based on the

unsupervised Inside-Outside algorithm. In any case, it is important to note that one and the same parameterized stochastic model M_Θ can be combined with different parsing methods as well as different learning methods, and that parsing methods and learning methods are largely independent of each other.

It is worth emphasizing that in order for the system to be usable in practice, there must be effective ways to implement parsing and learning methods, so that the actual analyses for a sentence can be computed with reasonable efficiency. Usually, this computation is divided into a *training phase*, where the learning method is applied once to the training sample T_t in order to estimate the parameters of the model M_Θ, and a *parsing phase*, where analyses are constructed and scored for individual sentences, although the exact division of labor between the phases depends on the methods involved. As noted in relation to the PCFG model, parsing methods in the data-driven approach are often closely related to grammar parsing algorithms, especially if the model M is a formal grammar or some other model with a closely related structure. However, for certain types of models it may not be possible to use standard grammar parsing methods, because the search space defined by the stochastic model is too complex. We will return to this problem when we discuss the efficiency problem for the data-driven approach.

In the previous section, we observed that grammar-based text parsing rests on the assumption that the text language L can be approximated by a formal language $L(G)$ defined by a grammar G. The data-driven approach is also based on an approximation, but this approximation is of an entirely different kind. While the grammar-based approximation in itself only defines permissible analyses for sentences and has to rely on other mechanisms for textual disambiguation, the data-driven approach tries to approximate the function of textual disambiguation directly. And while the grammar-based approximation is an essentially deductive approach, the data-driven approach is based on inductive inference from a finite sample $T_t = (x_1, \ldots, x_n)$ to the infinite language L.

It is a fundamental property of inductive inference that without making any a priori assumptions we have no rational basis for choosing one hypothesis over the other. For instance, there is an infinite number of languages L that are compatible with any finite text sample T_t. Thus, in order to support any kind of generalization beyond the training sample T_t, the inference scheme I must introduce an *inductive bias*, which can be defined as a minimal set of assertions B such that our inferences are entailed by B together with (a logical description of) the sample T_t (Mitchell, 1997). The bias of a particular model will in general depend both on the stochastic model M_Θ and on the learning method used.

Choosing the right inductive bias is essential for a good approximation, and research on machine learning of natural language during the last ten to fifteen years has produced many results about what bias may be appropriate for different problems in natural language processing. We will review some of this research later in this chapter, when we discuss the problem of disambiguation

in data-driven parsing. For the time being, we will simply observe that whereas the grammar-driven approach depends on a more or less satisfactory language approximation, the data-driven approach depends on inductive inference from a more or less representative language sample using a more or less appropriate inductive bias. These different starting points explain why problems such as robustness, disambiguation, accuracy and efficiency may appear quite different in the two extreme approaches. Let us now proceed to an examination of these problems in the context of data-driven text parsing.

If the grammar-based approach is sometimes characterized as being strong with respect to accuracy, but weaker with respect to robustness, disambiguation and efficiency, the reverse is often said to be true for the data-driven approach. In both cases, this is at best an oversimplification. Starting with *robustness*, there is no reason that the data-driven approach should be inherently more robust than the grammar-based approach. It all depends on properties of the formal model M as well as the inference scheme I used for generalization to unseen sentences. However, it is a contingent fact about most existing data-driven systems for text parsing that these components are defined in such a way that any possible input string x is assigned at least one analysis, which means that the robustness problem is eliminated.

This kind of absolute robustness can be illustrated by the framework of Data-Oriented Parsing (DOP) (Bod, 1995, 1998; Bod, Scha and Sima'an, 2003). More precisely, we will consider the model DOP3 in (Bod, 1998), which can be described as follows:

1. The formal model M defines as a permissible analysis for a string x any parse tree that can be composed from subtrees of trees in the text sample, using leftmost node substitution and allowing the insertion of words from x (even if these do not occur in the training sample).
2. The text sample $T_t = (x_1, \ldots, x_n)$ is a sample of sentences from a treebank containing the corresponding context-free parse trees $A_t = (y_1, \ldots, y_n)$.
3. The inductive inference scheme I is based on a stochastic model M_Θ that defines the probability $P(x, y)$ to be the sum of the probabilities $P(D_i)$, for every derivation D_i of y for x, which in turn is defined as the product of the probabilities $P(t_j)$, for every subtree t_j used in D_i.

It is a fundamental property of this model that, since any subtree has a non-zero probability, and since parse trees can be composed from arbitrary subtrees, any input string x can be assigned an analysis y with a non-zero probability $P(x, y)$. This in turn means that, provided that the model can be paired with efficient learning and parsing methods, the robustness problem is eliminated.

A consequence of the extreme robustness is that these data-driven parsers will analyze strings that are probably not in the text language L under any characterization. If we compare this to the grammar-driven language approximation, where the robustness problem arises from the fact that some sentences in L are not in the language $L(G)$ defined by the grammar, we can say that

the data-driven approach avoids the robustness problem by a kind of superset approximation, i.e., any sentence in L is a string that can be analyzed by the parser, but not necessarily vice versa.

Another consequence is that the number of analyses assigned to each input string will usually be very large. In this way, the development from PCFGs to robust DOP models can be seen as an extreme form of constraint relaxation, which is one of the methods used to alleviate the robustness problem in grammar-driven parsing but which can easily lead to an explosion in the number of analyses assigned to a sentence (cf. section 2.2.1). However, given a proper stochastic model, this is a rather controlled form of constraint relaxation, since all the different analyses can be ranked with respect to their score, which also means that we have a method for pruning the search space in a principled way, which is often necessary for reasons of efficiency.

Given the way in which the data-driven approach normally eliminates the robustness problem, there is little use for the second main technique to handle robustness in the grammar-driven approach, namely partial parsing. However, this does not mean that data-driven methods cannot be used to achieve partial parsing, e.g., chunking, as shown very early by Church (1988), using probabilistic methods, and later on by Ramshaw and Marcus (1995), using transformation-based learning. Other learning methods that have been used for data-driven partial parsing include memory-based learning (Veenstra, 1998; Tjong Kim Sang and Veenstra, 1999) and support vector machines (Kudo and Matsumoto, 2000). A collection of data-driven methods applied to partial parsing can be found in Cardie et al. (2000) and a survey of the field in Hammerton et al. (2002). Data-driven methods have also been used to construct cascaded partial parsers that approximate full parsing, e.g., using Hidden Markov Models (Brants, 1999) or memory-based partial parsing and grammatical relation finding (Argamon et al., 1998; Daelemans et al., 1999; Tjong Kim Sang and Veenstra, 2001; Buchholz, 2002).

As already noted, the problem of *disambiguation* can in many cases be even more severe in data-driven text parsing than for traditional grammar-driven systems, since the improved robustness that is the result of extreme constraint relaxation comes at the expense of massive overgeneration or leakage. However, this is compensated by the fact that the inductive inference scheme provides a mechanism for disambiguation, either by associating a score with each analysis, intended to reflect some optimality criterion, or by implicitly maximizing this criterion in a deterministic selection. In general, inductive learning methods can be grouped into three groups according to the type of mechanism they provide for ranking or selection (Jebara, 2004):

1. Generative models score analyses with the joint probability $P(x, y)$ of the string x and the analysis y.
2. Conditional models score analyses with the conditional probability $P(y \mid x)$ of the analysis y given the string x.

3. Discriminative models select the analysis y that maximizes the conditional probability $P(y \mid x)$ (without computing it).

Conditional and discriminative models are not always distinguished in the literature, and the term *discriminative* is often used to cover all models that attempt to maximize the conditional probability $P(y \mid x)$. It is also important to distinguish the structure of the model and the probability that is maximized, which is what we are concerned with here, from the method of parameter estimation, which may also be classified as generative, conditional or discriminative (Klein and Manning, 2002; Henderson, 2004). We will return to this distinction later on.

In the previous section, we discussed some of the early work on data-driven disambiguation, based on the generative PCFG model. The relatively poor parsing accuracy achieved with this model is usually attributed to two major weaknesses: the lack of sensitivity to lexical dependencies and the lack of sensitivity to structural preferences (Collins, 1999).[10] The first observation has motivated a wide range of experiments with lexicalized probabilistic models (Hindle and Rooth, 1991; Schabes, 1992; Resnik, 1992; Collins and Brooks, 1995; Charniak, 1997a). And even though the significance of lexicalization has later been called into question by Klein and Manning (2003), it is striking that virtually all current models for data-driven disambiguation make use of lexical information. Many of these models, which are based on binary relations between lexical items, can be subsumed under the notion of *bilexical grammar* (Eisner, 2000). Bilexical relations are especially important in models based on dependency representations of syntactic structure, such as those of Eisner (1996a,b), Yamada and Matsumoto (2003), and the models investigated in this book (Nivre et al., 2004; Nivre and Scholz, 2004).

The second weakness of the PCFG model is the lack of sensitivity to structural preferences, such as the preference for right-branching or left-branching structures in different languages (Collins, 1999). In order to capture such preferences, we need a different parameterization of syntactic structures, with more flexible independence assumptions, so that the probability of a certain structure can be conditioned on the most significant parts of the surrounding structure. This was the main motivation for the development of *history-based* models of natural language processing, which were first introduced by Black et al. (1992) and have been used extensively both for tagging and parsing. The idea is to map each pair (x, y) of an input string x and an analysis y to a sequence of decisions $D = (d_1, \ldots, d_n)$. In a generative model, the joint probability $P(x, y)$ can then be expressed using the chain rule of probabilities as follows:

$$P(x, y) = P(d_1, \ldots, d_n) = \prod_{i=1}^{n} P(d_i \mid d_1, \ldots, d_{i-1}) \tag{2.1}$$

[10] Further problems with the PCFG model are discussed by Briscoe and Carroll (1993); cf. also Klein and Manning (2003).

The conditioning context for each d_i, $(d_1, \ldots, d_{i-1})$, is referred to as the *history* and usually corresponds to some partially built structure. In order to get a tractable learning problem, histories are then grouped into equivalence classes by a function Φ (Black et al., 1992):

$$P(x, y) = P(d_1, \ldots, d_n) = \prod_{i=1}^{n} P(d_i \mid \Phi(d_1, \ldots, d_{i-1})) \tag{2.2}$$

Early versions of this scheme were integrated into grammar-driven systems. For example, Black et al. (1993) used a standard PCFG but could improve parsing performance considerably by using a history-based model for bottom-up construction of leftmost derivations. Briscoe and Carroll (1993) instead started from a unification-based grammar and employed LR parsing, using supervised learning to assign probabilities to transitions in an LALR(1) parse table constructed from the context-free backbone of the original grammar (cf. also Carroll and Briscoe, 1996).

Generative history-based models are most well-known from the influential work of Collins (1997, 1999) and Charniak (2000). These models are based on a stochastic process generating parse trees top-down, where the children of a given node are generated, not by complete productions as in the PCFG model, but by Markov processes generating one child at a time, conditioned on some partially built structure. Charniak (1997b) uses the popular term *Markov grammar* for this kind of model, although these models are not strictly speaking grammar-driven, since they will normally accept any input string in the same way as the DOP model considered earlier. The Collins and Charniak models are also similar in that they make heavy use of lexical dependencies, a property inherited from their earlier models (Collins, 1996; Charniak, 1997a). Generative models of a similar kind have also been proposed for dependency-based syntactic representations, e.g., by Eisner (1996a,b) and Wang and Harper (2004).

The various models proposed within the DOP framework (Bod, 1995, 1998; Bod, Scha and Sima'an, 2003) are usually not considered to be history-based, although they are normally generative models. However, it is not hard to relate them to the history-based approach. The first difference is that the DOP model assumes a many-to-one relationship between derivations and analyses, which means that the probability $P(x, y)$ has to be computed as a sum over all derivations $D_i = (d_1, \ldots, d_{n_i})$ of the analysis y for the input string x (assuming that $D_1, \ldots, D_m$ are all the derivations of analysis y for input x):

$$P(x, y) = \sum_{i=1}^{m} P(d_1, \ldots, d_{n_i}) = \sum_{i=1}^{m} \prod_{j=1}^{n_i} P(d_j \mid \Phi(d_1, \ldots, d_{j-1})) \tag{2.3}$$

The second difference is that derivations are defined only in terms of which fragments are used to build the analysis, where fragments are assumed to be independent of each other:

$$P(x,y) = \sum_{i=1}^{m} P(d_1, \ldots, d_{n_i}) = \sum_{i=1}^{m} \prod_{j=1}^{n_i} P(d_j) \tag{2.4}$$

The idea of summing over all derivations appears to be good for disambiguation but makes parsing intractable (Bod, 1998, chapter 4), a problem to which we will return later in this section. The DOP models have a lot in common with the stochastic version of LTAG parsing, which is also based on a sum-of-products model, albeit in combination with a grammar-based approach where permissible fragments and composition operations are defined by the LTAG formalism (Joshi and Sarkar, 2003).

History-based models can also be used to define conditional models, where the pair (x, y) is still modeled as a sequence of decisions but where the input string x is a conditioning variable:

$$P(y \,|\, x) = P(d_1, \ldots, d_n \,|\, x) = \prod_{i=1}^{n} P(d_i \,|\, \Phi(d_1, \ldots, d_{i-1}, x)) \tag{2.5}$$

Conditional history-based models were used by Jelinek et al. (1994), in the first system that did not make use of a hand-crafted grammar but extracted all the necessary information from treebank data, and by Magerman (1995), in the first system that showed a significant improvement over systems based on the standard PCFG model. While both these systems used probabilistic decision trees for parameter estimation, later versions of the conditional history-based approach are mostly based on maximum entropy models (Berger et al., 1996; Della Pietra et al., 1997), also known as exponential or log-linear models, as pioneered by Ratnaparkhi (1997, 1999). Conditional history-based models for dependency-based representations have been investigated in Eisner (1996a,b).

Whereas the early conditional models, including that of Ratnaparkhi (1997, 1999), only scored individual parsing decisions in isolation, later work has extended the idea to models that score complete analyses. These models are often used to rerank a small set of analyses produced by another model, typically a generative model, as in the work of Johnson et al. (1999); Collins (2000); Collins and Duffy (2002); Collins and Koo (2005); Charniak and Johnson (2005). This type of model has also been used very successfully for parse selection in grammar-driven frameworks based on linguistic theories such as LFG (Riezler et al., 2002), HPSG (Toutanova et al., 2002; Miyao et al., 2003), and CCG (Clark and Curran, 2004).

Conditional models, such as the maximum entropy model, are normally combined with conditional parameter estimation, i.e., estimation procedures that try to maximize the conditional likelihood of the training data. But conditional estimation can also be used with generative models. Thus, Johnson (2001) obtained a small (non-significant) improvement for a standard PCFG model, using maximum conditional likelihood estimation instead of the traditional maximum joint likelihood estimation. More recently, Henderson (2004) has shown that a history-based generative model based on left-corner parsing,

combined with conditional parameter estimation, outperforms both the generative model with joint estimation and a conditional model with conditional estimation. These results are in line with the argument of Klein and Manning (2002) to the effect that whereas conditional estimation methods often have an advantage over joint estimation, conditional model structure may in fact be harmful in many cases (although the empirical results in their article concern part-of-speech tagging rather than parsing).

There are also purely discriminative versions of the history-based model, i.e., models that implicitly try to maximize the conditional probability of each parsing decision without actually computing it. These models are usually combined with deterministic processing, since discriminative learning does not support a probabilistic scoring and ranking of complete analyses. In this category, we find most data-driven methods for partial parsing based on discriminative learning and deterministic left-to-right processing discussed earlier (Argamon et al., 1998; Daelemans et al., 1999; Kudo and Matsumoto, 2000; Tjong Kim Sang and Veenstra, 2001; Buchholz, 2002). Although these methods are seldom associated with the notion of history-based parsing in the literature, they do in fact implement a history-based strategy, classifying segments from left-to-right while basing their decisions on a combination of input features and previously classified segments.

Discriminative history-based methods for full parsing have been proposed for dependency-based representations by Yamada and Matsumoto (2003), who use a form of shift-reduce parsing for dependency-based representations in combination with support vector machines (Vapnik, 1995). Within the framework investigated in this book, Nivre et al. (2004) and Nivre and Scholz (2004) use a similar parsing algorithm but rely on memory-based learning (Daelemans and Van den Bosch, 2005) to predict parser actions. These frameworks are similar to early conditional parsing models, such as Ratnaparkhi (1997, 1999), in that inductive learning applies to individual parsing actions in isolation. A more holistic discriminative approach to full parsing can be found in the memory-based framework of Kübler (2004), where new analyses are constructed from arbitrarily large fragments of analyses for similar sentences in the training data, in a way which has close affinities with the DOP framework. Another close relative of DOP based on memory-based learning can be found in De Pauw (2003).

The problem of *accuracy* has been implicit throughout the discussion of disambiguation in this section. Since the data-driven approach, as defined here, always provides a mechanism for disambiguation, whether stochastic or deterministic, it is usually trivial to fulfill the requirement of disambiguation as such. Hence, research in this area during the last ten to fifteen years has mainly been focused on improving the accuracy of disambiguation. However, it is also worth noting that, because the data-driven approach incorporates an optimization criterion in the training phase, whether explicitly or implicitly, it is possible to optimize models for different criteria of accuracy. The relationship between estimation objectives and evaluation metrics has been the

subject of several studies (Goodman, 1996, 1998; Johnson, 2001; Klein and Manning, 2002; Sima'an, 2003).

With respect to the final problem of *efficiency*, the conventional wisdom seems to be that the data-driven approach is superior to the grammar-driven approach, but often at the expense of less adequate output representations (Kaplan et al., 2004). However, in reality we find as much variation among data-driven approaches as among grammar-driven approaches, and the overall picture is in fact very similar.

At one end of the scale, we find frameworks where the parsing problem is computationally intractable, such as the original DOP model (Sima'an, 1996a, 1999). Research on efficient parsing within the DOP framework has therefore focused on finding efficient approximations that preserve the advantage gained in disambiguation by considering several derivations of the same analysis. While early work focused on a kind of randomized search strategy called Monte Carlo disambiguation (Bod, 1995, 1998), the dominant strategy has now become the use of different kinds of PCFG reductions (Sima'an, 1996b; Goodman, 1996; Bod, 2001, 2003).

At the other end of the scale, we find highly efficient methods that perform parsing in linear time, such as the various deterministic methods for partial parsing. However, linear-time processing is achievable also for full parsing, either as a theoretical worst case (Nivre, 2003) or as an empirical average case (Ratnaparkhi, 1997, 1999).

In between, we find parsers based on history-based probabilistic models, whether generative or conditional, where parsing in principle consists in deriving all the possible analyses for a given input string and selecting the optimal analysis with respect to the probabilistic model. In this case, there is often a trade-off between accuracy in disambiguation and efficiency in processing. Broadly speaking, the more sophisticated models proposed in recent years have generally led to more accurate disambiguation but less efficient processing.

For example, with the standard PCFG model all possible analyses can be constructed in $O(n^3)$ time using a standard algorithm for context-free parsing. Moreover, given the independence assumptions of this model, the selection of the most probable analysis can be integrated into the parsing process using Viterbi optimization (Viterbi, 1967), as shown for the CKY algorithm by Ney (1991) and for Earley's algorithm by Stolcke (1995).

With the more complex history-based models, such as Collins (1997, 1999) and Charniak (2000), parsing becomes less efficient for two reasons. First, the drastic constraint relaxation leads to an explosion in the number of possible analyses for any given input string. Secondly, the more complex probability model does not allow the same reduction of the search space as the standard Viterbi algorithm for PCFGs. This means that, even if parsing does not become intractable, the time complexity may be such that an exhaustive search of the analysis space is no longer practical. For example, the complexity of the parsing algorithm used in Collins (1999) is $O(n^5)$. In practice, most systems

of this kind only apply the full probabilistic model to a subset of all possible analyses, resulting from a first pass based on an efficient approximation of the full model. This first pass is normally implemented as some kind of chart parsing with beam search, using an estimate of the final probability to prune the search space (Caraballo and Charniak, 1998). Another strategy for reducing the set of candidate analyses is to rely on an initial supertagging phase, as in the grammar-driven CCG approach (Clark and Curran, 2004; Curran and Clark, 2004).

Finally, for data-driven approaches the time required for training, although less critical than parsing time, should also be taken into consideration when discussing efficiency. For instance, learning methods that rely on numerical optimization, such as maximum entropy modeling, may require repeatedly reparsing the training corpus with the current model to determine the parameter updates that will improve the training criterion. In this respect, lazy learning methods such as memory-based learning have a clear advantage, since they reduce learning to the efficient storage of training instances. However, this of course means that more processing has to take place at parsing time, which may limit the advantage.

2.3.3 Converging Approaches

We have now considered two different strategies for text parsing, the grammar-driven approach and the data-driven approach. Although these strategies have different points of departure, we have seen that they in practice often lead to similar solutions when confronted with the mutually interacting requirements of robustness, disambiguation, accuracy and efficiency. In fact, our characterization of the two approaches is such that many contemporary frameworks for text parsing instantiate both. As illustrated in figure 2.6, we can distinguish approaches that are grammar-driven but not data-driven, such as CG (Karlsson, 1990; Karlsson et al., 1995) and its descendant FDG (Tapanainen and Järvinen, 1997; Järvinen and Tapanainen, 1998), and approaches that are data-driven but not grammar-driven, such as different varieties of history-based parsing (Ratnaparkhi, 1997, 1999; Collins, 1997, 1999; Charniak, 2000), as well as most incarnations of Data-Oriented Parsing (Bod, 1995, 1998; Bod, Scha and Sima'an, 2003). But we also find many frameworks that combine the use of formal grammars with data-driven methods to achieve robustness and disambiguation, such as broad-coverage parsers based on PCFG (Black et al., 1993), LTAG (Bangalore and Joshi, 1999), LFG (Riezler et al., 2002), HPSG (Toutanova et al., 2002; Miyao et al., 2003), and CCG (Clark and Curran, 2004).

Before we close the discussion of grammar-driven and data-driven text parsing, it may be useful to relate these concepts to a few other conceptual distinctions that are often made in the literature on natural language parsing. The first is the distinction between *deep parsing* and *shallow parsing*, which

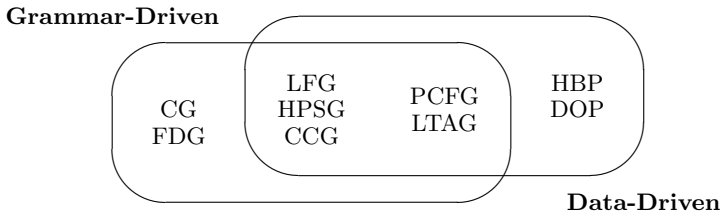


Fig. 2.6. Convergence of grammar-driven and data-driven text parsing

has to do with the amount of information contained in the syntactic representations produced by the parser, where deep parsing typically refers to the kind of representations found in linguistic theories like LFG (Kaplan and Bresnan, 1982), HPSG (Pollard and Sag, 1994) or CCG (Steedman, 2000), while shallow parsing can be exemplified with the skeletal constituent structures found in the first version of the Penn Treebank annotation scheme (Marcus et al., 1993) or the partial specification of grammatical functions in CG (Karlsson, 1990; Karlsson et al., 1995). In other words, this is a distinction between different kinds of syntactic representations and should not be confused with a distinction between different parsing methods. While it is true that most systems for deep parsing are grammar-based, there also exist data-driven approaches to deep parsing, such as the LFG-DOP model (Bod and Kaplan, 1998). And shallow parsing can be performed with grammar-driven as well as data-driven methods.

The second distinction is that between *full parsing* and *partial parsing*, which we have already touched upon several times. This distinction, which is sometimes confused with the previous distinction between deep and shallow parsing, has to do with the completeness of the analysis and can therefore only be defined relative to a specific target representation. Thus, a segmentation of the input string into base chunks can be regarded as a partial specification of a constituent analysis, and an assignment of grammatical functions to individual word tokens can be seen as a partial specification of a dependency structure. This means that both deep and shallow parsing can be implemented as full or partial parsing and that either grammar-driven or data-driven methods may be used in the realization.[11]

The third and final distinction is the distinction between *rule-based* and *example-based* (or *analogical*) language processing, which has to do with whether the analysis of new sentences is based on abstract rules (in a wide

[11] There is a further complication in that the term *partial parsing* is used both about fallback strategies adopted to cope with the robustness problem in full parsing (cf. section 2.3.1) and about approaches where the final goal is always a partial analysis, such as *chunking*. It is only partial parsing in the latter sense that is sometimes referred to as *shallow parsing*.

sense) or on concrete representations (as found in an annotated corpus). Whereas grammar-driven approaches are by necessity rule-based, data-driven approaches can involve more or less abstraction. Thus, most probabilistic parsing models are similar to grammars in that they are abstract generalizations over sets of concrete representations. This is true not only of models that involve a formal grammar, like the PCFG model, but also of most history-based models for parsing, which use training data to estimate the parameters of an abstract probabilistic model.[12] This is in contrast to the DOP framework, where concrete examples in the training corpus are stored as is and used as the raw material for constructing new analyses during processes, thus implementing an example-based approach to syntactic parsing. The same is true of memory-based approaches to parsing, such as Kübler (2004).

One of the conclusions that we want to draw from the discussion in this chapter is that the partly conflicting requirements of robustness, disambiguation, accuracy and efficiency give rise to a complex optimization problem, which we can try to solve in different ways but which always requires a joint optimization. The wide variety of different methods for text parsing can to a large extent be said to result from different optimization strategies and different goals.

The grammar-driven approach, in its purest form, starts from a system with optimal accuracy, in the sense that only sentences for which the correct analysis can be derived are covered, and gradually seeks to improve the system with respect to robustness and disambiguation. However, this development may compromise efficiency, which therefore has to be optimized together with robustness and disambiguation.

By contrast, the data-driven approach, in its most radical form, starts from a system with optimal robustness and disambiguation, in the sense that every sentence gets exactly one analysis, and gradually seeks to improve the system with respect to accuracy. Again, this may lead to problems of efficiency, which therefore has to be optimized together with accuracy. In both cases, we may furthermore put more or less priority on efficiency in relation to other optimization requirements.

When evaluating different strategies, we are of course interested in whether they lead to optimal performance overall. However, because of the interaction between different requirements, we can only define the optimum with respect to a given prioritization. It is true that accuracy is of prime importance, but if optimal accuracy leads to computational intractability, its practical interest is limited. Moreover, from a scientific point of view, we need to increase our knowledge about the complex interaction of conflicting requirements, which means that studies based on different goals and strategies can be valuable,

[12] The term *Markov grammar* for this type of model (Charniak, 1997b) is motivated by this similarity, even though the model normally does not involve a grammar in the formal sense (cf. section 2.3.2).

even if they do not always advance the state of the art in terms of raw performance (on any of the dimensions considered here).

2.3.4 Inductive Dependency Parsing

It seems appropriate to conclude this discussion of methods for text parsing by situating the framework of inductive dependency parsing in the wider landscape of different parsing methods, grammar-driven and data-driven, deep and shallow, full and partial, rule-based and example-based.

Inductive dependency parsing is an instance of the data-driven approach to text parsing, which in this study is combined with a rather extreme optimization strategy. The first step in this strategy is to construct a framework that is provably optimal with respect to robustness, disambiguation and efficiency. The second step is to use inductive learning to gradually improve parsing accuracy without sacrificing robustness, disambiguation or efficiency. The first step will be taken in chapter 3, while the second step will occupy us throughout chapters 4–5.

Robustness will be achieved with the kind of radical constraint relaxation found in many other data-driven approaches, which means that some analysis will be assigned to every input string. However, we will not resort to partial parsing but instead construct complete dependency representations for every input string, even though these representations will normally be rather shallow syntactic representations. Our task could therefore be characterized as full parsing with shallow dependency representations.

The use of dependency representations, which is a central element of the framework, will be discussed in more detail in chapter 3. It will be argued that, over and above any other motivation for using these representations, dependency structures provide the right level of underspecification to allow robust and efficient full parsing.

Disambiguation will be performed by deterministic processing in combination with discriminative learning, using a history-based model for parsing decisions. Although the general approach is compatible with many different learning methods, we will concentrate in this book on the use of memory-based learning of parsing decisions, which means that the approach can also be described as example-based, or analogical, rather than rule-based.

Efficiency in parsing is gained mainly through the use of deterministic processing, which means that parsing can be performed in linear time, which is arguably optimal since any reasonable parsing method will at least have to scan the input from start to end. Moreover, the use of a lazy learning method also gives good efficiency in training.

Given our overall optimization strategy, it is important to have precise evaluation criteria for the basic requirements of robustness, disambiguation, accuracy and efficiency. The last section of this chapter will be devoted to the definition of such criteria.

2.4 Evaluation Criteria

Most of the discussion in this chapter applies to natural language parsing in general and not only to the particular framework investigated in this book. However, when we come to concrete evaluation criteria, it is important that these reflect the general strategy underlying our research efforts. We will therefore concentrate in this section on evaluation criteria that are relevant for the version of inductive dependency parsing developed in this book, criteria that may in some respects be less suitable for other approaches to text parsing.

2.4.1 Robustness

Following the discussion in sections 2.3.1–2.3.2, we regard robustness in text parsing as the capacity of a system to analyze any input sentence. In other words, we do not attempt to draw any distinction between (lack of) coverage and (lack of) robustness. In this respect, we agree with Chanod (2001) that robustness is about exploring all constructions humans actually produce in natural language texts, 'be they grammatical, conformant to formal models, frequent or not' (Chanod, 2001, 188). This is admittedly a rather weak notion of robustness, which is easy to achieve in itself, but which can be considered a prerequisite for the optimization of other criteria. Stronger notions of robustness discussed in the literature, which relate to the degradation of accuracy with increasingly ill-formed input (Menzel, 1995; Basili and Zanzotto, 2002), can be regarded as additional requirements on top of the basic requirement of producing some analysis for every input.

Since we want the requirement of robustness to be independent of other requirements, in particular the requirement of disambiguation, we propose the following definition:

Definition 2.1. A system P for parsing texts in language L satisfies the requirement of *robustness* if and only if, for any text $T = (x_1, \ldots, x_n)$ in L, P assigns *at least* one analysis to every text sentence $x_i \in T$.

Given our optimization strategy, we want to treat robustness as an absolute requirement, which means that we will require formal proofs of this property. In other frameworks, where robustness is an optimization criterion rather than an absolute requirement, it may be more relevant to perform an empirical evaluation of the degree to which robustness can be achieved for representative input texts.

2.4.2 Disambiguation

Disambiguation in text parsing implies the selection of a single analysis for every text sentence from all the analyses that are compatible with the input string according to the formal framework adopted. From this perspective, it is irrelevant whether the input string is genuinely ambiguous or whether the

fact that there are several candidate analyses is due to the so-called leakage problem. It is also irrelevant whether the selection is achieved through an n-best ranking of a large space of candidate analyses or through a deterministic procedure that only constructs a single analysis for a given input sentence. More precisely, we define the requirement of *disambiguation* in the following way:

Definition 2.2. A system P for parsing texts in language L satisfies the requirement of *disambiguation* if and only if, for any text $T = (x_1, \ldots, x_n)$ in L, P assigns *at most* one analysis to every text sentence $x_i \in T$.

Since we want disambiguation to be an absolute requirement for our parsing methods, in the same way as robustness, we will require formal proofs of this property as well. In another context, it would be possible to regard disambiguation as a desideratum rather than a hard requirement, and to perform an empirical evaluation of a system's capacity for disambiguation when applied to representative samples of text.

The requirements of robustness and disambiguation are independent of each other, but any system satisfying both requirements must assign exactly one analysis to every text sentence. This is the notion of *robust disambiguation* that we will adopt as an absolute requirement for inductive dependency parsing.

2.4.3 Accuracy

Requiring a single analysis for every sentence in a text is obviously of limited interest unless we also require the analysis to be correct. According to the characterization of text parsing in section 2.2.2, we assume that there exists a single correct analysis for each sentence in a text. An absolute requirement of *accuracy* could therefore be stated as follows:[13]

Definition 2.3. A system P for parsing texts in language L satisfies the requirement of *accuracy* if and only if, for any text $T = (x_1, \ldots, x_n)$ in L, P assigns the correct analysis to every text sentence $x_i \in T$.

However, given the state of the art in natural language parsing, complete accuracy has to be regarded as an asymptotic goal, impossible to attain in practice for any non-trivial domain of natural language text, and especially in combination with robustness and disambiguation. Moreover, even if perfectly accurate parsing were possible, there would be no formal method for proving that a system satisfies this requirement, since the language L is not a formal language. This means that we must always rely on empirical evaluations using representative samples of text from the given language L.

[13] In a context where it is not assumed that every sentence has exactly one correct analysis, an alternative would be to replace 'the correct analysis' with 'at least one correct analysis'. In this study, the two formulations are practically equivalent, given the absolute requirement of robust disambiguation.

Given the optimization strategy adopted in this study, accuracy should be treated as an optimization criterion, where the goal is to maximize accuracy while maintaining robustness and disambiguation. We will adopt the standard methodology for evaluating accuracy in text parsing, which is to treat samples of treebank data for the language L as an empirical gold standard and use inferential statistics to generalize the results to arbitrary texts in L. As noted in section 2.2.2, this methodology is problematic in several ways, but we will postpone a discussion of these problems until chapter 5. There we will also define the different evaluation metrics that will be used for the evaluation of accuracy.

2.4.4 Efficiency

Finally, in order for methods to be usable in practical applications, they must allow tractable computation, which leads to the requirement of *efficiency*. As is customary, we will restrict ourselves to the computational resources of *time* and (to a lesser extent) *space*. In theory, tractability is usually taken to mean computable in deterministic polynomial time and space (relative to the size of the input), but many practical applications put higher demands on efficiency. Our strategy for optimization implies that we employ parsing algorithms with optimal complexity, which means that both time and space complexity should be linear in the size of the input. Thus:

Definition 2.4. A system P for parsing texts in language L satisfies the requirement of *efficiency* if and only if, for any text $T = (x_1, \ldots, x_n)$ in L, P processes every text sentence $x_i \in T$ in time and space that is linear in the length of x_i.

Although the theoretical complexity of an algorithm can be proven formally, practical efficiency also depends on other factors and is therefore another optimization criterion that can be maximized while maintaining robustness and disambiguation. As discussed in section 2.3.2, there is often a tradeoff between accuracy and efficiency in text parsing, which means that we need to explore the joint optimization of accuracy and efficiency.

Given our strategy for optimization, we need to evaluate efficiency in two different ways. On the one hand, we will provide formal proofs of time and space complexity, establishing asymptotic bounds on worst-case running time and memory requirements. On the other hand, we will perform empirical experiments using treebank data, measuring actual running time and memory consumption.

3

Dependency Parsing

Despite a long and venerable tradition in descriptive linguistics, dependency grammar has until quite recently played a fairly marginal role both in theoretical linguistics and in natural language processing. The increasing interest in dependency-based representations in natural language parsing in recent years appears to be motivated both by the potential usefulness of bilexical relations in disambiguation and by the gains in efficiency that result from the more constrained parsing problem for these representations.

In this chapter, we define a formal framework for parsing with dependency representations and present a parsing algorithm that is provably optimal with respect to the requirements of robustness, disambiguation and efficiency, as defined in the previous chapter. More precisely, we show that, for any input string, the algorithm constructs exactly one dependency structure in time that is linear in the length of the input. The significance of these results follows from the optimization strategy explored in this study, which is to start from optimal robustness, disambiguation and efficiency and gradually improve accuracy without sacrificing other requirements.

Before we turn to the formal framework and the analysis of the parsing algorithm, we will review some previous work on dependency grammar and parsing. Starting with the theoretical foundations of dependency grammar, we move on to consider both grammar-driven and data-driven methods for dependency parsing. In this chapter, we will limit our attention to systems for dependency parsing in a narrow sense, i.e., systems where the analysis assigned to an input sentence takes the form of a dependency structure. This means that we will not discuss systems that exploit dependency relations for the construction of another type of representation, such as the head-driven parsing models of Collins (1997, 1999). Moreover, we will restrict ourselves to systems for full parsing, which means that we will not deal with systems that produce a partial or underspecified representation of dependency structure, such as Constraint Grammar parsers.

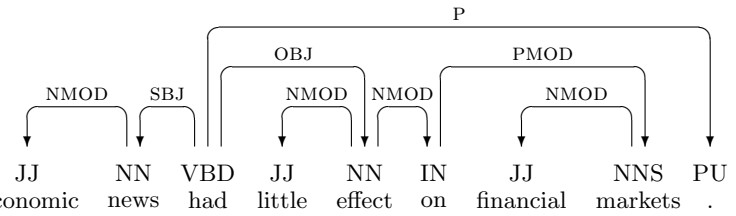


Fig. 3.1. Dependency structure for English sentence (same as figures 1.1 and 2.2)

3.1 Dependency Grammar

Although its roots may be traced back to Pāṇini's grammar of Sanskrit several centuries before the Common Era (Kruijff, 2002) and to medieval theories of grammar (Covington, 1984), dependency grammar has largely developed as a form for syntactic representation used by traditional grammarians, especially in Europe, and particularly in Classical and Slavic domains (Mel'čuk, 1988). This grammatical tradition can be said to culminate with the seminal work of Tesnière (1959), which is also the starting point of the modern theoretical tradition of dependency grammar.

This tradition comprises a large and fairly diverse family of grammatical theories and formalisms that share certain basic assumptions about syntactic structure, in particular the assumption that syntactic structure consists of *lexical* elements linked by binary asymmetrical relations called *dependencies*. Thus, the common formal property of dependency structures, as compared to representations based on constituency is the lack of phrasal nodes. This can be seen by contrasting the dependency representation in figures 1.1 and 2.2, repeated again for convenience in figure 3.1, with the constituency representation in figure 2.1 (cf. also section 2.1).

Among the more well-known theories of dependency grammar, besides the theory of structural syntax developed by Tesnière (1959), we find Word Grammar (WG) (Hudson, 1984, 1990), Functional Generative Description (FGD) (Sgall et al., 1986), Dependency Unification Grammar (DUG) (Hellwig, 1986, 2003), Meaning-Text Theory (MTT) (Mel'čuk, 1988), and Lexicase (Starosta, 1988). Constraint-based theories of dependency grammar have a strong tradition, represented by Constraint Dependency Grammar (CDG) (Maruyama, 1990; Harper and Helzerman, 1995; Menzel and Schröder, 1998) and its descendant Weighted Constraint Dependency Grammar (WCDG) (Schröder, 2002), Functional Dependency Grammar (FDG) (Tapanainen and Järvinen, 1997; Järvinen and Tapanainen, 1998), largely developed from Constraint Grammar (CG) (Karlsson, 1990; Karlsson et al., 1995), and finally Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001), later evolved into Extensible Dependency Grammar (XDG) (Debusmann et al.,

2004). A synthesis of dependency grammar and categorial grammar is found in the framework of Dependency Grammar Logic (DGL) (Kruijff, 2001).

We will make no attempt at reviewing all these theories here. Instead, we will try to characterize their common core of assumptions, centered upon the notion of dependency, and discuss major points of divergence, such as the issue of projective versus non-projective representations.

3.1.1 The Notion of Dependency

The fundamental notion of *dependency* is based on the idea that the syntactic structure of a sentence consists of binary asymmetrical relations between the words of the sentence. The idea is expressed in the following way in the opening chapters of Tesnière (1959):

> La phrase est un *ensemble organisé* dont les éléments constituants sont les *mots*. [1.2] Tout mot qui fait partie d'une phrase cesse par lui-même d'être isolé comme dans le dictionnaire. Entre lui et ses voisins, l'esprit aperçoit des *connexions*, dont l'ensemble forme la charpente de la phrase. [1.3] Les connexions structurales établissent entre les mots des rapports de *dépendance*. Chaque connexion unit en principe un terme *supérieur* à un terme *inférieur*. [2.1] Le terme supérieur reçoit le nom de *régissant*. Le terme inférieur reçoit le nom de *subordonné*. Ainsi dans la phrase *Alfred parle* [...], *parle* est le régissant et *Alfred* le subordonné. [2.2] (Tesnière, 1959, 11–13, emphasis in the original)[1]

In the terminology used in this book, a dependency relation holds between a *head* and a *dependent*. Alternative terms found in the literature are *governor* and *regent* for *head* (cf. Tesnière's *régissant*) and *modifier* for *dependent* (cf. Tesnière's *subordonné*).

Criteria for establishing dependency relations, and for distinguishing the head and the dependent in such relations, are obviously of central importance for dependency grammar. Such criteria have been discussed not only in the dependency grammar tradition, but also within other frameworks where the notion of syntactic head plays an important role, including all constituency-based frameworks that subscribe to some version of X-bar theory (Chomsky, 1970; Jackendoff, 1977). Here are some of the criteria that have

[1] English translation (by the author): 'The sentence is an *organized whole*, the constituent elements of which are *words*. [1.2] Every word that belongs to a sentence ceases by itself to be isolated as in the dictionary. Between the word and its neighbors, the mind perceives *connections*, the totality of which forms the structure of the sentence. [1.3] The structural connections establish *dependency* relations between the words. Each connection in principle unites a *superior* term and an *inferior* term. [2.1] The superior term receives the name *governor*. The inferior term receives the name *subordinate*. Thus, in the sentence *Alfred parle* [...], *parle* is the governor and *Alfred* the subordinate. [2.2]'

been proposed for identifying a syntactic relation between a head H and a dependent D in a construction C (Zwicky, 1985; Hudson, 1990):

1. H determines the syntactic category of C and can often replace C.
2. H determines the semantic category of C; D gives semantic specification.
3. H is obligatory; D may be optional.
4. H selects D and determines whether D is obligatory or optional.
5. The form of D depends on H (agreement or government).
6. The linear position of D is specified with reference to H.

It is clear that this list contains a mix of different criteria, some syntactic and some semantic, and one may ask whether there is in fact a single coherent notion of dependency corresponding to all the different criteria. This has led some theorists, such as Hudson (1990), to suggest that the concept of head has a prototype structure, i.e., that typical instances of this category satisfy all or most of the criteria while more peripheral instances satisfy fewer. Other authors have emphasized the need to distinguish different kinds of dependency relations. According to Mel'čuk (1988), the word forms of a sentence can be linked by three types of dependencies: *morphological*, *syntactic* and *semantic*. According to Nikula (1986), we must distinguish between syntactic dependency in *endocentric* and *exocentric* constructions (Bloomfield, 1933).

Thus, in figure 3.1, the NMOD relation holding between the noun *markets* and the adjective *financial* is an endocentric construction, where the head can replace the whole without disrupting the syntactic structure:

Economic news had little effect on [financial] markets. (3.1)

Endocentric constructions may satisfy all of the criteria listed above, although number 4 is usually considered less relevant, since dependents in endocentric constructions are taken to be optional and not selected by their heads. By contrast, the PMOD relation holding between the preposition *on* and the noun *markets* is an exocentric construction, where the head cannot readily replace the whole:

Economic news had little effect on [markets]. (3.2)

Exocentric constructions, by their definition, fail on criterion number 1, at least with respect to substitutability of the head for the whole, but may satisfy the remaining criteria. Considering the rest of the relations exemplified in figure 3.1, the SBJ and OBJ relations are clearly exocentric, and the NMOD relation from the noun *news* to the adjective *Economic* clearly endocentric, while the remaining NMOD relations (effect → little, effect → on) have a less clear status.

The distinction between endocentric and exocentric constructions is also related to the distinction between *head-complement* and *head-modifier* (or *head-adjunct*) relations found in many contemporary syntactic theories, since head-complement relations are exocentric while head-modifier relations are

endocentric. Many theories also recognize a third kind of relation, the *head-specifier* relation, typically exemplified by the determiner-noun relation, which is exocentric like the head-complement relation, but where there is no clear selection of the dependent element by the head.

The distinction between complements and modifiers is often defined in terms of *valency*, which is a central notion in the theoretical tradition of dependency grammar. Although the exact characterization of this notion differs from one theoretical framework to the other, valency is usually related to the semantic predicate-argument structure associated with certain classes of lexemes, in particular verbs but sometimes also nouns and adjectives. The idea is that the verb imposes requirements on its syntactic dependents that reflect its interpretation as a semantic predicate. Dependents that correspond to arguments of the predicate can be obligatory or optional in surface syntax but can only occur once with each predicate instance. By contrast, dependents that do not correspond to arguments can have more than one occurrence with a single predicate instance and tend to be optional. The *valency frame* of the verb is normally taken to include argument dependents, but some theories also allow obligatory non-arguments to be included (Sgall et al., 1986).

The notion of valency will not play a central role in the investigations in this book, but we will sometimes use the terms *valency-bound* and *valency-free* to make a rough distinction between dependents that are more or less closely related to the semantic interpretation of the head. Returning to figure 3.1, the subject and the object would normally be treated as valency-bound dependents of the verb *had*, while the adjectival modifiers of the nouns *news* and *markets* would be considered valency-free. The prepositional modification of the noun *effect* may or may not be treated as valency-bound, depending on whether the entity undergoing the effect is supposed to be an argument of the noun *effect* or not.

While most head-complement and head-modifier structures have a straightforward analysis in dependency grammar, there are also many constructions that have a relatively unclear status. This group includes constructions that involve grammatical function words, such as articles, complementizers and auxiliary verbs, but also structures involving prepositional phrases. For these constructions, there is no general consensus in the tradition of dependency grammar as to whether they should be analyzed as head-dependent relations at all and, if so, what should be regarded as the head and what should be regarded as the dependent. For example, some theories regard auxiliary verbs as heads taking lexical verbs as dependents; other theories make the opposite assumption; and yet other theories assume that verb chains are connected by relations that are not dependencies in the usual sense.

Another kind of construction that is problematic for dependency grammar (as for most theoretical traditions) is *coordination*. According to Bloomfield (1933), coordination is an endocentric construction, since it contains not only one but several heads that can replace the whole construction syntactically. However, this characterization raises the question of whether coordination can

be analyzed in terms of binary asymmetrical relations holding between a head and a dependent. Again, this question has been answered in different ways by different theories within the dependency grammar tradition.

In conclusion, the theoretical tradition of dependency grammar is united by the assumption that an essential part of the syntactic structure of sentences resides in binary asymmetrical relations holding between lexical elements. Moreover, there is a core of syntactic constructions for which the analysis given by different frameworks agree in all important respects. However, there are also important differences with respect to whether dependency analysis is assumed to cover all aspects of syntactic analysis, and with respect to the analysis of certain types of syntactic constructions. We will now turn to a discussion of some of the more important points of divergence in this tradition.

3.1.2 Varieties of Dependency Grammar

Perhaps the most fundamental open question in the tradition of dependency grammar is whether the notion of dependency is assumed to be not only *necessary* but also *sufficient* for the analysis of syntactic structure in natural language. This assumption is not made in the theory of Tesnière (1959), which is based on the three complementary concepts of *connection* (connexion), *junction* (jonction) and *transfer* (translation), where connection corresponds to dependency (cf. the quotation on page 47) but where junction and transfer are other kinds of relations that can hold between the words of a sentence. More precisely, junction is the relation that holds between coordinated items that are dependents of the same head or heads of the same dependent, while transfer is the relation that is relevant for a function word or other element that changes the syntactic category of a lexical element so that it can enter into different dependency relations. An example of the latter is the relation holding between the preposition *de* and *Pierre* in the construction *le livre de Pierre* (Pierre's book), where the preposition *de* allows the proper name *Pierre* to modify a noun, a dependency relation otherwise reserved for adjectives. Another way in which theories may depart from a pure dependency analysis is to allow a restricted form of constituency analysis, so that dependencies can hold between strings of words rather than single words. This possibility is exploited, to different degrees, in the frameworks of Hellwig (1986, 2003), Mel'čuk (1988) and Hudson (1990), notably in connection with coordination.

A second dividing line is that between mono-stratal and multi-stratal frameworks, i.e., between theories that rely on a single syntactic representation and theories that posit several layers of representation. In fact, most theories of dependency grammar, in contrast to frameworks for dependency parsing that will be discussed in section 3.2, are multi-stratal, at least if semantic representations are considered to be a stratum of the theory. Thus, in FGD (Sgall et al., 1986) there is both an *analytical* layer, which can be characterized as a surface syntactic representation, and a *tectogrammatical*

layer, which can be regarded as a deep syntactic (or shallow semantic) representation. In a similar fashion, MTT (Mel'čuk, 1988) recognizes both *surface syntactic* and *deep syntactic* representations (in addition to representations of deep phonetics, surface morphology, deep morphology and semantics). By contrast, Tesnière (1959) uses a single level of syntactic representation, the so-called *stemma*, which on the other hand includes junction and transfer in addition to syntactic connection.[2] The framework of XDG (Debusmann et al., 2004) can be seen as a compromise in that it allows multiple layers of dependency-based linguistic representations but requires that all layers, or *dimensions* as they are called in XDG, share the same set of nodes. This is in contrast to theories like FGD, where, e.g., function words are present in the analytical layer but not in the tectogrammatical layer.

The different requirements of XDG and FGD point to another issue, namely what can constitute a node in a dependency structure. Although most theories agree that dependency relations hold between *lexical* elements, rather than *phrases*, they can make different assumptions about the nature of these elements. The most straightforward view is that the nodes of the dependency structure are simply the word forms occurring in the sentence, which is the view adopted in most parsing systems based on dependency grammar. But it is also possible to construct dependency structures involving more abstract entities, such as lemmas or lexemes, especially in deeper syntactic representations. Another variation is that the elements may involve several word forms, constituting a *dissociate nucleus* (nucléus dissocié) in the terminology of Tesnière (1959), or alternatively correspond to smaller units than word forms, as in the morphological dependencies of Mel'čuk (1988).

A fourth dimension of variation concerns the inventory of specific dependency types posited by different theories, i.e., functional categories like SBJ, OBJ and NMOD that are used to label dependency arcs in the representation in figure 3.1. Broadly speaking, most theories either assume a set of more surface-oriented grammatical functions, such as *subject*, *object*, *adverbial*, etc., with a more or less elaborate subclassification, or a set of more semantically oriented role types, such as *agent*, *patient*, *goal*, etc., belonging to the tradition of *case roles* or *thematic roles* (Fillmore, 1968; Jackendoff, 1972; Dowty, 1989).[3] Multi-stratal theories often combine the two relation types. Thus, FGD (Sgall et al., 1986) uses grammatical functions in the analytical layer and semantic roles in the tectogrammatical layer. An alternative scheme of representation, which is found in MTT (Mel'čuk, 1988), is to use numerical indices for valency-bound dependents to reflect a canonical ordering of arguments (argument 1,

[2] Tesnière's representations also include *anaphors*, which are described as supplementary semantic connections without corresponding syntactic connections.

[3] The notion of a semantic role can be traced back to Pāṇini's *kānaka* theory (Misra, 1966), which is sometimes also seen as the earliest manifestation of dependency grammar. The notion of a grammatical function also has a long history that extends at least to the work of Appolonius Dyscolus in the second century of the Common Era (Robins, 1967).

2, 3, etc.) and to use descriptive labels only for valency-free dependents. Finally, it is also possible to use unlabeled dependency structures, although this is more common in practical parsing systems than in linguistic theories.

There are several open issues in dependency grammar that have to do with formal properties of representations. Since a dependency representation consists of lexical elements linked by binary asymmetrical relations, it can be defined as a *labeled directed graph*, where the set of nodes (or vertices) is the set of lexical elements (as defined by the particular framework), and the set of labeled arcs represent dependency relations from heads to dependents. In order to provide a complete syntactic analysis of a sentence, the graph must also be *connected* so that every node is related to at least one other node (Mel'čuk, 1988). Again, we refer to figure 3.1 as an illustration of this representation, where the nodes are the word tokens of the sentence (annotated with parts-of-speech) and the arcs are labeled with grammatical functions.[4]

Given this general characterization, which will be properly formalized in section 3.3, we may then impose various additional conditions on these graphs. Two basic constraints that are assumed in most versions of dependency grammar are the *single-head* constraint, i.e., the assumption that each node has at most one head, and the *acyclicity* constraint, i.e., the assumption that the graph should not contain cycles. These two constraints, together with connectedness, imply that the graph should be a rooted tree, with a single root node that is not a dependent of any other node. For example, the representation in figure 3.1 is a rooted tree with the verb *had* as the root node. Although these constraints are assumed in most versions of dependency grammar, there are also frameworks that allow multiple heads as well as cyclic graphs, such as WG (Hudson, 1984, 1990). Another issue that arises for multi-stratal theories is whether each level of representation has its own set of nodes, as in most theories, or whether they only define different arc sets on top of the same set of nodes, as in XDG (Debusmann et al., 2004).

However, the most important and hotly debated issues concerning formal representations have to do with the relationship between dependency structure and word order. According to Tesnière (1959), dependency relations belong to the *structural order* (l'ordre structural), which is different from the *linear order* (l'ordre linéaire) of a spoken or written string of words, and *structural syntax* is based on the relations that exist between these two dimensions. Most versions of dependency grammar follow Tesnière in assuming that the nodes of a dependency structure are not linearly ordered in themselves but only relative to a particular surface realization of this structure. A notable exception to this generalization is FGD, where the representations of both the analytical

[4] There seems to be no general consensus in the literature on dependency grammar as to whether the arcs representing dependency relations should be drawn pointing from heads to dependents or vice versa (or indeed with arrowheads at all). We have chosen to adopt the former alternative, both because it seems to be the most common representation in the literature and because it is consistent with standard practice in graph theory.

layer and the tectogrammatical layer are linearly ordered in order to capture aspects of information structure (Sgall et al., 1986). In addition, there are frameworks, such as TDG (Duchier and Debusmann, 2001), where the linear order is represented by means of a linearly ordered dependency structure, the Linear Precedence (LP) tree, while the proper dependency representation, the Immediate Dominance (ID) tree, is unordered.

However, whether dependency relations introduce a linear ordering or not, there may be constraints relating dependency structures to linear realizations. The best-known example is the constraint of *projectivity*, first discussed by Lecerf (1960), Hays (1964) and Marcus (1965), which is related to the contiguity constraint for constituency representations. A dependency graph satisfies the constraint of projectivity with respect to a particular linear order of the nodes if, for every arc $h \rightarrow d$ and node w, w occurs between h and d in the linear order only if w is dominated by h (where *dominates* is the reflexive and transitive closure of the arc relation). For example, the representation in figure 3.1 is an example of a *projective* dependency graph, given the linear order imposed by the word order of the sentence.

The distinction between *projective* and *non-projective* dependency grammar often made in the literature thus refers to the issue of whether this constraint is assumed or not. Broadly speaking, we can say that whereas most practical systems for dependency parsing do assume projectivity, most dependency-based linguistic theories do not. More precisely, most theoretical formulations of dependency grammar regard projectivity as the norm but also recognize the need for non-projective representations of certain linguistic constructions, e.g., long-distance dependencies (Mel'čuk, 1988; Hudson, 1990). It is also often assumed that the constraint of projectivity is too rigid for the description of languages with free or flexible word order.

Some multi-stratal theories allow non-projective representations in some layers but not in others. For example, FGD assumes that tectogrammatical representations are projective while analytical representations are not (Sgall et al., 1986). Similarly, TDG (Duchier and Debusmann, 2001) assume projectivity for LP trees but not for ID trees. Sometimes a weaker condition called *planarity* is assumed, which allows a node w to occur between a head h and a dependent d without being dominated by h only if w is a root (Sleator and Temperley, 1993).[5] Further relaxations of these constraints are discussed in Kahane et al. (1998) and Yli-Jyrä (2003).

The points of divergence considered up till now have all been concerned with aspects of representation. However, as mentioned at the end of the previous section, there are also a number of points concerning the substantive linguistic analysis where different frameworks of dependency grammar make different assumptions, in the same way that theories differ also within other traditions. We will limit ourselves to a brief discussion of two such points.

[5] This constraint is related to but not equivalent to the standard notion of planarity in graph theory (see, e.g., Grimaldi, 2004).

The first point concerns the issue of *syntactic* versus *semantic* heads. As noted in section 3.1.1, the criteria for identifying heads and dependents invoke both syntactic and semantic properties. In many cases, these criteria give the same result, but in others they diverge. A typical example is found in so-called case marking prepositions, exemplified in the following sentence:

I believe in the system. (3.3)

According to syntactic criteria, it is natural to treat the preposition *in* as a dependent of the verb *believe* and the noun *system* as a dependent of the preposition. According to semantic criteria, it is more natural to regard *system* as a direct dependent of *believe* and to treat *in* as a dependent of *system* (corresponding to a case marking affix in some other languages).[6] Most versions of dependency grammar treat the preposition as the head of the noun, but there are also theories that make the opposite assumption. Similar considerations apply to many constructions involving one function word and one content word, such as determiner-noun and complementizer-verb constructions. An elegant solution to this problem is provided by the theory of Tesnière (1959), according to which the function word and the content word form a *dissociate nucleus* (nucléus dissocié), united by a relation of *transfer* (translation). In multi-stratal theories, it is possible to treat the function word as the head only in more surface-oriented layers. For example, to return to example (3.3), FGD would assume that the preposition takes the noun as a dependent in the analytical layer, but in the tectogrammatical layer the preposition would be absent and the noun would instead depend directly on the verb.

The second point concerns the analysis of coordination, which presents problems for any syntactic theory but which seems to be especially hard to reconcile with the idea that syntactic constructions should be analyzed in terms of binary head-dependent relations. Consider the following example:

They operate ships and banks. (3.4)

It seems clear that the phrase *ships and banks* functions as a direct object of the verb *operate*, but it is not immediately clear how this phrase can be given an internal analysis that is compatible with the basic assumptions of dependency analysis, since the two nouns *ships* and *banks* seem to be equally good candidates for being heads. One alternative is to treat the conjunction as the head, as shown in figure 3.2 (*top*), an analysis that may be motivated on semantic grounds and is adopted in FGD. Another alternative, advocated by Mel'čuk (1988), is to treat the conjunction as the head only of the second conjunct and analyze the conjunction as a dependent of the first conjunct, as shown in figure 3.2 (*bottom*). The arguments for this analysis are essentially the same as the arguments for an asymmetric right-branching analysis in constituency-based frameworks. A third option is to give up a pure

[6] A third alternative is to treat both *in* and *system* as dependents of *believe*, since it is the verb that selects the preposition and takes the noun as an argument.

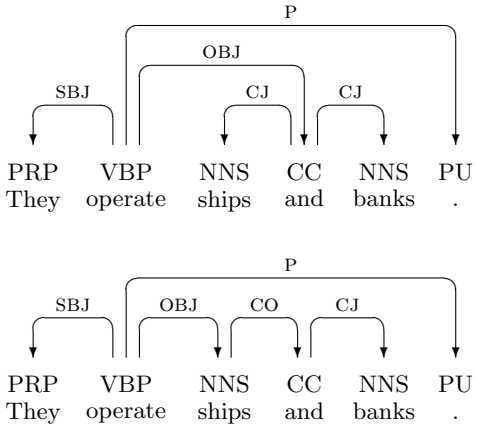


Fig. 3.2. Two analyses of coordination

dependency analysis and allow a limited form of phrase structure, as in WG (Hudson, 1990). A fourth and final variant is the analysis of Tesnière (1959), according to which both *ships* and *banks* are dependents of the verb, while the conjunction marks a relation of *junction* (jonction) between the two nouns.

3.2 Parsing with Dependency Representations

So far in this chapter, we have reviewed the theoretical tradition of dependency grammar, focusing on the common core of assumptions as well as major points of divergence, rather than on individual instantiations of this tradition. We will now turn to what is the main topic of this chapter, namely the computational implementation of syntactic analysis based on dependency representations, i.e., representations involving lexical nodes, connected by dependency arcs, possibly labeled with dependency types.

Such implementations may be intimately tied to a particular theory, such as the PLAIN system based on DUG (Hellwig, 1980, 2003), the implementation of Word Grammar by Fraser (1989, 1993), or the FDG parsing system (Tapanainen and Järvinen, 1997; Järvinen and Tapanainen, 1998).[7] On the whole, however, the connections between theoretical frameworks and computational systems are often rather indirect for dependency-based analysis, probably more so than for theories and parsers based on constituency analysis. This may be due to the relatively lower degree of formalization of dependency grammar theories in general, and this is also part of the reason why the topic

[7] Nowadays also known as Machinese Syntax (cf. Järvinen, 2003).

of this section is described as parsing with dependency *representations*, rather than parsing with dependency *grammar*.

In discussing dependency-based systems for syntactic parsing, we will adopt the division made in chapter 2 between *grammar-driven* and *data-driven* approaches, although these approaches are, as we have already had reason to observe, not mutually exclusive. We will conclude this section with a brief discussion of some of the potential advantages of using dependency representations in syntactic parsing.

3.2.1 Grammar-Driven Dependency Parsing

The earliest work on parsing with dependency representations was intimately tied to formalizations of dependency grammar that were very close to context-free grammar, such as the proposals of Hays (1964) and Gaifman (1965). In the formulation of Gaifman (1965) a *dependency system* contains three sets of rules:[8]

1. L_{I}: Rules of the form $X(Y_1 \cdots Y_i * Y_{i+1} \cdots Y_n)$, where i may equal 0 and/or n, which say that the category X may occur with categories $Y_1, \ldots, Y_n$ as dependents, in the order given (with X in position $*$).
2. L_{II}: Rules giving for every category X the list of words belonging to it (where each word may belong to more than one category).
3. L_{III}: A rule giving the list of all categories the occurrence of which may govern a sentence.

A sentence consisting of words $w_1, \ldots, w_n$ is analyzed by assigning to it a sequence of categories $X_1, \ldots, X_n$ and a relation of dependency d between occurrences of words such that the following conditions hold (where d^* is the transitive closure of d):

1. For no w_i, $d^*(w_i, w_i)$.
2. For every w_i, there is at most one w_j such that $d(w_i, w_j)$.
3. If $d^*(w_i, w_j)$ and w_k is between w_i and w_j, then $d^*(w_k, w_j)$.
4. The whole set of word occurrences is connected by d.
5. If $w_1, \ldots, w_i$ are left dependents and $w_{i+1}, \ldots, w_n$ right dependents of some word, then $X(X_1 \cdots X_i * X_{i+1} \cdots X_n)$ is a rule of L_{I}, where $X_1, \ldots, X_i, X_{i+1}, \ldots, X_n$ are the categories of $w_1, \ldots, w_i, w_{i+1}, \ldots, w_n$.
6. The word occurrence w_i that governs the sentence belongs to a category listed in L_{III}.

Gaifman remarks that 1–4 are general structure requirements that can be made on any relation defined on a finite linearly ordered set whether it is a set of categories or not, while 5–6 are requirements which relate the relation to the specific grammar given by the three sets of rules L_{I}–L_{III}. Referring back to the discussion of graph conditions in section 3.1.2, we may first of all

[8] The formulation of Hays (1964) is slightly different but equivalent.

note that Gaifman defines dependency relations to hold from dependent to head, rather than the other way round which is more common in the recent literature. Secondly, we see that condition 2 corresponds to the *single-head* constraint and condition 3 to the *projectivity* constraint. Conditions 1, 2 and 4 jointly entail that the graph is a rooted tree, which is presupposed in condition 6. Finally, it should be pointed out that this kind of dependency system only gives an unlabeled dependency analysis, since there are no dependency types used to label dependency relations.

Gaifman (1965) proves several equivalence results relating his dependency systems to context-free grammars. In particular, he demonstrates that the two systems are weakly equivalent, i.e., they both characterize the class of context-free languages. However, he also shows that whereas any dependency system can be converted to a strongly equivalent context-free grammar (modulo a specific mapping between dependency trees and context-free parse trees), the inverse construction is only possible for a restricted subset of context-free grammar (roughly grammars where all productions are lexicalized).

These results have been invoked to explain the relative lack of interest in dependency grammar within natural language processing for the subsequent twenty-five years or so, based on the erroneous conclusion that dependency grammar is only a restricted variant of context-free grammar (Järvinen and Tapanainen, 1998).[9] This conclusion is erroneous simply because the results only concern the specific version of dependency grammar formalized by Hays and Gaifman, which for one thing is restricted to projective dependency structures. However, it also worth emphasizing that with the increasing importance of problems like robustness and disambiguation, issues of (limited) generative capacity have lost some of their significance in the context of text parsing. Nevertheless, it seems largely true to say that, except for isolated studies of dependency grammar as an alternative to context-free grammar as the basis for transformational grammar (Robinson, 1970), dependency grammar has played a marginal role both in syntactic theory and in natural language parsing until fairly recently.

The close relation to context-free grammar in the formalization of dependency grammar by Hays and Gaifman means that essentially the same parsing methods can be used for both types of system. Hence, the parsing algorithm outlined in Hays (1964) is a bottom-up dynamic programming algorithm very similar to the CKY algorithm proposed for context-free parsing at about the same time (Kasami, 1965; Younger, 1967). The use of dynamic programming algorithms that are closely connected to context-free parsing algorithms such as CKY and Earley's algorithm (Earley, 1970) is a prominent trend also in more recent grammar-driven approaches to dependency parsing. One well-known example is the link grammar parser of Sleator and Temperley (1991,

[9] A similar lack of interest seems to have affected categorial grammar after the weak equivalence of a restricted type of categorial grammar with context-free grammar was proven by Bar-Hillel et al. (1960).

1993), which uses a dynamic programming algorithm implemented as a top-down recursive algorithm with memoization to achieve parsing in $O(n^3)$ time. Link grammar is not considered an instance of dependency grammar by its creators, and it departs from the traditional view of dependency by using undirected links, but the representations used in link grammar parsing are similar to dependency representations in that they consist of words linked by binary relations. Other examples include a modification of the CKY algorithm (Holan et al., 1997) and an Earley-type algorithm with left-corner filtering in the prediction step (Lombardo and Lesmo, 1996; Barbero et al., 1998).

A common property of all frameworks that implement dependency parsing as a form of lexicalized context-free parsing is that they are restricted to the derivation of projective dependency structures, although some of the frameworks allow post-processing that may introduce non-projective structures (Sleator and Temperley, 1991, 1993). Many of these frameworks can be subsumed under the notion of *bilexical grammar* introduced by Eisner (2000). In Eisner's formulation, a bilexical grammar consists of two elements:

1. A vocabulary Σ of terminal symbols (words), containing a distinguished symbol ROOT.
2. For each word $w \in \Sigma$, a pair of deterministic finite-state automata l_w and r_w. Each automaton accepts some regular subset of Σ^*.

The language $L(G)$ defined by a bilexical dependency grammar G is defined as follows:

1. A *dependency tree* is a rooted tree whose nodes are labeled with words from Σ, and where the root node is labeled with the special symbol ROOT. The children of a node are ordered with respect to each other and the node itself, so that the node has both *left children* that precede it and *right children* that follow it.
2. A dependency tree is *grammatical* according to G if and only if for every word token w that appears in the tree, l_w accepts the (possibly empty) sequence of w's left children (from right to left), and r_w accepts the sequence of w's right children (from left to right).
3. A string $x \in \Sigma^*$ is generated by G with analysis y if y is a grammatical dependency tree according to G and listing the node labels of y in infix order yields the string x followed by ROOT; x is called the *yield* of y.
4. The language $L(G)$ is the set of all strings generated by G.

The general parsing algorithm proposed by Eisner for bilexical grammar is again a dynamic programming algorithm, which proceeds by linking *spans* (substrings where roots occur either leftmost or rightmost or both) instead of *constituents*, thereby reducing the time complexity from $O(n^5)$ to $O(n^3)$. More precisely, the running time is $O(n^3g^3t)$, where g is an upper bound on the number of possible senses (lexical entries) of a single word, and t is an upper bound on the number of states of a single automaton.

Eisner shows how the framework of bilexical grammar, and the cubic-time parsing algorithm, can be modified to capture a number of different frameworks and approaches such as Milward's (mono)lexical dependency grammar (Milward, 1994), Alshawi's head automata (Alshawi, 1996), Sleator and Temperley's link grammar (Sleator and Temperley, 1991, 1993), and Eisner's own probabilistic dependency models that will be discussed below in section 3.2.2 (Eisner, 1996b,a).

Eisner's algorithm can be viewed as a method for finding a *projective spanning tree* in a directed graph. Generalizing this idea, McDonald, Pereira, Ribarov and Hajič (2005) show how non-projective dependency parsing can be performed by finding a spanning tree that is not required to be projective, using the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). This algorithm runs in $O(n^2)$ time, which means that non-projective parsing can in fact be performed more efficiently than projective parsing using this approach.

The second main tradition in grammar-driven dependency parsing is based on the notion of *eliminative* parsing, where sentences are analyzed by successively eliminating representations that violate constraints until only valid representations remain (cf. section 2.2.1). One of the first parsing systems based on this idea is the CG framework (Karlsson, 1990; Karlsson et al., 1995), which uses underspecified dependency structures represented as syntactic tags and disambiguated by a set of constraints intended to exclude ill-formed analyses. In CDG (Maruyama, 1990), this technique is extended to complete dependency structures by generalizing the notion of tag to pairs consisting of a syntactic label and an identifier of the head node. This kind of representation is fundamental for many different approaches to dependency parsing, since it provides a way to reduce the parsing problem to a tagging or classification problem. Other typical representatives of this tradition are the extended CDG framework of Harper and Helzerman (1995) and FDG (Tapanainen and Järvinen, 1997; Järvinen and Tapanainen, 1998), where the latter is a development of CG that combines eliminative parsing with a non-projective dependency grammar inspired by Tesnière (1959).

In the eliminative approach, parsing is viewed as a constraint satisfaction problem, where any analysis satisfying all the constraints of the grammar is a valid analysis. Constraint satisfaction in general is NP complete, which means that special care must be taken to ensure reasonable efficiency in practice. Early versions of this approach used procedures based on local consistency (Maruyama, 1990; Harper et al., 1995), which attain polynomial worst case complexity by only considering local information in the application of constraints. In the more recently developed TDG framework (Duchier, 1999, 2003), the problem is confronted head-on by using constraint programming to solve the satisfaction problem defined by the grammar for a given input string. The TDG framework also introduces several levels of representation (cf. section 3.1.2), arguing that constraints can be simplified by isolating different aspects of the grammar such as Immediate Dominance (ID) and Linear

Precedence (LP) and using constraints that relate different levels to each other (Duchier and Debusmann, 2001; Debusmann, 2001). This view is taken to its logical extension in the most recent version of the framework called Extensible Dependency Grammar (XDG), where any number of levels, or dimensions, can be defined in the grammatical framework (Debusmann et al., 2004)

From the point of view of text parsing, parsing as constraint satisfaction can be problematic in two ways. First, for a given input string, there may be no analysis satisfying all constraints, which leads to the robustness problem. Secondly, there may be more than one analysis, which leads to the problem of disambiguation. Menzel and Schröder (1998) extends the CDG framework of Maruyama (1990) with *graded*, or *weighted*, constraints, by assigning a weight w $(0.0 \leq w \leq 1.0)$ to each constraint indicating how serious the violation of this constraint is (where 0.0 is the most serious). In this extended framework, later developed into WCDG (Schröder, 2002), the best analysis for a given input string is the analysis that minimizes the total weight of violated constraints. While early implementations of this system used an eliminative approach to parsing (Menzel and Schröder, 1998), the more recent versions instead use a transformation-based approach, which successively tries to improve the analysis by transforming one solution into another guided by the observed constraint violations in the current solution. One advantage of this heuristic approximation strategy is that it can be combined with arbitrarily complex constraints, whereas standard eliminative procedures usually require constraints to be binary for efficiency reasons (Foth et al., 2004).

So far, we have distinguished two main trends in grammar-driven dependency parsing. The first is based on a formalization of dependency grammar that is closely related to context-free grammar, and therefore usually restricted to projective dependency structures, using standard techniques from context-free parsing to obtain good efficiency in the presence of massive ambiguity, in particular dynamic programming or memoization. The second is based on a formalization of dependency grammar in terms of constraints, not necessarily limited to projective structures, where parsing is naturally viewed as a constraint satisfaction problem which can be addressed using eliminative parsing methods, although the exact parsing problem is often intractable.

In addition to these two traditions, which both involve fairly complex grammars and parsing algorithms, there is a third tradition which is based on a simpler notion of dependency grammar together with a deterministic parsing strategy (possibly with limited backtracking). As in other parsing paradigms, the study of deterministic parsing can be motivated either by a wish to model human sentence processing or by a desire to make syntactic parsing more efficient (or possibly both). According to Covington (2001), these methods have been known since the 1960s without being presented systematically in the literature. The fundamental parsing strategy comes in different versions but we will concentrate here on the left-to-right (or incremental) version, which is formulated in the following way by Covington (2001):

> Accept words one by one starting at the beginning of the sentence, and try linking each word as head or dependent of every previous word.

This parsing strategy is compatible with many different grammar formulations. All that is required is that a grammar G defines a boolean function f_G that, for any two words w_1 and w_2, returns **true** if w_1 can be the head of w_2 according to G (and **false** otherwise).[10] Covington (2001) demonstrates how this parsing strategy can be used to produce dependency structures satisfying different conditions such as *uniqueness* (single head) and *projectivity* simply by imposing different constraints on the linking operation. Covington has also shown in previous work how this parsing strategy can be adapted to languages with free, flexible or rigid word order (Covington, 1990a,b, 1994). The time complexity of Covington's algorithm is $O(n^2)$ in the deterministic version.

The parsing algorithm investigated in this book can actually be derived as a special case of Covington's algorithm, although we will not give this formulation here, and our very first experiments used a simple grammar of the kind presupposed by Covington to perform unlabeled dependency parsing (Nivre, 2003). A similar approach can be found in Obrębski (2003), although this system is nondeterministic and derives a compact representation of all permissible dependency trees in the form of a directed acyclic graph. Yet another framework that shows affinities with the deterministic grammar-driven approach is that of Kromann (2004), although it is based on a more sophisticated notion of grammar called Discontinuous Grammar. Parsing in this framework is essentially deterministic but subject to repair mechanisms that are associated with local cost functions derived from the grammar.

Before we close the discussion of grammar-driven dependency parsing, we should also mention the work of Oflazer (2003), which is an extended finite-state approach to dependency parsing similar to the cascaded partial parsers used for constituency-based parsing by Abney (1996) and Roche (1997). Oflazer's system allows violable constraints for robust parsing and uses total link length to rank alternative analyses, as proposed by Lin (1996).

3.2.2 Data-Driven Dependency Parsing

As for natural language parsing in general, the first attempts at data-driven dependency parsing were also grammar-driven in that they relied on a formal dependency grammar and used corpus data to induce a probabilistic model for disambiguation (cf. section 2.3.1). Thus, Carroll and Charniak (1992) essentially use a PCFG model, where the context-free grammar is restricted to be equivalent to a Hays/Gaifman type dependency grammar. They report experiments trying to induce such a probabilistic grammar using unsupervised

[10] In this formulation, the parsing strategy is limited to unlabeled dependency graphs. In principle, it is possible to perform labeled dependency parsing by returning a set of permissible dependency types instead of **true**, but this makes the process nondeterministic in general.

learning on an artificially created corpus but with relatively poor results. On the whole, unsupervised learning for dependency parsing is a relatively unexplored area of research, with a few notable exceptions such as Yuret (1998) and Klein and Manning (2004).

A more successful and more influential approach was developed by Eisner (1996a,b), who defined several probabilistic models for dependency parsing and evaluated them using supervised learning with data from the Wall Street Journal section of the Penn Treebank. In later work, Eisner (2000) has shown how these models can be subsumed under the general notion of a *bilexical grammar* (BG), which means that parsing can be performed efficiently as discussed briefly in section 3.2.1. Eisner (2000) defines the notion of a *weighted bilexical grammar* (WBG) in terms of BG as follows (cf. section 3.2.1):

1. A *weighted DFA* A is a deterministic finite automaton that associates a real-valued *weight* with each arc and each final state. Each accepting path through A is assigned a weight, namely the sum of all arc weights on the path and the weight of the final state. Each string x accepted by A is assigned the weight of its accepting path.
2. A WBG G is a BG in which all the automata l_w and r_w are weighted DFAs. The weight of a dependency tree y under G is defined as the sum, over all word tokens w in y, of the weight with which l_w accepts w's sequence of left children plus the weight with which r_w accepts w's sequence of right children.

Eisner (1996b) presents three different probabilistic models for dependency parsing, which can be reconstructed as different weighting schemes within the framework of WBG. However, the first two models (models A and B) require that we distinguish between an underlying string $x \in \Sigma^*$, described by the WBG, and a surface string x', which results from a possibly nondeterministic, possibly weighted finite-state transduction R on x. The surface string x' is then grammatical with analysis (y, p) if y is a grammatical dependency tree whose yield x is transduced to x' along an accepting path p in R. To avoid the distinction between underlying strings and surface strings, we will restrict our attention to model C, which was found to perform significantly better than the other two models in the experiments reported in Eisner (1996a).

First of all, it should be pointed out that all the models in Eisner (1996b) involve part-of-speech tags, in addition to word tokens and (unlabeled) dependency relations, and define the joint probability of words, tags and dependency relations (called *links* in this context). Model C is defined as follows:

$$P(tw(1), \ldots, tw(n), links) = \prod_{i=1}^{n} P(lc(i) \mid tw(i)) P(rc(i) \mid tw(i)) \tag{3.5}$$

where n is the number of words, $tw(i)$ is the ith tagged word, and $lc(i)$ and $rc(i)$ are the left and right children of the ith word, respectively. The probability of generating each child is conditioned on the tagged head word and the

tag of the preceding child (left children being generated from right to left):

$$P(lc(i) \mid tw(i)) = \prod_{j=1}^{m} P(tw(lc_j(i)) \mid t(lc_{j-1}(i)), tw(i)) \tag{3.6}$$

$$P(rc(i) \mid tw(i)) = \prod_{j=1}^{m} P(tw(rc_j(i)) \mid t(rc_{j-1}(i)), tw(i)) \tag{3.7}$$

where m is the number of children on one side, $lc_j(i)$ is the jth left child of the ith word, and $t(lc_{j-1}(i))$ is the tag of the preceding left child (and analogously $rc_j(i)$ and $t(rc_{j-1}(i))$ for right children). This model can be implemented in the WBG framework by letting the automata l_w and r_w have weights on their arcs corresponding to the log of the probability of generating a particular left or right child given the tag of the preceding child. In this way, the weight assigned to a dependency tree y will be the log of $P(tw(1), \ldots, tw(n), \mathit{links})$ as defined above (Eisner, 2000).

Eisner's work on data-driven dependency parsing has been influential in two ways. First, it showed that generative probabilistic modeling and supervised learning could be applied to dependency representations to achieve a parsing accuracy comparable to the best results reported for constituency-based parsing at the time, although the evaluation metrics used in the two cases are not strictly comparable. Secondly, it showed how these models could be coupled with efficient parsing techniques that exploit the special properties of dependency structures. The importance of the second aspect can be seen in recent work by McDonald, Crammer and Pereira (2005), applying discriminative estimation methods to probabilistic dependency parsing. Thanks to the more efficient parsing methods offered by Eisner's methods for bilexical parsing, training can be performed without pruning the search space, which is impossible for efficiency reasons when using lexicalized constituency representations with comparable lexical dependencies. This approach can be extended to non-projective dependency parsing by combining the discriminative estimation methods with the Chu-Liu-Edmonds algorithm for finding a *maximum spanning tree* in a directed graph, without requiring the tree to be projective, as shown in McDonald, Pereira, Ribarov and Hajič (2005).

Collins et al. (1999) apply the generative probabilistic parsing models of Collins (1997, 1999) to dependency parsing, using data from the Prague Dependency Treebank. This requires preprocessing to transform dependency structures into flat phrase structures for the training phase and postprocessing to extract dependency structures from the phrase structures produced by the parser. The parser of Charniak (2000) has been adapted and applied to the Prague Dependency Treebank in a similar fashion, although this work remains unpublished.

Samuelsson (2000) proposes a probabilistic model for dependency grammar that goes beyond the models considered so far by incorporating labeled dependencies and allowing non-projective dependency structures. In this model,

dependency representations are generated by two stochastic processes: one top-down process generating the tree structure y and one bottom-up process generating the surface string x given the tree structure, defining the joint probability as $P(x, y) = P(y)P(x \mid y)$. Samuelsson suggests that the model can be implemented using a bottom-up dynamic programming approach, but the model has unfortunately never been implemented and evaluated.

Another probabilistic approach to dependency parsing that incorporates labeled dependencies is the stochastic CDG parser of Wang and Harper (2004), which extends the CDG model with a generative probabilistic model. Parsing is performed in two steps, where the first step assigns to each word a set of so-called SuperARVs, representing constraints on possible heads and dependents, and where the second step determines actual dependency links given the SuperARV assignment. Although the basic model and parsing algorithm is limited to projective structures, the system allows non-projective structures for certain *wh*-constructions. The system has been evaluated on data from the Wall Street Journal section of the Penn Treebank and achieves state-of-the-art performance using a dependency-based evaluation metric (Wang and Harper, 2004).

The first step in the parsing model of Wang and Harper (2004) can be seen as a kind of supertagging, which has turned out to be a crucial element in many recent approaches to statistical parsing, e.g., in LTAG (Joshi and Sarkar, 2003; Bangalore, 2003) and CCG (Clark and Curran, 2004; Curran and Clark, 2004). A similar two-step process is used in the statistical dependency parser of Bangalore (2003), which uses elementary LTAG trees as supertags in order to derive a dependency structure in the second step. Supertagging is performed using a standard HMM trigram tagger, while dependency structures are derived using a heuristic deterministic algorithm that runs in linear time. Another data-driven dependency parser based on supertagging is Nasr and Rambow (2004), where supertags are derived from a lexicalized extended context-free grammar and the most probable analysis is derived using a modified version of the CKY algorithm.

Most of the systems described in this section are based on a formal dependency grammar in combination with a generative probabilistic model, which means that parsing conceptually consists in the derivation of all analyses that are permissible according to the grammar and the selection of the most probable analysis according to the generative model. This is in contrast to the framework developed in this book, which does not involve a formal grammar, and where we will focus on purely discriminative models of inductive learning in combination with a deterministic parsing strategy.

The deterministic discriminative approach was first proposed by Kudo and Matsumoto (2002) and Yamada and Matsumoto (2003), using support vector machines (Vapnik, 1995) to train classifiers that predict the next action of a deterministic parser constructing unlabeled dependency structures. The parsing algorithm used in these systems implements a form of shift-reduce

parsing with three possible parse actions that apply to two neighboring words referred to as *target words*:[11]

1. A *Shift* action adds no dependency construction between the target words w_i and w_{i+1} but simply moves the point of focus to the right, making w_{i+1} and w_{i+2} the new target words.
2. A *Right* action constructs a dependency relation between the target words, adding the left node w_i as a child of the right node w_{i+1} and reducing the target words to w_{i+1}, making w_{i-1} and w_{i+1} the new target words.
3. A *Left* action constructs a dependency relation between the target words, adding the right node w_{i+1} as a child of the left node w_i and reducing the target words to w_i, making w_{i-1} and w_i the new target words.

The parser processes the input from left to right repeatedly as long as new dependencies are added, which means that up to $n - 1$ passes over the input may be required to construct a complete dependency tree, giving a worst case time complexity of $O(n^2)$, although the worst case seldom occurs in practice. The features used to predict the next parse action are the word forms and part-of-speech tags of the target words, of their left and right children, and of their left and right string context (in the reduced string). Yamada and Matsumoto (2003) evaluate the system using the standard data set from the Wall Street Journal section of the Penn Treebank and show that deterministic discriminative dependency parsing can achieve an accuracy that is close to the state-of-the-art with respect to dependency accuracy. Further improvements with this model are reported in Isozaki et al. (2004).

The framework of inductive dependency parsing investigated in this book has many properties in common with the system of Yamada and Matsumoto (2003), but there are three main differences. The first and most important difference is that our system constructs labeled dependency representations, i.e., representations where dependency arcs are labeled with dependency types. This also means that dependency type information can be exploited in the feature model used to predict the next parse action. The second difference is that the algorithm described below in section 3.4 is a head-driven arc-eager algorithm that constructs a complete dependency tree in a single pass over the data. The third and final difference is that our system, in its current incarnation, uses memory-based learning to induce classifiers for predicting the next parsing action based on conditional features, whereas Yamada and Matsumoto (2003) use support vector machines. However, as pointed out by Kudo and Matsumoto (2002), in a deterministic discriminative parser the learning method is largely independent of the rest of the system.

[11] The parser described in Kudo and Matsumoto (2002) is applied to Japanese, where dependencies are assumed to be strictly head-final, which means that only the actions *Shift* and *Right* are required.

3.2.3 The Case for Dependency Parsing

As noted several times already, dependency-based syntactic representations have played a fairly marginal role in the history of linguistic theory as well as that of natural language processing. Saying that there is increasing interest in dependency-based approaches to syntactic parsing may therefore not be saying very much, but it is hard to deny that the notion of dependency has become more prominent in the literature on syntactic parsing during the last decade or so.

At this point, it seems appropriate to ask what are the potential benefits of using dependency-based representations in syntactic parsing, as opposed to the more traditional representations based on constituency. According to Covington (2001), dependency parsing offers three *prima facie* advantages:

- Dependency links are close to the semantic relationships needed for the next stage of interpretation; it is not necessary to 'read off' head-modifier or head-complement relations from a tree that does not show them directly.
- The dependency tree contains one node per word. Because the parser's job is only to connect existing nodes, not to postulate new ones, the task of parsing is in some sense more straightforward. [...]
- Dependency parsing lends itself to word-at-a-time operation, i.e., parsing by accepting and attaching words one at a time rather than by waiting for complete phrases. [...]

To this it is sometimes added that dependency-based parsing allows a more adequate treatment of languages with variable word order, where discontinuous syntactic constructions are more common than in languages like English (Mel'čuk, 1988; Covington, 1990b). However, this argument is only plausible if the formal framework allows non-projective dependency structures, which is not the case for most dependency parsers that exist today.

For us, the first two advantages identified by Covington seem to be the most important. Having a more constrained representation, where the number of nodes is fixed by the input string itself, should enable conceptually simpler and computationally more efficient methods for parsing. At the same time, it is clear that a more constrained representation is a less expressive representation and that dependency representations are necessarily underspecified with respect to certain aspects of syntactic structure. For example, as pointed out by Mel'čuk (1988), it is impossible to distinguish in a pure dependency representation between an element modifying the head of a phrase and the same element modifying the entire phrase. However, this is precisely the kind of ambiguity that is notoriously hard to disambiguate correctly in syntactic parsing anyway, so it might be argued that this kind of underspecification is actually beneficial. And as long as the syntactic representation encodes enough of the structural relations that are relevant for semantic interpretation, then we are only happy if we can constrain the problem of deriving these representations.

In general, there is a tradeoff between the expressivity of syntactic representations and the complexity of syntactic parsing, and we believe that dependency representations are a good compromise in this respect. They are less expressive than most constituency-based representations but compensate for this by providing a relatively direct encoding of predicate-argument structure, which is relevant for semantic interpretation, and by being composed of bilexical relations, which are potentially beneficial for disambiguation. In this way, dependency structures are sufficiently expressive to be useful in natural language processing systems and at the same time sufficiently restricted to allow full parsing with high accuracy and efficiency under constraints of robustness and disambiguation. At least, this is our working hypothesis.

The remainder of this book will be devoted to the second half of this hypothesis, exploring the tradeoff between accuracy and efficiency under the constraints of robustness and disambiguation. By contrast, the first half of the hypothesis, the potential usefulness of dependency representations for different natural language applications, will have to remain an untested assumption. Although we can hope that the increasing interest in dependency parsing is indicative of their usefulness, any serious discussion of these issues would require an investigation taking us far beyond the scope of this book.

3.3 A Framework for Dependency Parsing

Having reviewed both grammar-driven and data-driven approaches to dependency parsing and motivated our choice of syntactic representation, it is now time to define the formal framework that will serve us in the investigation of inductive dependency parsing. Before we come to the formal definitions, it may be a good idea to position our approach with respect to the open issues in dependency grammar discussed in section 3.1.2. The following properties are characteristic:

- The syntactic analysis is limited to dependency structure only, although we allow word tokens to be pre-tagged with, e.g., parts-of-speech.
- The syntactic analysis is mono-stratal, which means that we assign a single dependency representation to each input string.
- The nodes of the dependency structure are the tokens of the input string.
- The labels of the dependency structure are not prescribed by the framework but are normally taken to be surface-oriented grammatical functions.
- The dependency graphs are rooted and connected; normally they are also taken to satisfy the single-head, acyclicity and projectivity constraints.
- The framework does not prescribe a specific analysis of problematic constructions such as coordination, disjoint syntactic and semantic heads, etc.

It is important to keep in mind that this is a framework for text parsing, where the input consists of tokenized text sentences and where each token, including punctuation, constitutes a node in the dependency representation.

It follows that the dependency analysis will normally be surface oriented. However, the constraints given by the framework are purely formal and do not concern the substantive side of the linguistic analysis. This is in harmony with the data-driven approach to text parsing, where the framework has to be compatible with whatever linguistic analysis is provided in the training data set, as long as it satisfies the formal conditions. In the empirical experiments reported in chapter 5 we will see that the same formal machinery can deal with fairly different assumptions about linguistic structure derived from the different data sets used.

3.3.1 Texts, Sentences and Tokens

Recapitulating parts of the characterization of text parsing from section 2.2.2, we make the following assumptions about texts, sentences and tokens:

Definition 3.1. A *text* is a sequence $T = (x_1, \ldots, x_n)$ of sentences.

Definition 3.2. A *sentence* is a sequence $x = (w_1, \ldots, w_n)$ of tokens.

Definition 3.3. A *token* is a sequence $w = (c_1, \ldots, c_n)$ of characters.

Tokens will normally be word forms, punctuation marks and other units that are recognized as the basic elements of text sentences. However, the exact nature of the tokenization process is not prescribed by the formal framework, which means that a tokenization that groups several word forms into multi-word units, or segments a single word form into several tokens (e.g., by compound splitting) is perfectly compatible with our definition of a sentence. As pointed out in section 2.2.2, this is a notion of *text sentence*, rather than *system sentence* (Lyons, 1977), since we make no assumptions about syntactic completeness or well-formedness. Readers who are unhappy with this use of the term *sentence* are welcome to use the more neutral term *string* instead, although it is important to remember that, since we accept as a token any string of characters that comes out of the tokenization process, our sentences are not strings over a well-defined alphabet.

Given a sentence $x = (w_1, \ldots, w_n)$, we define a function w_x that maps indices to tokens as follows:

$$w_x(i) = \begin{cases} w_i \text{ if } 1 \leq i \leq n \\ \text{undefined otherwise} \end{cases} \tag{3.8}$$

We will normally drop the subscript and simply write $w(i)$ when it is clear from the context which sentence is indicated. In order to represent annotation of tokens that result from preprocessing, we also allow additional functions that map each index to the appropriate annotation label. For example, we use a function p_x to represent a part-of-speech tagging $(p_1, \ldots, p_n)$ of x as follows:

$$p_x(i) = \begin{cases} p_i \text{ if } 1 \leq i \leq n \\ \text{undefined otherwise} \end{cases} \tag{3.9}$$

Economic_1 news_2 had_3 little_4 effect_5 on_6 financial_7 markets_8 $.9$

$w(1) = \text{Economic}$	$p(1) = \text{JJ}$
$w(2) = \text{news}$	$p(2) = \text{NN}$
$w(3) = \text{had}$	$p(3) = \text{VBD}$
$w(4) = \text{little}$	$p(4) = \text{JJ}$
$w(5) = \text{effect}$	$p(5) = \text{NN}$
$w(6) = \text{on}$	$p(6) = \text{IN}$
$w(7) = \text{financial}$	$p(7) = \text{JJ}$
$w(8) = \text{markets}$	$p(8) = \text{NNS}$
$w(9) = .$	$p(9) = \text{PU}$

Fig. 3.3. English sentence (cf. figure 3.1)

Although part-of-speech tagging is the only kind of annotation that will be considered in this book, one could also imagine annotation functions for lemmatization, morphological analysis or word sense disambiguation. We will use the term *annotation function* generally for any function over tokens that is available in a given setting, including the obligatory function w, and we will use the symbol $A_x = \{w_x, \ldots\}$ to denote the set of annotation functions for the string x. The functional notation is exemplified in figure 3.3, which represents the sentence from figure 3.1 using the w and p functions, i.e., $A_x = \{w_x, p_x\}$.

3.3.2 Dependency Graphs

A *dependency graph* is a labeled directed graph, the nodes of which are indices corresponding to the tokens of a sentence. Formally:

Definition 3.4. Given a set R of dependency types, a *dependency graph* for a sentence $x = (w_1, \ldots, w_n)$ is a labeled directed graph $G = (V, E, L)$, where:

1. $V = \mathbf{Z}_{n+1}$
2. $E \subseteq V \times V$
3. $L : E \to R$

Definition 3.5. A dependency graph G is *well-formed* if and only if:

1. The node 0 is a root (ROOT).
2. G is connected (CONNECTEDNESS).[12]

The set V of *nodes* (or *vertices*) is the set $\mathbf{Z}_{n+1} = \{0, 1, 2, \ldots, n\}$ ($n \in \mathbf{Z}^+$), i.e., the set of non-negative integers up to and including n (Grimaldi, 2004).

[12] Strictly speaking, we require the graph to be *weakly connected*, which entails that the corresponding undirected graph is connected, whereas a *strongly connected* graph has a *directed* path between any pair of nodes.

This means that every token index i of the sentence is a node ($1 \leq i \leq n$) and that there is a special node 0, which does not correspond to any token of the sentence and which will always be a root of the dependency graph (normally the only root).

Using position indices, rather than word forms, to represent tokens has certain practical advantages (Maruyama, 1990), one of which is that the relation of linear precedence is naturally represented by the arithmetic less-than relation, i.e., a node i precedes a node j if and only if $i < j$. In the following, we will reserve the term *token node* for a node that corresponds to a token of the sentence, and we will use the symbol V^+ to denote the set of token nodes of a sentence for which the set of nodes is V, i.e., $V^+ = V - \{0\}$. When necessary, we will write V_x and V_x^+ to indicate that V and V^+ are the nodes corresponding to a particular sentence $x = (w_1, \ldots, w_n)$. Note, however, that the only requirement imposed by x is that the number of nodes match the length of x, i.e., $|V^+| = n$ and $|V| = n + 1$.

The use of a special root node is common in the literature on dependency parsing and simplifies many definitions (Collins et al., 1999; Böhmová et al., 2003). For example, it is in many cases possible to assume that the dependency graph is a tree, even though the analysis scheme adopted does not require one of the tokens of the sentence to dominate every other token.

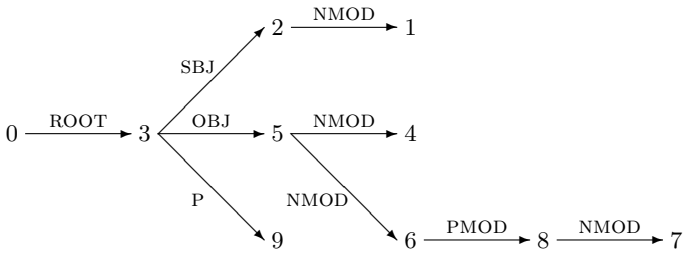
The set E of *arcs* (or *edges*) is a set of ordered pairs (i, j), where i and j are nodes. Since arcs are used to represent dependency relations, we will generally say that i is the *head* and j is the *dependent* of the arc (i, j).[13] As usual, we will use the notation $i \rightarrow j$ to mean that there is an arc connecting i and j (i.e., $(i, j) \in E$) and we will use the notation $i \rightarrow^* j$ for the reflexive and transitive closure of the arc relation E (i.e., $i \rightarrow^* j$ if and only if $i = j$ or there is a path of arcs connecting i to j).

The function L assigns a dependency type (arc label) $r \in R$ to every arc $e \in E$. We will use the notation $i \xrightarrow{r} j$ to mean that there is an arc labeled r connecting i to j (i.e., $(i, j) \in E$ and $L((i, j)) = r$).

Figure 3.4 shows how the dependency structure assigned to the example sentence from figure 3.1 can be represented as a dependency graph in the formal sense. Note that we take the set R of dependency types to include a special label ROOT, which is used to label arcs from the special root node 0. We will assume that every set R of dependency types includes such a label, which will be denoted r_0 when referring to arbitrary sets R.

The only conditions we have imposed on dependency graphs so far is that the special node 0 is a root (which is a pure technicality) and that the graph should be connected, which is a reasonable assumption given that the graph

[13] Unfortunately, this terminology is partially inconsistent with the terminology found in some textbooks, according to which j is the *head* and i is the *tail* of the arc (i, j) (Aho and Ullman, 1995). Other terms found in the literature are *predecessor*, *origin* and *source* for i, and *successor* and *terminus* for j (Aho and Ullman, 1995; Grimaldi, 2004).



$R = \{\text{ROOT}, \text{SBJ}, \text{OBJ}, \text{NMOD}, \text{PMOD}, \text{P}, \ldots\}$

$V = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$E = \{(0, 3), (3, 2), (3, 5), (3, 9), (2, 1), (5, 4), (5, 6), (6, 8), (8, 7)\}$

$L((0, 3)) = \text{ROOT}$
$L((3, 2)) = \text{SBJ}$
$L((3, 5)) = \text{OBJ}$
$L((3, 9)) = \text{P}$
$L((2, 1)) = \text{NMOD}$
$L((5, 4)) = \text{NMOD}$
$L((5, 6)) = \text{NMOD}$
$L((6, 8)) = \text{PMOD}$
$L((8, 7)) = \text{NMOD}$

Fig. 3.4. Dependency graph for English sentence (cf. figures 3.1 and 3.3)

should represent a coherent analysis of a sentence. In most cases, we want to impose further restrictions on well-formed dependency graphs, although we want the formal framework to be compatible with different views of dependency structures (cf. section 3.1.2). Thus, for the remainder of this book we will restrict our attention to the class of *projective dependency graphs*:

Definition 3.6. A *projective dependency graph* is a well-formed dependency graph satisfying the following additional conditions:

3. Every node has at most one head, i.e., if $i \rightarrow j$ then there is no node k such that $k \neq i$ and $k \rightarrow j$ (Single-Head).
4. The graph G is acyclic, i.e., if $i \rightarrow j$ then not $j \rightarrow^* i$ (Acyclicity).
5. The graph G is projective, i.e., if $i \rightarrow j$ then $i \rightarrow^* k$, for every node k such that $i < k < j$ or $j < k < i$ (Projectivity).

If we compare the graph conditions discussed in this section to those of Gaifman (1965) (cf. section 3.2.1, page 56), we see that our conditions 2, 3, 4 and 5 correspond directly to Gaifman's conditions 4, 2, 1 and 3, respectively. Thus,

the only difference is that we have no conditions corresponding to Gaifman's conditions 5 and 6, which relate the analysis to a particular grammar.[14] This is natural given that we are defining a framework for text parsing, where we do not require (nor preclude) the use of a formal grammar. Although we will assume all of conditions 1–5 for the investigations in this book, we will leave open the possibility of relaxing these constraints in other versions of inductive dependency parsing. We will return to this issue in chapter 6.

3.3.3 Dependency Parsing

Having defined text sentences and dependency graphs, we can now define the problem of *dependency parsing* as the mapping of text sentences to well-formed dependency graphs (relative to some set of graph conditions). Given our previous characterization of text parsing, we require this mapping to be a function and we assume that the correct mapping for each text sentence is defined by an independent criterion of accuracy (e.g., an empirical gold standard). In other words:

> Given a text $T = (x_1, \ldots, x_n)$ in language L, derive the single correct dependency graph G_i for every sentence $x_i \in T$.

In the last section of this chapter we will describe an algorithm that solves this problem with optimal efficiency, given an oracle that always chooses the correct action. The rest of the book will then be devoted to the approximation of this oracle using inductive inference, in particular memory-based learning.

3.4 Parsing Algorithm

The parsing algorithm described in this section was first presented in Nivre (2003), with a partial analysis of time complexity and robustness properties, although the algorithm was at that time restricted to unlabeled dependency graphs. It was extended to labeled dependency graphs in Nivre (2004b) and further analyzed with respect to incrementality in Nivre (2004a). We specify the algorithm by means of a transition system, which is nondeterministic in itself but which can be made deterministic by introducing an oracle that resolves nondeterministic choice points. The formulation in this book differs in some details from that in previous publications but it is equivalent in all important respects.

3.4.1 Configurations

We begin by defining a parser *configuration* for a sentence $x = (w_1, \ldots, w_n)$, relative to a set R of dependency types (including a special symbol r_0 for dependents of the root):

[14] One might see our condition 1 as a degenerate version of Gaifman's condition 5, since the special root node 0 will in most cases be the only permissible root node.

Definition 3.7. Given a set $R = \{r_0, r_1, \ldots r_m\}$ of dependency types and a sentence $x = (w_1, \ldots, w_n)$, a *parser configuration* for x is a quadruple $c = (\sigma, \tau, h, d)$, where:

1. σ is a stack of token nodes i ($1 \leq i \leq j$ for some $j \leq n$).
2. τ is a sorted sequence of token nodes i ($j < i \leq n$).
3. $h : V_x^+ \rightarrow V_x$ is a function from token nodes to nodes.
4. $d : V_x^+ \rightarrow R$ is a function from token nodes to dependency types.
5. For every token node $i \in V_x^+$, $h(i) = 0$ if and only if $d(i) = r_0$.

The idea is that the sequence τ represents the remaining input tokens in a left-to-right pass over the input sentence x; the stack σ contains partially processed nodes that are still candidates for dependency arcs, either as heads or dependents; and the functions h and d represent the (partially constructed) dependency graph for the input sentence x.

Representing the graph by means of two functions in this way is possible if we assume the Single-Head constraint. Since, for every token node j, there is at most one arc (i, j), we can represent this arc by letting $h(j) = i$. Strictly speaking, h should be a partial function, to allow the possibility that there is no arc (i, j) for a given node j, but we will avoid this complication by assuming that every node j for which there is no token node i such that $i \rightarrow j$ is headed by the special root node 0, i.e., $h(j) = 0$. In this way, it is easy to ensure that the graph is connected at all times. Given the Single-Head constraint, we can also let the labeling function assign dependency types to dependent nodes, rather than to arcs. Thus, if $i \xrightarrow{r} j$ then $h(j) = i$ and $d(j) = r$, with $h(j) = 0$ and $d(j) = r_0$ as a special case for a token that is not dependent on another token. Note that both h and d are defined only for token nodes, i.e., positive integers up to and including n ($V^+ = \mathbf{Z}_{n+1} - \{0\}$), whereas the range of h also includes the root node 0 ($V = \mathbf{Z}_{n+1}$). Formally, we establish the connection between configurations and dependency graphs as follows:

Definition 3.8. A configuration $c = (\sigma, \tau, h, d)$ for $x = (w_1, \ldots, w_n)$ defines the dependency graph $G_c = (V_x, E_c, L_c)$, where:

1. $E_c = \{(i, j) \mid h(j) = i\}$
2. $L_c = \{((i, j), r) \mid h(j) = i, d(j) = r\}$

In section 3.4.4 we will prove that, in the transition system described below, this always defines a well-formed projective dependency graph according to the definition in section 3.3.

We will use the following notational conventions for the components of a configuration:

1. Both the stack σ and the sequence of input tokens τ will be represented as lists, although the stack σ will have its head (or top) to the right for reasons of perspicuity. Thus, $\sigma|i$ represents a stack with top i and tail σ, while $j|\tau$ represents a list of input tokens with head j and tail τ, and the operator $|$ is taken to be left-associative for the stack and right-associative

for the input list. We use the symbols σ^- and τ^- to denote the tail of a non-empty σ and τ, i.e., $\sigma = \sigma^-|i$ and $\tau = j|\tau^-$ (for some i and j). Finally, we use ϵ to represent the empty list/stack.

2. For the functions h and d, we will use the notation $f[x \mapsto y]$, given a specific function f, to denote the function g such that $g(x) = y$ and $g(z) = f(z)$ for all $z \neq x$. Formally:

$$f[x \mapsto y](z) = \begin{cases} y & \text{if } z = x \\ f(z) & \text{otherwise} \end{cases}$$

Finally, we will refer to the token on top of the stack as *the top token*, the next input token as *the next token*, and the top token and the next token together as the *target tokens*.

Initial and terminal parser configurations are defined in the following way:

Definition 3.9. A configuration c for $x = (w_1, \ldots, w_n)$ is *initial* if and only if it has the form $c = (\epsilon, (1, \ldots, n), h_0, d_0)$, where:

1. $h_0(i) = 0$ for every $i \in V_x^+$.
2. $d_0(i) = r_0$ for every $i \in V_x^+$.

Definition 3.10. A configuration c for $x = (w_1, \ldots, w_n)$ is *terminal* if and only if it has the form $c = (\sigma, \epsilon, h, d)$ (for arbitrary σ, h and d).

In other words, we initialize the parser with an empty stack, with all the token nodes of the sentences remaining to be processed, and with a dependency graph where all token nodes are dependents of the special root node 0 and all arcs are labeled with the special label r_0, and we terminate whenever the list of input tokens is empty. As we will see, this happens when we have completed one left-to-right pass over the sentence.

We will use the following notation to refer to different classes of configurations, with or without reference to a specific sentence $x = (w_1, \ldots, w_n)$:

1. C (C_x) is the set of all possible configurations (for x).
2. C^0 (C_x^0) is the set of initial configurations (for x).
3. C^ϵ (C_x^ϵ) is the set of terminal configurations (for x).
4. C^n (C_x^n) is the set of non-terminal configurations (for x), i.e., $C^n = C - C^\epsilon$ ($C_x^n = C_x - C_x^\epsilon$).

It can be noted that C_x^0, for a given sentence $x = (w_1, \ldots, w_n)$, is always a singleton set $C_x^0 = \{(\epsilon, (1, \ldots, n), h_0, d_0)\}$.

3.4.2 Transitions

A transition is a partial function $t : C^n \to C$. In other words, a transition maps non-terminal configurations to new configurations but may be undefined for some non-terminal configurations. The parsing algorithm uses four transitions, two of which are parameterized by a dependency type $r \in R$. Formally:

Definition 3.11. A *transition* is a partial function $t : C^n \rightarrow C$.

Definition 3.12. Given a set of dependency types R, the following transitions are possible for every $r \in R$:

1. LEFT-ARC(r):
   $(\sigma|i, j|\tau, h, d) \rightarrow (\sigma, j|\tau, h[i \mapsto j], d[i \mapsto r])$
   if $h(i) = 0$

2. RIGHT-ARC(r):
   $(\sigma|i, j|\tau, h, d) \rightarrow (\sigma|i|j, \tau, h[j \mapsto i], d[j \mapsto r])$
   if $h(j) = 0$

3. REDUCE:
   $(\sigma|i, \tau, h, d) \rightarrow (\sigma, \tau, h, d)$
   if $h(i) \neq 0$

4. SHIFT:
   $(\sigma, i|\tau, h, d) \rightarrow (\sigma|i, \tau, h, d)$

The transition LEFT-ARC(r) (LA(r)) makes the top token i a (left) dependent of the next token j with dependency type r, i.e., $j \xrightarrow{r} i$, and immediately pops the stack. This transition can apply only if $h(i) = 0$, i.e., if the top token is previously attached to the root 0. The node i is popped from the stack because it must be complete with respect to left and right dependents at this point (given the assumption of projectivity).

The transition RIGHT-ARC(r) (RA(r)) makes the next token j a (right) dependent of the top token i with dependency type r, i.e., $i \xrightarrow{r} j$, and immediately pushes j onto the stack. This transition can apply only if $h(j) = 0$, i.e., if the next token is previously attached to the root 0.[15] The node j is pushed onto the stack since it must be complete with respect to its left dependents at this point, but it cannot be popped because it may still need new dependents to the right.

The transition REDUCE (RE) pops the stack. This transition can apply only if $h(i) \neq 0$, i.e., if the top token i is already attached to a token node. This transition is needed for popping a node that was pushed in a RIGHT-ARC(r) transition and which has since found all its right dependents.

The transition SHIFT (SH) pushes the next token i onto the stack. This transition can apply unconditionally as long as there are input tokens remaining. It is needed for processing nodes that have their heads to the right, as well as nodes that are to remain attached to the special root node.

The transitions LEFT-ARC(r) and REDUCE have in common that they reduce the size of the stack by 1 without affecting the length of the input sequence. We will therefore call these transitions POP-transitions. In a similar

[15] This condition is in fact superfluous, since it is impossible for the next input token to be attached to any other node, but it is included for symmetry.

fashion, RIGHT-ARC(r) and SHIFT have in common that they reduce the length of the input sequence by 1 while increasing the size of the stack by 1 and will therefore be called PUSH-transitions.

We will use the symbol T_R to refer to the set of possible transitions for a given set R of dependency types. Thus, for the parsing algorithm defined here, $T_R = \{\mathrm{LA}(r), \mathrm{RA}(r), \mathrm{RE}, \mathrm{SH} \mid r \in R\}$. Since transitions are functions, we will also use the notation $t(c) = c'$ to signify that transition t maps configuration c to c'. (If t is undefined for c, we write $t(c) = \perp$ instead.)

Having defined configurations and transitions, we can now define transition sequences in the obvious way:

Definition 3.13. A *transition sequence* is a sequence of configurations $C_{0,m} = (c_0, c_1, \ldots, c_m)$, where:

1. The first configuration c_0 is an initial configuration ($c_0 \in C^0$).
2. For every $i > 0$, there is a transition $t \in T_R$ such that $c_i = t(c_{i-1})$.

Definition 3.14. A transition sequence $C_{0,m}$ is *terminating* if and only if it ends in a terminal configuration, i.e., $C_{0,m} = (c_0, c_1, \ldots, c_m)$ and $c_m \in C^\epsilon$.

Finally, we need to connect transition sequences to sentences and dependency graphs. The basic idea is that a transition sequence corresponds to the analysis of a sentence if its initial configuration has the token nodes of the sentence as its input sequence. Moreover, if the transition sequence is terminating then the dependency graph $G = (V, E, L)$ defined by the terminal configuration is the dependency graph assigned to the sentence. In formal terms:

Definition 3.15. A transition sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ corresponds to a sentence x if and only if $c_0 \in C^0_x$, i.e., $c_0 = (\epsilon, (1, \ldots, n), h_0, d_0)$.

Definition 3.16. A terminating transition sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ corresponding to a sentence x assigns to x the dependency graph $G_m = (V_x, E_m, L_m)$ defined by c_m.

Note that we simplify the notation by writing $G_m = (V_x, E_m, L_m)$, rather than $G_{c_m} = (V_x, E_{c_m}, L_{c_m})$, when the defining configuration is indexed by a transitition sequence (cf. definition 3.8). Figure 3.5 shows a terminating transition sequence that corresponds to the example sentence from figure 3.1. The dependency graph assigned to the sentence is the one described formally in figure 3.4.

3.4.3 Deterministic Parsing

The transition system defined in the previous section is nondeterministic in itself, since there is normally more than one transition applicable to a given configuration. More precisely, the system can be characterized as follows:

$$
\begin{array}{rlll}
 & (\ \epsilon, & (1,\ldots,9),\ h_0, & d_0 \\
\text{SH}\rightarrow & (\ (1), & (2,\ldots,9),\ h_0, & d_0 \\
\text{LA}(\text{NMOD})\rightarrow & (\ \epsilon, & (2,\ldots,9),\ h_1 = h_0[1 \mapsto 2], & d_1 = d_0[1 \mapsto \text{NMOD}] \\
\text{SH}\rightarrow & (\ (2), & (3,\ldots,9),\ h_1, & d_1 \\
\text{LA}(\text{SBJ})\rightarrow & (\ \epsilon, & (3,\ldots,9),\ h_2 = h_1[2 \mapsto 3], & d_2 = d_1[2 \mapsto \text{SBJ}] \\
\text{SH}\rightarrow & (\ (3), & (4,\ldots,9),\ h_2, & d_2 \\
\text{SH}\rightarrow & (\ (3,4), & (5,\ldots,9),\ h_2, & d_2 \\
\text{LA}(\text{NMOD})\rightarrow & (\ (3), & (5,\ldots,9),\ h_3 = h_2[4 \mapsto 5], & d_3 = d_2[4 \mapsto \text{NMOD}] \\
\text{RA}(\text{OBJ})\rightarrow & (\ (3,5), & (6,\ldots,9),\ h_4 = h_3[5 \mapsto 3], & d_4 = d_3[5 \mapsto \text{OBJ}] \\
\text{RA}(\text{NMOD})\rightarrow & (\ (3,5,6), & (7,8,9),\ h_5 = h_4[6 \mapsto 5], & d_5 = d_4[6 \mapsto \text{NMOD}] \\
\text{SH}\rightarrow & (\ (3,\ldots,7), & (8,9),\ h_5, & d_5 \\
\text{LA}(\text{NMOD})\rightarrow & (\ (3,5,6), & (8,9),\ h_6 = h_5[7 \mapsto 8], & d_6 = d_5[7 \mapsto \text{NMOD}] \\
\text{RA}(\text{PMOD})\rightarrow & (\ (3,\ldots,8), & (9),\ h_7 = h_6[8 \mapsto 6], & d_7 = d_6[8 \mapsto \text{PMOD}] \\
\text{RE}\rightarrow & (\ (3,5,6), & (9),\ h_7, & d_7 \\
\text{RE}\rightarrow & (\ (3,5), & (9),\ h_7, & d_7 \\
\text{RE}\rightarrow & (\ (3), & (9),\ h_7, & d_7 \\
\text{RA}(\text{P})\rightarrow & (\ (3,9), & \epsilon,\ h_8 = h_7[9 \mapsto 3], & d_8 = d_7[9 \mapsto \text{P}]
\end{array}
$$

Fig. 3.5. Transition sequence for English sentence (cf. figures 3.1 and 3.4)

1. If the stack is empty then the only possible transition is SHIFT.
2. If the stack is non-empty then the possible transitions are RIGHT-ARC(r), SHIFT and one of the following:
   a) LEFT-ARC(r) if the token i on top of the stack is attached to the root ($h(i) = 0$).
   b) REDUCE if the token i on top of the stack is attached to a non-root ($h(i) \neq 0$).
3. For the transitions LEFT-ARC(r) and RIGHT-ARC(r) there is always a nondeterministic choice of the dependency type $r \in R$.

In order to perform deterministic parsing, the transition system needs to be supplemented with a mechanism for predicting the next transition at each nondeterministic choice point, as well as choosing a dependency type r for the transitions LEFT-ARC(r) and RIGHT-ARC(r). Such a mechanism can be called an *oracle* (Kay, 2000). Strictly speaking, an oracle always makes the correct prediction, an ideal that we can only hope to approximate in practice, and the term *guide* is therefore often used to denote an (imperfect) approximation of an oracle (Boullier, 2003). In this study, we will use the term *guide* for any function that satisfies the formal conditions required to resolve the nondeterminism of our transition system, and reserve the term *oracle* for the special case of an infallible guide. In predicting the next transition, a guide may take into account not only the current configuration c but also the annotation functions A_x for the given sentence x. Hence, we define a guide to be a function of two arguments, a configuration c and a set of annotation functions A_x, using the symbol A_f to denote the set of possible annotation functions.

Definition 3.17. A *guide* is a function $g : (C^n \times 2^{A_f}) \rightarrow T_R$ (where C^n is the set of non-terminal configurations, 2^{A_f} is the set of all sets of annotation functions, and T_R the set of possible transitions) satisfying the condition that $g(c, A_x)$ is a transition applicable to c (for every $c \in C^n_x$), that is:

1. If $g(c, A_x) \in \{\mathrm{LA}(r), \mathrm{RA}(r), \mathrm{RE}\}$ then c has a non-empty stack $\sigma|i$.
2. If $g(c, A_x) = \mathrm{LA}(r)$ and the stack of c is $\sigma|i$, then $h(i) = 0$.
3. If $g(c, A_x) = \mathrm{RE}$ and the stack of c is $\sigma|i$, then $h(i) \neq 0$.
4. If $g(c, A_x) = \mathrm{RA}(r)$ and the input of c is $j|\tau$, then $h(j) = 0$.

Definition 3.18. An *oracle* is a guide o such that, if c is a configuration for the sentence x, $o(c, A_x) = t$ if and only if t is the transition out of c that leads to the correct analysis of x.

For practical purposes, an oracle is an idealized notion that we can only try to approximate. In chapters 4–5 we will explore a data-driven approach to guided parsing, inducing classifiers from treebank data. But guides can also be constructed using a grammar-driven approach, e.g., by combining grammar rules with hand-crafted heuristics (Nivre, 2003).

However, for the analysis of the parsing algorithm it is irrelevant which kind of guide (or oracle) we are using. For the remainder of this chapter we will therefore make the idealized assumption that there exists an oracle o that determines the correct transition given a non-terminal configuration. For the analysis of time complexity, we will also assume that the oracle function can be computed in constant time. Moreover, we will assume for the time being that the transition from one configuration to the next is also a constant-time operation.

Assuming that we have an oracle o satisfying the conditions stated above, the algorithm for deterministic dependency parsing is very simple and straightforward:

PARSE($x = (w_1, \ldots, w_n)$)
1 $c \leftarrow (\epsilon, (1, \ldots, n), h_0, d_0)$
2 **while** $c = (\sigma, \tau, h, d)$ is not terminal
3 **if** $\sigma = \epsilon$
4 $c \leftarrow$ SHIFT(c)
5 **else**
6 $c \leftarrow [o(c, A_x)](c)$
7 $G \leftarrow (V_x, E_c, L_c)$
8 **return** G

As long as the parser is in a non-terminal configuration, it applies the SHIFT transition if the stack is empty and otherwise the transition $o(c, A_x)$ predicted by the oracle. When a terminal configuration is reached, the dependency graph defined by this configuration is returned.

3.4.4 Algorithm Analysis

In this section we will prove two theorems about the parsing algorithm defined in the preceding sections. We will also state a third theorem that is relevant here but which will not be proven until the next chapter. The first theorem says that every sentence has a terminating transition sequence whose length is directly proportional to the length of the sentence; the second theorem says that every such terminating transition sequence results in a projective dependency graph; and the third theorem says that every projective dependency graph has a corresponding terminating transition sequence. In section 3.4.5 we will argue that this means that the algorithm is optimal with respect to the evaluation criteria for robustness, disambiguation and efficiency defined in section 2.4. However, before we turn to discuss the implications of our theorems, we will also give a brief informal analysis of the way in which the algorithm is related to other algorithms for dependency parsing that can be found in the literature.

Theorem 3.19 establishes two crucial properties of the parsing algorithm. First, there exists a terminating transition sequence corresponding to every sentence. Secondly, any terminating transition sequence corresponding to a sentence $x = (w_1, \ldots, w_n)$ consists of at most $2n - 1$ configurations. Together these results imply that parsing can be performed in linear time, given that oracle predictions and transitions can be performed in constant time.

Theorem 3.19. Given a sentence $x = (w_1, \ldots, w_n)$, the deterministic parsing algorithm terminates after at most $2n - 1$ transitions.[16]

Proof. We first show that a transition sequence starting in an initial configuration $c_0 = (\epsilon, (1, \ldots, n), h_0, d_0)$ can contain at most n PUSH-transitions and that such a sequence must be terminating:

1. PUSH-transitions have as a precondition that the input sequence τ is non-empty and decreases its length by 1. Since τ has length n in c_0 and there are no transitions that increase the length of τ, the maximum number of PUSH-transitions in a transition sequence starting in c_0 is n. For the same reason, a transition sequence $C_{0,m} = (c_0, \ldots, c_m)$ containing n PUSH-transitions must end in a configuration $c_m = (\sigma, \epsilon, h, d)$ which is terminal.

We then show that a transition sequence containing n PUSH-transitions can contain at most $n - 1$ POP-transitions:

2. POP-transitions have as a precondition that the stack σ is non-empty and have as an effect that the size of σ is decreased by 1. Since σ is empty in the initial configuration c_0 and the only transitions that increase the size

[16] $2n - 1$ transitions corresponds to $2n - 1$ iterations of the **while** loop in the deterministic algorithm. This means that the resulting transition sequence will contain $2n$ configurations including the initial configuration.

of σ are PUSH-transitions, of which there can be no more than n instances, it follows that the maximum number of POP-transitions in a transition sequence starting in c_0 cannot be greater than n. In fact, since the nth PUSH-transition results in a terminal configuration, the maximum number of POP-transitions is $n - 1$.

Given that the number of POP-transitions is bounded by the number of PUSH-transitions, and given that at least one PUSH-transition (SHIFT) is applicable to every non-terminal configuration c_i, we conclude that, for every initial configuration $c_0 = (\epsilon, (1, \ldots, n), h_0, d_0)$ there is at least one transition sequence $C_{0,m} = (c_0, \ldots, c_m)$ containing exactly n PUSH-transitions and at most $n - 1$ POP-transitions, leading to a terminal configuration $c_m = (\sigma, \epsilon, h, d)$. □

Theorem 3.19 implies that the running time of the deterministic parsing algorithm defined in section 3.4.3 is $O(n)$, where n is the length of the input sentence, provided that the time required for line 4 and 6 is $O(1)$. An inspection of the transitions defined in section 3.4.2 shows that the application of a transition t to a configuration c_i, to derive the next transition $c_{i+1} = t(c_i)$, only requires the following operations:

1. Pushing or popping the stack σ.
2. Deleting the head of the input list τ.
3. Updating the function h to $h[i \mapsto j]$ (given i and j).
4. Updating the function d to $d[i \mapsto r]$ (given i and r).

All of these operations can be performed in constant time with an appropriate choice of data structures, e.g., using arrays to store h and d and any standard implementation of lists for σ and τ.[17] This leaves the computation of the oracle function $o(c_i, A_x) = t$, which must also be performed in constant time. We will return to this issue in chapter 4, where we will discuss the different types of guides needed for inductive dependency parsing and show that they can be implemented to run in $O(1)$ time.

A consideration of the data structures needed for parsing, again disregarding the implementation of the oracle, indicates that the space complexity of the deterministic parsing algorithm is also $O(n)$, although we will not formally prove this. Since parsing is deterministic, only one configuration c needs to be stored at any time (cf. section 3.4.3). The four components of a configuration $c = (\sigma, \tau, h, d)$ all require space that is proportional to the length n of the sentence times the space needed to store the information about an individual token. Since the latter only amounts to a token index (for σ, τ and h) or a dependency type (for d), it is evident that the overall space requirement is $O(n)$.

[17] We also need to make sure that these data structures can accomodate arbitrarily large inputs, since there is no upper bound on the length of a sentence, although any given sentence is of course finite.

Let us turn now to theorem 3.21, which concerns properties of the dependency graph defined by a terminating transition sequence. More precisely, it says that every terminating transition sequence defines a projective dependency graph according to definition 3.6. Together with theorem 3.19, this guarantees that the parsing algorithm satisfies the requirements of robustness and disambiguation, a topic to which we will return in section 3.4.5. First of all, we introduce a simple lemma, which will be used in the proof of theorem 3.21 and again in chapter 4.

Lemma 3.20. If $C_{0,m}$ is a terminating transition sequence for a sentence $x = (w_1, \ldots, w_n)$ then, for every configuration $c_p = (\sigma_p, \tau_p, h_p, d_p)$ $(0 < p < m)$ and token j, if $h_p(j) \neq 0$ then $h_q(j) = h_p(j)$ for every $q > p$.

Proof. The lemma follows directly from the conditions associated with the transitions Left-Arc(r) and Right-Arc(r), which require that $h(j) = 0$ for the potential dependent (definition 3.12).

Theorem 3.21. Every terminating transition sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ corresponding to a sentence x defines a dependency graph $G_m = (V_x, E_m, L_m)$ satisfying the following five conditions (cf. section 3.3.2):

1. Root
2. Connectedness
3. Single-Head
4. Acyclicity
5. Projectivity

Proof. The proof consists of five parts, one for each of the five conditions:

1. Root: This condition follows directly from the definition of the arc relation $E_m = \{(i, j) \mid h_m(j) = i\}$ (cf. section 3.4.2). Since h_m is not defined for the node 0, there can be no arc $(i, 0) \in E_m$.
2. Connectedness: In order to prove that G_m is connected, we begin by observing that the graph G_0 defined by the initial configuration c_0 is connected, since every node except 0 is a dependent of 0. We then show that every transition t_k, where $c_k = t_k(c_{k-1})$ and $1 \leq k \leq m$, preserves this property. Since neither Reduce nor Shift modifies the h function, we can concentrate on Left-Arc(r) and Right-Arc(r), which both modify the graph by replacing an arc of the form $(0, j)$ with an arc of the form (i, j). This means that the resulting graph G_k will be connected if G_{k-1} is connected and there is a path from 0 to i in G_{k-1} (i.e., $0 \rightarrow^* i$), since i will then belong to the same connected component before and after the transition. To prove that $0 \rightarrow^* i$ in G_{k-1}, we consider the two cases separately:

a) If $t_k = \text{LEFT-ARC}(r)$ then $i \in \tau_{k-1}$, which entails that $(0, i) \in E_{k-1}$ (since both $\text{LEFT-ARC}(r)$ and $\text{RIGHT-ARC}(r)$ have as a postcondition that the dependent token is not a member of the input sequence). Hence, $0 \rightarrow^* i$ in G_{k-1}.

b) If $t_k = \text{RIGHT-ARC}(r)$, then $i \in \sigma_{k-1}$ and either $(0, i) \in E_{k-1}$, if i was pushed in a SHIFT transition, or there is some node $i' \in \sigma_{k-1}$ such that $0 < i' < i$ and $(i', i) \in E_{k-1}$. Since the same holds for any token on the stack (including i'), there must be a path from 0 to i which is maximally of length i. Hence, $0 \rightarrow^* i$ in G_{k-1}.

We conclude that G_m is connected.

3. SINGLE-HEAD: This condition also follows from the definition of the arc relation $E_m = \{(i, j) \mid h_m(j) = i\}$ (cf. section 3.4.2). Since h_m is a (total) function with domain V^+ there is exactly one node i such that $(i, j) \in E$ for every token node j.

4. ACYCLICITY: To prove that G is acyclic we make use of a theorem from graph theory saying that for a loop-free undirected graph $G = (V, E)$ the following two claims are equivalent (Grimaldi, 2004, theorem 12.5):

   a) G is connected and $|V| = |E| + 1$.
   b) G contains no cycles and $|V| = |E| + 1$.

   Since we have already proven CONNECTEDNESS, we only have to show that $|V_x| = |E_m| + 1$ and that G_m is loop-free, i.e., that E_m contains no arcs of the form (i, i). The first property again follows from the definition of the arc relation $E_m = \{(i, j) \mid h_m(j) = i\}$, which defines exactly one arc for every node except 0. The second property follows from the observation that G_0 is loop-free together with the fact that every transition that modifies h_k to h_{k+1} adds an arc between two distinct nodes i and j.

5. PROJECTIVITY: We need to prove that if $i \xrightarrow{r} j$ then, for every node k such that $i < k < j$ or $i > k > j$, $i \rightarrow^* k$. Given Lemma 3.20, there exists a unique configuration $c_p = (\sigma_p, \tau_p, h_p, d_p)$ such that either $\sigma_p = \sigma_{p^-}|i$, $\tau_p = j|\tau_{p^-}$ and $t_{p+1} = \text{RIGHT-ARC}(r)$ (for the case $i < j$) or $\sigma_p = \sigma_{p^-}|j$, $\tau_p = i|\tau_{p^-}$ and $t_{p+1} = \text{LEFT-ARC}(r)$ (for the case $j < i$). In either case, the desired result follows if we can show that, according to h_p, every k such that $i < k < j$ (first case) or $i > k > j$ (second case) is dominated by i or j, i.e., $i \rightarrow^* k$ or $j \rightarrow^* k$. Since the two cases are exactly symmetrical, we only prove the claim for the first case, using a proof by induction over the length l_{ij} of the token sequence $x_{(i,j)} = (i, \ldots, j)$.

   - **Basis:** If $l_{ij} = 2$, then $x_{(i,j)} = (i, j)$ and the claim holds vacuously.
   - **Induction:** Assume the claim holds for $l_{ij} \leq l$ (for some $l \geq 2$) and assume $l_{ij} = l + 1$. It follows that $j > i + 1$, which means that the transition t_p from c_{p-1} to c_p must be a POP-transition. (After a PUSH-

transition, $j = i + 1$ by definition.) Hence, there is some node k' such that $\sigma_{p-1} = \sigma_{p^-}|i|k'$ and $\tau_{p-1} = \tau_p = j|\tau_{p^-}$. Since both $l_{ik'} \leq l$ and $l_{k'j} \leq l$, we can use the inductive hypothesis twice to infer:
a) For every k such that $i < k < k'$, $i \rightarrow^* k$ or $k' \rightarrow^* k$.
b) For every k such that $k' < k < j$, $j \rightarrow^* k$ or $k' \rightarrow^* k$.
We know that t_p is a POP-transition. If $t_p = \text{LEFT-ARC}(r')$, then $j \rightarrow k'$. If $t_p = \text{REDUCE}$, there must be an earlier transition $t_{p-q} = \text{RIGHT-ARC}(r')$, where $\sigma_{p-q-1} = \sigma_{p-q-1^-}|i$ and $\tau_{p-q-1} = k'|\tau_{p-q-1^-}$, which implies $i \rightarrow k'$. In both cases, we can conclude that, for every k such that $i < k < j$, $i \rightarrow^* k$ or $j \rightarrow^* k$.

This concludes the proof of theorem 3.21. □

Theorem 3.21 is in a sense the equivalent of a consistency proof for a grammar parsing algorithm, since it guarantees that the transition system only derives analyses that are licensed by the formal model. Theorem 3.22 establishes that any analysis permitted by the formal model can be derived in the transition system, thus corresponding to the completeness proof for a grammar parsing algorithm.

Theorem 3.22. For every projective dependency graph $G_m = (V_x, E_m, L_m)$ for a sentence x, there is a terminating transition sequence $C_{0,m}$ that assigns G_m to x.

Proof. We will postpone the demonstration of theorem 3.22 to section 4.1.6, where we give a constructive proof in the form of an algorithm that actually computes a corresponding transition sequence for any projective dependency graph. By proving the correctness of this algorithm, we prove theorem 3.22 as a corollary.

The importance given to theorems 3.19, 3.21 and 3.22 in this investigation is motivated by the evaluation criteria defined in section 2.4. However, before we turn to a discussion of these criteria again, we will make a few observations on our parsing algorithm in relation to other work on natural language parsing, in particular dependency parsing.

As observed in Nivre (2004a), the algorithm is in many ways similar to that of Yamada and Matsumoto (2003). The most important difference is that, whereas the algorithm of Yamada and Matsumoto (2003) corresponds directly to a strict bottom-up shift-reduce parser for a context-free grammar in Chomsky Normal Form (Chomsky, 1959), our algorithm uses an arc-eager strategy where left dependents are processed bottom-up while right dependents are processed top-down.[18] In addition, Yamada and Matsumoto (2003)

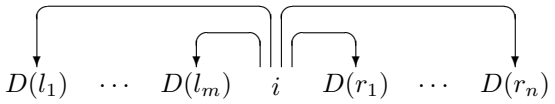
[18] To say that right dependents are processed top-down is true insofar as the arc from the head to the right dependent is added before the dependent is complete. However, the absence of nonterminal nodes in dependency parsing means that there is no top-down *prediction* taking place.

allow multiple passes over the input, whereas our algorithm is restricted to a single left-to-right pass.

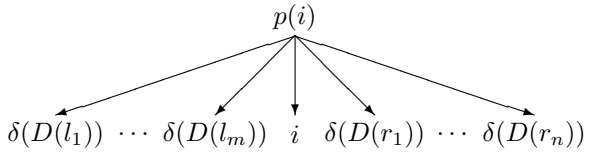
One advantage of the left-to-right arc-eager parsing strategy is that it is optimal with respect to incrementality, in the sense that it minimizes the number of unattached words on the stack (where words attached to the special root are regarded as unattached), a result that is shown both theoretically and experimentally in Nivre (2004a). In this way, our algorithm can be seen as a special case of Covington's incremental strategy for dependency parsing (Covington, 2001), although we do not define it here as a strict word-at-a-time operation. However, it is possible to show that the work done to integrate a new word w_i in Covington's algorithm corresponds to a continuous subsequence $c_k, \ldots, c_l$ of the transition sequence for our algorithm, where c_k is the first configuration where $\tau = (i, \ldots, n)$ and c_l is the first configuration where i is on the top of the stack σ. Moreover, by exploiting the properties of a projective dependency graph, our algorithm achieves a time complexity of $O(n)$, whereas the algorithm defined by Covington (2001) is $O(n^2)$ in its deterministic version.

Finally, it may be interesting to relate our algorithm to methods for context-free parsing. Comparing methods for dependency parsing and constituency parsing is not completely straightforward, since the mapping from dependency representations to constituency representations is one-to-many. Let us therefore consider a specific mapping to lexicalized constituent trees, which is only defined for the case where there is a single node i attached to the special root 0.[19] If $D(i)$ is the (unlabeled) dependency tree rooted at i, then there is a corresponding constituent tree $T = \delta(D(i))$, where the mapping δ is defined recursively as follows:

If $D(i)$ has the form



then $\delta(D(i)) =$



where $p(i)$ is the part-of-speech of the token i.

[19] This is essentially the mapping from dependency to constituency used by Carroll and Charniak (1992).

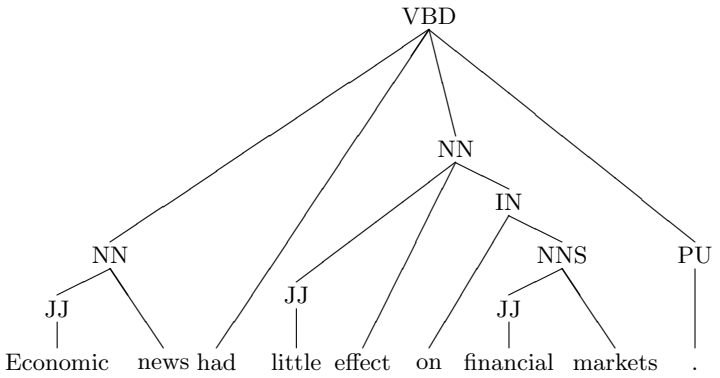


Fig. 3.6. Derived constituent structure for English sentence

Figure 3.6 shows the constituency tree corresponding to the dependency analysis of our example sentence from figure 3.1, which includes fewer nonterminal nodes than the constituent analysis of this sentence given in figure 2.1. Given this mapping, our parsing algorithm corresponds quite closely to an arc-eager head-corner algorithm (Kay, 1989; Van Noord, 1997), where each constituent with head i is built inside out from the head by adding children in the order $i, l_m, \ldots, l_1, r_1, \ldots, r_n$. The fact that the algorithm is arc-eager (Abney and Johnson, 1991) in this context means that children to the right of the head may be added before they are complete. A more precise investigation of the relation to constituency-based parsing is an interesting topic in itself but is outside the scope of this study.

3.4.5 Evaluation Criteria Revisited

The optimization strategy underlying our approach to dependency-based text parsing requires a framework that is provably optimal with respect to robustness, disambiguation and efficiency, within which we can gradually improve parsing accuracy without sacrificing other properties (cf. section 2.3.4). It is now time to verify that the formal framework and parsing algorithm developed in this chapter satisfy these requirements.

According to the evaluation criteria defined in section 2.4, both robustness and disambiguation are taken to be absolute requirements, which jointly entail that exactly one analysis is assigned to every text sentence. In virtue of theorem 3.19, we know that our parsing algorithm, given an oracle, constructs exactly one dependency graph for every sentence. And given theorem 3.21, we know that this dependency graph is a well-formed representation according to the framework defined in section 3.3. In this sense, the methods are provably optimal with respect to robustness and disambiguation.

Whether the single well-formed representation assigned to each text sentence is also the correct analysis of that sentence is of course an entirely different question. But this is the problem of accuracy, which will be the focus of our attention in the next two chapters, where we will attempt to construct guides, approximating oracles, by applying inductive machine learning techniques to treebank data. However, theorem 3.22, although still assumed without proof, at least guarantees that the correct analysis is reachable in the search space of the parsing algorithm. Alternatively, we may conclude from theorem 3.22 that our parsing algorithm satisfies the absolute criterion of accuracy in the presence of an oracle (cf. section 2.4.3).

Theorem 3.19, together with certain assumptions about the implementation of oracles and transitions, entails that the time complexity of deterministic parsing in the present framework is $O(n)$, where n is the length of the input sentence. This result is arguably theoretically optimal as well, since any reasonable method for parsing natural language sentences will at least have to scan the sequence of tokens once, which means that linear time is a lower bound on parsing time. The space complexity is also $O(n)$, although we have only given an informal argument to support this claim, which means that the parsing algorithm satisfies the criterion of efficiency stated in section 2.4.4. However, as discussed in chapter 2, the optimization of accuracy may easily compromise efficiency, which means that the two notions must be optimized together. In the empirical experiments in chapter 5, we will therefore need to evaluate efficiency as well as accuracy. Moreover, since we will adopt a data-driven approach, we need to consider efficiency in *training* as well as efficiency in *parsing*.

4

Inductive Dependency Parsing

Machine learning based on various forms of inductive inference has been used for a wide range of problems in natural language processing, with syntactic parsing being one of the more prominent problems during the last decade. In particular, methods for parsing unrestricted natural language text under requirements of robustness and disambiguation have to an increasing extent been characterized by the data-driven approach, sometimes in combination with a grammar-based strategy.

In this chapter, we will examine how the parsing methods developed in the previous chapter can be combined with the data-driven approach using a form of guided parsing. In this scheme, we use inductive machine learning to construct parser guides from treebank data. These guides are essentially classifiers that predict the next transition given the current configuration at each nondeterministic choice point. In this way, we can maintain the high efficiency of deterministic processing while developing more and more accurate guides in order to improve parsing accuracy.

An empirical evaluation of this methodology, with respect to accuracy as well as efficiency, will be presented in chapter 5. Here we will focus instead on the model of inductive inference, which belongs to the class of conditional history-based models, and the way in which this model can be combined with a deterministic parsing strategy and with discriminative machine learning. We will also discuss the different kinds of contextual features that can be used as a basis for prediction, and we will introduce memory-based learning, which is the learning method that will be used in the experiments reported in the next chapter. Finally, we will give a very brief description of the implementation of all these elements in a system called MaltParser, which has been used to carry out all the experiments reported in this book.

4.1 A Framework for Inductive Dependency Parsing

In order to situate our approach within the larger context of data-driven parsing methods, we begin by recapitulating our analysis of the data-driven approach in terms of inductive inference based on a formal model of syntactic representations and a representative sample of the relevant text language (cf. section 2.3.2). We then define our model of inductive inference as a conditional history-based model, combine it with a deterministic parsing strategy and derive a learning problem that can be solved using discriminative classifier induction. Finally, we show how training data for this learning problem can be derived from a treebank using a variant of the deterministic parsing algorithm.

4.1.1 Data-Driven Text Parsing

In the data-driven approach to text parsing, the mapping from input strings to output analyses is defined by an inductive mechanism applied to a text sample $T_t = (x_1, \ldots, x_n)$ from the language L to be analyzed. As explained in section 2.3.2, we can generally understand a data-driven text parsing system as consisting of three essential components:

1. A formal model M defining permissible analyses for sentences in L.
2. A sample of text $T_t = (x_1, \ldots, x_n)$ from L, with or without the correct analyses $A_t = (y_1, \ldots, y_n)$.
3. An inductive inference scheme I defining actual analyses for the sentences of any text $T = (x_1, \ldots, x_n)$ in L, relative to M and T_t (and possibly A_t).

The model M may be a formal grammar, defining an exact string language, in which case permissible representations will be restricted to this language. In our case, the model M is given by the formal framework for dependency parsing defined in section 3.3, which does not impose any restriction on the strings being analyzed. Given a set of dependency types R, the permissible representations for a sentence $x = (w_1, \ldots, w_n)$ is the set of all well-formed projective dependency graphs $G = (V_x, E, L)$ with node set $V_x = \mathbf{Z}_{n+1}$ and labeling function $L : E \rightarrow R$. Alternatively, in terms of the parsing method defined in section 3.4, we can characterize the permissible representations as the set of dependency graphs defined by some terminating transition sequence corresponding to x. Since the only requirement is that the set of token nodes in V_x^+ are in a one-to-one mapping with the tokens in the sentence, and that the dependency type labels are restricted to a given set R, it is clear that this imposes no restriction on the set of strings that can be parsed by the system. Hence, this is not a grammar-driven approach to text parsing.

The sample of text T_t, which is our basis for inductive generalization, will normally be called the *training corpus*. Although there exist *unsupervised* learning methods that apply to raw, unannotated text, such as the Inside-Outside algorithm for estimating the parameters of a PCFG, we will follow the mainstream tradition in data-driven parsing and use *supervised* learning,

which requires the text sample to be annotated with representations defined by the model M. This means that the training corpus will be a *treebank* of the language L, i.e., a text corpus where each sentence is annotated with its correct analysis relative to the given model M (Abeillé, 2003b; Nivre, forthcoming). There are several methodological problems connected to the use of treebank data for inductive learning, having to do with the representativity of the data, the validity and reliability of the annotation, and the problem of converting annotations from one type of representation to another. These problems will be dealt with in chapter 5, where we report the experiments performed to evaluate the framework of inductive dependency parsing. For the remainder of this chapter, we will simply assume that we have available a sample of text $T_t = (x_1, \ldots, x_n)$ from the language L that we want to analyze and that every sentence $x_i \in T_t$ has been annotated with its correct analysis $y_i \in A_t$, where y_i is a well-formed dependency graph as defined in the preceding chapter.

While the formal model M has been described in detail in chapter 3 and problems connected to the training sample T_t will be dealt with in chapter 5, the inductive inference scheme I is the central topic of this chapter. This is the heart of the data-driven approach to text parsing, since it defines the way in which a parsing system can generalize from sentences found in the training corpus T_t to previously unseen sentences encountered in new texts.

4.1.2 Inductive Inference

As seen in section 2.3.2, an inductive inference scheme can be conceptualized and implemented in many different ways but can generally be decomposed into three main elements:

1. A parameterized *stochastic model* M_Θ assigning a score $S(x, y)$ to each permissible analysis y of a sentence x, relative to a set of parameters Θ.
2. A *parsing method*, i.e., a method for computing the best analysis y for a sentence x according to $S(x, y)$ (given an instantiation of Θ).
3. A *learning method*, i.e., a method for instantiating Θ based on inductive inference from the training sample T_t.

It is important to remember that this is a conceptual decomposition, which does not always correspond to system components or temporal processes. For example, while it is usually possible to divide the work done by the learning method and the parsing method into a *training phase*, which is applied once to the training corpus, and a *parsing phase*, which is applied to every new sentence without reprocessing the training corpus, the exact division of labor between these phases is dependent on the learning strategy. With an *eager* learning method, all the inference is performed during the training phrase; with a *lazy* learning method, most of the inductive inference is postponed until the parsing phase.

For the time being we will disregard the practical implementation of both parsing and learning and concentrate on the characterization of models and

parameters for inductive dependency parsing. We will return to the parsing problem in section 4.1.4 and to the learning problem in section 4.1.5.

4.1.3 History-Based Models

The general approach of inductive dependency parsing is compatible with a variety of different models and parameterizations, but in this book we will limit our attention to conditional history-based models (cf. section 2.3.2), which are easily combined with the parsing methods developed in the previous chapter. In a history-based model, the parameterization essentially involves three steps (cf. Collins, 1999):

1. Define a one-to-one mapping between syntactic analyses y and decision sequences $D = (d_1, \ldots, d_m)$ such that D is a canonical derivation of y.
2. Define the score $S(x, y)$, for every sentence x and analysis y, in terms of each decision d_i in the corresponding decision sequence $D = (d_1, \ldots, d_m)$, conditioned on the history $H = (d_1, \ldots, d_{i-1})$.
3. Define a function Φ that groups histories (and decision sequences) into equivalence classes, thereby reducing the number of parameters in Θ to make the learning problem manageable.

In a conditional history-based model, the score $S(x, y)$ defined by the model is the conditional probability $P(y \mid x)$ of the analysis y given the sentence x, which means that the input sentence is a conditioning variable for each decision in the decision sequence:

$$P(y \mid x) = P(d_1, \ldots, d_m \mid x) = \prod_{i=1}^{m} P(d_i \mid d_1, \ldots, d_{i-1}, x) \tag{4.1}$$

In order to get a manageable learning problem, it is normally necessary to introduce a function Φ, which defines an equivalence relation among histories. This gives us the final form of the parameterized model:

$$P(y \mid x) = P(d_1, \ldots, d_m \mid x) = \prod_{i=1}^{m} P(d_i \mid \Phi(d_1, \ldots, d_{i-1}, x)) \tag{4.2}$$

The parameters of this model are simply the conditional probabilities $P(d \mid H)$, for all possible decisions d and non-equivalent histories H.

In the framework investigated in this book, the mapping from analyses to decision sequences is given by the transition system defined in section 3.4.2, where every terminating transition sequence $C_{0,m} = (c_0, \ldots, c_m)$ defines exactly one dependency graph G (definition 3.16). The inverse mapping from dependency graphs to transition sequences will be discussed in section 4.1.5 below, because it will be needed to derive training instances for the inductive learner, and we will also demonstrate that any well-formed dependency

graph can be mapped to a transition sequence. It is important to remember that, although we will only consider deterministic parsing strategies in this book, the transition system itself is nondeterministic, which means that it associates a set of transition sequences, and dependency graphs, with any given sentence. And other parsing strategies might explore a larger part of this space of alternatives.

Given that an analysis in our framework is a dependency graph G defined by a transition sequence $C_{0,m} = (c_0, \ldots, c_m)$, where the transition t_i defines the mapping from configuration c_{i-1} to c_i, i.e., $c_i = t_i(c_{i-1})$, the conditional history-based model can now be expressed as follows:

$$P(G \mid x) = P(c_0, \ldots, c_m \mid x) = \prod_{i=1}^{m} P(t_i \mid c_0, \ldots, c_{i-1}, x) \tag{4.3}$$

However, when considering the input sentence x as a conditioning variable, we want to be able to take into account not only the sequence of tokens but any information that is available about the sentence as a result of preprocessing. Therefore, we replace x with the set of annotation functions A_x, which also includes the function w_x mapping string positions to tokens (cf. section 3.3.1):

$$P(G \mid A_x) = P(c_0, \ldots, c_m \mid A_x) = \prod_{i=1}^{m} P(t_i \mid c_0, \ldots, c_{i-1}, A_x) \tag{4.4}$$

In order to reduce the number of model parameters, we first group together all histories that end in the same configuration. In other words, we make a kind of Markov assumption to the effect that the probability of a transition from configuration c_i is independent of all earlier configurations in the transition sequence. However, this assumption is not as drastic as it may seem, since a configuration $c = (\sigma, \tau, h, d)$ records almost all relevant information about the preceding transition sequence in the state of σ, τ, h and d. Therefore, we simply use the function Φ to define equivalence classes of pairs (c, A_x), consisting of a configuration c and an input sentence x represented by its annotation functions A_x:

$$P(G \mid A_x) = P(c_0, \ldots, c_m \mid A_x) = \prod_{i=1}^{m} P(t_i \mid \Phi(c_{i-1}, A_x)) \tag{4.5}$$

We will use the term *parser condition* to refer to a pair of the form (c, A_x), where c is a parser configuration and A_x is the set of annotation functions for a sentence x, and we will use the term *parser state* for an equivalence class of parser conditions defined by the function Φ. We will also say that Φ is a function from parser conditions to parser states, although we will continue to write Φ as a function of two arguments, one configuration and one set of annotation functions.

The model parameters are the conditional probabilities $P(t \mid \Phi(c, A_x))$, for all possible transitions $t \in T_R$ and distinct parser states $\Phi(c, A_x)$. The cardinality of the parameter set therefore depends on two factors. The first is

the number of distinct transitions, which is $|T_R| = 2|R| + 2$ (where $|R|$ is the number of distinct dependency types), since there are $|R|$ different instances of LEFT-ARC(r) and RIGHT-ARC(r) plus REDUCE and SHIFT. The second is the number of distinct parser states $\Phi(c, A_x)$, which depends on the definition of Φ but which will normally be many orders of magnitude greater than $|T_R|$.

In section 4.2 we will show how the parameterization Φ can be defined by a set of feature functions $\{\phi_1, \ldots, \phi_p\}$ that extract relevant features from the current parser condition. The definition of these feature functions will in turn determine the exact number of parser states and model parameters.

4.1.4 Parsing Methods

Given a conditional history-based model, the conditional probability $P(y_j \mid x)$ of analysis y_j given input x can be used to rank a set of alternative analyses $\{y_1, \ldots, y_k\}$ of the input sentence x, derived by a nondeterministic parser. If the model allows a complete search of the analysis space, we can in this way be sure to find the analysis y_j that maximizes the probability $P(y_j \mid x)$ according to the model:

$$\arg\max_{y_j} P(y_j \mid x) = \arg \max_{(d_1,\ldots,d_m)} \prod_{i=1}^{m} P(d_i \mid \Phi(d_1, \ldots, d_{i-1}, x)) \tag{4.6}$$

With a deterministic parsing strategy, we instead try to find the most probable analysis y_j without exploring more than one decision sequence, based on the following approximation:

$$\arg\max_{y_j} P(y_j|x) \approx (d_1^*, \ldots, d_m^*) : d_i^* = \arg\max_{d_i} P(d_i|\Phi(d_1, \ldots, d_{i-1}, x)) \tag{4.7}$$

A deterministic parsing strategy is in this context a *greedy algorithm*, making a locally optimal choice in the hope that this will lead to a globally optimal solution (Cormen et al., 1990). The main problem with the greedy strategy is that it may not lead to a globally optimal solution. The main advantage is that it improves parsing efficiency by avoiding an exhaustive search of the analysis space. An additional advantage is that it reduces the effective number of parameters of the stochastic model, since only the mode of the distribution $P(d_i \mid \Phi(d_1, \ldots, d_{i-1}, x))$ needs to be estimated for each distinct condition $\Phi(d_1, \ldots, d_{i-1}, x)$. This also means that a larger class of learning methods can be used, including discriminative methods as well as methods for estimating generative and conditional probability models.

In section 3.4.3, we gave the following specification of the deterministic algorithm for dependency parsing:

PARSE($x = (w_1, \ldots, w_n)$)
1 $c \leftarrow (\epsilon, (1, \ldots, n), h_0, d_0)$
2 **while** $c = (\sigma, \tau, h, d)$ is not terminal
3 **if** $\sigma = \epsilon$
4 $c \leftarrow$ SHIFT(c)
5 **else**
6 $c \leftarrow [o(c, A_x)](c)$
7 $G \leftarrow (V_x, E_c, L_c)$
8 **return** G

This algorithm assumes the existence of an oracle function o predicting the next transition for a given nondeterministic configuration c in the correct transition sequence for the sentence x (cf. section 3.4.3). In inductive dependency parsing, we replace the oracle o with a *guide* g, which predicts the next transition for a given configuration c, based on the parser state $\Phi(c, A_x)$. This is a form of *guided parsing* (Boullier, 2003), where the parser is guided by the function g at each nondeterministic choice point, which gives us the following parsing algorithm:

GUIDED-PARSE($x = (w_1, \ldots, w_n)$)
1 $c \leftarrow (\epsilon, (1, \ldots, n), h_0, d_0)$
2 **while** $c = (\sigma, \tau, h, d)$ is not terminal
3 **if** $\sigma = \epsilon$
4 $c \leftarrow$ SHIFT(c)
5 **else**
6 $c \leftarrow [g(c, A_x)](c)$
7 $G \leftarrow (V_x, E_c, L_c)$
8 **return** G

The only difference with respect to the original algorithm is that we have replaced the oracle function $o(c, A_x)$ with a guide function $g(c, A_x)$, which means that we no longer assume that the guide always returns the correct answer. In *inductive* dependency parsing, as opposed to other forms of guided dependency parsing, we use inductive machine learning to construct the guide. Given training data from a dependency treebank, we induce a classifier g that maps every distinct parser state $\Phi(c, A_x)$ to a transition. We overload notation by using the symbol g to refer both to the *classifier*, which is a function from parser states to transitions, and for the *guide*, which is a function from parser conditions to transitions and which respects the condition that the transition returned is applicable to the configuration included in the parser condition. More precisely, the guide g returns the transition returned by the classifier g if this is a permissible transition, but returns SHIFT otherwise (since SHIFT is applicable to any non-terminal transition). Formally:

$$g(c, A_x) = \begin{cases} g(\Phi(c, A_x)) & \text{if } g(\Phi(c, A_x)) \text{ is applicable to } c \\ \text{SHIFT} & \text{otherwise} \end{cases} \tag{4.8}$$

where $g(\Phi(c, A_x))$ is the value of the classifier for the parser state $\Phi(c, A_x)$.

Given the conditional history-based model underlying our parsing strategy, the *optimal classifier* g can be characterized as follows:

$$g(\Phi(c, A_x)) = \arg\max_{t_i} P(t_i \mid \Phi(c, A_x)) \tag{4.9}$$

Since there is no guarantee that the most probable transition, given $\Phi(c, A_x)$, is also the transition required by the correct analysis of x, it may be the case that $g(\Phi(c, A_x)) \neq o(c, A_x)$, which means that the guide g defined in terms of the optimal classifier is not a true oracle even in theory. In practice, we will have to use an estimated approximation $\hat{g}$ of the optimal classifier g, which means that it may also be the case that $\hat{g}(\Phi(c, A_x)) \neq g(\Phi(c, A_x))$. Thus, finding the best possible estimate $\hat{g}$ for the optimal classifier g, given a sample T_t of treebank data with analyses A_t, is the central learning problem in this framework.

Going back to our characterization of the inductive inference scheme for history-based models, we can say that choosing a deterministic form of guided parsing as our parsing method reduces the complexity of the model M_Θ in two ways. First of all, the parameter set Θ only contains the modes of the conditional distributions $P(t_i|\Phi(c, A_x))$:

$$\Theta = \{t_i \mid \arg\max_{t_i} P(t_i|\Phi(c, A_x))\} \tag{4.10}$$

Secondly, the score $S(x, y)$ assigned to an analysis y of sentence x by the model M_Θ is binary:

$$S(x, y) = \begin{cases} 1 \text{ if } y = \textsc{Guided-Parse}(x) \\ 0 \text{ otherwise} \end{cases} \tag{4.11}$$

However, it is important to remember that the history-based model defined in section 4.1.3 is also compatible with many other parsing methods, which do not reduce the model complexity in this way.

Before we turn from parsing methods to learning methods, it is worth noting that the linear time complexity of the deterministic parsing algorithm is based on the assumption that computing the oracle function $o(c, A_x)$ is a constant-time operation. If we want to preserve this complexity for inductive dependency parsing, we therefore have to ensure that the computation of the guide function $g(c, A_x)$ can also be performed in constant time. We will return to this issue when we discuss the definition of feature functions in section 4.2.

4.1.5 Learning Methods

The learning problem that we have derived from the conditional history-based model, in combination with the deterministic parsing algorithm, consists in the induction of a classifier that can be used to construct a guide, as defined in the previous section. In terms of machine learning, this is an instance

of *function approximation* (Mitchell, 1997), where the *target function* is the optimal classifier g while the *learned function* $\hat{g}$ is an approximation of g.

There are many different learning methods that could be used to solve this problem. Since the target function g is defined in terms of a conditional probability, it may seem natural to use a probabilistic learning method. In a *generative* model we could estimate the joint probability $P(\Phi(c, A_x), t)$, for every parser state $\Phi(c, A_x)$ and transition $t \in T_R$, and then derive the required conditional probability by conditioning and marginalizing:[1]

$$\hat{P}(t \mid \Phi(c, A_x)) = \frac{\hat{P}(\Phi(c, A_x), t)}{\sum_{t_i \in T_R} \hat{P}(\Phi(c, A_x), t_i)} \tag{4.12}$$

When only the conditional probability is needed, we may be able to make more efficient use of the training data by estimating the conditional distribution $P(t \mid \Phi(c, A_x))$ directly. Thus, an early version of inductive dependency parsing was based on conditional maximum likelihood estimation (Nivre, 2004b). However, given that it is only the *mode* of the conditional distribution that is needed, i.e., the transition t that maximizes $P(t \mid \Phi(c, A_x))$, we can take this argument one step further and argue that a *discriminative* learning method might be even more efficient.

One way of relating discriminative learning to our conditional model is to say that, instead of estimating the complete conditional distribution $P(t \mid \Phi(c, A_x))$, discriminative methods try to optimize the mapping from inputs $\Phi(c, A_x)$ to outputs t by only estimating the *mode* of this distribution (Jebara, 2004). For example, memory-based learning tries to find the optimal output by extrapolating from the most similar inputs seen previously, but without explicitly estimating a conditional probability (Daelemans and Van den Bosch, 2005). Other discriminative learning methods are artificial neural networks (Bishop, 1996) and support vector machines (Vapnik, 1995).

Using a discriminative learning method means that we can formulate the learning problem as a pure classification problem, where an *input instance* is a parser state $\Phi(c, A_x)$ and an *output class* is a transition $t \in T_R$. Using a supervised learning method, our task is then to induce a classifier $\hat{g}$ given a set of training instances D_t. Ideally, we would like the training set to be a sample of the function that we want to approximate:

$$D_g = \{(\Phi(c, A_x), t) \mid g(\Phi(c, A_x)) = t\} \tag{4.13}$$

However, since the function g is not known, it is not clear how such a sample could be established. What we have instead is a training corpus T_t, from which we can obtain a sample of training instances defined in terms of the oracle function o and the parameterization function Φ. For every sentence $x = (w_1, \ldots, w_n)$, let $C^{o,x}_{0,m} = (c_0, \ldots, c_m)$ be the unique transition sequence such that $c_0 = (\epsilon, (1, \ldots, n), h_0, d_0)$ and $c_i = [o(c_{i-1}, A_x)](c_{i-1})$ (for $i > 0$). The sample of training instances for Φ given T_t is:

[1] We use the notation $\hat{P}(\cdot)$ to denote an estimate of $P(\cdot)$.

$$D_{\Phi} = \{(\Phi(c, A_x), t) \mid o(c, A_x) = t, c \in C_{0,m}^{o,x}, x \in T_t\} \tag{4.14}$$

This set can be defined in a two-step process, where we first extract a set of pairs (c, t) from the training corpus T_t:

$$D_t = \{(c, t) \mid o(c, A_x) = t, c \in C_{0,m}^{o,x}, x \in T_t\} \tag{4.15}$$

This set is independent of the parameterization function Φ and can be reused with different parameterizations to define proper training sets:

$$D_{\Phi} = \{(\Phi(c, A_x), t) \mid (c, t) \in D_t\} \tag{4.16}$$

In order to construct a specific instance of the inductive dependency parser, we therefore have to solve three independent subproblems:

1. Derive the set D_t from the training corpus T_t.
2. Define the parameterization Φ and derive the training set D_{Φ} from D_t.
3. Induce a classifier $\hat{g}$ from the training set D_{Φ} using inductive learning.

The first problem can be solved using a form of guided parsing, using an oracle defined by the gold standard dependency graph from the treebank, as we will show in the next section. The second problem will be the topic of section 4.2, where we discuss the way in which different types of features can be defined in terms of parser conditions. The third problem will be addressed in section 4.3, where we introduce memory-based learning, which is the family of learning methods that will be used in the experiments reported in chapter 5.

4.1.6 Oracle Parsing

Given a training corpus $T_t = (x_1, \ldots, x_n)$, we want to extract the set of training instances $D_t = \{(c, t) \mid o(c, A_x) = t, c \in C_{0,m}^{o,x}, x \in T_t\}$. If we let D_x be the set of instances derived from a particular sentence x, i.e., $D_x = \{(c, t) \mid o(c, A_x) = t, c \in C_{0,m}^{o,x}\}$, then we can construct D_t by taking the union of D_x for all the sentences $x \in T_t$:

$$D_t = \bigcup_{x \in T_t} D_x \tag{4.17}$$

For each sentence $x \in T_t$, let $G_g = (V_x, E_g, L_g)$ be the dependency graph assigned to x by the gold standard annotation, and let $h_g : V_x^+ \rightarrow V_x$ and $d_g : V_x^+ \rightarrow R$ be defined as follows:

1. $h_g(i) = j$ if and only if $(j, i) \in E_g$
2. $d_g(i) = r$ if and only if $\exists j : ((j, i), r) \in L_g$

We can then derive the set D_x of instances for each sentence x by the following algorithm, which is a variant of the deterministic parsing algorithm defined in section 3.4.3:

ORACLE-PARSE($x = (w_1, \ldots, w_n), h_g, d_g$)
1 $c \leftarrow (\epsilon, (1, \ldots, n), h_0, d_0)$
2 $D_x \leftarrow \emptyset$
3 **while** $c = (\sigma, \tau, h, d)$ is not terminal
4 **if** $\sigma = \epsilon$
5 $c \leftarrow$ SHIFT(c)
6 **else**
7 $t \leftarrow$ ORACLE(c, h_g, d_g)
8 $D \leftarrow D_x \cup (c, t)$
9 $c \leftarrow t(c)$
10 **return** D_x

The main difference, apart from the fact that we accumulate pairs (c, t) in the variable D_x, is that the oracle function o is replaced by a call to the function ORACLE, which predicts the next transition using the gold standard functions h_g and d_g:

ORACLE($c = (\sigma|i, j|\tau, h, d), h_g, d_g$)
1 **if** $h_g(i) = j$
2 **return** LEFT-ARC($d_g(i)$)
3 **else if** $h_g(j) = i$
4 **return** RIGHT-ARC($d_g(j)$)
5 **else if** $\exists k \in \sigma$ ($h_g(j) = k$ **or** $h_g(k) = j$)
6 **return** REDUCE
7 **else**
8 **return** SHIFT

For any configuration $c = (\sigma, \tau, h, d)$ that is passed as an argument to the function ORACLE, we know that both the stack σ and the input sequence τ are non-empty, because c is non-terminal (τ) and nondeterministic (σ). Hence, we can always assume that there is a token i on top of the stack σ and a token j at the head of the input list τ. Now, if i and j are linked by a dependency arc according to h_g, then the correct transition is LEFT-ARC(r) or RIGHT-ARC(r), with the dependency type r specified by d_g. If there is no arc between i and j, then REDUCE is the correct choice if and only if j is linked to a token to the left of i (below i in the stack σ); otherwise, the correct transition is SHIFT.

To check whether j is linked to a token to the left of i, we need to check if there is a token $k \in \sigma$ that is either the head of j ($h_g(j) = k$) or a dependent of j ($h_g(k) = j$). The first of these conditions can be checked simply by inspecting $h_g(j)$, since it holds if and only if $0 < h_g(j) < i$. The second condition may in a naive implementation require searching the entire stack σ. However, if there is a token k to the left of i such that $h_g(k) = j$, then we must already have encountered k in a previous configuration. And if we use an auxiliary stack to store tokens that have their head to the right, according to h_g, then we only need to compare the top of this stack with j.

It is worth pointing out that, whereas every transition sequence $C_{0,m} = (c_1, \ldots, c_m)$ for a sentence x defines a unique dependency graph $G_m = (V_x, E_m, L_m)$, the inverse relation is strictly speaking not a function, since there are a limited number of situations where two distinct transition sequences define the same dependency graph. This happens in configurations where the smallest arc, according to the gold standard graph G_g, that spans both the top token i and the next token j does not involve either i or j but tokens k and l, such that $k < i$ and $j < l$. In this case, both i and j must be popped from the stack before the arc connecting k and l can be added, but the order in which i and j are reduced is immaterial. In other words, either REDUCE or SHIFT is a possible transition and both of them will lead to the correct dependency graph. The algorithm defined above always prefers SHIFT in this situation, which means that j will be reduced before i. In other words, the ORACLE-PARSE algorithm constructs a canonical transition sequence for every dependency graph, consistently choosing SHIFT in cases of harmless SHIFT-REDUCE conflicts.

We conclude the discussion of oracle parsing with a correctness proof for the algorithm ORACLE-PARSE. Theorem 4.1 says that the functions h_m and d_m derived by the algorithm are identical to the input functions h_g and d_g for any sentence x and projective dependency graph $G_g = (V_x, E_g, L_g)$, which entails that the transition sequence $C_{0,m}$ used to construct the training data set D_x assigns G_g to x. As promised in section 3.4.4, this indirectly proves also theorem 3.22, since we can use ORACLE-PARSE to constructively prove that there exists a corresponding transition sequence for any projective dependency graph.

Theorem 4.1. For every sentence $x = (w_1, \ldots, w_n)$ with dependency graph $G_g = (V_x, E_g, L_g)$, if $c_m = (\sigma_m, \epsilon, h_m, d_m)$ is the terminal configuration in the computation of ORACLE-PARSE(x, h_g, d_g), then $h_g = h_m$ and $d_g = d_m$.

Proof. We need to show that $h_g(i) = h_m(i)$ and $d_g(i) = d_m(i)$ for every $i \in V_x^+$. We begin by noting that the following conditions hold for every token $i \in V_x^+$, in virtue of the transition system used by ORACLE-PARSE:

1. In the initial configuration c_0, $h_0(i) = 0$ and $d_0(i) = r_0$ (definition 3.9).
2. Before the terminal configuration c_m is reached, i must be shifted onto the stack, i.e., there exist p $(0 \leq p < m)$ and q $(p < q \leq m)$ such that $c_p = (\sigma_p, i|\tau_{p^-}, h_p, d_p)$ and $c_q = (\sigma_{q^-}|i, \tau_{p^-}, h_q, d_q)$ (definition 3.10).
3. The values of $h(i)$ and $d(i)$ can only change in a transition from a configuration c where i occurs at the head of the input list, i.e., $c = (\sigma, i|\tau, h, d)$, or on top of the stack, i.e., $c = (\sigma|i, \tau, h, d)$. In the former case, $h(i)$ and $d(i)$ are modified only by a RIGHT-ARC(r) transition; in the latter case, only by a LEFT-ARC(r) transition (definition 3.12).

There are three cases to consider, based on the value of $h_g(i)$:[2]

[2] Line numbers in this proof refer to the ORACLE algorithm defined on page 97.

1. If $h_g(i) = 0$ (and $d_g(i) = r_0$), we only need to show that $h_0(i) = h_m(i)$ and $d_0(i) = h_m(i)$, i.e., that i cannot be involved as the dependent in a LEFT-ARC(r) or RIGHT-ARC(r) transition. Given condition 3 above, this reduces to two subcases:

   a) In a configuration $c = (\sigma|i, \tau, h, d)$, LEFT-ARC($r$) is excluded because $h_g(i) = 0$ (contradicting the condition in line 1).

   b) In a configuration $c = (\sigma, i|\tau, h, d)$, RIGHT-ARC($r$) is excluded because $h_g(i) = 0$ (contradicting the condition in line 3).

   We conclude that $h_g(i) = h_0(i) = h_m(i)$ and $d_g(i) = d_0(i) = d_m(i)$.

2. If $h_g(i) \neq 0$ and $h_g(i) < i$, we need to show that there is some transition where $h(i)$ and $d(i)$ are changed from $h_0(i)$ and $d_0(i)$ to $h_g(i)$ and $d_g(i)$. (Together with Lemma 3.20, this entails that $h_g(i) = h_m(i)$ and $d_g(i) = d_m(i)$.) In virtue of condition 2 above, we know that there exists a configuration $c_p = (\sigma_p, i|\tau_{p^-}, h_p, d_p)$. There are three cases to consider for the transition out of this configuration:

   a) If $c_p = (\sigma_{p^-}|j, i|\tau_{p^-}, h_p, d_p)$, $h_g(j) = i$ and $d_g(j) = r$, ORACLE returns LEFT-ARC(r), so $c_{p+1} = (\sigma_{p^-}, i|\tau_{p^-}, h_p[j \mapsto i], d_p[j \mapsto r])$ (line 1–2).

   b) If $c_p = (\sigma_{p^-}|j, i|\tau_{p^-}, h_p, d_p)$, $h_g(i) = j$ and $d_g(i) = r$, ORACLE returns RIGHT-ARC(r), so $c_{p+1} = (\sigma_{p^-}|j|i, \tau_{p^-}, h_p[i \mapsto j], d_p[i \mapsto r])$ (line 3–4).

   c) If $c_p = (\sigma_{p^-}|j, i|\tau_{p^-}, h_p, d_p)$, $h_g(i) \neq j$, $h_g(j) \neq i$ and $h_g(i) \in \sigma_{p^-}$, ORACLE returns REDUCE, so $c_{p+1} = (\sigma_{p^-}, i|\tau_{p^-}, h_p, d_p)$ (line 5–6).

   In case (b), the goal has been reached. In cases (a) and (c), i remains at the head of the input list, while the size of the stack decreases by 1. Hence, as long as $h_g(i) \in \sigma_{p^-}$, there will eventually be a configuration $c_{q-1} = (\sigma_{q-1^-}|h_g(i), i|\tau_{p^-}, h_{q-1}, d_{q-1})$ followed by a RIGHT-ARC(r) transition, where $d_g(i) = r$. Assume $h_g(i) \notin \sigma_{p^-}$. Then $h_g(i)$ must have been popped from the stack in an earlier transition, which entails that there is a node k such that $h_g(i) < k < i$ and either $h_g(h_g(i)) = k$ (if $h_g(i)$ was popped in a LEFT-ARC(r') transition) or there is a token l such that $l < h_g(i)$ and k is linked to l (if $h_g(i)$ was popped in a REDUCE transition). But in either case this is a contradiction, since G_g is projective. Hence, we may conclude that $h_g(i) \in \sigma_{p^-}$. It follows that $h_g(i) = h_q(i) = h_m(i)$ and $d_g(i) = d_q(i) = d_m(i)$.

3. If $h_g(i) > i$, we again need to show that there is some transition where $h(i)$ and $d(i)$ are changed from $h_0(i)$ and $d_0(i)$ to $h_g(i)$ and $d_g(i)$. In virtue of condition 2 above, we know that there exists a configuration $c_q = (\sigma_{q^-}|i, \tau_{p^-}, h_q, d_q)$. Moreover, since $h_g(i) > i$, $h_q(i) = 0$ and $d_q(i) = r_0$. We know that $h_g(i) \in \tau_{p-}$ (since $h_g(i) > i$) and that $h_g(i)$ must eventually be pushed onto the stack (condition 2). We now show that this can only happen in a configuration where $h(i) = h_g(i)$ and $d(i) = d_g(i)$ (which

together with Lemma 3.20 entails that $h_g(i) = h_m(i)$ and $d_g(i) = d_m(i)$). More precisely, this follows from the following two propositions:

a) The node i can only be popped from the stack in a configuration of the form $c_r = (\sigma_{q^-}|i, j|\tau_{r^-}, h_r, d_r)$ $(r \geq q)$, where $h_g(i) = j$. Assume $h_g(i) \neq j$. Then LEFT-ARC(r) is obviously excluded (line 1). And REDUCE is excluded because this would entail that there exists some k such that $h_g(j) = k$ or $h_g(k) = j$ and $k < i < j < h_g(i)$ (line 5), which contradicts the assumption that G_g is projective.

b) The node $h_g(i)$ can only be pushed onto the stack in a configuration of the form $c_s = (\sigma_s, h_g(i)|\tau_{s^-}, h_s, d_s)$ $(s > q)$, where $i \notin \sigma_s$. Assume $i \in \sigma_s$. Then SHIFT is excluded because there exists some k (namely i) such that $k \in \sigma_s$ and $h_g(k) = h_g(i)$ (line 5). And RIGHT-ARC(r) is excluded because this would entail that there exists some k such that $k = h_g(h_g(i))$ and $i < k < h_g(i)$ (line 3), which contradicts the assumption that G_g is projective.

Hence, $h_g(i) = h_{r+1}(i) = h_m(i)$ and $d_g(i) = d_{r+1}(i) = d_m(i)$.

This concludes the proof of theorem 4.1. □

4.2 Features and Models

One of the key elements in the model of inductive dependency parsing defined in the previous section is the function Φ that defines an equivalence relation on the set of parser conditions and thereby defines what properties of a condition are relevant for the prediction of the next transition. This is reflected in the definition of the learning problem, where the set of possible input instances is simply the range of the function Φ, which we call the set of parser states.

In this section, we will discuss how Φ can be defined in terms of a set of feature functions, each of which extracts a relevant feature of the current parser condition. We will begin by defining a formal model for the specification of feature functions and move on to discuss the specific feature functions that will be used in the experiments reported in chapter 5. Finally, we will introduce the concept of a feature model, which corresponds to an instantiation of the function Φ, defined by a specific set of feature functions.

The formalization of feature functions is necessary for the implementation of feature models in the MaltParser system, described briefly in section 4.4, but it is not essential for the experimental study of memory-based inductive dependency parsing reported in chapter 5, where only a subset of the definable features will be used. Readers who are not interested in the formal aspects of feature functions can therefore skip most of the technical discussion in section 4.2.1 without missing anything that will be important later on.

4.2.1 Feature Functions

The role of the function Φ in our model is to determine which properties of a parser condition are relevant for the prediction of the next transition. In general, Φ can be defined by a set of simpler functions ϕ_i, which we call *feature functions*. Although the order of these functions is normally irrelevant, we will assume that they are ordered in a sequence $\Phi_{1,p} = (\phi_1, \ldots, \phi_p)$, which will save us the trouble of introducing a special name for each feature function ϕ_i, since it can be identified by its position in the sequence.

If $\Phi_{1,p} = (\phi_1, \ldots, \phi_p)$, then each function ϕ_i corresponds to a *feature*, or *attribute*, of a parser condition (c, A_x). Applying Φ to (c, A_x) is equivalent to applying each feature function ϕ_i to (c, A_x) in turn, which means that $\Phi(c, A_x) = (v_1, \ldots, v_p)$ if and only if $\phi_i(c, A_x) = v_i$ for every $\phi_i \in \Phi$. In this way, the value of $\Phi(c, A_x)$ corresponds to the standard representation of an *instance* as a sequence of *features*, often called a *feature vector*, which is widely used in machine learning (Mitchell, 1997).

Recall from section 3.3.1 that every function f in the set A_x of annotation functions for a sentence $x = (w_1, \ldots, w_n)$ is a function from the set of token nodes $V^+ = \{1, \ldots, n\}$ to some set of values V_f, where V_w is the set of possible word forms and V_p is the set of permissible part-of-speech categories, etc. Using A_f to denote the set of possible annotation functions, the notion of a feature function can be characterized as follows:

Definition 4.2. Given a set of configurations C, a set of annotation functions A_f, and a set of values V_ϕ, a *feature function* is a function $\phi : (C \times 2^{A_f}) \to V_\phi$.

We can then define parameterization functions in terms of feature functions:

Definition 4.3. Given a sequence of feature functions $\Phi_{1,p} = (\phi_1, \ldots, \phi_p)$, the corresponding *parameterization function* is the function $\Phi : (C \times 2^{A_f}) \to (V_{\phi_1} \times \cdots \times V_{\phi_p})$ such that $\Phi(c, A_x) = (v_1, \ldots, v_p)$ if and only if $\phi_i(c, A_x) = v_i$ (for $1 \leq i \leq p$, $c \in C$ and $A_x \in 2^{A_f}$).

The number of distinct parser states $\Phi(c, A_x)$ can now be defined as $|V_\Phi| = |V_{\phi_1}| \cdot \ \ldots \ \cdot |V_{\phi_p}|$, and the total number of model parameters in M_Θ is $|\Theta| = |T_R| \cdot |V_\Phi|$ for the general model and $|\Theta| = |V_\Phi|$ for the reduced model with a deterministic parsing strategy (cf. section 4.1.4). But even for the general model, it is normally the case that $|\Theta|$ is $O(|V_\Phi|)$.

The definition of a feature function is very general and compatible with many different ways of specifying such functions. In this study, we will restrict our attention to feature functions ϕ that can be defined in terms of the the composition of two simpler functions a_ϕ and f_ϕ as follows:

$$\phi(c, A_x) = v \ \Leftrightarrow \ [f_\phi \circ a_\phi](c) = v \tag{4.18}$$

where a_ϕ and f_ϕ satisfy the following conditions:

$$a_\phi : C \to V^+ \tag{4.19}$$

$$f_\phi \in A_x \cup \{d_c\} \tag{4.20}$$

The basic idea is that a_ϕ is an *address function*, mapping the configuration c to a specific token $i \in V^+$, and that f_ϕ is an *attribute function*, picking out a specific attribute v of i. This attribute may be given by one of the annotation functions in A_x or by the dependency type function d_c. Note that the function d_c is parameterized for the current configuration c, since this function is updated dynamically from one configuration to the next, whereas the annotation functions in A_x remain constant during the analysis of a given sentence x.

The reason for restricting the class of feature functions in this way is twofold. First, we want to ensure that feature functions can be computed efficiently, so that overall parsing efficiency is not compromised. Computing the value of each feature function must be done for every new parser condition, both in the construction of training instances during the training phase (cf. section 4.1.5) and in the construction of instances for the classifier during the parsing phase (cf. section 4.1.4). Secondly, we want to define a formal specification language for feature functions, so that implementations of inductive dependency parsing do not need to rely on hard-coded feature functions but can allow users to specify arbitrary feature functions within the space of permissible functions. The MaltParser system described in section 4.4 implements this functionality.

In the remainder of this section we will discuss the formal specification of feature functions, in particular the specification of the address function a_ϕ. Part of this discussion will be rather technical, but we will try to illustrate all the formal definitions with concrete examples. We will use a configuration from the transition sequence in figure 3.5 as our running example, more precisely the configuration c_{14}, resulting from a REDUCE transition. Figure 4.1 shows the relevant properties of this configuration, together with the annotation functions w_x and p_x for the sentence in question (cf. figure 3.3).

First of all, we define functions that extract a token from the stack σ_c or the input sequence τ_c of the current configuration.

Definition 4.4. For every configuration $c = (\sigma_c, \tau_c, h_c, d_c)$ and $i \geq 0$:

1. $\sigma_i(c) = \sigma_c[i]$
2. $\tau_i(c) = \tau_c[i]$

where $x[i]$ returns the ith element of the list x (starting from 0).

Note that σ_i and τ_i (for $i \geq 0$) are partial functions, which are undefined if the length of the relevant list is less than or equal to i. For example, the function σ_0, when applied to a configuration c, returns the top token (if any), while the function τ_1 returns the token following the next token in the input sequence τ_c (if τ_c has length two or more). For our example in figure 4.1, $\sigma_0(c_{14}) = 5$, while $\tau_1(c_{14}) = \bot$ (because τ_{14} has length one).

Given the basic functions σ_i and τ_i, we can construct complex address functions by composition with functions that map tokens to tokens according to their relations in the dependency graph, as defined by the function h_c in

$$
\begin{aligned}
c_{14} &= ((3,5), (9), h_7, d_7) \\
\sigma_{14} &= (3,5) \\
\tau_{14} &= (9)
\end{aligned}
$$

$h_7(1) = 2$	$d_7(1) = \text{NMOD}$	$w_x(1) = \text{Economic}$	$p_x(1) = \text{JJ}$
$h_7(2) = 3$	$d_7(2) = \text{SBJ}$	$w_x(2) = \text{news}$	$p_x(2) = \text{NN}$
$h_7(3) = 0$	$d_7(3) = \text{ROOT}$	$w_x(3) = \text{had}$	$p_x(3) = \text{VBD}$
$h_7(4) = 5$	$d_7(4) = \text{NMOD}$	$w_x(4) = \text{little}$	$p_x(4) = \text{JJ}$
$h_7(5) = 3$	$d_7(5) = \text{OBJ}$	$w_x(5) = \text{effect}$	$p_x(5) = \text{NN}$
$h_7(6) = 5$	$d_7(6) = \text{NMOD}$	$w_x(6) = \text{on}$	$p_x(6) = \text{IN}$
$h_7(7) = 8$	$d_7(7) = \text{NMOD}$	$w_x(7) = \text{financial}$	$p_x(7) = \text{JJ}$
$h_7(8) = 6$	$d_7(8) = \text{PMOD}$	$w_x(8) = \text{markets}$	$p_x(8) = \text{NNS}$
$h_7(9) = 0$	$d_7(9) = \text{ROOT}$	$w_x(9) = .$	$p_x(9) = \text{PU}$

Fig. 4.1. Configuration c_{14} with functions w_x and p_x (cf. figures 3.3 and 3.5)

the current configuration. To this end we define three higher-order functions, that map an arbitrary address function to a new address function by composing it with a function returning the head, leftmost dependent or rightmost dependent of a token.

Definition 4.5. For every function $a : C \to V^+$:

1. $h(a) = h_c \circ a$
2. $l(a) = l_c \circ a$
3. $r(a) = r_c \circ a$

where $l_c(i)$ and $r_c(i)$ are partial functions returning the leftmost and rightmost dependent, respectively, of a token $i \in V^+$.

Equipped with the basic functions σ_i and τ_i and the higher-order functions h, l and r, we can now give an inductive definition of the class of address functions.

Definition 4.6. The set of *address functions* is the smallest set A satisfying the following conditions:

1. For every $i \geq 0$, $\sigma_i, \tau_i \in A$.
2. For every $a \in A$, $h(a), l(a), r(a) \in A$.

It is worth pointing out again that all address functions are partial and fail to return a token as soon as one of the underlying functions (σ_i, τ_i, h_c, l_c or r_c) is undefined.

In order to exemplify the use of complex address functions, we consider the functions $h(\sigma_0)$ and $r(r(h(\sigma_0)))$, which return the head of the top token and the rightmost dependent of the rightmost dependent of the head of the top token, respectively. For the example in figure 4.1, these functions return the

Function	Description
σ_0	The top token
$\sigma_n (n > 0)$	The nth stack token (not counting the top token)
τ_0	The next token
$\tau_n (n > 0)$	The nth input token (not counting the next token)
$h(\sigma_0)$	The head of the top token
$l(\sigma_0)$	The leftmost dependent of the top token
$r(\sigma_0)$	The rightmost dependent of the top token
$l(\tau_0)$	The leftmost dependent of the next token

Fig. 4.2. Commonly used address functions

tokens 3 and 6, respectively, since $h(\sigma_0)(c_{14}) = h_7(\sigma_0(c_{14})) = h_7(5) = 3$ and $r(r(h(\sigma_0)))(c_{14}) = r_7(r_7(h_7(\sigma_0(c_{14})))) = r_7(r_7(h_7(5))) = r_7(r_7(3)) = r_7(5) = 6$.

Although the framework allows address specifications of almost arbitrary complexity, most of the features considered in this study will be based on a relatively small number of functions, in combination with different attribute functions. The most commonly used address functions are listed with explanations in figure 4.2. The list does not include the functions $h(\tau_0)$ and $r(\tau_0)$, since the parsing algorithm precludes the possibility of the next token having a head (other than 0) or a right dependent in the current configuration.

Having considered the construction of address functions at some length, we are now in a position to define a set of feature functions, using higher-order functions that map an address function a to a new function ϕ from parser conditions (c, A_x) to values v of an attribute function f, such that $\phi(c, A_x) = f(a(c))$. Formally:

Definition 4.7. If a is an address function, then for any configuration c and set of annotation functions A_x:

1. $f(a)(c, A_x) = f_x(a(c))$ for every $f_x \in A_x$
2. $d(a)(c, A_x) = d_c(a(c))$

The attribute function is either one of the annotation functions $f_x \in A_x$ (including the token function w_x) or the function d_c belonging to the current configuration c. In the former case, we have a *static feature*, since the value of the attribute function $f_x(i)$, for a given token i, remains constant during the parsing of a sentence x. In the latter case, we have a *dynamic feature*, because $d_c(i)$ will change dynamically between the different configurations of a transition sequence. Static features will be discussed further in section 4.2.2 below, while dynamic features are treated in section 4.2.3.

Finally, a short note on the implementation of feature functions. As noted in section 4.1.4, the linear time complexity of the inductive parsing algorithm

is dependent on the assumption that the guide function $g(c, A_x)$ can be computed in constant time. A naive implementation of the functions l_c and r_c, using only the function h_c in the current configuration c, would require an exhaustive search of the set of input tokens to find the leftmost or rightmost dependent. However, this can easily be avoided by adding an explicit representation of the functions l_c and r_c. These functions can be updated in constant time for any transition t as follows:

- If $t = \textsc{Left-Arc}(r)$ and $c = (\sigma|i, j|\tau, h, d)$ then $l_c(j) \leftarrow i$.
- If $t = \textsc{Right-Arc}(r)$ and $c = (\sigma|i, j|\tau, h, d)$ then $r_c(i) \leftarrow j$.
- If $t = \textsc{Reduce}$ or $t = \textsc{Shift}$ then no update is needed.

Given these functions, any component function of a complex address function can be computed in constant time. We can therefore conclude that an address function constructed from k component functions can be computed in time which is $O(k)$ regardless of the length of the input sentence. Moreover, since the application of an attribute function to the value returned by the address function is a constant time operation, it is clear that the time required to compute a feature function is constant in the length of the input.

4.2.2 Static Features

A static feature function has the form $f(a)$, where a is an address function and f refers to one of the annotation functions in A_x. In other words, static features are based on information available as input, which remains constant throughout the parsing process. On the other hand, since the address defined by a is relative and not absolute, the actual value of a static feature function will of course vary in the course of a transition sequence.

One important class of static features are those with the attribute function w_x, which we call *lexical features*, since they are defined in terms of the actual word form w_i of a token i, where $w_x(i) = w_i$. As discussed in section 2.3.2, the importance of lexical features for disambiguation has been a dominant theme in research on natural language parsing over the last ten to fifteen years. And despite studies such as Gildea (2001), Dubey and Keller (2003), Klein and Manning (2003) and Bikel (2004), which can be taken to show that the significance of lexicalization has been overstated, it remains a fact that all state-of-the-art systems for robust disambiguation make use of lexical information in some way. The benefit of using lexical features in the inductive dependency parsing was demonstrated in Nivre et al. (2004) and is further investigated in the experiments in chapter 5.

In the previous section, we introduced the most commonly occurring address functions. In a similar fashion, figure 4.3 introduces the lexical features that will be used in the experiments later on. The most central features are $w(\sigma_0)$ and $w(\tau_0)$, which extract the word form of the top token and the next token, respectively. But we will also make use of lexical features for lookahead

Function	Description
$w(\sigma_0)$	Word form of the top token
$w(\tau_0)$	Word form of the next token
$w(\tau_n)(n > 0)$	Word form of the nth input token
$w(h(\sigma_0))$	Word form of the head of the top token

Fig. 4.3. Lexical features

tokens, i.e., tokens occurring n positions after the next token, denoted by $w(\tau_n)$, and for the head of the top token, symbolized by $w(h(\sigma_0))$.

Returning to our example configuration in figure 4.1, we get the following values for some of the features defined in figure 4.3:

$$\begin{aligned} w(\sigma_0)(c_{14}, A_x) &= w_x(5) = \text{effect} \\ w(\tau_0)(c_{14}, A_x) &= w_x(9) = . \\ w(\tau_1)(c_{14}, A_x) &= w_x(\perp) = \perp \\ w(h(\sigma_0))(c_{14}, A_x) &= w_x(3) = \text{had} \end{aligned} \tag{4.21}$$

Besides lexical features, static features can be defined in terms of any kind of annotation introduced as a result of preprocessing and encoded in a function $f : V^+ \rightarrow V_f$ included in A_x. The only kind of preprocessing that will be used in our experiments is part-of-speech tagging, which means that the only annotation function that will be used in features is the function p_x that maps each token to its part-of-speech (as defined by the part-of-speech tagger applied in preprocessing). We call these features *part-of-speech features*.

If the role of lexicalization in syntactic parsing has recently been the matter of some debate, the role of part-of-speech tagging is even more of a moot point. In early work on treebank parsing it was more or less standard practice to have a separate tagging phase prior to parsing proper (Charniak, 1996), but with the emergence of lexicalized models it was found that better parsing accuracy could often be obtained if the part-of-speech analysis was integrated in the parsing process (Charniak, 1997a; Collins, 1997). More recently, it has been argued that the main reason for using parts-of-speech in data-driven parsing is that they provide a back-off model for lexical features and thereby counteract the sparse data problem (Charniak, 2000; Van den Bosch and Buchholz, 2002).

In a study of memory-based shallow parsing, Van den Bosch and Buchholz (2002) showed that a model incorporating words but no parts-of-speech, while inferior with small training data sets, outperforms a model involving parts-of-speech but no words for training sets over a certain size (which in their experiments was around 50 000 sentences). However, it was still the case that a model incorporating *both* words and parts-of-speech gave the best overall

Function	Description
$p(\sigma_0)$	Part-of-speech of the top token
$p(\sigma_n)(n > 0)$	Part-of-speech of the nth stack token
$p(\tau_0)$	Part-of-speech of the next token
$p(\tau_n)(n > 0)$	Part-of-speech of the nth input token

Fig. 4.4. Part-of-speech features

performance, which indicates that the smoothing effect obtained by including parts-of-speech is beneficial also with large training sets.

In the experiments reported in the next chapter we make use of part-of-speech features for the two target tokens, i.e., the top token and the next token, as well as neighboring tokens both on the stack and in the sequence of remaining input tokens. We use the term *stack tokens* to refer to tokens that occur below the top token on the stack and the term *lookahead tokens* to refer to tokens that occur after the next token in the input sequence. Figure 4.4 shows the notational conventions that will be used to refer to part-of-speech features. By way of example, here are the values of a sample of part-of-speech features for the configuration in figure 4.1:

$$\begin{aligned} p(\sigma_0)(c_{14}, A_x) &= p_x(5) = \text{NN} \\ p(\sigma_1)(c_{14}, A_x) &= p_x(3) = \text{VBD} \\ p(\tau_0)(c_{14}, A_x) &= p_x(9) = \text{PU} \\ p(\tau_1)(c_{14}, A_x) &= p_x(\bot) = \bot \end{aligned} \tag{4.22}$$

4.2.3 Dynamic Features

A dynamic feature function has the form $d(a)$, where a is an address function and d denotes the dependency type function d_c that belong to the current parser configuration c and that is updated dynamically during the parsing process. One of the differences between the parsing methods investigated in this book and many other approaches to dependency parsing is that the parser produces labeled dependency graphs directly, rather than first producing an unlabeled dependency graph and then assigning labels to dependency arcs. This fact can be exploited when defining relevant feature functions, since the labels of previously added arcs are available in the state of the function d_c. We call these features *dependency type features*, or *dependency features* for short.

Figure 4.5 introduces the dependency features that will be used in our experiments. There are three features defined in relation to the top token,

Function	Description
$d(\sigma_0)$	Dependency type of the top token
$d(l(\sigma_0))$	Dependency type of the leftmost dependent of the top token
$d(r(\sigma_0))$	Dependency type of the rightmost dependent of the top token
$d(l(\tau_0))$	Dependency type of the leftmost dependent of the next token

Fig. 4.5. Dependency features

extracting the dependency types relating this token to its head ($d(\sigma_0)$), its leftmost dependent ($d(l(\sigma_0))$) and its rightmost dependent ($d(r(\sigma_0))$). In addition, we consider the leftmost dependent of the next input token ($d(l(\tau_0))$). We exemplify these dependency features by applying them to the configuration in figure 4.1:

$$\begin{aligned} d(\sigma_0)(c_{14}, A_x) &= d_7(5) = \text{OBJ} \\ d(l(\sigma_0))(c_{14}, A_x) &= d_7(4) = \text{NMOD} \\ d(r(\sigma_0))(c_{14}, A_x) &= d_7(6) = \text{NMOD} \\ d(l(\tau_0))(c_{14}, A_x) &= d_7(\bot) = \bot \end{aligned} \tag{4.23}$$

While dependency features are the only dynamic features used in this study, it would also be possible to define features based on the function h_c that records the index of a token's head. Although the exact numerical index is unlikely to be a useful feature, the comparison of features could be used to define *distance-based features*, which have been used with some success in other data-driven approaches to syntactic parsing (Collins, 1999), although these functions are seldom based on a purely quantitative notion of distance. Moreover, features that compare the relative position of two tokens would require a more complex definition of feature functions, and we will therefore leave this as a possible topic for future research.

4.2.4 Feature Models

In this section, we have shown how the parameterization function Φ can be defined by a sequence of feature functions $\Phi_{1,p} = (\phi_1, \ldots, \phi_p)$, where each feature function has the form $f(a)$ for some address function a and attribute function $f \in \{w, p, d\}$. Since each of the functions ϕ_i defines a feature of the current parser parser condition, we will say that the complex function Φ defines a *feature model*.

One of the questions posed in the experiments reported in chapter 5 is how different features influence the performance of an inductive dependency parser, with respect to accuracy as well as efficiency. We will address this question by a series of experiments, where we vary the feature model Φ while

keeping other things constant. In this context, it is often convenient to be able to define a complex model in terms of two or more simpler models. For this purpose, we define the *concatenation* of two models in the obvious way:

Definition 4.8. Let Φ_1 and Φ_2 be two parameterization functions (or feature models), defined by two sequences of feature functions $\Phi^1_{1,p} = (\phi^1_1, \ldots, \phi^1_p)$ and $\Phi^2_{1,q} = (\phi^2_1, \ldots, \phi^2_q)$. The *concatenation* of Φ_1 and Φ_2, denoted $\Phi_1 + \Phi_2$, is the function Φ defined by $\Phi_{1,p+q} = (\phi^1_1, \ldots, \phi^1_p, \phi^2_1, \ldots, \phi^2_q)$.

The models that will be examined in chapter 5 can be seen as concatenations of three types of models, based on the three types of features discussed in this section:

1. Part-of-speech models
2. Dependency models
3. Lexical models

Before we conclude the discussion of features and models in this chapter, we will define the three types of models and introduce the notational conventions that will be used to designate these models in the experiments in chapter 5.

Part-of-speech models will be designated Φ^p_{ij} $(i, j \geq 0)$. All part-of-speech models include the features $p(\sigma_0)$ and $p(\tau_0)$. The parameter i specifies how many successive stack tokens will be included in addition to the top token. That is, every feature $p(\sigma_n)$, for $n \leq i$, is included. In a similar fashion, the parameter j specifies how many lookahead tokens will be included over and above the next input token. Thus, every feature $p(\tau_n)$, for $n \leq j$, is included. We illustrate this class of models by applying the model Φ^p_{01} to the configuration in figure 4.1:

$$\begin{aligned} \Phi^p_{01}(c_{14}, A_x) &= (p(\sigma_0)(c_{14}, A_x), p(\tau_0)(c_{14}, A_x), p(\tau_1)(c_{14}, A_x)) \\ &= (\mathrm{NN}, \mathrm{PU}, \bot) \end{aligned} \tag{4.24}$$

Dependency models will be designated Φ^d_{ijk} $(i, j, k \in \{0, 1\})$. All dependency models include the feature $d(\sigma_0)$. In addition, it may include some or all of the features $d(l(\sigma_0))$, $d(r(\sigma_0))$ and $d(l(\tau_0))$, and the indices i, j and k are basically boolean variables indicating the presence or absence of these features (in the order just listed). We illustrate this class of models by applying the model Φ^d_{011} to the configuration in figure 4.1:

$$\begin{aligned} \Phi^d_{011}(c_{14}, A_x) &= (d(\sigma_0)(c_{14}, A_x), d(r(\sigma_0))(c_{14}, A_x), d(l(\tau_0))(c_{14}, A_x)) \\ &= (\mathrm{OBJ}, \mathrm{NMOD}, \bot) \end{aligned} \tag{4.25}$$

Lexical models, finally, will be designated Φ^w_{ij} $(i, j \geq 0)$. The parameter i specifies how many lexical features are extracted from the stack, starting with the top token $(i \geq 1)$ and possibly including the head of the top token $(i = 2)$. The parameter j specifies how many successive input tokens will be included, starting with the next input token $(i \geq 1)$ and possibly adding an extra

lookahead token ($j = 2$). We illustrate this class of models by applying the model Φ^w_{11} to the configuration in figure 4.1:

$$\begin{aligned} \Phi^w_{11}(c_{14}, A_x) &= (w(\sigma_0)(c_{14}, A_x), w(\tau_0)(c_{14}, A_x)) \\ &= (\text{effect}, .) \end{aligned} \tag{4.26}$$

4.3 Memory-Based Learning

In the deterministic version of inductive dependency parsing investigated in this book, the central learning problem is to induce a mapping from parser states to parser transitions. This problem can be solved using *memory-based learning*, a discriminative machine learning method that has been successfully applied to a wide range of problems in natural language processing (Daelemans and Van den Bosch, 2005). Although the general approach of inductive dependency parsing is not directly committed to any particular method for inductive learning, memory-based learning seems well suited for the task, with a local approximation of the target function that is potentially sensitive to subregularities and exceptional instances (Daelemans et al., 2002).

In this section we introduce the basic concepts of memory-based learning and discuss the different algorithms and parameters that can be used in the implementation of this approach. For the experiments reported in chapter 5 we rely on the software package TiMBL (Tilburg Memory-Based Learner) developed by Walter Daelemans, Antal van den Bosch and their colleagues at Tilburg University and the University of Antwerp (Daelemans et al., 2004), and our presentation of memory-based learning is deeply influenced by their work, which is presented comprehensively in Daelemans and Van den Bosch (2005). We close the section by relating our use of memory-based learning in dependency parsing to previous work on memory-based language processing, in particular memory-based parsing.

4.3.1 Memory-Based Learning and Classification

Memory-based learning and problem solving is based on two fundamental principles: learning is the simple storage of experiences in memory, and solving a new problem is achieved by reusing solutions from similar previously solved problems (Daelemans and Van den Bosch, 2005). It is inspired by the *nearest neighbor* approach in statistical pattern recognition and artificial intelligence (Fix and Hodges, 1952), as well as the *analogical modeling* approach in linguistics (Skousen, 1989, 1992). In machine learning terms, it can be characterized as a *lazy* learning method, since it defers processing of input until needed and processes input by combining stored data (Aha, 1997).[3]

[3] Memory-based learning is also known as instance-based learning, exemplar-based learning and case-based learning.

In contrast to *eager* learning methods, such as the family of generative probabilistic methods that are used in many data-driven parsers, memory-based learning performs generalization without abstraction. In addition, it uses similarity-based reasoning as an implicit smoothing method to deal with low-frequency events (Daelemans and Van den Bosch, 2005). Both of these properties make the method potentially well suited for problems in natural language processing, which are often characterized by distributions containing a long tail of low-frequency events, where it is notoriously difficult to distinguish noise from significant exceptions (Daelemans et al., 2002).

Conceptually, memory-based learning algorithms can be seen as variants of the k-nearest neighbor algorithm (k-NN) (Cover and Hart, 1967; Devijver and Kittler, 1982; Aha et al., 1991). Given the task of inducing a classifier $\hat{g} : S \rightarrow T$ from a set of training instances $D_t = \{(s_1, t_1), \ldots (s_n, t_n)\}$, where $s_i \in S$ is an input instance and $t_i \in T$ is its class, this type of algorithm can be described as follows:

- Learning consists in storing the set D_t of training instances in memory.
- Classifying a new instance r is performed in two steps:
  1. Compare r to every stored input instance s_i ($(s_i, t_i) \in D_t$):
     a) Compute the distance $\Delta(r, s_i)$ between r and s_i.
     b) Update the set of k closest instances (nearest neighbors).
  2. Take the majority class t of the k nearest neighbors as the class of r.

Even though the basic memory-based strategy remains the same, there are many parameters that can be varied to modify the resulting classifier. The most obvious parameter is perhaps the value of k, which can be varied from 1 to n (where n is the number of training instances in D_t). A small k value leads to a very local approximation of the function g, which is more sensitive to local exceptions and subregularities but also less robust when faced with noisy data. A large k value gives a more global approximation, which is less sensitive to local variations, whether due to noise or to significant exceptions. For the task of predicting the next parser transitions, a k value of about 5 has turned out to be optimal for many feature models (Nivre et al., 2004), although this is something that is investigated further in the experiments reported in chapter 5.

Another important parameter is the distance metric Δ, which can be varied in many different ways. For example, *feature weighting* can be used to give different weights to features in the representation of instances; *value weighting* can be used to differentiate penalties for mismatches between feature values; and *exemplar weighting* can be used to weight stored instances differently. Exemplar weighting will not be exploited in the investigations in this book, but both feature weighting and value weighting will be discussed in section 4.3.2 below and studied experimentally in chapter 5. A third kind of parameter is the voting procedure, where the main alternative to a simple majority vote is to use a weighting scheme that gives more weight to closer instances, using so-called *distance-weighted class voting* (Dudani, 1976).

In the following section we will discuss the parameters of memory-based learning that are relevant to our study of inductive dependency parsing and to the experiments reported in chapter 5. We will focus on the way that these parameters are implemented in TiMBL, since this is the software that is used in all our experiments. On the other hand, we will only deal with a small subset of all the features that are available in this software package. For more information about TiMBL, see the TiMBL Reference Guide (Daelemans et al., 2004); see also Daelemans and Van den Bosch (2005).

4.3.2 Learning Algorithm Parameters

Let $D_t = \{(s_1, t_1), \ldots, (s_n, t_n)\}$ be the set of training instances, where each input instance is represented as a vector of feature values $s_i = (s_1^i, \ldots, s_p^i)$ and each output t_i is a class taken from some set T. We recall that in the case of inductive dependency parsing, an input instance is a parser state $\Phi(c, A_x) = (\phi_1(c, A_x), \ldots, \phi_p(c, A_x)) = (v_1, \ldots, v_p)$ while the output class is a transition $t \in T_R$. We will assume that all features are symbolic, i.e., that their values are not numeric, a restriction that holds for all the features considered in this book, where feature values are word forms, parts-of-speech or dependency types. However, memory-based learning as such is not restricted to symbolic features.

We will begin by discussing the implementation of the k-NN algorithm in TiMBL, which differs in two ways from the standard formulation (Aha et al., 1991). First of all, the TiMBL version of k-NN considers the k nearest *distances*, rather than the k nearest instances. Since the training set may contain several instances at the same distance from a given instance, the number m of instances included by TiMBL may therefore be greater than k.

The second difference concerns the method used for tie-breaking, i.e., for deciding which class to choose in case there is no majority class in the nearest neighbor set. The default method in TiMBL, which will be used in all the experiments in chapter 5, is to use a three-step procedure:

1. Increase the value of k by 1 and choose the majority class in the larger neighbor set, if such a class exists.
2. Otherwise, choose the majority class in the entire training set D_t, if such a class exists.
3. Otherwise, choose the class t_1 of the first instance (s_1, t_1) encountered in the training set D_t.

The next parameter to discuss is the choice of the distance metric Δ. When dealing with symbolic features, the most straightforward metric is the Overlap metric, also referred to as Hamming distance, Manhattan metric, city-block distance, and L1 metric (Daelemans and Van den Bosch, 2005). The distance between two input instances $r = (r_1, \ldots, r_p)$ and $s = (s_1, \ldots, s_p)$ according to this metric is simply the number of mismatching features. Formally:

$$\Delta(r,s) = \sum_{i=1}^{p} \delta(r_i, s_i) \tag{4.27}$$

The δ function in this definition is a 0-1 mismatch function:

$$\delta(r_i, s_i) = \begin{cases} 0 \text{ if } r_i = s_i \\ 1 \text{ if } r_i \neq s_i \end{cases} \tag{4.28}$$

More sophisticated distance metrics can usually be understood as variations on the Overlap metric. One common variation is to associate a weight w_i with each feature ϕ_i and calculate the distance as a sum of weighted mismatches:

$$\Delta(r,s) = \sum_{i=1}^{p} w_i \, \delta(r_i, s_i) \tag{4.29}$$

Although it is possible to assign weights to features manually, based on some kind of *a priori* knowledge, it is much more common to derive weights automatically from training data using information-theoretic concepts which are also used in decision tree learning. Thus, Information Gain (IG) weighting considers the average amount of information about the correct class label contributed by each feature:

$$w_i = H(T) - \sum_{v \in V_i} P(v) H(T|v) \tag{4.30}$$

In this equation, T is the set of class labels, V_i is the set of values for feature ϕ_i, and $H(T)$ and $H(T|v)$ is the entropy of the class labels, *a priori* and conditioned on the value v, respectively:

$$H(T) = -\sum_{t \in T} P(t) \log_2 P(t) \tag{4.31}$$

$$H(T|v) = -\sum_{t \in T} P(t,v) \log_2 P(t|v) \tag{4.32}$$

One problem with IG weighting is that it tends to overestimate the relevance of features with large value sets. Quinlan (1986) has therefore introduced a normalized version, called Gain Ratio (GR), where IG is divided by the entropy of the value set:

$$w_i = \frac{H(T) - \sum_{v \in V_i} P(v) H(T|v)}{H(V_i)} \tag{4.33}$$

Although GR weighting still has a bias towards features with large value sets, it often gives good performance in practice. Other weighting schemes, which attempt to correct the bias of IG and GR, have been proposed based on the χ^2 statistic (White and Liu, 1994).

Another way of modifying the Overlap metric is to use a more sophisticated mismatch function, which differentiates the penalty of a mismatch based on the similarity of the feature values involved. This is the idea behind the (Modified) Value Difference Metric (MVDM), proposed by Stanfill and Waltz (1986) and refined by Cost and Salzberg (1993), which quantifies the distance between two feature values v_j and v_k belonging to the same value set V_i by considering their cooccurrence with target classes:

$$\delta(v_j, v_k) = \sum_{t \in T} |P(t|v_j) - P(t|v_k)| \tag{4.34}$$

Although MVDM introduces a kind of *value* weighting, rather than feature weighting, it will also have an indirect feature weighting effect, since $\delta(v_j, v_k)$ will on average be larger for informative features that have a more skewed conditional class distributions than for less informative features with more uniform distributions (Daelemans and Van den Bosch, 2005). One problem with MVDM is that it is sensitive to sparse data. TiMBL therefore offers the possibility of setting a frequency threshold l, so that MVDM is applied only if both of the values compared occur at least l times in the training data; otherwise the 0-1 mismatch function is used instead.

All of the modifications to the distance metric considered so far have the potential drawback that they increase the complexity of distance computations and thereby compromise the efficiency of classification. However, in the TiMBL implementation both feature weights w_i (for $1 \leq i \leq p$) and value distances $\delta(v_j, v_k)$ (for $v_j, v_k \in V_i$) can be computed and stored at learning time, which means that only table lookup is required at classification time.

In addition to feature weighting and value weighting, a weighting scheme can also be applied in the voting procedure that determines the majority class. Dudani (1976) proposed a voting rule in which the vote of each instance s_i is weighted by a function w_i of its distance to the new instance r:

$$w_i = \begin{cases} \frac{d_k - d_i}{d_k - d_1} & \text{if } d_k \neq d_1 \\ 1 & \text{if } d_k = d_1 \end{cases} \tag{4.35}$$

In this equation, d_i is the distance of s_i to r, d_1 is the distance of the nearest neighbor, and d_k is the distance of the most distant instance in the neighbor set. In addition to this inverse-linear (IL) weighting scheme, Dudani (1976) proposed the inverse distance (ID) weight:

$$w_i = \frac{1}{d_i} \text{ if } d_i \neq 0 \tag{4.36}$$

In order to make the weighting applicable also to neighbors with zero distance, it is customary to add a small constant ϵ to the denominator (Wettschereck, 1994):

$$w_i = \frac{1}{d_i + \epsilon} \tag{4.37}$$

In chapter 5 we will investigate how the different parameters of memory-based learning affect the performance of inductive dependency parsing. The parameters that will be varied are the following:

1. Number of nearest distances: k
2. Distance metric: Overlap or MVDM (with frequency threshold l)
3. Feature weighting: IG, GR, or none
4. Distance-weighted class voting: IL, ID, or none

In experiments where other parameters are varied, such as feature models or training sets, we will usually keep the learning algorithm parameters constant. However, rather than using the default values of the TiMBL system, which usually gives suboptimal performance, we will use the following settings, which have been shown to give good performance in previous experiments (Nivre et al., 2004; Nivre and Scholz, 2004):

1. Number of nearest distances: $k = 5$
2. Distance metric: MVDM with $l = 3$
3. Feature weighting: None
4. Distance-weighted class voting: ID

Finally, a remark on the efficiency of memory-based learning and classification. Given the lazy learning approach, training a memory-based classifier is usually very efficient, given that this basically consists in storing instances in memory, in a so-called instance base, and precomputing metrics such as feature weights and value distances for MVDM. By contrast, classification is less efficient, with a worst-case complexity of $O(n)$, where n is the number of instances in the instance base. The TiMBL software package implements a tree-based indexing scheme that speeds up classification in practice, although the worst-case complexity remains the same. It also offers the possibility of compressing the instance base, e.g., by constructing a decision tree based on feature weights. This decision tree yields an approximation of the exhaustive k-NN search, which improves efficiency but usually has a negative effect on classification accuracy. TiMBL also offers several hybrid solutions, that exploit the trade-off between efficiency and accuracy in different ways. These alternative algorithms will not be explored in this book, mainly because previous experiments have shown that classification performance degrades considerably especially for more complex feature models. We refer the reader to the TiMBL Reference Guide for more information on the tree-based indexing used by TiMBL as well as alternatives to the k-NN algorithm.

4.3.3 Memory-Based Language Processing

Memory-based learning and classification has been applied to a wide range of problems in natural language processing, exemplified in the following list (see also Daelemans and Van den Bosch, 2005):

- Hyphenation and syllabification (Daelemans and Van den Bosch, 1992)
- Assignment of word stress (Daelemans et al., 1994)
- Grapheme-to-phoneme conversion (Stanfill and Waltz, 1986; Lehnert, 1987; Weijters, 1991; Daelemans and Van den Bosch, 1996)
- Morphological analysis (Van den Bosch and Daelemans, 1999)
- Part-of-speech tagging (Cardie, 1993; Daelemans et al., 1996; Zavrel and Daelemans, 1997)
- Prepositional phrase attachment (Zavrel et al., 1997)
- Word sense disambiguation (Ng and Lee, 1996; Fujii et al., 1998; Dagan et al., 1999; Veenstra and Daelemans, 2000; Escudero et al., 2000)
- Named entity recognition (De Meulder and Daelemans, 2003; Hendrickx and Van den Bosch, 2003)
- Semantic role labeling (Van den Bosch et al., 2004; Kouchnir, 2004)
- Text categorization and filtering (Masand et al., 1992; Yang and Chute, 1994; Riloff and Lehnert, 1994)

Most of these problems have a natural formulation as a classification problem, where some kind of linguistic entity, such as a grapheme, a syllable, a word, or an entire document, is mapped to a finite set of discrete categories. This formulation of the problem makes memory-based learning a natural choice.

However, syntactic parsing is *prima facie* not a classification problem of this kind, especially not if we consider full parsing. Even though the input is a linguistic entity such as a sentence, the output usually comes from an infinite set of complex structures, such as constituency trees or dependency graphs. In order to apply the memory-based approach to syntactic parsing, it is therefore necessary to reformulate the problem so that it can be solved using discriminative learning. Broadly speaking, there are three different reformulations that have been proposed in the literature, which we may call holistic parsing, cascaded partial parsing, and history-based parsing.

The holistic approach is in a way the most straightforward application of the memory-based approach to full syntactic parsing and is based on the idea of storing complete sentences with their analyses in the instance base. Parsing a new sentence is performed by finding the most similar sentences in the instance base and adapting their analyses to the input sentence, possibly backing off to smaller fragments if necessary. This approach is most clearly exemplified in the work of Streiter (2001a,b) and Kübler (2004), but the DOP framework (Bod, 1995, 1998, 2003) is essentially based on the same idea, especially in its non-probabilistic incarnation where priority is given to analyses composed of large fragments (Bod, 2000). A variation on this theme is De Pauw (2003), who uses a memory-based model to score analyses in a parse forest derived using a grammar-driven parsing method.

The cascaded approach starts from a partial parsing or chunking analysis, which can be cast as a classification problem using the so-called BIO

representation[4] (Ramshaw and Marcus, 1995) and which has been performed successfully with memory-based methods by, among others, Veenstra (1998) and Tjong Kim Sang and Veenstra (1999). One way of extending this partial analysis to a more complete syntactic analysis is to use a cascade of partial parsers, where the input of each parser includes the output of previous parsers, in combination with methods for identifying grammatical relations holding between chunks. Memory-based approaches to cascaded partial parsing and grammatical relation finding include Argamon et al. (1998), Buchholz et al. (1999), Daelemans et al. (1999), Krymolowski and Dagan (2000), Kübler and Hinrichs (2001), Buchholz (2002) and Dagan and Krymolowski (2003).

The history-based approach is the most indirect way of performing parsing through classification, since the input instances are not linguistic entities but states of some parsing system, and the classes are not linguistic categories or structures but actions of this parsing system (cf. sections 2.3.2 and 4.1.3). In this way, Veenstra and Daelemans (2000) used memory-based learning to predict the actions of a shift-reduce parser, although this method was only tested on an artificially created corpus. Inductive dependency parsing using memory-based learning to guide a deterministic parser is exactly the same idea, although combined with a different kind of syntactic representation and a different parsing algorithm.

4.4 MaltParser

Using memory-based learning and classification to guide a deterministic parser is one instantiation of the general approach of inductive dependency parsing. In this section, we describe a system called MaltParser, which has been used to perform all the experiments on memory-based dependency parsing reported in chapter 5, but which is designed as a more general framework for inductive dependency parsing.

The system can be described as a data-driven parser-generator framework. While a traditional parser generator constructs a parser given a grammar, a data-driven parser generator constructs a parser given a treebank. However, MaltParser also takes as input the specification of a feature model, as defined in section 4.2, which means that different parsers can be induced from the same treebank without recompiling the system. Moreover, the design of the system is intended to facilitate the variation not only of feature models but also of parsing algorithms and learning methods, although these parameters will be kept constant in the experiments reported in this book.

[4] For a given phrase or chunk type, each token is tagged as **B**eginning, being **I**nside, or being **O**utside a constituent of that type.

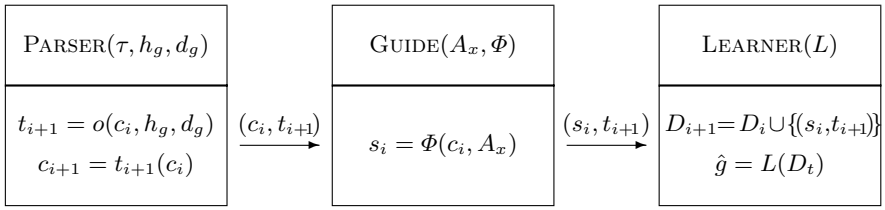


Fig. 4.6. Architecture for training

4.4.1 Architecture

In the data-driven approach to text parsing, we can usually distinguish two different phases, the *training phase* and the *parsing phase* (cf. section 4.1.2). Although these phases are different in nature, they can often be decomposed into very similar or even identical subtasks. For the framework investigated in this book, the training phase consists of two steps. The first step involves parsing every sentence x of the training corpus T_t using the oracle parsing algorithm, extracting the feature vector $\Phi(c_i, A_x)$ for every nondeterministic configuration c_i, and storing the pair $(\Phi(c_i, A_x), t_{i+1})$ in the set of training instances D_t. The second step is the induction of a classifier $\hat{g}$ from D_t using a particular learning method. The parsing phase consists in parsing every sentence x of the input text T using the inductive parsing algorithm, extracting the feature vector $\Phi(c_i, A_x)$ for every nondeterministic configuration c_i, and querying the classifier for $\hat{g}(\Phi(c_i, A_x)) = t_{i+1}$.

Comparing these two phases, we note that the extraction of feature vectors is performed in exactly the same way during training and parsing, although the vectors are used for learning in one case and for prediction in the other. Moreover, we have previously seen that the parsing algorithms used for training and parsing differ only minimally from each other. This suggests that a data-driven parsing system should be designed in such a way that the same basic components for parsing and feature extraction can be used both in the training phase and in the parsing phase. In addition, since we want to be able to vary parsing methods, feature models and learning methods independently of each other, these components should be encapsulated and separated from each other. This gives rise to an architecture with three main components (in addition to input/output modules and overall control structure):

1. Parser
2. Guide
3. Learner

In this architecture, the Parser constructs dependency graphs by applying transitions to parser configurations, the Guide extracts feature vectors from

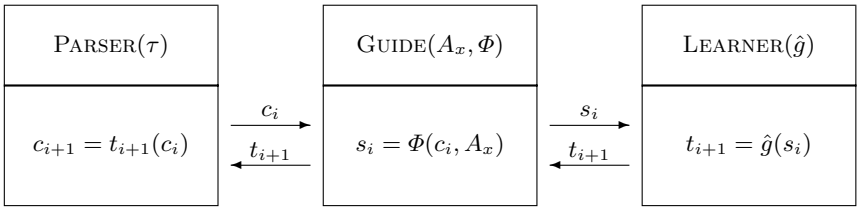


Fig. 4.7. Architecture for parsing

parser conditions and passes data between the Parser and Learner, and the Learner handles the mapping from feature vectors to transitions. In practice, the Learner will normally be an interface to a standard machine learning package such as TiMBL.

Figure 4.6 depicts the data flow in the architecture during the training phase. For a given sentence $x_i \in T_t$, the Parser takes as input the token sequence τ and the gold standard functions (h_g, d_g). If the current configuration c_i is nondeterministic, the Parser derives the correct transition t_{i+1} from (h_g, d_g) using the oracle function o, passes (c_i, t_{i+1}) to the Guide as a training instance, and derives the next configuration c_{i+1} by applying t_{i+1} to c_i. The Guide takes as input a feature model Φ (constant for all the sentences of the training corpus) and the annotation functions A_x corresponding to the current sentence x. When receiving the instance (c_i, t_{i+1}) from the Parser, the Guide uses the feature model Φ to extract the parser state s_i, represented as a feature vector, and passes the training instance (s_i, t_{i+1}) to the Learner. The Learner, parameterized by an inductive learning algorithm L, collects instances in the training set D_t and finally applies L to induce a classifier $\hat{g}$ when the entire training corpus has been parsed.

Figure 4.7 shows the data flow during the parsing phase. In this case, there is no gold standard analysis to guide the Parser, which only takes the input sequence as input. If the current configuration c_i is nondeterministic, the Parser requests a prediction of the next transition by passing c_i to the Guide. However, once the predicted transition t_{i+1} is returned by the Guide, the Parser applies t_{i+1} to the current configuration c_i in exactly the same way as during training. Furthermore, the Guide extracts the parser state $s_i = \Phi(c_i, A_x)$ in exactly the same way during parsing and training, using the model Φ defined by the current feature specification s_ϕ. The only difference is that, instead of passing an instance (s_i, t_{i+1}) to the Learner as training data, during parsing it sends the state s_i and receives the predicted transition t_{i+1}, which is then passed on to the Parser. The Learner, finally, uses the function $\hat{g}$, induced in the training phase, to map the state s_i to the transition $t_{i+1} = \hat{g}(s_i)$.

One of the advantages of this architecture is that parsing is completely separated from learning, which makes it possible to vary parsing methods

and learning methods independently. The Parser has no knowledge of the feature model Φ and behaves in exactly the same way regardless of which features are used for learning and prediction. The Learner only has to learn a mapping from feature vectors to transitions, without knowing either how the features are extracted or how the transitions are to be used. Finally, the Guide has no knowledge about either parsing algorithms or learning algorithms, but only handles the abstraction from configuration to states and passes data between the Parser and the Learner. The Learner can also encapsulate the interface to an external machine learning package, converting the feature vector constructed by the Guide to whatever special format is required by the external module. In this way, different machine learning packages can be plugged in without modifying the Guide module.

4.4.2 Implementation

The architecture presented above is realized in MaltParser (Nivre and Hall, 2005), a version of which is freely available for research and educational purposes, together with a suite of tools for data conversion and evaluation.[5] The version of MaltParser used for the experiments in this book supports the following functionality:

- **Parser:** The Parser implements the deterministic parsing algorithm in two versions: Oracle-Parse for training and Guided-Parse for parsing.
- **Guide:** The Guide accepts specifications of arbitrary feature models, but features are limited to dependency features, part-of-speech features and lexical features.
- **Learner:** The Learner supports memory-based learning via an interface to TiMBL.

The most recent version of the system extends this functionality by providing alternative parsing algorithms, notably Covington's incremental algorithms for non-projective dependency parsing (Covington, 2001), and alternative learning methods, such as support vector machines using the LIBSVM tools (Wu et al., 2004).

[5] URL: http://www.msi.vxu.se/users/nivre/research/MaltParser.html

5

Treebank Parsing

The problem of parsing unrestricted natural language text has been defined in this study as the problem of assigning to each sentence in a text its correct syntactic analysis. Conceived in this fashion, text parsing is essentially an empirical problem and the accuracy of a text parsing system can only be evaluated by comparing the analysis produced by the system to some kind of gold standard. The standard method for carrying out this kind of evaluation is to apply the system to a sample of text taken from a treebank, i.e., from a corpus where each sentence is annotated with its correct analysis. In the data-driven approach to text parsing, treebank data may also be used in the training corpus, i.e., in the sample of text on which we base our inductive inference. Using treebank data for training and evaluation is what we normally understand by the term *treebank parsing*.

This chapter presents an experimental evaluation of inductive dependency parsing based on treebank parsing. We use treebank data to train parser guides, as described in the preceding chapter, and we use treebank data to evaluate the quality of the resulting parsers, with respect to accuracy as well as efficiency. Before we turn to the evaluation, we briefly discuss treebanks and their use in research on syntactic parsing more generally, touching on some of the methodological problems that arise in using treebank data for parser evaluation. We then describe our experimental methodology, including the data sets used, the models and algorithms evaluated, and the evaluation metrics used to assess performance. The main part of the chapter is devoted to the presentation and discussion of experimental results, focusing on the influence of different kinds of parameters related to the feature model and to the learning algorithm. We conclude the chapter with a final evaluation of the best models and a comparison with related results in the literature.

5.1 Treebanks and Parsing

A treebank can be defined as a linguistically annotated corpus that includes some kind of syntactic analysis over and above part-of-speech tagging. The term *treebank* appears to have been coined by Geoffrey Leech (Sampson, 2003) and obviously alludes to the fact that the most common way of representing the syntactic analysis is by means of a tree structure. However, in current usage, the term is in no way restricted to corpora annotated with tree-shaped representations, but applies to all kinds of syntactically analyzed corpora (Abeillé, 2003a; Nivre, forthcoming).

Treebanks have been around in some shape or form at least since the 1970s. One of the earliest efforts to produce a syntactically annotated corpus was made by Ulf Teleman and colleagues at Lund University, resulting in more than 300,000 words of both written and spoken Swedish, annotated manually with grammatical functions and a limited form of phrase structure, an impressive achievement at the time but unfortunately documented only in Swedish (Teleman, 1974; Einarsson, 1976a,b; Nivre, 2002). This treebank will be reused in the experiments below.

However, it is only in the last ten to fifteen years that treebanks have been produced on a large scale for a wide range of languages, usually by combining automatic processing with manual annotation or post-editing. A fairly representative overview of available treebanks for a number of languages can be found in Abeillé (2003b), together with a discussion of certain methodological issues. This volume is well complemented by the proceedings of the annual workshops on Treebanks and Linguistic Theories (TLT) (Hinrichs and Simov, 2002; Nivre and Hinrichs, 2003; Kübler et al., 2004; Civit et al., 2005).

While corpus linguistics provided most of the early motivation for developing treebanks and continues to be one of the most important usage areas, the use of treebanks in natural language parsing has increased dramatically in recent years and has probably become the primary driving force behind the development of new treebanks. Broadly speaking, we can distinguish two main uses of treebanks in this area. The first is the use of treebank data in the evaluation of syntactic parsers, which will be discussed in section 5.1.1. The second is the application of inductive machine learning to treebank data, exemplified by the majority of data-driven approaches to text parsing. These two uses are in principle independent of each other, and the use of treebank data in evaluation is not limited to data-driven parsing systems. However, the data-driven method for research and development normally involves an iterative training-evaluation cycle, which makes not only inductive inference but also empirical evaluation an integral part of the methodology. This gives rise to certain methodological problems, which will be treated in section 5.1.2. Finally, in section 5.1.3 we will address the specific requirements on treebank data for dependency parsing.

5.1.1 Treebank Evaluation

Empirical evaluation of systems and components for natural language processing is currently a very active field. With respect to syntactic parsing there are essentially two types of data that are used for evaluation. On the one hand, we have so-called *test suites*, i.e., collections of sentences that are compiled in order to cover a particular range of syntactic phenomena without consideration of their frequency of occurrence (Lehmann et al., 1996). On the other hand, we have treebank samples, which are extracted to be representative with respect to the frequency of different phenomena. Both types of data are clearly relevant for the evaluation of syntactic parsers, but it is also clear that the resulting evaluation will focus on different properties. Test suite evaluation typically measures the coverage of a syntactic parser in terms of the number of constructions that it can handle, without considering the relative frequency of these constructions, although it can also give diagnostic information on issues such as overgeneration and overacceptance (cf. Oepen and Flickinger, 1998). Treebank evaluation, on the other hand, measures the average performance that we can expect from the parser when applied to naturally distributed data from the same source as the evaluation corpus. Given the view of text parsing adopted in this study, it is clear that treebank evaluation is the most relevant form of empirical evaluation.

Parser evaluation may focus on several different dimensions. For instance, robustness (or coverage) can be evaluated by calculating how large a proportion of the input sentences receive an analysis, and disambiguation (or leakage) can be evaluated by computing the average number of analyses assigned to a sentence, normalized with respect to sentence length (Black et al., 1993). For this kind of evaluation it is not even necessary to have annotated treebank data. However, as noted by Carroll et al. (1998), these measures are very weak in themselves, unless they are complemented by some kind of qualitative evaluation of the analyses assigned to a given sentence. For the investigations in this book, they are even less interesting, since our parsing methods guarantee exactly one analysis per sentence for any input text.

Another dimension that can in principle be evaluated without annotated treebank data is efficiency. Measuring time or memory consumption during parsing and relating it to the size of the input only requires a sample of text. Again, however, it is clear that this is a very weak form of evaluation, unless it is combined with an assessment of analysis quality. In the case of dependency parsing, it is trivial to construct an optimally efficient parser that simply analyzes each word as a dependent of the preceding word.

This brings us to the evaluation of accuracy, which is clearly the most important aspect of treebank evaluation. First, as we have just seen, it is often a necessary condition for the interpretation of other forms of evaluation. In most cases, it is simply not meaningful to compare two systems with respect to robustness, disambiguation or efficiency unless we have some way of comparing their respective accuracy. More importantly, however, the notion of empirical

accuracy is at the very heart of the notion of text parsing, as defined in this study. Whereas grammar parsing can be evaluated in terms of formal notions such as consistency and completeness, there is simply no alternative to an empirical evaluation of accuracy for text parsing. And the standard methodology for this kind of evaluation is to use a sample of treebank data as an empirical gold standard.

The basic idea is straightforward. If the treebank sample is representative of the text language that we want to analyze, then applying the parser to the text and comparing the output of the parser to the original annotation will allow us to estimate the average accuracy of the parser when applied to an arbitrary text taken from the same population. In the same fashion, comparing the output of two different parsers applied to the same sample should allow us to test the hypothesis that their average accuracy is different. On the face of it, this is a standard application of statistical inference to experimental data. In reality, there are a number of problems that arise in connection with this evaluation method, problems related to data selection, to treebank annotation and to evaluation metrics.

Starting with the problems of data selection, it is worth remembering that any application of statistical inference is based on the assumption that we have a random sample of the variable under consideration, or at least a set of independent and identically distributed (i.i.d.) variables (Lindgren, 1993). To what extent a treebank sample satisfies these conditions depends on a number of factors, some of which are not under the control of the researcher wishing to perform evaluation, such as the sampling procedure used when collecting the data for the treebank in the first place. Even if data for the treebank has been collected by means of a sampling procedure, this sampling is usually performed on the level of text blocks, such as documents or paragraphs, which means that the sampling conditions are not satisfied on the level of individual sentences. This problem becomes even more serious if we measure accuracy on the word or phrase level, as is the case for many accuracy metrics, since it is quite obvious that the individual words or phrases of a single sentence are not i.i.d. variables. By taking these factors into account when selecting data for evaluation, we can mitigate the effects of statistical dependence between measurements and avoid the overestimation of statistical significance, even if we can never in practice attain the ideal situation of having a strict random sample.

Besides problems having to do with the sampling procedure itself, we must also ask ourselves which population we are sampling from. Throughout this study we have referred to the problem of parsing unrestricted natural language text, but it is highly questionable whether we can ever sample this population by selecting data from existing treebanks. Many treebanks are limited to a single genre of text, usually newspaper text which is easily accessible, with the Wall Street Journal section of the Penn Treebank (Marcus et al., 1993) being the most well-known example in parser evaluation. And even so-called balanced corpora usually draw their data from a limited set of text types,

often including newspaper text, literary works, and various forms of technical and scientific writing, as in the influential Brown Corpus (Kucera and Francis, 1967). Although this is not literally *unrestricted* natural language text, it is still the best we can get if we want to perform a statistical evaluation of accuracy in text parsing. But we have to keep in mind that our results will only be valid for the population from which our data have been sampled, and that the delimitation of this population is often far from clear-cut.

Another way in which treebank data depart from the ideal of unrestricted natural language text is that it has already been tokenized and segmented into sentences. Although our definition of text parsing presupposes that a text consists of a sequence of sentences, and that a sentence consists of a sequence of tokens, it is a non-trivial problem to segment naturally occurring text into sentences and tokens (Palmer, 2000), which means that we are bound to overestimate parsing accuracy when using test samples that are tokenized and segmented into sentences. Still, unless we are specifically interested in the influence of tokenization and sentence segmentation on parsing accuracy, this is a reasonable idealization in practice, which also has the advantage that it makes it easier to compare different parsers on exactly the same set of sentences and tokens.

A more serious problem is the role of the gold standard annotation in the evaluation process. The basic assumption in treebank evaluation of accuracy is that the gold standard provides the correct analysis for each sentence. In practice, this assumption is problematic for several reasons. First of all, any annotated corpus is bound to contain plain errors in the annotation, which means that in some cases the gold standard will provide an incorrect analysis of the sentence in question. If errors are rare and randomly distributed, this can be regarded as a minor problem.

Secondly, we may question the assumption that every sentence in a text has a single correct analysis, even relative to a fixed model of syntactic representation. Besides sentences that are genuinely ambiguous even in context, there is the problem of syntactic indeterminacy (Matthews, 1981), which means that more than one syntactic analysis may be compatible with an unambiguous semantic interpretation of a sentence and that one of these analyses therefore has to be chosen more or less arbitrarily. This may in turn lead to inconsistent annotation of the same syntactic construction, a problem that is supposed to be eliminated by detailed annotation guidelines, but which nevertheless exists in practice, as shown by Dickinson and Meurers (2003).

Finally, when using treebank data for evaluation it is often necessary to convert the annotation from one type of representation to another in order to fit the output of the parsers to be evaluated. Thus, many treebank annotation schemes include empty categories, which are normally removed when evaluating parsers that do not include such elements. Another relevant example is the conversion of constituency-based representations to dependency-based representations, which is necessary in the experiments reported below. This kind of conversion can seldom be performed with perfect accuracy, which means

that the converted annotation will contain a larger proportion of questionable analyses than the original one.

Despite all these problems, however, an annotated treebank can in most cases be regarded as a reasonable approximation to a gold standard, or at least as a sufficiently objective standard for the evaluation of accuracy in text parsing. The final methodological issue to be discussed concerning treebank evaluation is how to measure the correspondence between the output of a parser and the gold standard annotation, i.e., the choice of *evaluation metric*. Given a test sample $T_e = (x_1, \ldots, x_n)$, with the corresponding gold standard annotation $A_g = (y_1^g, \ldots, y_n^g)$ and the output $A_p = (y_1^p, \ldots, y_n^p)$ of some parser p, an obvious metric to use is the proportion of sentences where the parser output completely matches the gold standard annotation, usually referred to as the *exact match* (EM) criterion (where δ is the so-called Kronecker's δ that has value 1 if the two arguments are identical and 0 otherwise):

$$\text{EM} = \frac{1}{n} \sum_{i=1}^{n} \delta(y_i^g, y_i^p) \tag{5.1}$$

The EM metric has the advantage that the variables observed are sentences, rather than words or phrases, which makes statistical independence assumptions somewhat less problematic. At the same time, it is a rather crude metric, since an error in the analysis of a single word or constituent has exactly the same impact on the result as the failure to produce any analysis whatsoever.

Consequently, the most widely used evaluation metrics today are based on various kinds of partial correspondence between the parser output and the gold standard parse. The most well-known of these evaluation metrics are the PARSEVAL measures (Black et al., 1991; Grishman et al., 1992), which consider the number of matching constituents between the parser output and the gold standard, and which have been widely used in parser evaluation, in particular using data from the Penn Treebank. By comparing the number m of matching constituents to the number p of constituents produced by the parser and the number c of constituents in the gold standard analysis, we can measure the *bracketed precision* ($\frac{m}{p}$) and the *bracketed recall* ($\frac{m}{c}$). Only considering the bracketing has the advantage that it enables comparisons between parsers that use different sets of categories to label constituents. However, if constituent labels are also taken into account, we get *labeled precision* and *labeled recall* instead. Finally, it is common to include statistics on the mean number of *crossing brackets* per sentence (or the proportion of sentences that have zero crossing brackets), where a crossing bracket occurs if a parser constituent overlaps a gold standard constituent without one being properly contained in the other.

Although the PARSEVAL measures make very few assumptions about the form of syntactic representations, they do presuppose that representations are constituency-based. For dependency-based representations, the closest correspondent to these metrics is the *attachment score* (AS) (Eisner, 1996a,b;

Collins et al., 1999), which measures the proportion of words in a sentence that are attached to the correct head according to the gold standard. If we let h_g denote the gold standard assignment of dependents to heads for the sentence $x = (w_1, \ldots, w_k)$ and let h_p denote the assignment produced by the parser p, then we can define the *unlabeled* attachment score (AS_U) of p with respect to x as follows:

$$\mathrm{AS}_U(x) = \frac{1}{k} \sum_{i=1}^{k} \delta(h_g(i), h_p(i)) \tag{5.2}$$

If we also take dependency labels into account, as proposed by Lin (1995), we get a *labeled* version of the attachment score (AS_L), which is applicable to parsers that produce labeled dependency graphs (using d_g and d_p for the assignment of dependency labels to words by analogy with h_g and h_p):

$$\mathrm{AS}_L(x) = \frac{1}{k} \sum_{i=1}^{k} \delta(h_g(i), h_p(i)) \cdot \delta(d_g(i), d_p(i)) \tag{5.3}$$

When calculating the attachment score for the entire test sample, we may either calculate the mean per sentence (sometimes called the *macro-average*) or the mean per word (the *micro-average*). Although it can be argued that the former is more natural, the latter is more common in the literature. One good reason for this is that the sentence score for very short sentences can only assume a discrete set of values, which may distort the overall scores.

The PARSEVAL measures have been criticized for being too permissive in some situations while sometimes penalizing the same error more than once (Lin, 1995; Carroll and Briscoe, 1996; Carpenter and Manning, 1997; Carroll and Briscoe, 1996). Regardless of these problems, however, the PARSEVAL measures and the attachment scores for dependency representations have the disadvantage that they are only applicable to one kind of representation. As an alternative to these metrics, several researchers have therefore proposed evaluation schemes based on dependency structure, where both the treebank annotation and the parser output, whether constituency-based or dependency-based, are converted into sets of more abstract dependency relationships (Lin, 1995, 1998; Carroll et al., 1998; Kübler and Telljohann, 2002; Carroll et al., 2003). Recently, this has led to the development of *dependency banks* for parser evaluation (Carroll et al., 2003; King et al., 2003; Forst et al., 2004). Having more abstract representations also makes the scheme less sensitive to the indeterminacy problem in annotation. The only drawback with this methodology is the overhead involved in converting parser representations to the more abstract dependency relationships and the possible errors that may be introduced in this process. In situations where only one kind of representation is relevant, it may therefore still be justified to use the more representation-dependent metrics. Thus, in the experiments reported below we will mainly use metrics based on exact match and attachment scores for dependency-based representations.

5.1.2 Treebank Learning

In the data-driven approach to text parsing, treebank data is crucial not only for the evaluation but also for the development of parsing systems, since the core component of this approach is the application of inductive inference to a representative sample of data in the training phase. In principle, the use of treebank data in the training phase is independent of its use in evaluation, but in practice they are intimately connected since the development of a data-driven parser normally involves an iterative training-evaluation cycle, where different parameters are varied systematically to improve overall performance.

During both development and final evaluation, it is essential that the data used for evaluation is distinct from the data used for training. In both cases, we are interested in estimating the expected accuracy, i.e., the accuracy that we can expect on average when applying the parser to an independent test set, and training set accuracy is in general a very poor estimate of this quantity. Training set accuracy increases consistently with model complexity, but a model with very high training set accuracy often overfits the training data and does not generalize well (Hastie et al., 2001).

However, repeatedly using the same test set for evaluation will produce a similar effect, which means that the test set accuracy may substantially overestimate the expected accuracy on unseen data. It is important in this context to distinguish two different but related problems: *model selection* and *model assessment* (Hastie et al., 2001). Model selection is the problem of estimating the performance of different models in order to choose the (approximate) best one; model assessment is the problem of estimating the expected accuracy of the finally selected model.

In a data-rich situation, the standard solution is to randomly divide the available data into three parts: a training set, a validation set, and a test set. The training set is used for inductive inference; the validation set is used (repeatedly) to estimate accuracy for model selection; and the test set is used for the assessment of the accuracy of the final chosen model. A well-known example of this methodology is the standard split of the Wall Street Journal section of the Penn Treebank into sections 02–21 for training, one of the sections 00, 22 and 24 for validation, and section 23 for final testing.

In a data-poor situation, there are various techniques that can be used to approximate the validation step without having a separate validation set, either by analytical methods, such as Bayesian Information Criterion (BIC) or Minimum Description Length (MDL), or by efficient sample re-use, such as cross-validation and bootstrap methods (Hastie et al., 2001). Although these techniques can also to some extent be used for model assessment, it is more common to combine them with an independent test set for the final evaluation.

Whether treebank parsing should be considered a data-rich or a data-poor situation depends to some extent on which languages we are interested in. For English, there are several treebanks of reasonable size available, which explains the standard training-validation-test setup usually applied to the Wall Street

Journal data. However, it is also worth pointing out that, even if section 23 is only used once in every published study based on this data set, it has over the years been used repeatedly by the same and different research groups, which in fact amounts to a kind of repeated testing, albeit at a higher level of abstraction. Thus, the value of this data set as a basis for estimation of expected accuracy is by now highly dubious, and it is probably better regarded today as a benchmark set.

For a language like Swedish, which is of special interest in this study, the availability of treebank data is rather limited, which motivates the use of cross-validation for model selection, reserving a separate test set for the final evaluation. Finally, it is worth remembering that for most languages of the world, there are simply no treebank data available at all, which rules out supervised learning methods completely.

5.1.3 Treebanks for Dependency Parsing

When the data-driven approach to text parsing is combined with supervised learning methods, training data must be annotated with the same kind of syntactic representations that are used in the parsing system. In our case, this means that we require treebanks that are annotated with dependency graphs. The availability of such treebanks has increased substantially in recent years. In addition to the Prague Dependency Treebank of Czech (Hajič, 1998; Hajič et al., 2001), which is probably the most well-known treebank of this kind, we find the METU Treebank of Turkish (Oflazer et al., 2003), the Danish Dependency Treebank (Kromann, 2003), the Eus3LB Corpus of Basque (Aduriz et al., 2003), the Turin University Treebank of Italian (Bosco and Lombardo, 2004), and the parsed corpus of Japanese described in Kurohashi and Nagao (2003). Furthermore, there are hybrid treebanks, which include both constituency and dependency annotation, such as the TIGER Treebank of German (Brants et al., 2002) and the Alpino Treebank of Dutch (Van der Beek et al., 2002).

In fact, whereas many of the early large-scale treebank projects, such as the Lancaster Parsed Corpus (Garside et al., 1992) and the original Penn Treebank (Marcus et al., 1993), were based on constituency annotation only, most annotation schemes today include some kind of functional analysis that can be regarded as a partial dependency analysis. This is true of the Penn Treebank II annotation scheme (Bies et al., 1995), which adds functional tags to the original phrase structure annotation, and similar combinations of constituent structure and grammatical functions are found in the SUSANNE annotation scheme (Sampson, 1995), in the ICE-GB Corpus of British English (Nelson et al., 2002), and in the adaptations of the Penn Treebank II schemes that have been developed for Chinese (Xue et al., 2004), Korean (Han et al., 2002), Arabic (Maamouri and Bies, 2004) and Spanish (Moreno et al., 2003).

Constituency-based treebanks, with or without functional annotation, can in principle be converted to dependency treebanks. As shown by Gaifman (1965), it is straightforward to convert a constituency tree to an unlabeled

dependency tree, provided that every constituent c has a unique head child c_h. The dependency tree is obtained by recursively letting the head d of each non-head child c_d of c be a dependent to the head h of the head child c_h of c (where a terminal node $c_h = h$ is its own head) (cf. Xia and Palmer, 2001). This method has been used in several studies to convert constituency-based treebank annotations to dependency structures, notably using data from the Penn Treebank (Collins, 1996, 1997, 1999; Xia and Palmer, 2001; Xia, 2001; Yamada and Matsumoto, 2003; Nivre and Scholz, 2004), but also from the German treebanks TüBa-D (Kübler and Telljohann, 2002) and TIGER (Bohnet, 2003; Ule and Kübler, 2004).

In practice, there are normally a number of factors that interact to make some of the converted dependency representations less than optimal. Besides problems that are inherent in the dependency-based approach to syntactic representations, such as the existence of constructions that are not readily analyzed as single-headed, the main problem is that it may be difficult to identify the head child in a constituency representation even when such a child exists, since most constituency-based annotation schemes do not mark heads explicitly. The standard solution to this problem is to use *head percolation tables* (Magerman, 1995; Collins, 1996, 1999) that provide heuristic rules for identifying the head child in a constituent of a specific type. Figure 5.1 shows the head percolation table used by Yamada and Matsumoto (2003) and Nivre and Scholz (2004). The first column contains the constituent labels found in the Penn Treebank. For each constituent label, the second column specifies the direction of search (from the right [R] or from the left [L]) and the second column gives a list of potential head child categories, partially ordered by descending priority (with the vertical bar | symbolizing equal priority). For example, for a constituent of type NP, we start searching from the right for a child of type POS, NN, NNP, NNPS or NNS. The first child matching this condition is chosen as the head. If no child matching this condition is found, we proceed to search for a child of type NX, etc. If the entire list is exhausted, the first child encountered when searching in the specified direction is chosen as the head.

When applied to the Penn Treebank, a head percolation table of this kind gives a quite reasonable conversion to dependency structures for the majority of constituent types. However, for certain types of constituents, such as complex noun phrases involving coordination, it is extremely difficult to devise a set of rules that guarantees an adequate conversion in all cases. The most elaborate scheme in this respect is probably the rules used by Collins (1999), where the head percolation table is supplemented by special rules for noun phrases and coordination (cf. also Bikel, 2004).

The problem of identifying head children in constituency representations is mitigated if an extensive functional annotation is present. Thus, in converting the Swedish treebank Talbanken (Einarsson, 1976a,b) to a dependency treebank, the problem of identifying head children can be solved almost completely by only considering the functional annotation (Nilsson et al., 2005).

NP	R	POS|NN|NNP|NNPS|NNS NX JJR CD JJ JJS RB QP NP
ADJP	R	NNS QP NN $ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB
ADVP	L	RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN
CONJP	L	CC RB IN
FRAG	L	
INTJ	R	
LST	L	LS :
NAC	R	NN|NNS|NNP|NNPS NP NAC EX $ CD QP PRP VBG JJ JJS JJR ADJP FW
PP	L	IN TO VBG VBN RP FW
PRN	R	
PRT	L	RP
QP	R	$ IN NNS NN JJ RB DT CD NCD QP JJR JJS
RRC	L	VP NP ADVP ADJP PP
S	R	TO IN VP S SBAR ADJP UCP NP
SBAR	R	WHNP WHPP WHADVP WHADJP IN DT S SQ SINV SBAR FRAG
SBARQ	R	SQ S SINV SBARQ FRAG
SINV	R	VBZ VBD VBP VB MD VP S SINV ADJP NP
SQ	R	VBZ VBD VBP VB MD VP SQ
UCP	L	
VP	L	VBD VBN MD VBZ VB VBG VBP VP ADJP NN NNS NP
WHADJP	R	CC WRB JJ ADJP
WHADVP	L	CC WRB
WHNP	R	WDT WP WP$ WHADJP WHPP WHNP
WHPP	L	IN TO FW
NX	R	POS|NN|NNP|NNPS|NNS NX JJR CD JJ JJS RB QP NP
X	R	

Fig. 5.1. Head percolation table for the Penn Treebank

However, the presence of functional annotation is even more crucial if we want to convert constituent representations to *labeled* dependency graphs, since dependency type labels can normally be inferred from the functional annotation but only indirectly from the constituency annotation.

The experiments presented in this chapter are based on data from Swedish and English. The choice of these languages does not reflect the availability of dependency treebanks but rather a desire to develop better parsing systems for Swedish, on the one hand, and to compare the performance of the system to available benchmarks for English, on the other hand. An unfortunate consequence of this choice is that the experiments will in both cases be performed on dependency treebanks that are the result of conversion from another kind of annotation. Experiments on genuine dependency treebanks, notably the Prague Dependency Treebank and the Danish Dependency Treebank, have also been performed but will not be reported here. The main reason is that

Genom	PR	AAPR
skattereformen	NNDDSS	AA
införs	VVPSSMPA	FV
individuell	AJ	SSAT
beskattning	VN	SS
av	PR	SSETPR
arbetsinkomster	NN SS	SSET
.	IP	IP

Fig. 5.2. Swedish sentence annotated according to MAMBA

this requires special treatment of non-projective dependency graphs, which would take us too far afield in this study. Preliminary results on the use of inductive dependency parsing in combination with graph transformation techniques to capture non-projective structures are reported by Nivre and Nilsson (2005).

5.2 Experimental Methodology

As noted above, the experimental evaluation of the deterministic and memory-based version of inductive dependency parsing is based on data from two languages, Swedish and English. We will systematically vary parameters of the feature model and the learning algorithm in order to study their influence on parsing accuracy and efficiency. We will also compare the results to relevant previous research. In this methodological section, we first describe the data sets used in the experiments, the parameters varied in the experiments, and the metrics used to evaluate parsing accuracy and efficiency.

5.2.1 Treebank Data

The Swedish data for the experiments come from Talbanken (Einarsson, 1976a,b), a syntactically annotated corpus of written and spoken Swedish, created in a series of projects at the University of Lund in the 1970s. In this study, we use the Professional Prose section, containing informative and argumentative text from brochures, newspapers, and books. The syntactic annotation follows the MAMBA scheme (Teleman, 1974), which is described by its creators as an eclectic combination of constituent structure, dependency structure and topological field analysis. Figure 5.2 shows an example of the MAMBA annotation applied to a Swedish sentence taken from Talbanken.[1]

[1] Word-by-word gloss: 'Through tax-reform-DEF introduce-PAST-PASSIVE individual taxation of work-income-PLUR.' Translation: 'Through the tax reform individual taxation of work incomes is introduced.'

The annotation consists of two layers, the first being a lexical analysis, consisting of part-of-speech information including morphological features, and the second being a syntactic analysis, in terms of grammatical functions. Both layers are flat in the sense that they consist of tags assigned to individual word tokens, but the syntactic layer also gives information about constituent structure, as exemplified with respect to the grammatical subject in figure 5.2. All the words belonging to the subject noun phrase *individuell beskattning av arbetsinkomster* (individual taxation of work incomes) are annotated with the tag SS for subject, but the head noun *beskattning* (taxation) is marked as such by having only this tag, while the pre-modifying adjective *individuell* (individual) is also tagged AT for adjectival modifier and the words of the post-modifying prepositional phrase *av arbetsinkomster* (of work incomes) are tagged ET for nominal post-modifier. Within the prepositional phrase, the noun *arbetsinkomster* (work incomes) is marked as the head, while the preposition *av* (of) gets an additional tag PR for the prepositional function.

The constituent structure recognized in MAMBA is rather flat, especially on the clause level where the analysis to a large extent is modeled after the topological field analysis proposed by Diderichsen (1946) for the Scandinavian languages. The main constituents recognized in the clause are the following:

- Verb (–V)
- Subject (–S)
- Object (–O)
- Predicative (–P)
- Adverbial (–A)

A more fine-grained classification of these constituents is obtained by varying the first letter of the two-letter tag. Thus, finite verbs are tagged FV, non-finite verbs IV; logical subjects are tagged ES, formal subjects FS, and other subjects SS, etc. In addition to the constituents recognized in Diderichsen's topological field model, there is a limited phrase structure analysis of noun phrases, prepositional phrases, adjective phrases, and subordinate clauses, with special tags for internal grammatical functions. Altogether, there are 42 distinct grammatical function tags in the MAMBA annotation scheme (Teleman, 1974).

Thanks to the rich functional annotation it is relatively straightforward to convert the MAMBA annotation to dependency graphs. For the majority of phrases, the syntactic head is explicitly marked and the function tags assigned to other constituents can be used to label dependency arcs. However, there are two types of structures that require special treatment. The first is the clause structure, where there is no explicit indication of a syntactic head, and where the following categories are considered as candidate heads in descending order of priority:

1. The leftmost finite verb (FV).
2. The leftmost non-finite verb (IV).

ADV	Adverbial modifier
APP	Apposition
ATT	Attribute (adnominal modifier)
CC	Coordination (conjunction or second conjunct)
DET	Determiner
ID	Non-first element of multi-word expression (idiom)
IM	Infinitive dependent on infinitive marker
INF	Infinitival complement
IP	Punctuation
OBJ	Object
PR	Complement of preposition
PRD	Predicative complement
ROOT	Dependent of special root node
SUB	Subject
UK	Head verb of subordinate clause dependent on complementizer
VC	Verb chain (nonfinite verb dependent on other verb)
XX	Unclassifiable dependent

Fig. 5.3. Dependency types in Swedish treebank

3. The head of the leftmost predicative complement (–P).
4. The head of the leftmost subject, object or adverbial (–S, –O, –A).
5. The leftmost word.

The second type of structure is coordination, where the MAMBA annotation treats every conjunct as a head. In our conversion to dependency structure, we adopt a Mel'čuk style analysis of coordination and treat the leftmost conjunct as the head (cf. section 3.1.2).

After the initial conversion to dependency graphs, we apply two types of transformations to these graphs. The first is related to some of the traditional open issues in dependency grammar, where we prefer a different analysis than the one assumed in the MAMBA annotation. More precisely:

1. Prepositional phrases are headed by the preposition, which takes the head noun of the nominal complement as a dependent.
2. Subordinate clauses with an overt complementizer (except relative clauses) are headed by the complementizer, which takes the finite verb of the subordinate clause as a dependent.
3. Infinitival verb phrases with an overt infinitive marker are headed by the infinitive marker, which takes the infinitive verb as a dependent.

The second type of transformation concerns three kinds of structure that do not have a clear-cut dependency analysis, namely idioms (including multiword proper names and compound function words), verb chains and coordinate structures. By analogy with Tesnière's notion of dissociate nuclei, these

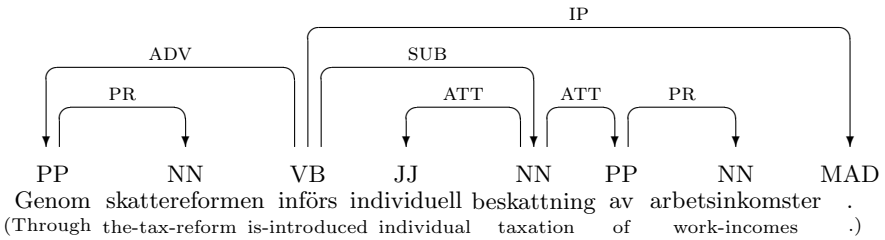


Fig. 5.4. Dependency graph for Swedish sentence, converted from Talbanken

constructions are treated as left-headed chains, where each subsequent element is a dependent of the immediately preceding one, and where left and right dependents of the entire unit are attached to the leftmost and rightmost element, respectively (and internal dependents to the leftmost element). In the case of verb chains, this means that left dependents will formally be treated as dependents of the leftmost verb (normally the finite verb in Swedish), while right dependents will be attached to the rightmost verb (possibly a non-finite verb).

The final thing to note about the conversion of the MAMBA annotation is the set R of dependency types used as arc labels. On the one hand, we have collapsed some of the finer distinctions in the original set of grammatical functions, where no less than twelve different types of adverbials are distinguished. On the other hand, we have added a few dependency types for relations that are not marked explicitly in the MAMBA annotation, notably for verb chains and coordination. This gives us a set R of 17 dependency labels (including the special root label r_0 = ROOT), which are listed with explanations in figure 5.3.

The part-of-speech tags included in the original annotation of Talbanken have not been used in the experiments, mainly because there is no part-of-speech tagger available for this tagset. Since we want to be able to apply the parsing system to new texts, we have therefore used a statistical tagger trained on the much larger Stockholm-Umeå Corpus (Ejerhed and Källgren, 1997), using a tagset consisting of 150 tags, to preprocess the Swedish data both for training and for evaluation. The estimated accuracy of the tagger, when evaluated on held-out data from the Stockholm-Umeå Corpus is 94.4%. Figure 5.4 shows the result of converting the sentence in figure 5.2 using the procedure described in this section and tagging it with the statistical part-of-speech tagger (although the morphological features of the part-of-speech tags have been suppressed for readability reasons).

The dependency treebank obtained by converting the Professional Prose section of Talbanken consists of 6316 sentences and 97623 tokens (including punctuation), which gives a mean sentence length of 15.46 tokens. For previous experiments, the sentences of this treebank have been randomly divided into

```
( (S
     (NP-SBJ (JJ Economic) (NN news) )
     (VP (VBD had)
       (NP
         (NP (JJ little) (NN effect) )
         (PP (IN on)
           (NP (JJ financial) (NNS markets) ))))
     (. .) ))
```

Fig. 5.5. English sentence annotated according to Penn Treebank II

ten equally large sections, numbered 0–9, where sections 1–8 have been used as training data and section 9 as validation data, saving section 0 for later studies (Nivre et al., 2004; Nivre and Nilsson, 2004; Nivre, 2004a). In this study, we instead use nine-fold cross-validation on sections 1–9 for model selection, and use section 0 as the test set for the final model assessment. The data set used for cross-validation consists of 5685 sentences and 87757 tokens, while the final test set consists of 631 sentences and 9841 tokens.

The English data are taken from the Penn Treebank (Marcus et al., 1993), which has been the most widely used treebank for parser evaluation over the last decade. In this study, we use the Wall Street Journal section of the treebank, with the Penn Treebank II annotation scheme (Bies et al., 1995), which combines constituency analysis with a limited functional annotation. Figure 5.5 repeats the example sentence used in chapters 1–4, this time in the original annotation format using the full node labels, composed of bracketing labels and grammatical function labels.

We assume that the Penn Treebank II annotation scheme is familiar to most readers and proceed directly to a discussion of the way in which this annotation can be converted to dependency graphs. For the unlabeled dependency graphs we rely on the standard method described in section 5.1.3, using the head percolation table of Yamada and Matsumoto (2003), which is a slight modification of the rules used by Collins (1999). Using this conversion scheme permits us to make exact comparisons with the parser of Yamada and Matsumoto (2003), as well as the parsers of Collins (1997) and Charniak (2000), which are evaluated on the same data set in Yamada and Matsumoto (2003). The head percolation table can be found in figure 5.1.

In addition to the structural conversion, we also have to derive dependency types to use as arc labels. Compared to the Swedish treebank, the functional annotation in the Penn Treebank is much less comprehensive, which makes this a non-trivial problem. Given an arc $i \rightarrow j$, derived from a local constituent tree where w_i is the head of the head child h and w_j is the head of a non-head child d, let M, H and D be the original labels on the mother node, head child h and child d, respectively, except that H and D are replaced by

AMOD	Modifier of adjective or adverb (phrase adverbial)
DEP	Other dependent (default label)
NMOD	Modifier of noun (including complement)
OBJ	Object
P	Punctuation
PMOD	Modifier of preposition (including complement)
PRD	Predicative complement
ROOT	Dependent of special root node
SBAR	Head verb of subordinate clause dependent on complementizer
SBJ	Subject
VC	Verb chain (nonfinite verb dependent on other verb)
VMOD	Modifier of verb (sentence or verb phrase adverbial)

Fig. 5.6. Dependency types in English treebank

TAG if they are part-of-speech tags. The labels M, H and D, stripped of their function tags, have been used by Collins (1999) to construct complex dependency labels (M, H, D, dir), where dir is L or R (for left and right dependency, respectively). In our experiments, we instead use these labels to formulate a set of rules for choosing the arc label r, $i \xrightarrow{r} j$. In order of descending priority, the rules are as follows:

1. If D is a punctuation category, r = P.
2. If D contains the function tag SBJ, r = SBJ.
3. If D contains the function tag PRD, r = PRD.
4. If M = VP, H = TAG and D = NP (without any function tag), r = OBJ.
5. If M = VP, H = TAG and D = VP, r = VC.
6. If M = SBAR and D = S, r = SBAR.
7. If M = VP, S, SQ, SINV or SBAR, r = VMOD.
8. If M = NP, NAC, NX or WHNP, r = NMOD.
9. If M = ADJP, ADVP, QP, WHADJP or WHADVP, r = AMOD.
10. If M = PP or WHPP, r = PMOD.
11. Otherwise, r = DEP.

The complete set R of dependency types, including the root label r_0 = ROOT, is listed in figure 5.6. The explanations given reflect the intended interpretation of each category, although it is clear that the rules for choosing dependency types will also cover cases that do not fit the descriptions. A notoriously difficult problem is the treatment of complex noun phrases, which have a very flat structure in the Penn Treebank. For example, coordinated noun phrases will often be analyzed as structures where the last conjunct is the head, while preceding conjuncts as well as the coordinating conjunction are analyzed as dependents of the NMOD type. Similar problems can be identified for most of the rules and categories. However, having a set of dependency types that are

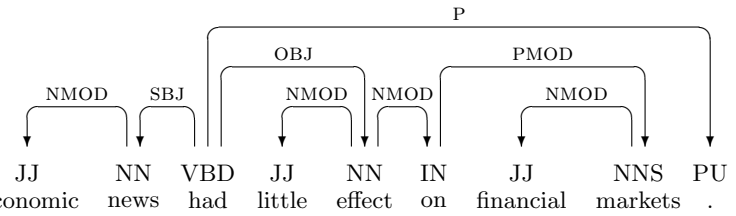


Fig. 5.7. Dependency graph for English sentence, converted from the Penn Treebank (cf. figures 1.1, 2.2 and 3.1)

Table 5.1. Data sets for training, validation and test; Sec: section, W: number of tokens, S: number of sentences, W/S: mean number of tokens per sentence

	Swedish				**English**			
Data set	**Sec**	**W**	**S**	**W/S**	**Sec**	**W**	**S**	**W/S**
Training	1–9	87757	5685	15.44	01-21	950028	39832	23.85
Validation	–	–	–	–	00	46451	1921	24.18
Test	0	9841	631	15.60	23	56684	2416	23.46

similar in nature and cardinality to the Swedish set will make results more comparable.

As regards part-of-speech tagging, we use the gold standard tags from the Penn Treebank for training, and a statistical tagger trained on section 2-21 for validation and final testing. The tagger has an accuracy of 96.1% on section 23. In the final evaluation, we will also evaluate the parser on gold standard tags to see how large proportion of the errors can be attributed to tagging errors. Figure 5.7 shows the result of converting the sentence in figure 5.5 using the procedure described in this section (with the gold standard tags from the treebank).

The dependency treebank obtained by converting the Wall Street Journal section of the Penn Treebank consists of 49208 sentences and 1173766 tokens, which gives a mean sentence length of 23.85 words. We use sections 2-21 for training (39832 sentences, 950028 tokens), section 00 for validation and model selection (1921 sentences, 46451 tokens), and section 23 for the final model assessment (2416 sentences, 56684 tokens).

Table 5.1 gives an overview of all the data sets used in the experiments. There is no separate validation data set for Swedish, given that we use cross-validation for model selection.

5.2.2 Models and Algorithms

As emphasized on several occasions, the parsing methods evaluated in these experiments are only one possible instantiation of the general framework of inductive dependency parsing. This means that, although the experiments will involve a systematic study of the influence of different variables on accuracy and efficiency, there are also many factors that will be kept constant. This holds in particular for the parsing algorithm, which will in all cases be the deterministic arc-eager algorithm presented and analyzed in section 3.4. But it also holds for the learning method, in the sense that we will only consider memory-based learning algorithms, as described in section 4.3, although we will explore many variants of this general approach to machine learning.

The first part of the experiment, presented in section 5.3, will be devoted to parameters of the feature model, exploring different combinations of the three types of features discussed in section 4.2: part-of-speech features, dependency features, and lexical features. We will begin with simple models based only on part-of-speech features and gradually increase model complexity by adding first dependency features and then lexical features. The different models will mainly be evaluated with respect to parsing accuracy, but a selected subset will also be evaluated for efficiency. Finally, we will consider the learning curves of different models, i.e., parsing accuracy as a function of the size of the training corpus. Throughout the first part, the parameters of the learning algorithm will be kept constant. As described in section 4.3.2, we will use the following settings for the k-NN classification provided by the memory-based learner:

1. Number of nearest distances: $k = 5$
2. Distance metric: MVDM with $l = 3$
3. Feature weighting: None
4. Distance-weighted class voting: ID weighting.

In terms of the TiMBL system, this corresponds to the following parameter settings: -k 5 -m M -L 3 -w 0 -d ID (cf. Daelemans et al., 2004).

The second part of the experiment, presented in section 5.4, will explore some of the options provided by TiMBL for the memory-based learning and classification. In particular, we will consider the influence of different k values and distance metrics, in interaction with different schemes for feature weighting and distance-weighted voting. Throughout the second part, the parameters of the feature model will in principle be kept constant, but we will consider two different feature models, one that is lexicalized and one that is not.

The first two parts of the experiment constitute the validation or model selection phase, in the terminology of section 5.1.2. The third and final part of the experiment is the final evaluation or model assessment phase, where we apply the best models with the best settings to a test data set that has not been used in the validation phase. However, it is important to keep in mind that, because of the complex interaction of feature models, learning

algorithm parameters and properties of the data sets, there is no guarantee that the models and settings selected for the final evaluation are in fact truly optimal even for the given data sets.

5.2.3 Evaluation

We will use two different metrics to evaluate parsing accuracy, *attachment score* (AS) and *exact match* (EM). AS measures the proportion of *tokens* that are correctly analyzed, while EM measures the proportion of *sentences* that are assigned a completely correct dependency graph. Both metrics come in an unlabeled version, which only considers the attachment of dependents to head, and a labeled version, which also takes the dependency type labels into account.

Definition 5.1. Given a test sample $T_e = (x_1, \ldots, x_m)$, consisting of m sentences, where each sentence $x_i = (w_1, \ldots, w_k)$ consists of k_i tokens, and the total number of tokens in the sample is n (i.e., $n = \sum_{i=1}^{m} k_i$). Let $A_g = (G_1^g, \ldots, G_m^g)$ be the dependency graphs of the gold standard annotation, let $A_p = (G_1^p, \ldots, G_m^p)$ be the dependency graphs produced by parser p, let $G_i^g = (V_{x_i}, E_i^g, L_i^g)$ and $G_i^p = (V_{x_i}, E_i^p, L_i^p)$, and let h_i^g, d_i^g, h_i^p and d_i^p be the following functions:

$$\begin{aligned} h_i^g(j) = l &\;\Leftrightarrow\; (l, j) \in E_i^g \\ d_i^g(j) = r &\;\Leftrightarrow\; \exists l : ((l, j), r) \in L_i^g \\ h_i^p(j) = l &\;\Leftrightarrow\; (l, j) \in E_i^p \\ d_i^p(j) = r &\;\Leftrightarrow\; \exists l : ((l, j), r) \in L_i^p \end{aligned}$$

The *unlabeled attachment score* AS_U of p with respect to T_e and A_g is:

$$\mathrm{AS}_U = \frac{1}{n} \sum_{i=1}^{m} \sum_{j=1}^{k_i} \delta(h_i^g(j), h_i^p(j))$$

The *labeled attachment score* AS_L of p with respect to T_e and A_g is:

$$\mathrm{AS}_L = \frac{1}{n} \sum_{i=1}^{m} \sum_{j=1}^{k_i} \delta(h_i^g(j), h_i^p(j)) \cdot \delta(d_i^g(j), d_i^p(j))$$

The *unlabeled exact match* EM_U of p with respect to T_e and A_g is:

$$\mathrm{EM}_U = \frac{1}{m} \sum_{i=1}^{m} \delta(h_i^g, h_i^p)$$

The *labeled exact match* EM_L of p with respect to T_e and A_g is:

$$\mathrm{EM}_L = \frac{1}{m} \sum_{i=1}^{m} \delta(h_i^g, h_i^p) \cdot \delta(d_i^g, d_i^p)$$

In the final evaluation in section 5.5, we will also provide a breakdown of the attachment score for different dependency types, which will be computed as *unlabeled attachment score*, (labeled) *precision*, (labeled) *recall* and (labeled) F_β *measure* ($\beta = 1$) for each dependency type r.

Definition 5.2. Let T_e be a test sample and let h_i^g, d_i^g, h_i^p and d_i^p be defined as in definition 5.1. The *unlabeled attachment score* $AS_U(r)$ of dependency type r for parser p with respect to T_e and A_g is:

$$\mathrm{AS}_U(r) = \frac{|\{w_j \in T_e \mid d_i^g(j) = r, h_i^g(j) = h_i^p(j)\}|}{|\{w_j \in T_e \mid d_i^g(j) = r\}|}$$

The *precision* $\mathrm{P}(r)$ of r for p with respect to T_e and A_g is:

$$\mathrm{P}(r) = \frac{|\{w_j \in T_e \mid d_i^p(j) = d_i^g(j) = r, h_i^g(j) = h_i^p(j)\}|}{|\{w_j \in T_e \mid d_i^p(j) = r\}|}$$

The *recall* $\mathrm{R}(r)$ of r for p with respect to T_e and A_g is:

$$\mathrm{R}(r) = \frac{|\{w_j \in T_e \mid d_i^p(j) = d_i^g(j) = r, h_i^g(j) = h_i^p(j)\}|}{|\{w_j \in T_e \mid d_i^g(j) = r\}|}$$

The F *measure* $\mathrm{F}(r)$ of r for p with respect to T_e and A_g is:

$$\mathrm{F}(r) = \frac{2 \cdot \mathrm{P}(r) \cdot \mathrm{R}(r)}{\mathrm{P}(r) + \mathrm{R}(r)}$$

While the $\mathrm{AS}_U(r)$ score only measures how often a dependent of type r is assigned the correct head (regardless of the assigned label), the $\mathrm{P}(r)$ score tells us how often the parser is completely correct when using the label r and the $\mathrm{R}(r)$ score how often a dependent of type r is parsed completely correctly, while $\mathrm{F}(r)$ is the harmonic mean of $\mathrm{P}(r)$ and $\mathrm{R}(r)$.

In all scores reported for evaluation metrics concerning accuracy, punctuation tokens will be omitted from the counts. Figure 5.8 lists the parts-of-speech that are counted as punctuation categories in the two treebanks.

Efficiency will be evaluated by the following three metrics:

1. Training time: The time required to construct the instance base for the memory-based classifier, including the parsing of the training corpus using the gold standard parsing algorithm and the precomputation of metrics by TiMBL.
2. Parsing time: The time required to parse one sentence of the test corpus, excluding the initialization of the parser.
3. Memory consumption: The amount of memory allocated for parsing the test corpus.

Although parsing time is measured for each individual sentence, the results will mostly be presented in aggregated form, as the total parsing time for

Swedish		English	
MAD	Major delimiter	.	Sentence-final punctuation
MID	Minor delimiter	,	Comma
		:	Colon, semi-colon
PAD	Paired delimiter	-LRB-	Left bracket character
		-RRB-	Right bracket character
		"	Straight double quote
		‘	Left open single quote
		“	Left open double quote
		’	Right close single quote
		”	Right close double quote
		#	Pound sign
		$	Dollar sign

Fig. 5.8. Punctuation categories in Swedish and English treebank

a given test corpus, as the mean parsing time per sentence, or as the mean number of words parsed per second (cf. table 5.1 for quantitative properties of the data sets). Time is measured using standard system calls, with measurements reported in seconds (s) or milliseconds (ms), while memory consumption is measured through the UNIX command `top` and reported in number of megabytes (MB). All experiments are run on a SunBlade 2000 with one 1.2GHz UltraSPARC-III processor and 1GB of memory.

The results presented in section 5.3–5.4 are based on the cycle of training and validation for model selection. For the smaller Swedish data set, validation is performed by means of nine-fold cross-validation on sections 1–9, and all results presented are the arithmetic mean of the results from the nine folds. For the larger English data set, we consistently use sections 02–21 for training and section 00 for validation. The results presented in section 5.5 are the final results for model assessment, which involve training on sections 1–9 and testing on section 0 for Swedish, training on sections 02-21 and testing on section 23 for English. For the final evaluation, we use McNemar's test to assess the statistical significance of differences in accuracy (both attachment score and exact match).

5.3 Feature Model Parameters

We start our investigation of different feature models from a baseline model, where the only features used to predict the next transition are the parts-of-speech of the top token and the next token. In the notation introduced in section 4.2.4, the baseline is written Φ^p_{00}, and its two features $p(\sigma_0)$ and $p(\tau_0)$. In the next three sections, we will gradually increase the complexity of the model by adding a larger part-of-speech context, dependency features, and

Table 5.2. Accuracy as a function of part-of-speech context only; AS: attachment score, EM: exact match; U: unlabeled, L: labeled; Swedish (cross-validation), English (section 00)

Model	Swedish				English			
	AS		EM		AS		EM	
	U	L	U	L	U	L	U	L
Φ^p_{00}	71.0	64.5	15.6	12.3	58.4	56.2	3.7	3.1
Φ^p_{01}	73.8	67.5	17.8	13.8	75.9	73.1	8.5	6.8
Φ^p_{10}	74.8	67.8	23.4	15.5	60.5	58.0	5.0	4.0
Φ^p_{11}	**77.9**	**70.9**	**27.2**	**18.0**	**77.7**	**74.8**	**13.4**	**10.2**
Φ^p_{21}	**78.4**	**71.0**	**28.4**	**18.6**	**77.7**	**74.8**	**14.2**	**10.7**
Φ^p_{31}	78.0	70.0	28.1	18.4	77.1	74.2	13.6	10.0
Φ^p_{12}	**77.4**	**70.1**	**26.6**	**17.8**	**79.0**	**76.1**	14.4	10.0
Φ^p_{13}	77.1	69.7	26.1	17.3	78.8	75.9	13.6	9.5
Φ^p_{14}	77.1	69.7	26.3	17.3	78.8	75.9	**14.6**	**10.2**
Φ^p_{15}	77.0	69.7	26.0	17.3	78.5	75.5	14.1	9.6
Φ^p_{16}	77.0	69.6	25.9	17.2	78.7	75.7	14.3	9.3

lexical features. Finally, we will evaluate a subset of the models with respect to efficiency and also investigate their learning curves.

5.3.1 Part-of-Speech Context

The role of part-of-speech features in data-driven approaches to parsing is far from clear-cut, as noted in section 4.2.2, but they are nevertheless used in most models and appear to have a positive effect especially by providing a backoff model for lexical features (Charniak, 2000; Van den Bosch and Buchholz, 2002).

Table 5.2 shows the accuracy obtained for Swedish and English with feature models that differ only with respect to the number of tokens included in the part-of-speech context. Remember that Φ^p_{mn} is a model that includes the $m+1$ top tokens on the stack and the $n+1$ next input tokens (cf. section 4.2.4).

Our first observation is that the baseline model achieves a very modest parsing accuracy, both with respect to attachment score and exact match, especially for English. We see that adding a lookahead of just one token (Φ^p_{01}) makes a tremendous difference for English, and is clearly beneficial for Swedish as well. We see that adding one more token from the stack (Φ^p_{10}) also has a positive effect, although in this case the difference is greater for Swedish, where it has an especially strong effect on the exact match evaluation. For English, the strong positive effect on the exact match evaluation shows up only when the extra stack token is combined with an extra input token (Φ^p_{11}).

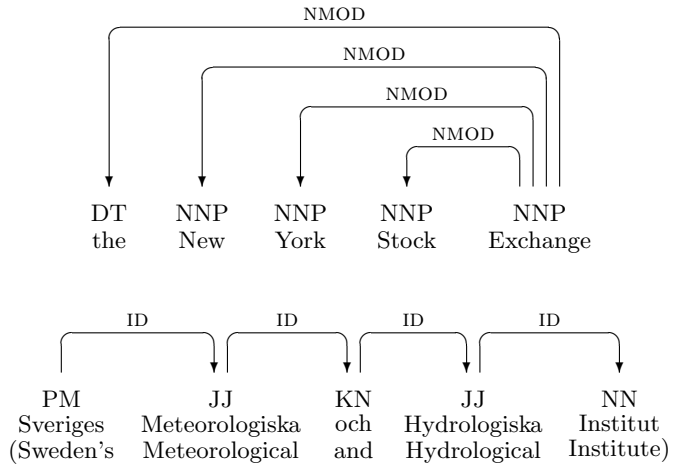


Fig. 5.9. Nominal compounds and multi-word names in English and Swedish

Considering the results at a superficial level, it seems that looking forward is more important for English, while looking backwards is more important for Swedish. However, this difference can to a very large extent be explained by the proliferation of nominal compounds in English, in combination with the particular dependency analysis inherited from the converted phrase structure annotation and the deterministic parsing strategy used. To illustrate this phenomenon, let us consider a typical example from the Wall Street Journal data:

the New York Stock Exchange (5.4)

Apart from the determiner *the*, this noun phrase consists of four consecutive words, which in the Penn Treebank annotation are all tagged as proper nouns (NNP). According to the head percolation table, a noun phrase of this kind is always headed by the rightmost noun, in this case the noun *Exchange*, with all the preceding words as dependents. In order to parse such a structure correctly, the parser must keep shifting until the head noun is the next input token and then perform a series of LEFT-ARC(NMOD) transitions until all the dependents have been attached to the noun. However, without any form of lookahead it will be very hard to predict when the last noun has been encountered, especially with a feature model that only includes part-of-speech features.

The same problem does not arise with the Swedish data. First of all, Swedish compounds are normally written as single words, which means that nominal compounds will not be encountered as syntactic units in the Swedish data. Moreover, in structures similar to the English example, such as multi-word proper names, the Swedish annotation marks the leftmost element as the

head and treats each subsequent element as a dependent of the immediately preceding element. Therefore, the problem of predicting when the head has been found does not occur when parsing Swedish. The difference between the two styles of analysis are illustrated in figure 5.9, which contrast the annotation of English nominal compounds in the converted Penn Treebank with the annotation of Swedish multi-word units in Talbanken.

Moving on to the middle section of table 5.2, we see that adding a second extra stack token (Φ^p_{21}) gives a marginal increase in accuracy, especially with respect to exact match, but that adding a third token (Φ^p_{31}) gives a decrease across the board. In the lower section, we see essentially the same pattern with respect to lookahead, although with a differentiation between languages. For English, adding a second lookahead token (Φ^p_{12}) is beneficial but extending the context even further is not. For Swedish, accuracy starts to go down already with the second lookahead token. However, we will see later that the effect of increasing the part-of-speech context is also sensitive to the presence of other features.

Summarizing the results for part-of-speech features only, it seems that the model Φ^p_{21} gives the best performance for Swedish, while the model Φ_{12} is optimal for English. In addition, the model Φ^p_{11} gives reasonable performance for both languages (and even outperforms Φ^p_{12} with respect to labeled exact match for English). We will therefore keep all three models when we go on to add dependency features in the next section.

5.3.2 Dependency Structure

The use of dynamic dependency type features for making parsing decisions is largely unchartered territory in the literature, which is due to the fact that data-driven dependency parsers normally do not construct labeled dependency graphs and therefore do not have access to dependency type labels during parsing. In this section, we investigate the effect of adding to the part-of-speech models the previously assigned dependency types of the top token ($d(\sigma_0)$), its leftmost and rightmost dependents ($d(l(\sigma_0))$, $d(r(\sigma_0))$), and the leftmost dependent of the next token ($d(l(\tau_0))$) (cf. section 4.2.4). The results are shown in table 5.3.

The most important observation is that adding the dependency type of the top token (Φ^d_{000}) gives a substantial increase in parsing accuracy across the board, for both languages, all part-of-speech models, and all evaluation metrics. Adding information about other dependencies has a more marginal impact, and the leftmost dependent of the top token even has a negative effect on attachment score for English. By and large, however, the models that incorporate all dependency features (Φ^d_{111}) are the best performing models for both languages, regardless of part-of-speech context, a result that holds without exception for the exact match criterion.

These results can be related to research on constituency-based parsing in the following way. The positive effect of including the dependency type of

Table 5.3. Accuracy as a function of dependency type features; AS: attachment score, EM: exact match; U: unlabeled, L: labeled; Swedish (cross-validation), English (section 00)

Model	Swedish				English			
	AS		EM		AS		EM	
	U	L	U	L	U	L	U	L
Φ^p_{11}	77.9	70.9	27.2	18.0	77.7	74.8	13.4	10.2
$\Phi^p_{11} + \Phi^d_{000}$	80.3	72.5	29.2	19.0	80.9	77.7	16.3	11.7
$\Phi^p_{11} + \Phi^d_{100}$	80.3	72.8	29.6	20.3	80.5	77.5	16.3	11.8
$\Phi^p_{11} + \Phi^d_{010}$	81.2	73.4	31.6	19.9	81.2	78.1	18.1	12.9
$\Phi^p_{11} + \Phi^d_{001}$	81.6	73.9	31.1	19.8	**81.8**	**78.8**	18.9	13.9
$\Phi^p_{11} + \Phi^d_{111}$	**82.3**	**74.9**	**33.6**	**22.6**	81.6	78.7	**19.5**	**15.0**
Φ^p_{21}	78.4	71.0	28.4	18.6	77.7	74.8	14.2	10.7
$\Phi^p_{21} + \Phi^d_{000}$	80.3	72.4	30.0	19.3	80.6	77.3	16.6	12.0
$\Phi^p_{21} + \Phi^d_{100}$	80.2	72.6	29.5	20.0	80.3	77.2	16.4	12.1
$\Phi^p_{21} + \Phi^d_{010}$	81.4	73.5	32.0	20.2	81.2	77.9	19.0	13.4
$\Phi^p_{21} + \Phi^d_{001}$	81.6	73.8	31.6	20.0	81.3	78.3	19.2	14.4
$\Phi^p_{21} + \Phi^d_{111}$	**82.2**	**74.8**	**33.4**	**22.1**	**81.5**	**78.5**	**19.8**	**15.2**
Φ^p_{12}	77.4	70.1	26.6	17.8	79.0	76.1	14.4	10.0
$\Phi^p_{12} + \Phi^d_{000}$	80.4	72.4	29.2	18.9	82.4	79.2	18.4	12.4
$\Phi^p_{12} + \Phi^d_{100}$	80.5	72.8	29.1	19.7	82.3	79.3	19.2	14.1
$\Phi^p_{12} + \Phi^d_{010}$	81.5	73.6	31.9	19.9	83.1	80.0	20.5	14.7
$\Phi^p_{12} + \Phi^d_{001}$	81.8	73.9	31.4	20.0	**83.5**	**80.5**	21.2	15.0
$\Phi^p_{12} + \Phi^d_{111}$	**82.5**	**75.1**	**33.5**	**22.2**	83.4	**80.5**	**21.9**	**17.0**

the top token mirrors the observation that grandparent nodes are important in phrase structure parsing (Collins, 1999; Charniak, 2000). Both types contribute to disambiguation by clarifying the grammatical function of the node in question. For example, observing that the top token has the dependency type *subject* or *object* can be helpful in exactly the same way as knowing that a particular NP is immediately dominated by an S node or a VP node in phrase structure parsing. The mixed evidence concerning dependents of the target nodes, in particular the rightmost dependent of the top token and the leftmost dependent of the next token, has bearing on the issue of whether it is relevant to consider sibling nodes in phrase structure parsing. For instance, while Charniak (2000) conditions on up to four preceding siblings, Collins (1997, 1999) achieves similar accuracy without including any information of this kind.

Comparing the different part-of-speech models, we see that the model Φ^p_{12} still gives the highest accuracy for English. However, after the addition of

Table 5.4. Accuracy as a function of part-of-speech context (with dependency type features); AS: attachment score, EM: exact match; U: unlabeled, L: labeled; Swedish (cross-validation), English (section 00)

Model	Swedish				English			
	AS		EM		AS		EM	
	U	L	U	L	U	L	U	L
$\Phi^p_{12} + \Phi^d_{111}$	**82.5**	**75.1**	**33.5**	**22.2**	83.4	80.5	21.9	17.0
$\Phi^p_{13} + \Phi^d_{111}$	82.4	74.9	33.2	21.9	**83.8**	**80.8**	22.6	17.1
$\Phi^p_{14} + \Phi^d_{111}$	82.1	74.5	32.5	21.3	83.6	80.7	**23.0**	**17.3**
$\Phi^p_{15} + \Phi^d_{111}$	82.0	74.5	31.6	20.9	83.5	80.6	22.8	17.2
$\Phi^p_{16} + \Phi^d_{111}$	82.0	74.4	31.4	20.5	83.7	**80.8**	22.7	17.0

the dependency type features, this model also gives the highest accuracy for Swedish with respect to attachment score. And with respect to exact match, it is now the model Φ^p_{11}, not Φ^p_{21}, that gets the top score. This result can probably be explained by the fact that the head of the top token, if present in a given configuration, is always identical to the second topmost token on the stack, which means that there is considerable redundancy in the information given by the features $p(\sigma_1)$ and $d(\sigma_0)$. In order to further investigate the interaction of part-of-speech features and dependency type features, we have also repeated part of the experiment presented in table 5.2, increasing the lookahead with respect to part-of-speech features, but this time in the presence of all dependency type features. The results are given in table 5.4.

For English, it is no longer detrimental to increase the lookahead. The best performing models now include three (Φ^p_{13}) or four (Φ^p_{14}) extra input tokens, and it is possible to increase the lookahead up to six tokens without more than marginal degradation. This seems to indicate that the positive effect of an increased lookahead is dependent on having a more richly articulated feature model to start from. For Swedish, we can observe similar results, although the best performing model is still limited to two lookahead tokens, and the degradation with increasing lookahead is somewhat steeper. This can probably be explained as a problem of sparse data, since the Swedish training corpus is one order of magnitude smaller than the English one. On the whole, however, we see very small differences as a function of varying the lookahead above one token.

5.3.3 Lexicalization

Lexicalization is usually considered a necessary condition for accurate disambiguation in data-driven parsing. Table 5.5 shows the result of adding lexical features to a subset of the models considered in previous sections. The upper

Table 5.5. Accuracy as a function of lexical features (with part-of-speech context and dependency features); AS: attachment score, EM: exact match; U: unlabeled, L: labeled; Swedish (cross-validation), English (section 00)

Model	Swedish				English			
	AS		EM		AS		EM	
	U	L	U	L	U	L	U	L
$\Phi^p_{12} + \Phi^d_{111}$	82.5	75.1	33.5	22.2	83.4	80.5	21.9	17.0
$\Phi^p_{12} + \Phi^d_{111} + \Phi^w_{10}$	83.7	78.5	34.9	26.3	85.2	83.2	25.7	22.5
$\Phi^p_{12} + \Phi^d_{111} + \Phi^w_{01}$	84.6	78.7	37.2	25.3	85.0	82.5	25.7	20.2
$\Phi^p_{12} + \Phi^d_{111} + \Phi^w_{11}$	85.6	81.5	39.1	30.2	86.6	84.8	29.9	26.2
$\Phi^p_{12} + \Phi^d_{111} + \Phi^w_{21}$	85.7	81.5	39.4	30.3	86.9	85.1	30.1	26.4
$\Phi^p_{12} + \Phi^d_{111} + \Phi^w_{12}$	**85.9**	**81.7**	39.4	30.1	**87.3**	85.4	30.1	26.2
$\Phi^p_{12} + \Phi^d_{111} + \Phi^w_{22}$	**85.9**	81.6	**39.8**	**30.4**	**87.3**	**85.6**	**31.1**	**27.7**
$\Phi^p_{11} + \Phi^d_{111} + \Phi^w_{22}$	**85.9**	**81.7**	39.5	**30.4**	86.3	84.6	28.8	25.7
$\Phi^p_{12} + \Phi^d_{111} + \Phi^w_{22}$	**85.9**	81.6	**39.8**	**30.4**	87.3	85.6	31.1	**27.7**
$\Phi^p_{13} + \Phi^d_{111} + \Phi^w_{22}$	85.8	81.5	39.6	30.0	**87.6**	**85.8**	**31.2**	27.3
$\Phi^p_{14} + \Phi^d_{111} + \Phi^w_{22}$	85.7	81.4	39.1	29.6	87.5	85.7	31.0	27.0
$\Phi^p_{15} + \Phi^d_{111} + \Phi^w_{22}$	85.5	81.1	39.0	29.4	87.5	85.7	31.0	26.8
$\Phi^p_{16} + \Phi^d_{111} + \Phi^w_{22}$	85.4	80.9	38.6	29.0	87.4	85.7	30.2	26.3

part of the table is based on the model $\Phi^p_{12} + \Phi^d_{111}$, incorporating all dependency type features and a lookahead of two tokens, and shows the effect of adding the word form of the top token (feature $w(\sigma_0)$, model Φ^w_{10}), the word form of the next token (feature $w(\tau_0)$, model Φ^w_{01}), and the word forms of both target tokens (model Φ^w_{11}), followed by the further addition of the head of the top token (feature $w(h(\sigma_0))$, model Φ^w_{21}), one lookahead token (feature $w(\tau_1)$, model Φ^w_{12}), and both of these tokens (model Φ^w_{22}) (cf. section 4.2.4).

First of all, we see that the benefit of lexicalization is evident for inductive dependency parsing as well as for other data-driven approaches. With very few exceptions, parsing accuracy increases steadily with the addition of each lexical feature, although the magnitude of the improvement quickly decreases. However, there is a clear difference in this respect between Swedish, where improvement is marginal for any features after the first two, and English, where accuracy increases more steadily, especially with respect to the exact match metrics. Again, this difference can probably be explained by reference to data sparseness for Swedish, which is even more sensitive for lexical features than for part-of-speech features.

The lower part of table 5.5 again explores the variation of lookahead, while keeping the number of lexical features constant at the maximum (Φ^w_{22}). We see that increasing the lookahead to three tokens is beneficial for English but

not for Swedish, while decreasing it to one has a strong negative effect for English but is barely noticeable for Swedish, where the labeled attachment score even goes up. These results give further support to the assumption that the Swedish data is too sparse to make effective use of the discriminative power of a more complex feature model.

Let us summarize the results concerning the influence of the feature model on parsing accuracy. For part-of-speech features, it is essential to include a context of at least one token in each direction, in addition to the top token and the next token. Increasing the context further on the stack side appears to give no improvement, while increasing the lookahead is beneficial to the extent that there are sufficient quantities of training data available. Dependency type features have a positive influence on parsing accuracy, especially with respect to the exact match metrics, and the dependency type of the top token is the single most important feature. Lexicalization, finally, has a substantial positive effect on parsing accuracy, but the addition of lexical features over and above the word form of the target tokens is dependent on the availability of large quantities of training data.

Given the results so far, we will now restrict our attention to a subset of the models considered, which will be evaluated with respect to efficiency and learning curves. In section 5.4, an even smaller subset will be used as a basis for the exploration of learning algorithm parameters. Altogether, we will consider five models, in order of increasing complexity:

$$
\begin{aligned}
B &= \Phi^{p}_{00} \\
P &= \Phi^{p}_{12} \\
D &= \Phi^{p}_{12} + \Phi^{d}_{111} \\
L_2 &= \Phi^{p}_{12} + \Phi^{d}_{111} + \Phi^{w}_{11} \\
L_4 &= \Phi^{p}_{12} + \Phi^{d}_{111} + \Phi^{w}_{22}
\end{aligned}
$$

The first model (B) is the baseline model, where the only features included are the parts-of-speech of the target tokens. The second model (P) is the best pure part-of-speech model for English, with a context of one stack token and two lookahead tokens. The third model (D) is the model obtained by adding all dependency type features to the P model. The last two models (L_2 and L_4) are lexicalized models, based on the D model, and incorporating two and four lexical features, respectively. Note that L_4 is not the best performing model for English, where the model $\Phi^{p}_{13} + \Phi^{d}_{111} + \Phi^{w}_{22}$ has a higher accuracy on the validation set. We will return to this model in the final evaluation in section 5.5. Table 5.6 gives an overview of the accuracy of the five selected models.

5.3.4 Efficiency

One of the the main tenets of this study is that, even though accuracy is the single most important evaluation criterion in natural language parsing,

Table 5.6. Accuracy as a function of feature model (selection); AS: attachment score, EM: exact match; U: unlabeled, L: labeled; Swedish (cross-validation), English (section 00)

Model	Swedish				English			
	AS		EM		AS		EM	
	U	L	U	L	U	L	U	L
B	71.0	64.5	15.6	12.3	58.4	56.2	3.7	3.1
P	77.4	70.1	26.6	17.8	79.0	76.1	14.4	10.0
D	82.5	75.1	33.5	22.2	83.4	80.5	21.9	17.0
L_2	85.6	81.5	39.1	30.2	86.6	84.8	29.9	26.2
L_4	85.9	81.6	39.8	30.4	87.3	85.6	31.1	27.7

it cannot be regarded in isolation from other requirements, such as robustness, disambiguation and efficiency. We have chosen to treat robustness and disambiguation as absolute requirements, but we still have to consider the trade-off between accuracy and efficiency. Increasing model complexity tends to correlate positively with parsing accuracy (if overfitting can be avoided) but negatively with efficiency. It is therefore important to see how the differences in accuracy observed so far correlate with differences in efficiency.

Table 5.7 shows the evaluation of our five selected models with respect to the efficiency metrics defined in section 5.1.1. The first column reports training time (T); the next three columns show parsing time, expressed as total parsing time (P), mean parsing time per sentence (S), and mean number of words parsed per second (W); the fifth column gives the memory consumption (M). Time is expressed in seconds for T and P, milliseconds for S, and memory consumption is reported in megabytes. It should be remembered that the size of both training and test corpora are about one order of magnitude larger for English than for Swedish (cf. table 5.1). (The results for Swedish are as usual the mean score of a nine-fold cross-validation.)

Starting with the training time, we see that the memory-based approach is very efficient, as can be expected, with training times of three to four minutes for the most complex models on section 2-21 of the Wall Street Journal section of the Penn Treebank. We also see that training time scales very well, with an approximately linear growth both with respect to the number of features (2, 5, 9, 11, and 13 for the selected models) and with respect to the size of the training corpus (one order of magnitude difference between Swedish and English).

With respect to parsing efficiency, the memory-based approach has a drawback in that all training instances have to be kept in memory during parsing. Even with the optimized storing and indexing provided by TiMBL, this is bound to show up in the measurements of time and memory consumption. Nevertheless, we see that although the English parsers generally require more

Table 5.7. Efficiency as a function of feature model; T: training time (s), P: parsing time (s), S: mean parsing time per sentence (ms), W: mean number of words parsed per second, M: memory requirements during parsing (MB)

Model	Swedish					English				
	T	**P**	**S**	**W**	**M**	**T**	**P**	**S**	**W**	**M**
B	4.1	3.4	5.4	2753.7	4	42.5	16.3	8.5	2844.5	4
P	7.0	5.3	8.4	1771.7	14	73.6	63.4	33.0	732.8	61
D	9.9	11.3	17.9	831.7	19	109.7	72.2	37.6	643.1	96
L_2	14.8	158.7	251.5	59.2	42	178.2	971.8	505.9	47.8	298
L_4	18.2	549.3	870.5	17.1	56	226.8	2861.3	1489.5	16.2	420

memory than the Swedish ones, the ratio is always less than 10:1 for the same feature model. And with respect to parsing time, there is very little difference at all when considering the mean number of words per second, except for the model P, which is surprisingly slow for English. The difference in parsing time per sentence can largely be explained by the fact that the mean sentence length is longer for English, and the difference in total parsing time of course by the different sizes of the test corpora.

The picture that clearly emerges from table 5.7 is that model complexity is the most important factor with respect to parsing time. For both languages, we see that parsing speed drops by one order of magnitude with the introduction of lexical features. Comparing total parsing times, the ratio between L_2 and D is about 14:1 for both languages. The crucial difference between lexicalized and non-lexicalized models is not the number of features, but the number of values per feature. Whereas both part-of-speech features and dependency features have value sets with a cardinality ranging from about 10 to 50, the number of values for lexical features are in the order of 10^4–10^5. Adding more lexical features slows down parsing even further, as can be seen in the difference between L_4 and L_2, which is about 3:1 and which is caused by the addition of two more lexical features. This can be compared with the difference between D and P, which is about 2:1 for Swedish and barely noticeable for English, despite the addition of four new features.

The decrease in efficiency with increasing model complexity has to be seen in relation to the gain in accuracy. Going back to table 5.6, we note that the addition of four dependency features from the P model to the D model gives a gain in accuracy of 4–5 percentage points for attachment score, and even more for exact match, with less than a 50% drop in parsing speed. By contrast, the addition of two more lexical features from the L_2 model to the L_4 model improves attachment score by 0.1–0.8 percentage points while reducing parsing speed by about 70%. Whether we are willing to pay this price or not is dependent on the requirements of our applications, which may put different constraints on the lowest acceptable accuracy or efficiency, but it illustrates

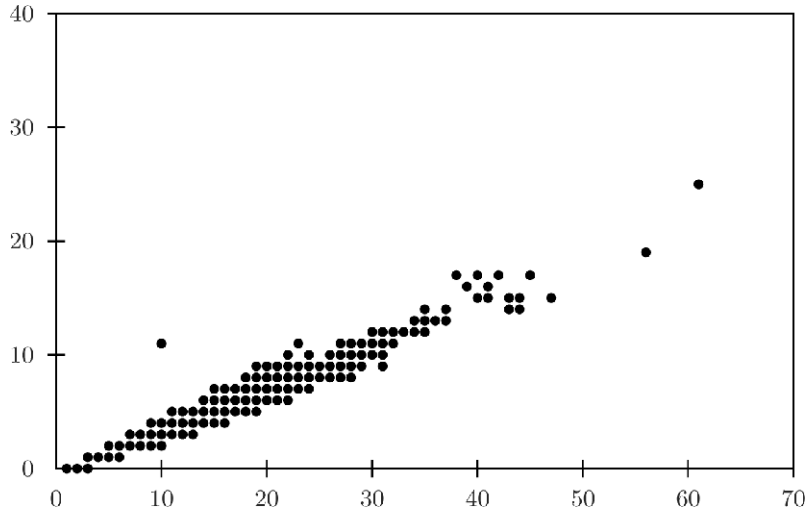


Fig. 5.10. Parsing time (ms) as a function of sentence length (number of words); Model B; Swedish, section 9 (630 data points)

the inevitable trade-off between accuracy and efficiency in data-driven parsing and the consequent need for joint optimization.

To complete the analysis of efficiency, we will consider the relation between sentence length and parsing time. In section 3.4, we established that the time complexity of the parsing algorithm is $O(n)$, where n is the number of words in the sentence, provided that transitions can be performed in constant time. Using a memory-based classifier to predict the next transition allows us to perform transitions in time that is constant in the number of words for a given feature model and training data set, but the size of this constant clearly depends on the complexity of the feature model and the size of the training data set. Moreover, because of the optimized storage and indexing techniques used in TiMBL, classifying a new instance may or may not require an exhaustive search of the instance base, which means that we can expect a larger variation in classification time for more complex models.

Figures 5.10–5.12 show a plot of parsing time (ms) as a function of sentence length for the models B, D and L_2 applied to section 9 of the Swedish treebank (after training on sections 1–8). The plot is based on 630 data points, each point representing one sentence. Figures 5.13–5.15 show the same plots for section 00 of the English treebank (after training on section 02–21 as usual), this time including 1920 sentences.

The linear behavior of the parser is most clearly discernible for the B model, where the time spent on classification is relatively small but also relatively constant given the very simple feature model. With increasing model complexity, the correlation between sentence length and parsing time becomes

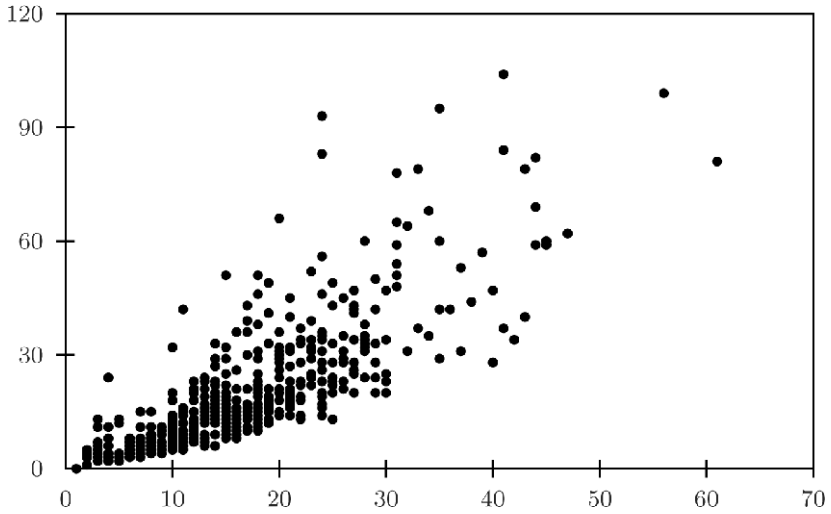


Fig. 5.11. Parsing time (ms) as a function of sentence length (number of words); Model D; Swedish, section 9 (630 data points)

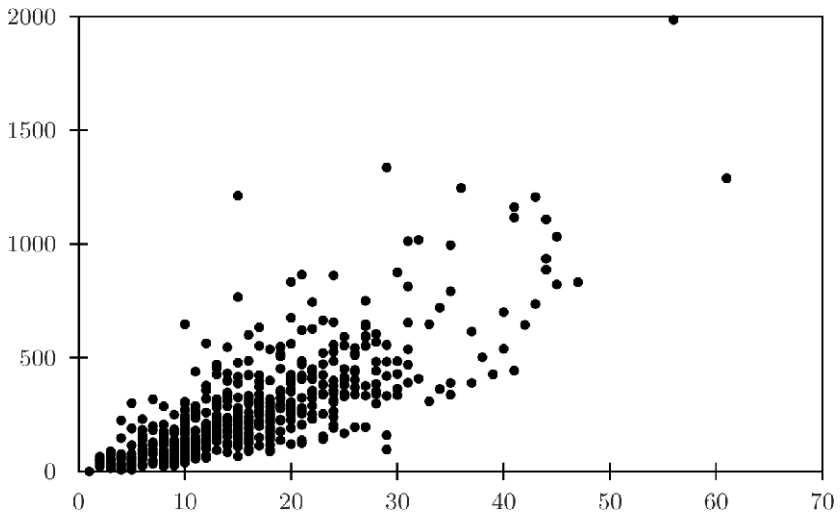


Fig. 5.12. Parsing time (ms) as a function of sentence length (number of words); Model L_2; Swedish, section 9 (630 data points)

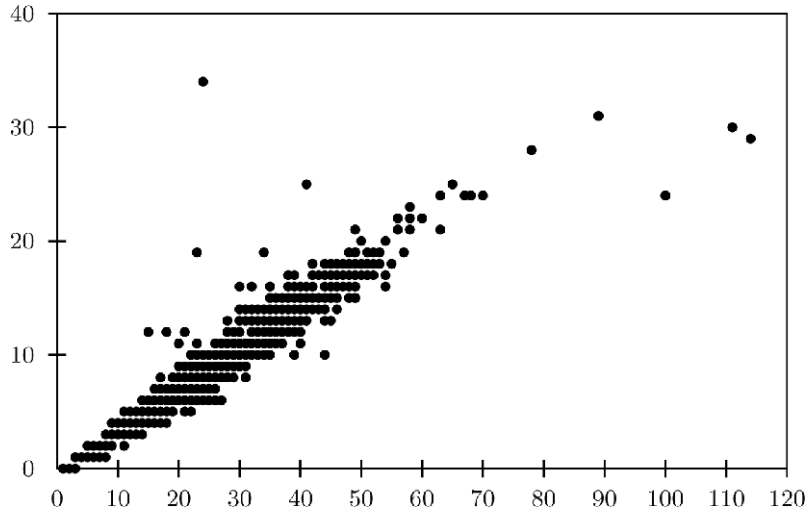


Fig. 5.13. Parsing time (ms) as a function of sentence length (number of words); Model *B*; English, section 00 (1920 data points)

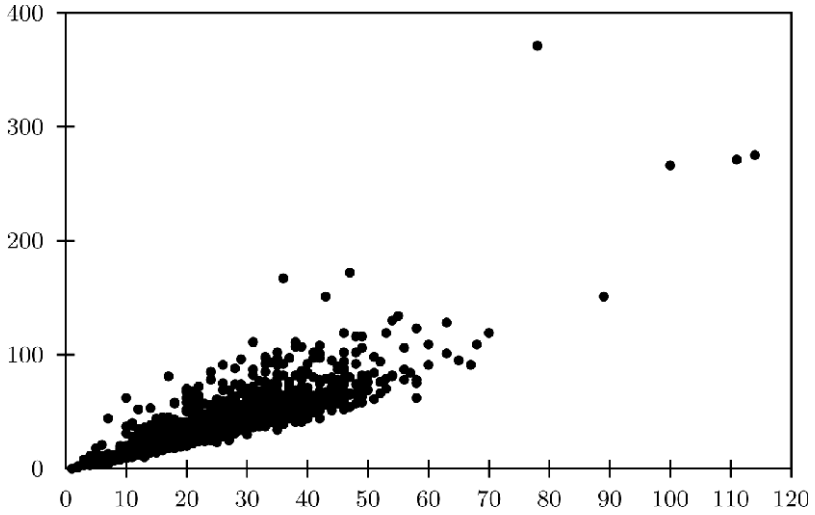


Fig. 5.14. Parsing time (ms) as a function of sentence length (number of words); Model *D*; English, section 00 (1920 data points)

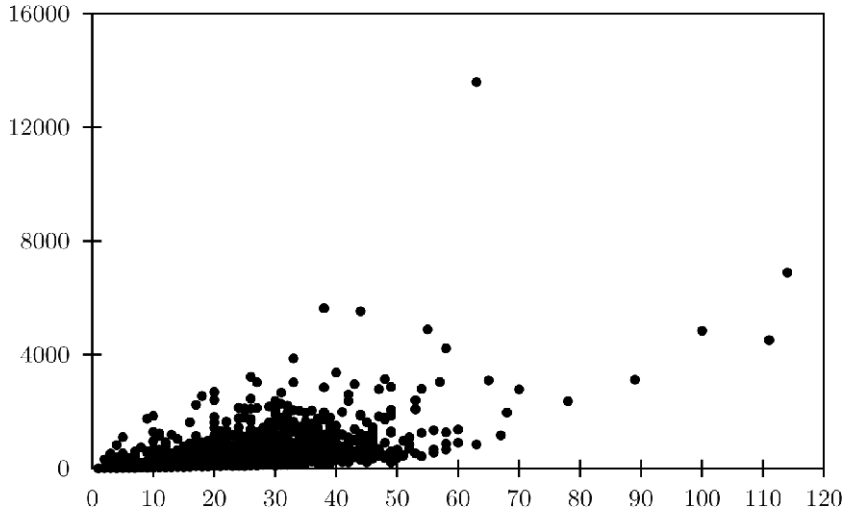


Fig. 5.15. Parsing time (ms) as a function of sentence length (number of words); Model L_2; English, section 00 (1920 data points)

increasingly noisy, as classification time begins to dominate parsing time. Even though the plots for some of the more complex models do not quite resemble straight lines, the data is very sparse for high values of the sentence length variable, and there are no grounds to reject the assumption that parsing time remains linearly related to sentence length even for more complex models, although the variance clearly increases due to the increased variance in classification time.

5.3.5 Learning Curves

The final aspect of feature models that will be considered in this evaluation is their sensitivity to the amount of training data available, which can be assessed by considering their learning curves. Figures 5.16–5.17 plot the accuracy of the feature models D and L_2 for Swedish as a function of the size of the training corpus. Figure 5.16 depicts the development of attachment score (labeled and unlabeled), while figure 5.17 shows exact match (labeled and unlabeled). The training corpus varies from 1 to 8 sections, and the values depicted are the mean of a nine-fold cross-validation as usual. Figures 5.18–5.19 give the same type of information for English, except that each increment of the training corpus represents two sections, i.e., one tenth of the entire training corpus, and measurements are only based on the validation set, section 00.

First of all, we may note that the lexicalized L_2 model generally has a steeper learning curve than the non-lexicalized D model, which is only to be expected given that the data is much more sparse for lexical features than

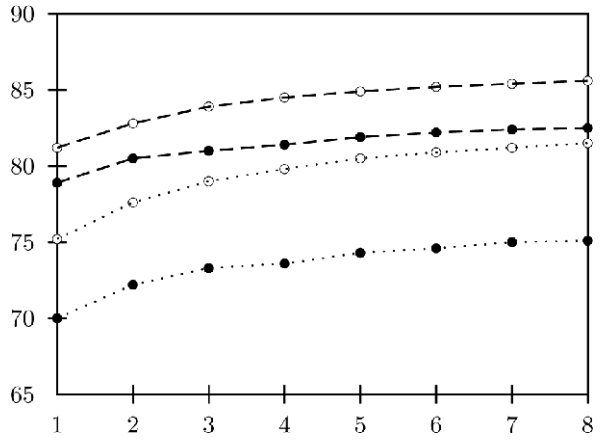


Fig. 5.16. Learning curve for attachment score; AS_U (dashed) and AS_L (dotted); models D (•) and L_2 (○); Swedish

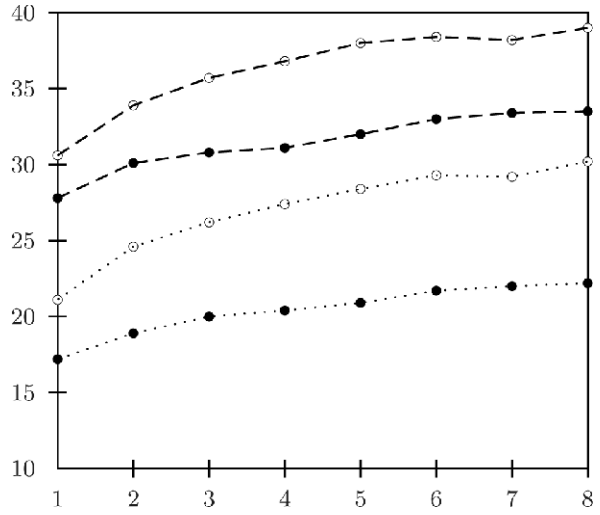


Fig. 5.17. Learning curve for exact match; EM_U (dashed) and EM_L (dotted); models D (•) and L_2 (○); Swedish

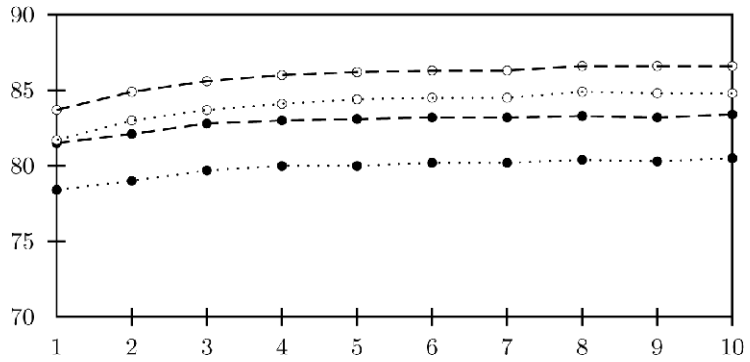


Fig. 5.18. Learning curve for attachment score; AS_U (dashed) and AS_L (dotted); models D (•) and L_2 (○); English

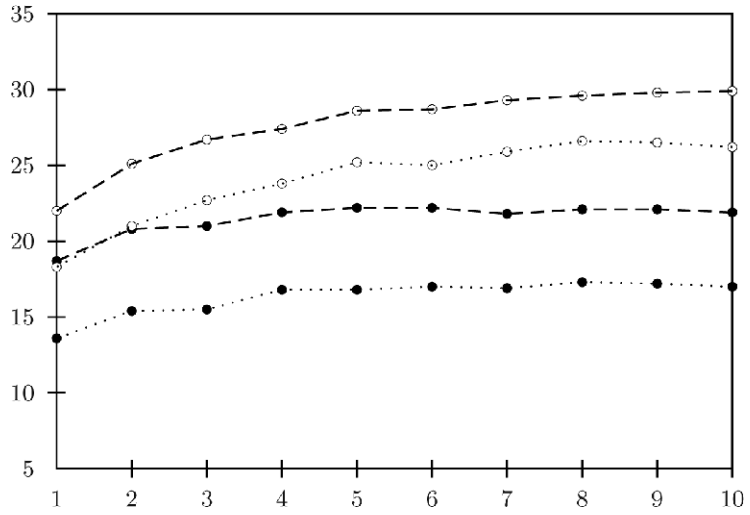


Fig. 5.19. Learning curve for exact match; EM_U (dashed) and EM_L (dotted); models D (•) and L_2 (○); English

for part-of-speech and dependency features. At the same time, it is worth pointing out that the L_2 model outperforms the D model at all data points, i.e., even when only training on a single section of the Swedish treebank, which only contains about 600 sentences and 10,000 tokens. This indicates that the analogy-based smoothing provided by the memory-based learner works well even for sparse data sets.

If we compare the two types of evaluation metrics, we see that exact match has a steeper learning curve than attachment score, which might be taken to show that the former is a more discriminative metric. Especially the L_2 model shows a steady increase in exact match accuracy for both languages.

Comparing the two languages, finally, it is evident that the curves are steeper for Swedish than for English. This is a direct reflection of the different sizes of the data sets involved, where the complete training corpus for Swedish is of about the same size as the smallest fraction considered for the English. In this way, the Swedish curves could almost be considered as a zoom-in of the first data point of the English curves. Some of the curves for English are really very flat, especially for attachment score and the D model. This is both good news and bad news. It is good news in the sense that (almost) optimal accuracy can often be obtained with only half of the training data, which can make parsing more efficient. But it is bad news in the sense that accuracy is unlikely to improve with the addition of more training data. The picture is somewhat different for the exact match metrics, where especially the L_2 model continues to improve up until the maximum size of the training corpus.

5.4 Learning Algorithm Parameters

Having examined the influence of different feature models on both accuracy and efficiency, we will now turn to the role of the learning algorithm, which in our case means exploring the parameter space of memory-based learning and classification, as provided by the TiMBL system. It is important to remember that feature selection and parameter optimization can be highly dependent on each other (Daelemans and Hoste, 2002; Daelemans et al., 2003). In principle, we should therefore explore the combined space of feature models and learning algorithm parameters in order to arrive at a truly optimal combination. In practice, this is often impossible because of the combinatorial effect, and the standard approach is therefore to keep one factor constant while varying the other systematically, which is also the methodology adopted in this study. However, because the factors are not independent, we need to make sure that the factor kept constant has a nearly optimal value, or the results can be highly misleading. This is the reason that we did not use the default settings of TiMBL when exploring different feature models in the preceding section, but instead used parameter settings that had been found optimal in previous studies. By the same reasoning, we will not use arbitrary feature models as our basis for parameter optimization in this section, but will limit our attention

to two of the best models. More precisely, we will study the D model, which was the best performing non-lexicalized model in the preceding section, and the L_2 model, which was a model that gave close to optimal performance for both Swedish and English and that strikes a good balance between accuracy and efficiency.

5.4.1 Neighbor Space and Distance Metric

Two fundamental parameters of the k-NN classification provided by memory-based learning are the number k of nearest neighbors (distances) taken into account and the metric used to compute the distance between instances. In our first experiment, we compare the Overlap and MVDM metrics, while varying the value of k from 1 to 9 in increments of 2.[2] The results are shown in table 5.8.

Starting with the Overlap metric, we see that performance is reasonable but not optimal with a k value of 1 but degrades drastically as the k value increases. The reason for this behavior is that, with the simple Overlap metric, there will typically be an abundance of ties in nearest neighbor position, and increasing the k value will lead to larger and larger portions of the instance base being used to classify each instance (Daelemans and Van den Bosch, 2005). This means that the set of local neighborhood approximations will eventually give way to a globally defined approximation, which simply assigns the overall majority class to every instance.

In terms of dependency parsing, this means that the parser will select the SHIFT transition in every configuration, constructing a dependency graph where every token is attached to the special root node. The attachment score of such a parser is simply the proportion of tokens attached to the special root node in the gold standard treebank, which happens to be 7.0% for the Swedish treebank and 4.7% for the English treebank. The non-lexicalized D model, which has fewer features and fewer values per feature, reaches this stage already at $k = 7$, while the lexicalized L_2 model degrades at a slightly slower pace. The degradation is accelerated by the fact that TIMBL uses the k nearest distances, rather than the k nearest neighbors, but the result will eventually be the same anyway.

Using the more sophisticated MVDM metric gives a completely different picture. Increasing the k value from 1 to 3 has a positive effect for both models with respect to all metrics, but further changes have very little effect overall. The highest accuracy is found at $k = 3$ for the D model in both languages. For the L_2 model, the optimum appears to be $k = 5$ for Swedish and $k = 7$ for English, although the differences are generally small. In the following experiments, we will therefore use $k = 3$ for the D model but use both $k = 5$ and $k = 7$ for the L_2 model.

[2] In order to avoid ties, it is recommended to use odd integers as values of k.

Table 5.8. Accuracy as a function of distance metric (Overlap, MVDM) and k value; AS: attachment score, EM: exact match; U: unlabeled, L: labeled; Swedish (cross-validation), English (section 00)

Model	**Metric**	k	**Swedish**				**English**			
			AS		**EM**		**AS**		**EM**	
			U	**L**	**U**	**L**	**U**	**L**	**U**	**L**
D	Overlap	1	**78.4**	**70.0**	**29.6**	**20.0**	**81.9**	**78.8**	**20.4**	**15.4**
		3	63.0	52.4	18.4	12.0	65.4	61.1	7.1	4.5
		5	31.7	24.1	7.4	4.9	37.7	34.3	1.2	1.0
		7	7.0	7.0	3.8	3.8	4.7	4.7	0.4	0.4
		9	7.0	7.0	3.8	3.8	4.7	4.7	0.4	0.4
	MVDM	1	80.8	72.7	31.8	20.9	82.9	79.9	20.9	16.1
		3	**82.5**	75.0	**33.1**	**21.5**	**83.4**	**80.5**	**21.9**	**17.1**
		5	**82.5**	**75.1**	32.8	20.7	83.1	80.1	21.8	16.5
		7	**82.5**	**75.1**	33.0	21.0	82.8	79.8	21.7	16.1
		9	82.2	74.8	32.3	20.7	82.7	79.6	20.9	15.0
L_2	Overlap	1	**82.2**	**76.2**	**34.3**	**24.7**	**85.2**	**83.0**	**27.0**	**22.5**
		3	72.5	63.5	22.8	14.8	74.1	71.1	12.0	9.0
		5	44.9	34.8	11.2	6.7	51.0	47.5	2.7	1.5
		7	9.5	8.9	4.0	3.8	19.7	18.4	0.5	0.4
		9	7.0	7.0	3.8	3.8	4.7	4.7	0.4	0.4
	MVDM	1	83.1	78.6	34.9	26.0	85.1	83.2	26.2	23.4
		3	84.7	80.6	37.2	28.6	86.2	84.5	29.2	25.8
		5	**84.8**	**80.8**	**37.6**	**29.1**	**86.3**	84.5	**29.7**	25.2
		7	84.7	80.7	37.4	29.0	**86.3**	**84.6**	**29.7**	**26.3**
		9	84.7	80.6	37.1	28.8	86.2	84.4	29.6	26.0

While the MVDM metric is in most cases superior to the Overlap metric already at $k = 1$ and improves with larger k values, it is also more sensitive to data sparseness. TiMBL therefore provides a back-off from MVDM to Overlap through a frequency threshold l, which means that the distance metric switches from MVDM to Overlap whenever one or both values compared occur less than l times in the training data. Table 5.9 shows the effect of increasing this threshold from 1 to 5 in increments of 1 for the D model with $k = 3$ and the L_2 model with $k = 5$ and $k = 7$. We see that the D model is completely insensitive to this parameter, with identical results for all settings, indicating that data sparseness is not a problem for the non-lexicalized D model, not even for Swedish. For the L_2 model, we see a small but steady improvement with a higher threshold, especially for Swedish with a more limited amount of training data. For $k = 5$ the optimal threshold appears to be 3 for both

Table 5.9. Accuracy as a function of switching threshold l (MVDM to Overlap); AS: attachment score, EM: exact match; U: unlabeled, L: labeled; Swedish (cross-validation), English (section 00)

Model	k	l	Swedish				English			
			AS		EM		AS		EM	
			U	L	U	L	U	L	U	L
D	3	1	82.5	75.0	33.7	21.5	83.4	80.5	21.9	17.1
		2	82.5	75.0	33.7	21.5	83.4	80.5	21.9	17.1
		3	82.5	75.0	33.7	21.5	83.4	80.5	21.9	17.1
		4	82.5	75.0	33.7	21.5	83.4	80.5	21.9	17.1
		5	82.5	75.0	33.7	21.5	83.4	80.5	21.9	17.1
L_2	5	1	84.8	80.8	37.6	29.1	86.3	84.5	29.7	25.9
		2	85.2	81.1	38.4	29.3	**86.5**	84.7	29.9	**26.1**
		3	85.3	**81.2**	**38.8**	**29.7**	**86.5**	**84.8**	29.9	25.9
		4	**85.4**	81.1	**38.8**	29.6	**86.5**	84.7	**30.0**	26.0
		5	85.3	81.0	38.5	29.3	**86.5**	84.7	**30.0**	25.8
	7	1	84.7	80.7	37.4	29.0	86.3	84.6	29.7	26.3
		2	85.0	80.9	38.2	29.1	86.5	84.7	29.9	**26.5**
		3	85.1	**81.0**	38.4	**29.2**	**86.6**	**84.8**	29.9	26.3
		4	**85.2**	**81.0**	**38.5**	**29.2**	**86.6**	84.6	29.9	26.2
		5	85.1	80.9	38.3	**29.2**	**86.6**	**84.8**	**30.0**	26.2

languages, which confirms earlier experiments (Nivre et al., 2004; Nivre and Scholz, 2004). For $k = 7$ the best results are obtained with a threshold of 5 for English, whereas for Swedish accuracy is consistently worse than for $k = 5$. In the following, we will therefore consider $k = 5$, $l = 3$ to be optimal for Swedish but consider both $k = 5$, $l = 3$ and $k = 7$, $l = 5$ for English.

5.4.2 Weighting Schemes

In addition to distance metric and k value, the k-NN classification may be tuned by different weighting schemes. In this section, we will consider two types of weighting. On the one hand, we have applied feature weighting with Information Gain (IG) and Gain Ratio (GR). On the other hand, we have used distance-weighted class voting with inverse distance (ID) and inverse-linear (IL) weighting. The results of these experiments are reported in table 5.10.

The overall tendency is that these weighting schemes have a negative influence on parsing accuracy. One possible explanation is that the MVDM metric in itself has a feature weighting effect, as observed in section 4.3, and that additional weighting will therefore result in overfitting. For the D model, the best results are obtained with IG feature weighting, which gives marginally

Table 5.10. Accuracy as a function of weighting scheme; FW: feature weighting (GR: gain ratio, IG: information gain); DWCV: distance weighted class voting (ID: inverse distance, IL: inverse linear); AS: attachment score, EM: exact match; U: unlabeled, L: labeled; Swedish (cross-validation), English (section 00)

Model	k	l	FW	DWCV	**Swedish**				**English**			
					AS		**EM**		**AS**		**EM**	
					U	**L**	**U**	**L**	**U**	**L**	**U**	**L**
D	3	1	–	–	**82.5**	75.0	**33.7**	21.5	**83.4**	**80.5**	21.9	**17.1**
			GR	–	82.1	74.9	32.9	**22.3**	83.0	80.1	21.7	17.0
			IG	–	82.2	**75.1**	33.0	21.5	**83.4**	**80.5**	**22.0**	**17.1**
			–	ID	82.0	74.4	33.3	21.8	**83.4**	80.4	21.7	**17.1**
			–	IL	81.0	73.0	32.2	21.3	82.9	79.9	21.5	16.6
L_2	5	3	–	–	85.3	81.2	38.8	29.7	86.5	**84.8**	**29.9**	25.9
			GR	–	84.6	80.4	37.1	28.7	86.0	84.1	29.0	25.5
			IG	–	83.8	79.8	34.6	26.7	86.0	84.5	27.3	24.4
			–	ID	**85.6**	**81.5**	**39.1**	**30.2**	**86.6**	**84.8**	**29.9**	**26.2**
			–	IL	85.0	80.6	38.1	28.9	86.0	84.2	28.3	25.0
	7	5	–	–	–	–	–	–	86.6	84.8	30.0	26.2
			GR	–	–	–	–	–	85.7	83.8	28.2	24.9
			IG	–	–	–	–	–	86.0	84.5	27.5	24.9
			–	ID	–	–	–	–	**86.8**	**85.0**	**30.3**	**26.9**
			–	IL	–	–	–	–	86.5	84.7	29.4	26.0

higher AS_L for Swedish and EM_U for English, but also lower AS_U and EM_U for Swedish. By an appeal to Occam's razor we will therefore conclude that the optimal parameter settings for non-lexicalized models are as follows:

1. Number of nearest distances: $k = 3$
2. Distance metric: MVDM with $l = 1$
3. Feature weighting: None
4. Distance-weighted class voting: None

Turning to L_2, we find that distance-weighted class voting with ID weighting has a consistent positive effect, whereas all the other weighting schemes are detrimental. We also see that, with the addition of ID weighting, $k = 7$ and $l = 5$ gives better performance than $k = 5$ and $l = 3$ for English. Hence, we conclude that the following settings are optimal for our lexicalized models:

1. Number of nearest distances: $k = 5$ (Swedish), $k = 7$ (English)
2. Distance metric: MVDM with $l = 3$ (Swedish), $l = 5$ (English)
3. Feature weighting: None
4. Distance-weighted class voting: ID

With the exception of the higher k and l values for English, these are also the settings that were used in the validation of feature models, and which have been found optimal in previous studies (Nivre et al., 2004; Nivre and Scholz, 2004).

5.5 Final Evaluation

In this section, we will assess the quality of the feature models and parameter settings that produced the best results during validation by applying them to an independent test set. In this way, we can hope to get a more unbiased estimate of the expected accuracy and efficiency with respect to new data. However, it is important to remember that, even though the test sets have not been used in the validation phase, they are nevertheless sampled from the same treebanks as the respective training and validation sets. Our estimates will therefore be valid for text that belongs to the same population, but not for text from other sources in Swedish or English.

We will also compare the performance of the best models to the state of the art in dependency-based parsing. This will raise certain questions about the sources of parsing errors, and we will try to tease apart the influence of three such sources: errors in part-of-speech tagging, errors in the function approximation, and errors due to the greedy, deterministic parsing strategy.

5.5.1 Accuracy and Efficiency

Table 5.11 shows the accuracy obtained on the test sets for the best lexicalized and non-lexicalized models, using the optimal parameter settings of the learning algorithm. We have included both the best models from section 5.3 and the models used as the basis for parameter optimization in section 5.4. This yields five models altogether, since the best non-lexicalized model for Swedish from section 5.3 is identical to the D model used in section 5.4. We use the notation D' and L'_4 to denote the models that are exactly like D and L_4 except that they include a part-of-speech lookahead of three tokens instead of two.

If we compare the results to those obtained during validation, they are comparable in all cases, which indicates that the models have not been overfitted to the training and validation data. Differences between validation scores and test scores are generally less than one percentage point for attachment scores, whereas the exact match scores show a variation of up to two percentage points (with a decrease in accuracy for Swedish and an increase for English). This is only natural, given that the number of observations is one order of magnitude greater for the attachment scores, which are word-based, than for the exact match scores, which are sentence-based.

As for the comparison between feature models, the only differences that are statistically significant for Swedish are those between the non-lexicalized

Table 5.11. Final evaluation: accuracy; AS: attachment score, EM: exact match; U: unlabeled, L: labeled; Swedish (section 0), English (section 23)

Model	Swedish				English			
	AS		EM		AS		EM	
	U	L	U	L	U	L	U	L
$D = \Phi^p_{12} + \Phi^d_{111}$	83.2	75.8	33.4	20.2	83.5	80.5	23.2	17.3
$D' = \Phi^p_{13} + \Phi^d_{111}$	83.3	75.7	31.8	19.2	83.6	80.7	23.5	18.0
$L_2 = \Phi^p_{12} + \Phi^d_{111} + \Phi^w_{11}$	**86.3**	**82.0**	37.7	29.6	87.4	85.7	30.8	26.8
$L_4 = \Phi^p_{12} + \Phi^d_{111} + \Phi^w_{22}$	86.1	81.8	37.3	29.8	87.8	86.0	32.7	**28.7**
$L'_4 = \Phi^p_{13} + \Phi^d_{111} + \Phi^w_{22}$	**86.3**	**82.0**	**39.2**	**30.8**	**88.1**	**86.3**	**32.8**	28.4

models D and D', on the one hand, and the lexicalized models L_2, L_4 and L'_4, on the other. Between these groups, the difference is statistically significant beyond the 0.01 level for all metrics (McNemar's test).[3] Within the groups, however, there are no significant differences.[4] For English we find basically the same pattern, but in addition to the differences between non-lexicalized and lexicalized models, which are all significant beyond the 0.0001 level, there are also significant differences between the model with two lexical features (L_2) and the models with four lexical features (L_4 and L'_4) with $p < 0.01$ for all metrics. Comparing L_4 and L'_4, finally, the differences appear to be significant for the attachment scores but not for the exact metrics. However, this result should be taken with a pinch of salt, since the attachment scores are based on observations of word tokens, which are very far from being independent of each other. Therefore, the lack of a significant difference in the sentence-based exact match comparison throws serious doubt on the value of the differences in attachment score. In conclusion, it therefore seems fair to say that the difference in accuracy between lexicalized and non-lexicalized models is statistically significant for both languages, and that the addition of two extra lexical features makes a significant difference for English, with the larger data sets, but not for Swedish. Any conclusions beyond these are not clearly warranted by the experimental results.

In order to get a more fine-grained picture of accuracy, we will now consider the accuracy for different dependency types. Table 5.12 gives unlabeled attachment score (AS_U), labeled precision (P), recall (R) and F measure (F) for the top scoring model L'_4 on the Swedish test set. Broadly speaking, we can divide dependency types according to accuracy into three sets. In the high-accuracy set, with a labeled F measure from 84% to 98%, we find all

[3] For EM_U $p = 0.01$; for the other three metrics $p < 0.0001$.

[4] It is worth remembering that the Swedish test set only contains 631 sentences, which means that the difference between 29.6% and 29.8% in the EM_L score for L_2 and L_4 is the difference of a single sentence.

Table 5.12. Final evaluation: attachment score (AS_U), precision (P), recall (R) and F measure per dependency type: Swedish (model L'_4)

Label	**n**	**AS_U**	**P**	**R**	**F**
ADV	1607	79.8	75.8	76.8	76.3
APP	42	23.8	38.1	19.0	25.4
ATT	950	81.3	79.9	78.5	79.2
CC	963	82.5	78.1	79.8	78.9
DET	947	92.6	88.9	90.2	89.5
ID	254	72.0	72.5	58.3	64.6
IM	133	98.5	98.5	98.5	98.5
INF	10	100.0	100.0	30.0	46.2
OBJ	585	88.0	78.2	77.3	77.7
PR	985	94.2	88.6	92.7	90.6
PRD	244	90.6	76.7	77.0	76.8
ROOT	607	91.3	84.6	91.3	87.8
SUB	957	89.8	86.7	82.5	84.5
UK	213	85.0	89.4	83.6	86.4
VC	238	93.7	82.1	90.6	86.1
XX	29	82.8	85.7	20.7	33.3
Total	8782	86.3	82.0	82.0	82.0

dependency types where the head is a closed class word: IM (marker → infinitive), PR (preposition → noun), UK (complementizer → verb) and VC (auxiliary verb → main verb). We also find the type DET (noun → determiner), which has similar characteristics although the determiner is not treated as the head in the Swedish annotation. The high-accuracy set also includes the central dependency types ROOT and SUB, which normally identify the finite verb of the main clause and the grammatical subject, respectively.

In the medium-accuracy set, with a labeled F measure in the range of 75–80%, we find the remaining major dependency types, ADV (adverbial), ATT (nominal modifier), CC (coordination), OBJ (object) and PRD (predicative). However, this set can be divided into two subsets, the first consisting of ADV, ATT and CC, which have an unlabeled attachment score not too much above the labeled F measure, indicating that parsing errors are mainly due to incorrect attachment. This is plausible also because ADV and ATT are the dependency types typically involved in modifier attachment ambiguities, and coordination is a source of attachment ambiguities as well. The second subset contains OBJ and PRD, which both have an unlabeled attachment score close to 90%, which means that they are often correctly attached but may be incorrectly labeled. This is again plausible, since these types identify nominal arguments of the verb (other than the subject), which can often occur in the same syntactic contexts.

Table 5.13. Final evaluation: attachment score (AS_U), precision (P), recall (R) and F measure per dependency type: English (model L'_4)

Label	**n**	**AS$_U$**	**P**	**R**	**F**
AMOD	2072	78.2	80.7	73.0	76.7
DEP	259	42.9	56.5	30.1	39.3
NMOD	21002	91.2	91.1	90.8	91.0
OBJ	1960	86.5	78.9	83.5	81.1
PMOD	5593	90.2	87.7	89.5	88.6
PRD	832	90.0	75.9	71.8	73.8
ROOT	2401	86.4	78.8	86.4	82.4
SBAR	1195	86.0	87.1	85.1	86.1
SBJ	4108	90.0	90.6	88.1	89.3
VC	1771	98.8	93.4	96.6	95.0
VMOD	8175	80.3	76.5	77.1	76.8
Total	49368	88.1	86.3	86.3	86.3

Finally, we have a low-accuracy set, with a labeled F measure below 70%, where the common denominator is mainly that these dependency types are rare: INF (infinitive complements), APP (appositions), XX (unclassifiable). The only exception to this generalization is the type ID (idiom constituent), which is not that rare but which is rather special for other reasons. All types in this set except APP have a relatively high unlabeled attachment score, but their labels are seldom used correctly. An extreme case is INF, which has both an unlabeled attachment score and a labeled precision of 100%, although there are only 10 instances in total in the test set, but which has a much lower labeled recall (30%). The dependency type APP, finally, is used for a family of loosely connected modifiers of either nouns or verbs, which apparently are very difficult to attach correctly.

Table 5.13 gives the same kind of breakdown across dependency types for the top scoring model L'_4 on the English test set, where we can distinguish a similar division into three sets according to accuracy level. In the high-accuracy set, with a labeled F measure from 86% to 95%, we find SBJ (subject) and three dependency types where the head is a closed class word: PMOD (preposition → complement/modifier), VC (auxiliary verb → main verb) and SBAR (complementizer → verb). In addition, this set includes the type NMOD, which includes the noun-determiner relation as an important subtype.

In the medium-accuracy set, with a labeled F measure from 74% to 82%, we find the types AMOD, VMOD, OBJ, PRD and ROOT. The former two dependency types mostly cover adverbial functions, and have a labeled accuracy not too far below their unlabeled attachment score, which is an indication that the main difficulty lies in finding the correct head. By contrast, the argument functions OBJ and PRD have a much better unlabeled attachment score,

which shows that they are often attached to the correct head but misclassified. This tendency is especially pronounced for the PRD type, where the difference is more than 15 percentage points, which can probably be explained by the fact that this type is relatively infrequent in the annotated English data.

The low-accuracy set for English only includes the default classification DEP. The very low accuracy for this dependency type can be explained by the fact that it is both a heterogeneous category and the least frequent dependency type in the data.

If we compare the results across languages, we can distinguish the following general patterns:

- Dependents of closed class words have high accuracy, labeled as well as unlabeled. This includes the following construction types:
  1. Preposition → Noun (Swedish PR, English PMOD[5])
  2. Complementizer → Verb (Swedish UK, English SBAR)
  3. Auxiliary verb → Main verb (Swedish and English VC)
- Core arguments of the verb have high unlabeled accuracy. This includes:
  1. Subjects (Swedish SUB, English SBJ)
  2. Objects (Swedish and English OBJ)
  3. Predicative complements (Swedish and English PRD)

  Subjects also have high labeled accuracy, whereas objects and predicative complements are more easily confused with each other.
- Modifiers generally have medium accuracy, both labeled and unlabeled.
- Atypical, heterogeneous and rare dependency types have low accuracy, especially when labels are taken into account.

One apparent difference between the languages is that nominal modifiers (NMOD) have a very high accuracy for English (90.5% AS_U, 90.4% F), whereas the closest corresponding dependency type for Swedish (ATT) has both unlabeled and labeled accuracy below 80%. However, this can probably be explained by the fact that the English category NMOD contains two prominent subcategories that contribute significantly to the overall result. These subcategories are determiners, which have a very high accuracy also in Swedish (91.5% AS_U, 89.6% F), and constituents of noun-noun compounds, which can usually be identified relatively easily but which are absent in the Swedish data because of different orthographic conventions.

Another difference is that the type ROOT, which normally identifies the finite verb of the main clause, has a considerably higher accuracy for Swedish, where it belongs to the high-accuracy set, than for English, where it is found in the middle set. This is probably related to the greater sentence complexity in the English data set, where a greater mean sentence length can be expected to correlate with a greater mean number of clauses per sentence, which tends to make the identification of the main clause more difficult.

[5] The PMOD type also contains modifiers of prepositions but is heavily dominated by nominal complements.

Table 5.14. Final evaluation: efficiency; T: training time (s), P: parsing time (s), S: mean parsing time per sentence (ms), W: mean number of words parsed per second, M: memory requirements during parsing (MB)

Model	Swedish					English				
	T	**P**	**S**	**W**	**M**	**T**	**P**	**S**	**W**	**M**
$D = \Phi^p_{12} + \Phi^d_{111}$	9.5	10.1	16.0	974.4	19	102.8	71.7	29.7	790.6	96
$D' = \Phi^p_{13} + \Phi^d_{111}$	12.0	13.6	21.6	723.6	26	118.9	93.7	38.8	605.0	155
$L_2 = \Phi^p_{12} + \Phi^d_{111} + \Phi^w_{11}$	16.0	167.0	264.7	58.9	45	171.0	1140.5	472.1	49.7	295
$L_4 = \Phi^p_{12} + \Phi^d_{111} + \Phi^w_{22}$	18.0	574.0	909.7	17.1	57	217.9	3327.8	1380.0	17.0	416
$L'_4 = \Phi^p_{13} + \Phi^d_{111} + \Phi^w_{22}$	20.9	661.2	1050.0	14.9	66	234.6	3641.8	1510.0	15.6	457

Finally, it is worth remembering that the two data sets differ in the treatment of coordination, which falls under a separate dependency type (CC) for Swedish, which is comparable in accuracy to the modifier constructions. For English, coordination is not analyzed as a separate category, which means that coordinate structures can be present in any category, although most instances are likely to be found in the VMOD and NMOD categories. Since the analysis assigned to coordinate structures in this way is often linguistically inadequate, it is fair to say that the accuracy results for English give a too optimistic estimate of the true accuracy for parsing unrestricted text.

To complete the picture on model assessment, we also present an evaluation of efficiency on the training and test sets. The results are presented in table 5.14. As can be expected, there are no significant deviations from the results obtained during validation. The higher absolute parsing times (P) for English are due to the fact that the test set is larger than the validation set, but the mean number of words parsed per second is very similar. Relating efficiency to accuracy, we may note that for Swedish the L_4 and L'_4 models more than triple the parsing time without a statistically significant improvement in accuracy, which makes the L_2 model appear as the best choice for a joint optimization of accuracy and efficiency. For English, the addition of two more lexical features gives a significant improvement in accuracy, which means that the more complex models will be optimal for applications where we can accept the decrease in parsing speed. Finally, it is worth pointing out that the differences observed between Swedish and English in this respect are more probably related to the size of the data sets than anything else.

5.5.2 Related Work

In this section, we will try to relate the results from the final evaluation to the state of the art in dependency-based text parsing. For Swedish this is rather difficult, since there is no comparable evaluation reported in the literature, let alone based on the same data. Most of the parsers developed for Swedish use

constituency-based representations, either for full parsing (Sågvall Hein, 1982) or partial parsing (Kokkinakis and Johansson Kokkinakis, 1999; Megyesi, 2002; Bigert, 2005). Voutilainen (2001) presents a partial and informal evaluation of a Swedish FDG parser, based on manually checked parses of about 400 sentences from newspaper text, and reports F measures of 95% for subjects and 92% for objects. These results clearly indicate a higher level of accuracy than that attained in the experiments reported here, but without knowing the details of the data selection and evaluation procedure it is very difficult to draw any precise conclusions. In any case, the results reported in this study may serve as a benchmark for future evaluations of Swedish dependency parsing. The results are encouraging, given the limited amount of data available for training, but it must also be kept in mind that the Swedish data set does not exhibit the same level of complexity as the English one.

For English there is much more relevant work to compare with. The first large-scale evaluation of dependency parsing on the Wall Street Journal data was performed by Eisner (1996b,a). However, Eisner excluded certain types of sentences from the evaluation, in particular sentences involving coordination, which means that the results are not strictly comparable. The same goes for the evaluations of grammar-driven parsers such as the statistical CDG parser of Wang and Harper (2004), the XLE LFG parser of Kaplan et al. (2004) and the CCG parser of Clark and Curran (2004), which are all based on different ways of extracting dependencies from the Penn Treebank data, the latter two using the PARC 700 Dependency Bank (King et al., 2003) and the CCGbank (Hockenmaier, 2003a), respectively.

By contrast, the results reported by Yamada and Matsumoto (2003) and Isozaki et al. (2004) are based on exactly the same data samples and conversion methods (except that they only consider unlabeled dependencies). In addition, Yamada and Matsumoto (2003) derive comparable results for the parsers of Collins (1997) and Charniak (2000), by applying the same conversion to the output of these parsers. More recently, the same data sets have also been used by McDonald, Crammer and Pereira (2005). Table 5.15 presents a comparison of our results with those obtained with the other systems, limited to unlabeled accuracy metrics. In addition to the usual metrics AS_U and EM_U, we also break down the attachment score into *dependency accuracy* (DA), which is the attachment score for all tokens not attached to the special root node, and *root accuracy* (RA), which is the attachment score for all tokens attached to the special root node. Given the conversion of the Penn Treebank annotation to dependency graphs, there is exactly one token per sentence attached to the special root node in the gold standard.

It is clear that, with respect to unlabeled accuracy, our parser does not quite reach state-of-the-art performance, even if we limit the competition to deterministic methods such as those of Yamada and Matsumoto (2003) and Isozaki et al. (2004). We believe that there may be three different reasons for this. First of all, the part-of-speech tagger used for preprocessing in our experiments has a lower accuracy than the one used by Yamada and Matsumoto

Table 5.15. Comparison with related work; AS_U: unlabeled attachment score, DA_U: dependency accuracy, RA_U: root accuracy, EM_U: unlabeled exact match

Study	$\mathbf{AS}_U$	$\mathbf{DA}_U$	$\mathbf{RA}_U$	$\mathbf{EM}_U$
Collins (1997) (Model 3)	91.7	91.5	95.2	43.3
Charniak (2000)	**92.2**	**92.1**	95.2	**45.2**
Yamada and Matsumoto (2003)	90.4	90.3	91.6	38.4
Isozaki et al. (2004)	91.4	91.2	**95.7**	40.7
McDonald, Crammer and Pereira (2005)	91.0	90.9	94.2	37.5
This study	88.1	88.2	86.4	32.8

(2003) (96.1% vs. 97.1%).[6] Although this is not a very interesting explanation, it undoubtedly accounts for part of the difference. We will return to this in our error analysis in the next section.

A more important factor is the relatively low root accuracy of our parser, which may reflect a weakness in the one-pass parsing strategy with respect to the global structure of complex sentences. Although the systems of Yamada and Matsumoto (2003) and Isozaki et al. (2004) are also deterministic, they perform multiple passes over the input, building the structures bottom-up. In the case of Isozaki et al. (2004), parsing is also preceded by a separate root detection phase, which explains the improved root accuracy of this system over Yamada and Matsumoto (2003).

It is noteworthy that our parser has lower root accuracy than dependency accuracy, whereas the inverse holds for all the other parsers. The problem becomes even more visible when we compare dependency and root accuracy for sentences of different lengths, as shown in table 5.16. Here we see that for really short sentences (up to 10 words) root accuracy is indeed higher than dependency accuracy, but while dependency accuracy degrades very gracefully with sentence length, the root accuracy drops more drastically (which also very clearly affects the exact match score). It is also interesting to compare with the results for Swedish, where the mean sentence length is considerably smaller, and where root accuracy is indeed as high as 91.3% (cf. table 5.12). This may be taken to suggest that some kind of preprocessing in the form of clausing or root detection could improve overall parsing accuracy.

Although it seems clear that the inferior root accuracy is primarily related to the single-pass deterministic parsing strategy, it is less clear whether the lower dependency accuracy is due to the parsing strategy or to lower prediction accuracy of the classifiers involved. Whereas our system uses memory-based learning and classification, the systems of Yamada and Matsumoto (2003) and Isozaki et al. (2004) are based on support vector machines. In a recent study by Sagae and Lavie (2005), using data from the Penn Treebank and

[6] Isozaki et al. (2004) apparently used the tags provided by the Collins parser, although this is not entirely clear from the description in the paper.

Table 5.16. Accuracy in relation to sentence length; AS_U: unlabeled attachment score, DA_U: unlabeled dependency accuracy, RA_U: root accuracy, EM_U: unlabeled exact match

Length	$\mathbf{AS}_U$	$\mathbf{DA}_U$	$\mathbf{RA}_U$	$\mathbf{EM}_U$
≤ 10	93.8	93.4	95.5	85.4
$11 \leq 20$	89.4	89.3	90.0	43.4
$21 \leq 30$	87.7	87.9	84.4	22.4
$31 \leq 40$	87.5	87.7	83.1	11.1
$41 \leq \infty$	86.8	87.1	73.7	5.3

a parsing methodology very similar to ours albeit with constituency-based representations, it is found that support vector machines give consistently higher accuracy than memory-based learning. We will return to this issue in the error analysis in the next section.

Turning finally to the assessment of labeled accuracy, we are not aware of any strictly comparable results, but Blaheta and Charniak (2000) report an F measure of 98.9% for the assignment of grammatical role labels to phrases that were correctly parsed by the parser described in Charniak (2000), using the same data set. If null labels are excluded, the F score drops to 95.6%. The corresponding F measures for our system, based on the labeled precision and recall for tokens that are assigned the correct head, are 98.0% and 97.8%, treating the default label DEP as the equivalent of a null label. The experiments are not strictly comparable, since they involve different sets of functional categories (where only the labels SBJ and PRD are equivalent) and one is based on phrase structure and the other on dependency structure, but it nevertheless seems fair to conclude that the labeling accuracy of our parser is close to the state of the art, even if its capacity to derive correct structures is not. It should also be kept in mind that the labeling here is performed in the same deterministic one-pass parsing process as the derivation of the structure, whereas Blaheta and Charniak (2000) apply an elaborate probabilistic model to the output of a probabilistic parser.

To make the comparison complete, it should also be pointed out that, if the memory-based deterministic approach dependency parsing does not quite reach the state of the art in terms of accuracy, it is highly competitive in terms of efficiency. This holds both with respect to parsing time, with a linear time parsing algorithm, and with respect to training time, where the memory-based approach vastly outperforms, e.g., support vector machines.

5.5.3 Error Analysis

Although we will not be able to present a detailed error analysis, we will try to tease apart some of the error sources involved in deterministic memory-based dependency parsing. More precisely, we will consider the influence of

Table 5.17. Accuracy as a function of tagging accuracy (English, model L_4'); AS: attachment score, EM: exact match; U: unlabeled, L: labeled

		AS		EM	
Tagging	**Accuracy**	**U**	**L**	**U**	**L**
Gold standard	100.0	89.7	88.3	36.0	31.8
Nakagawa et al. (2002)	97.1	88.7	87.1	34.4	29.8
Hall (2003)	96.1	88.1	86.3	32.8	28.4

part-of-speech tagging errors and the role of the inductively defined parser guide in relation to the deterministic parsing strategy.

Table 5.17 presents the parsing accuracy obtained with the best model for the English data, with three different ways of assigning part-of-speech tags in the preprocessing. The first uses the gold standard tags in the Penn Treebank annotation; the second uses the output of the tagger described in Nakagawa et al. (2002), based on revision learning with support vector machines and used in the parsing experiments of Yamada and Matsumoto (2003);[7] the third uses the output of the tagger described in Hall (2003), based on hidden Markov models with suffix probabilities and used in all the experiments of this study.

We see that there is a substantial improvement in parsing accuracy when using the gold standard tags, ranging from 1.6 percentage points for unlabeled attachment accuracy to 3.4 percentage points for labeled exact match. A more detailed look at different dependency types (not shown in the table) reveals that the greatest improvement is found for the argument types OBJ and PRD, where the labeled F measure increases by 4.1 (OBJ) and 3.2 (PRD) percentage points. This indicates that a significant proportion of the cases where these two types are confused during parsing are due to tagging errors. Using gold standard tags also improves the root accuracy of the parser from 86.4% to 90.0%, which is a very substantial difference.

Using the tagger of Nakagawa et al. (2002) instead of our own HMM tagger also makes a significant difference. In fact, although the error reduction in tagging is only about 25%, the error reduction in parsing is roughly 40% when compared to the scores obtained with gold standard tags, and as much as 50% for unlabeled exact match. It thus seems that the tagging errors avoided by the better tagger are of a kind that are important for syntactic parsing. However, without a detailed analysis of the differences between the taggers, it is difficult to say anything more precise than this.

The second kind of error analysis that we will present is an attempt to distinguish the influence of prediction errors, caused by errors in the learned approximation of the guide function, and errors that are the combined effect of previous prediction errors and the greedy, deterministic parsing strategy.

[7] Special thanks to Hiroyasu Yamada for supplying us with the output of this tagger for section 23 of the Wall Street Journal section of the Penn Treebank.

Table 5.18. Classification accuracy, number of instances, and number of exact matches (model L'_4); Swedish (section 0), English (section 23)

Data set	Accuracy	Instances	Exact
Swedish	90.7	16714	1064
English tagged	94.4	102419	15185
English gold	95.2	102419	15962

Table 5.18 presents the pure classification accuracy of the best memory-based models for learning and classification when tested on the set of instances derived from a gold standard parse of the respective test sets, section 0 of the Swedish treebank and section 23 of the English treebank. This proportion tells us how often the estimated guide prediction $\hat{g}(\Phi(c, A_x))$ coincides with the true oracle prediction $o(c, A_x)$ for the given data sets. The table also gives the number of instances in each test set (equivalent to the number of nondeterministic parser configurations in parsing the test set) and the number of parser states that had an exact match in the instance base.

First, we may note that the number of exact matches in the instance base is relatively low, about 6% for Swedish and about 15% for English. The higher proportion for English is expected, since the training data set is one order of magnitude larger. With a low percentage of exact matches it is crucial to have an adequate smoothing model, and it seems that the similarity-based smoothing built into the memory-based classification solves this problem very well, with classification accuracy above 90% for all conditions. For English we can also observe that tagging errors have a relatively small impact on classification accuracy as such, although the effect is magnified in parsing because of subsequent errors caused by the initial prediction error.

Comparing classification accuracy and parsing accuracy is not completely straightforward, but the most relevant metric is labeled attachment score, since the transitions chosen in the classification task are parameterized for dependency types. For Swedish, a classification accuracy of 90.7% corresponds to a labeled attachment score of 82.0%, which indicates a drop in accuracy of about 9% because of the greedy parsing strategy. For English with noisy tagging, a classification accuracy of 94.4% corresponds to a labeled attachment score of 86.3%, which is roughly 8% deterioration. For English with gold standard tags, there is only a 7% drop from 95.2% to 88.3%. As expected, the drop in accuracy from pure classification to parsing is smaller the better the classification accuracy, since a lower probability of prediction errors also leads to a lower probability of errors caused by earlier prediction errors. As a first approximation, we can predict the labeled attachment accuracy to be the square of the classification accuracy (CA) (treating both AS_L and CA as proportions ranging from 0 to 1):

$$\mathrm{AS}_L = \mathrm{CA}^2 \tag{5.5}$$

This prediction overestimates the parsing accuracy more for English than for Swedish. This can probably be explained by the fact that the English sentences are generally longer and more complex, which by itself increases the probability of errors being caused by previous prediction errors. In order to more accurately predict the parsing accuracy from the classification accuracy, we therefore need to introduce a constant c related to the complexity of the text samples being parsed:

$$\mathrm{AS}_L = c \cdot \mathrm{CA}^2 \tag{5.6}$$

In our experiments, the value of c appears to be approximately 0.97 for English and almost 1.0 for Swedish. Since the negative effect of error propagation is likely to increase with sentence length, we might be able to predict the value of c from the mean sentence length in the test corpus. For the test sets used here, a good approximation can be obtained by setting $c = 1 - 0.003(n - 15)$, where n is the mean number of words per sentence. Whether a model of this kind can be generalized to other languages and data sets is a question that can only be answered by further research.

In conclusion, it seems that the classification accuracy attained by the memory-based approach to learning is sufficient for highly accurate parsing, especially if sufficient amounts of training data are available, as indicated by the significantly higher accuracy for the English data sets. However, in order to convert this potential into state-of-the-art parsing accuracy, the parsing process clearly needs to be improved. This improvement may take the form of enhanced preprocessing, as suggested by the results of Isozaki et al. (2004), or it may require abandoning the strictly deterministic parsing strategy. We will return to these issues in the concluding chapter.

6

Conclusion

In this book we have explored the framework of inductive dependency parsing as a methodology for accurate and efficient syntactic parsing of unrestricted natural language text. Starting from the basic requirements of robustness and disambiguation, as they appear in our definition of the text parsing problem, we have sought to develop methods where accuracy and efficiency can be jointly optimized. The basic ingredients are dependency-based parsing and inductive machine learning, restricted in this study to deterministic parsing supported by memory-based learning and classification.

As is customary, we will use the concluding chapter of the book to look forward as well as backward. In the first part of the chapter, we will outline what we take to be our main contributions to the study of natural language parsing in general and data-driven dependency parsing in particular. In the second part, we will point to some important directions for future research, motivated by the desire to improve inductive dependency parsing as well as by the need to increase our knowledge about natural language parsing in general.

6.1 Main Contributions

The contributions of this study can be divided into four broad categories, roughly corresponding to chapters 2–5. The first is a conceptual analysis of the problems and methods involved in natural language parsing (chapter 2). The second is a formal analysis of dependency parsing and of a particular deterministic algorithm for projective dependency parsing (chapter 3). The third is a general model for inductive dependency parsing and an instantiation of this model through memory-based learning and classification (chapter 4). The fourth is an empirical evaluation of inductive dependency parsing, in its deterministic memory-based version, based on treebank data from Swedish and English (chapter 5).

One of the main points made in the conceptual discussion in chapter 2 is that text parsing, the problem of parsing unrestricted natural language text, is

in many ways radically different from grammar parsing, the problem of parsing strings according to a formal grammar. While grammar parsing is an abstract and mathematically well-defined problem, which can be studied using formal methods, text parsing is an essentially empirical problem, where the central requirement of accuracy can only be evaluated by statistical inference based on representative text samples. This makes it an open question whether and to what extent methods for grammar parsing have a role to play in text parsing. We are certainly not the first to notice this ambiguity in the application of the term *parsing* to natural language, but we feel that it has perhaps not been given the attention it deserves in the literature. And even though we only consider one specific formulation of the text parsing problem, which may not be accepted by everyone, and only discuss a subset of the problems involved in this task, we hope that this discussion can make some small contribution to our scientific understanding of natural language parsing.

Another distinction that is well-known in the literature, but which may also be in need of some conceptual clarification, is the distinction between the grammar-driven and the data-driven approach to text parsing. First of all, we have emphasized that the two approaches are not incompatible and are actually combined in many contemporary parsing systems. Secondly, we have proposed that a fruitful way of understanding the distinction is to view the two approaches as different strategies for handling the complex optimization of robustness, disambiguation, accuracy and efficiency. The grammar-driven approach is based on a formal language approximation, which provides a solid basis for accurate parsing, but which needs to be complemented with techniques for handling robustness and disambiguation. The data-driven approach is instead based on inductive inference for disambiguation, which makes a formal language characterization superfluous and thereby favors robustness. Whether used alone or in combination, both strategies also have to deal with the trade-off between robust and accurate disambiguation, on the one hand, and efficiency, on the other. In this book, we have only exploited the data-driven approach to dependency parsing, but the general framework is compatible also with a grammar-driven approach, a topic that we will return to later in this chapter.

Chapter 3 presents a formalization of dependency-based text parsing, which provides the formal framework for the study of specific methods in later chapters. Given a text, composed of a sequence of sentences, the parsing problem consists in assigning to each sentence a labeled dependency graph, using the tokens of the sentence as nodes, and using typed syntactic relations as arcs. Since tokens are represented by string positions, rather than actual word forms, the framework separates the structural analysis of a sentence, represented by the graph structure, from its surface realization, represented by the word string, possibly with additional annotation. Dependency graphs are assumed to be rooted and connected, but additional requirements such as single-headedness, acyclicity and projectivity are optional. Moreover, no specific assumptions are made concerning the form of the syntactic analysis,

over and above the fact that it can be represented as a labeled dependency graph. In this way, we hope to provide a framework that is general enough to encompass different instantiations of dependency-based text parsing and thereby provide a basis for comparative studies, although such investigations are beyond the scope of this study.

The main contribution of chapter 3 is the specification and analysis of an efficient algorithm for projective dependency parsing, based on a head-driven arc-eager parsing strategy. The algorithm is defined in terms of a nondeterministic transition system, which requires an oracle, or guide, to predict the next transition at each nondeterministic choice point. Guided parsing can be implemented in a variety of ways, using the grammar-driven or the data-driven approach, without changing the fundamental properties of the parsing algorithm.

We prove three theorems about the parsing algorithm. Theorem 3.19 says that any sentence of length n is parsed in at most $2n - 1$ transitions. This means that time complexity is linear in the length of the input, provided that transitions can be performed in constant time. Theorem 3.21 says that the dependency graph given at termination, for any input sentence, satisfies the formal conditions Root, Connectedness, Single-Head, Acyclicity and Projectivity. This means that any sentence can be assigned a well-formed dependency analysis. Taken together, these results guarantee that the parsing algorithm is optimal with respect to robustness and disambiguation, since any input sentence is assigned exactly one well-formed dependency graph. Moreover, time complexity is also optimal, since the single well-formed dependency graph is constructed in linear time.

While theorems 3.19 and 3.21 have no bearing on accuracy, i.e., on whether the single well-formed dependency graph assigned to a given sentence is the correct analysis or not, theorem 3.22 establishes that any projective dependency graph has a corresponding transition sequence, which means that, given an oracle that predicts the next transition, the parsing algorithm will always derive the correct analysis for a given input sentence. In the second half of the book, we investigate to what extent it is possible to approximate oracles using inductive machine learning and to achieve accurate and efficient parsing by combining the deterministic parsing algorithm with a data-driven approach to text parsing.

There are three essential components in the data-driven approach: a formal model defining permissible syntactic representations; an empirical sample of text constituting the basis for generalization; and an inductive inference scheme for assigning syntactic representations to new sentences (based on the formal model and the empirical text sample). In chapter 4, we show how the framework for dependency parsing developed in chapter 3 can be used as the formal model in a data-driven system, and we go on to define a history-based inductive inference scheme, which is parameterized by a feature model, a parsing method and a learning method. We show how the deterministic parsing algorithm can be used as the parsing method, implementing a greedy

search for the optimal analysis according to the history-based model, and how memory-based learning and classification can be used to solve the inductive learning problem defined by the history-based model in combination with the deterministic parsing strategy. We also demonstrate how a variation of the parsing algorithm can be used to generate training data for the learner, given a training corpus with gold standard dependency annotations. The correctness proof for this algorithm, theorem 4.1, is of central importance insofar as it also constitutes a constructive proof of theorem 3.22.

The feature model, which defines equivalence classes of histories, can be defined by a sequence of feature functions, each of which is equivalent to the composition of an address function and an attribute function. This formalization forms the basis of a description language for feature models, which is used in the implemented MaltParser system to allow the specification of arbitrary feature models without recompiling the system. To a large extent, the specification of feature models is independent of both the parsing method and the learning method used. At the end of chapter 4, we briefly describe the architecture of the MaltParser system, which is based on a strict modularization of parsing method, feature model, and learning method.

Given the empirical view of text parsing, evaluation by necessity requires empirical experiments. Chapter 5 presents a fairly exhaustive evaluation of inductive dependency parsing, based on deterministic parsing and memory-based learning, using treebank data from Swedish and English. The evaluation focuses on the influence of different parameters of the feature model and the learning algorithm on accuracy, but the evaluation also takes efficiency into account.

The experiments confirm the importance of proper feature selection for data-driven parsing, evident from many previous studies. By and large, we can say that accuracy increases with model complexity until it is affected by data sparseness, which in our experiments happens sooner for Swedish than for English. At the same time, parsing efficiency decreases with model complexity, as the time needed to predict the next transition increases, which means that accuracy and efficiency must be jointly optimized, although the optimal point may vary from one context to the other.

The experimental results corroborate the importance of lexicalization for accurate disambiguation, both for Swedish and for English, which is in line with previous studies on data-driven parsing, although conflicting results exist (Klein and Manning, 2003; Dubey and Keller, 2003; Arun and Keller, 2005). Another conclusion that can be drawn is that dynamically defined dependency type features have a positive influence on parsing accuracy, a result which has not been reported in the literature before, although it can probably be related to results for structurally defined features in constituency-based parsing.

The experiments also demonstrate the importance of optimizing parameters of the learning algorithm, confirming the results of previous studies of data-driven natural language processing. For memory-based learning and classification, a relatively large k value in combination with the MVDM

distance metric has previously been found to work well for higher-level tasks such as parsing (Daelemans and Van den Bosch, 2005), a finding that is also supported by the experimental results reported in this study.

For Swedish, the final evaluation establishes a new benchmark, since no large-scale evaluation of dependency parsing has previously been performed for this language. The results indicate that reasonable parsing accuracy can be achieved with fairly limited amounts of training data, given a suitable dependency annotation. For English, the final evaluation shows that the accuracy obtained is not quite state-of-the-art, which can be explained by the combined effect of classification errors and a greedy, deterministic parsing strategy. It is also possible that the learner suffers from the fact that the labeled dependency graphs used for training are derived from a constituency-based annotation with only limited functional annotation. On the other hand, the parser compares well with other systems in terms of efficiency, both in training and in parsing. In any case, the results are significant in that they show what kind of accuracy can be achieved using a deterministic one-pass strategy in combination with a purely discriminative strategy for inductive learning, imposing hard constraints on robustness and disambiguation, and maintaining a high level of efficiency.

6.2 Future Directions

The dependency-based system for text parsing evaluated in chapter 5 is only one instantiation of a more general approach to inductive dependency parsing. This instantiation is the result of making specific decisions at a number of points, such as which conditions to impose on dependency graphs, what kind of parsing algorithm to use, and what kind of learning methods to employ. Our suggestions for future research can to a large extent be understood as the exploration of different choices at each of these points.

Although the framework of dependency parsing defined in chapter 3 only requires dependency graphs to be rooted and connected, we have restricted our attention in this study to projective dependency graphs. Given that the data sets used for training and evaluation in chapter 5 only contain projective structures, this restriction may seem well motivated, but it is problematic in a more general context. For example, it is sometimes claimed that one of the advantages of dependency grammar over approaches based on constituency is that it allows a more adequate treatment of languages with variable word order, where discontinuous syntactic constructions are more common than in languages like English (Mel'čuk, 1988; Covington, 1990b). But this argument is really only plausible if the formal framework allows non-projective dependency structures. From the point of view of computational implementation this can be problematic, since the inclusion of non-projective structures makes the parsing problem more complex and therefore compromises efficiency and

in practice also accuracy and robustness, which is why most robust dependency parsers are restricted to projective structures.

This is in contrast to dependency treebanks, exemplified by the Prague Dependency Treebank (Hajič et al., 2001), the Danish Dependency Treebank (Kromann, 2003), and the METU Treebank of Turkish (Oflazer et al., 2003), which generally allow sentences to be annotated with non-projective dependency structures in order to capture discontinuous constructions. The fact that projective dependency parsers can never exactly reproduce the analyses found in non-projective treebanks is often neglected because of the relative scarcity of problematic constructions. While the proportion of sentences containing non-projective dependencies is often 15–25%, the total proportion of non-projective arcs is normally only 1–2% (Nivre and Nilsson, 2005). As long as the main evaluation metric is attachment score per word, with state-of-the-art accuracy mostly below 90%, the penalty for not handling non-projective constructions is almost negligible, especially since these constructions are hard to parse correctly in the first place. Still, from a theoretical point of view, projective parsing of non-projective structures has the drawback that it rules out perfect accuracy even as an asymptotic goal.

Developing methods that can handle non-projective dependency structures accurately and efficiently is a very important goal of future research. However, it is still an open question whether this is best achieved by introducing more complex parsing algorithms, that can construct non-projective dependency graphs directly, or by combining projective parsing with different kinds of pre- or post-processing to derive non-projective structures. Thus, Nivre and Nilsson (2005) show how the deterministic memory-based parsing method evaluated in chapter 5 can be combined with graph transformation techniques to give a significant improvement of parsing accuracy for the Prague Dependency Treebank. Since non-projective constructions often involve long-distance dependencies, this line of research is closely related to recent work on the recovery of empty categories and non-local dependencies in constituency-based parsing (Johnson, 2002; Dienes and Dubey, 2003; Hockenmaier, 2003b; Jijkoun and De Rijke, 2004; Cahill et al., 2004; Levy and Manning, 2004; Campbell, 2004).

Another kind of decision concerns the parsing algorithm, in particular the choice between deterministic and nondeterministic parsing strategies. The empirical evaluation indicates that the greedy optimization strategy inherent in deterministic parsing puts an upper bound on the accuracy that can be attained. An obvious option to explore in the future is therefore to introduce a mild form of nondeterminism, which could improve parsing accuracy without sacrificing efficiency. However, this also requires a mechanism for scoring candidate analyses, which means that we could then no longer rely on purely discriminative learning methods.

This leads directly to a third decision point, namely the choice of learning method. If we want to derive more than one analysis in order to select the most likely candidate, we need a learning method that can estimate the

conditional probabilities associated with the history-based model, and not just the mode of each conditional distribution. Although memory-based classification can give information about the distance of the nearest neighbors, and about the conditional distribution in the neighbor space, it is not straightforward to derive comparable probability estimates for events from different neighbor spaces. This suggests that Bayesian inference, and other methods for estimating probabilities, may be worth exploring instead. However, even if we stick to the deterministic parsing algorithm investigated in this book, it may be interesting to compare alternative learning methods, such as support vector machines or maximum entropy modeling. Regardless of learning method, it may also be possible to improve results through the use of active learning (Cohn et al., 1994; Thompson et al., 1999; Tang et al., 2002). For Swedish, where the available data is relatively sparse, active learning can be used to select new sentences for annotation. For English, where the learning curves are relatively flat when using random sampling, it may be possible to improve accuracy by a more efficient use of the available data.

A fourth direction for future research is to provide a synthesis of the data-driven and the grammar-driven approach to text parsing. Even though the data-driven approach is fundamental to inductive dependency parsing, there is nothing that prevents us from integrating grammatical constraints, either to guide the inductive learning itself, or to filter out predictions that violate the constraints. However, this must be done in such a way that the robustness of the data-driven approach is maintained. Combining inductive and deductive learning is very hard in general, but it could potentially be a way to enhance existing data-driven methods and to get an optimal mix of the two strategies.

Another research topic that has received increased attention lately is the notion of incremental parsing. One of the interesting properties of the deterministic arc-eager parsing algorithm, which has not been discussed in detail in this study, is that it appears to be optimal with respect to incrementality as well as time complexity (Nivre, 2004a), in the sense that it minimizes the number of unattached tokens on the stack. However, in order to make parsing really incremental, we need to eliminate the need for preprocessing in the form of part-of-speech tagging. This can be done either by integrating tagging into the parsing process or by eliminating part-of-speech features from our feature models. The latter option is explored in Nivre (2005).

In order to gain a better understanding of different methods for syntactic parsing of unrestricted natural language text, it is very important that the empirical basis for evaluation is widened to as many languages as possible. In this book, we have evaluated our approach to inductive dependency parsing on Swedish and English, but there are also results available for Czech (Nivre and Nilsson, 2005), Bulgarian (Marinov and Nivre, 2005), Danish (Nivre and Hall, 2005) and Italian (Chanev, 2005), as well as ongoing work on Arabic, Chinese, Dutch, Japanese, German, Portuguese, Slovene, Spanish and Turkish. Comparing the performance of inductive dependency parsing across a wide range of languages will show to what extent the methodology is language-independent.

Finally, empirical evaluation not only requires data from more languages but also more diversified evaluation methods. In this study, we have tried to get a more complete picture by using both attachment score and exact match metrics to evaluate accuracy, but we are also aware that these measures do not tell the whole story. In the future, we therefore want to apply other evaluation schemes, in particular the more abstract dependency-based evaluation discussed briefly in chapter 5 (cf. Lin, 1998; Carroll et al., 1998, 2003). This leads naturally to more semantically oriented evaluation schemes, focusing on the recovery of predicate-argument structure and other aspects of semantic analysis. After all, syntactic parsing is in most applications only a preliminary to semantic interpretation, even though it is a fascinating problem in itself.

References

Abeillé, A. (2003a). Introduction. In Abeillé, A. (ed.), *Treebanks: Building and Using Parsed Corpora*, Kluwer Academic Publishers, pp. xiii–xxvi.

Abeillé, A. (ed.) (2003b). *Treebanks: Building and Using Parsed Corpora*. Kluwer Academic Publishers.

Abney, S. (1991). Parsing by chunks. In Berwick, R., Abney, S. and Tenny, C. (eds), *Principle-Based Parsing*, Kluwer Academic Publishers, pp. 257–278.

Abney, S. (1996). Partial parsing via finite-state cascades. *Journal of Natural Language Engineering* **2**: 337–344.

Abney, S. (1997). Part-of-speech tagging and partial parsing. In Young, S. and Bloothooft, G. (eds), *Corpus-Based Methods in Language and Speech Processing*, Kluwer Academic Publishers, pp. 118–136.

Abney, S. and Johnson, M. (1991). Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research* **20**: 233–250.

Aduriz, I., Aranzabe, M. J., Arriola, J. M., Atutxa, A., Díaz de Ilarraza, A., Garmendia, A. and Oronoz, M. (2003). Construction of a Basque dependency treebank. In Nivre, J. and Hinrichs, E. (eds), *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, Växjö University Press, pp. 201–204.

Aha, D. (ed.) (1997). *Lazy Learning*. Kluwer Academic Publishers.

Aha, D. W., Kibler, D. and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning* **6**: 37–66.

Aho, A. V., Sethi, R. and Ullman, J. D. (1986). *Compilers: Principles, Techniques, and Tools*. Addison-Wesley.

Aho, A. V. and Ullman, J. D. (1995). *Foundations of Computer Science*. Computer Science Press.

Ajdukiewicz, K. (1935). Die syntaktische Konnexität. *Studia Philosophica* **1**: 1–27.

Alshawi, H. (1996). Head automata and bilingual tiling: Translation with minimal representations. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 167–176.

Argamon, S., Dagan, I. and Krymolowski, Y. (1998). A memory-based approach to learning shallow natural language patterns. *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics (ACL-COLING)*, pp. 67–73.

Arun, A. and Keller, F. (2005). Lexicalization in crosslinguistic probabilistic parsing: The case of French. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 302–313.

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM conference. *Proceedings of the International Conference on Information Processing*, pp. 125–132.

Baker, J. (1979). Trainable grammars for speech recognition. *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pp. 547–550.

Bangalore, S. (2003). Localizing dependencies and supertagging. In Bod, R., Scha, R. and Sima'an, K. (eds), *Data-Oriented Parsing*, CSLI Publications, University of Chicago Press, pp. 283–298.

Bangalore, S. and Joshi, A. K. (1999). Supertagging: An approach to almost parsing. *Computational Linguistics* **25**: 237–267.

Bar-Hillel, Y. (1953). A quasi-arithmetical notation for syntactic description. *Language* **29**: 47–58.

Bar-Hillel, Y., Gaifman, C. and Shamir, E. (1960). On categorial and phrase-structure grammars. *Bulletin of the Research Council of Israel* **9F**: 1–16.

Barbero, C., Lesmo, L., Lombardo, V. and Merlo, P. (1998). Integration of syntactic and lexical information in a hierarchical dependency grammar. In Kahane, S. and Polguère, A. (eds), *Proceedings of the Workshop on Processing of Dependency-Based Grammars (ACL-COLING)*, pp. 58–67.

Barton, G. E., Berwick, R. C. and Ristad, E. S. (1987). *Computational Complexity and Natural Language*. MIT Press.

Basili, R. and Zanzotto, F. M. (2002). Parsing engineering and empirical robustness. *Natural Language Engineering* **8**: 97–120.

Berger, A., Della Pietra, S. A. and Della Pietra, V. J. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics* **22**: 39–71.

Bies, A., Ferguson, M., Katz, K. and MacIntyre, R. (1995). Bracketing guidelines for Treebank II style, Penn Treebank project. University of Pennsylvania, Philadelphia.

Bigert, J. (2005). *Automatic and Unsupervised Methods in Natural Language Processing*. PhD thesis, KTH, Stockholm.

Bikel, D. M. (2004). Intricacies of Collins' parsing model. *Computational Linguistics* **30**: 479–511.

Bishop, C. M. (1996). *Neural Networks for Pattern Recognition*. Oxford University Press.

Black, E., Abney, S., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Roukos, S.,

Santorini, B. and Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. *Proceedings of the DARPA Speech and Natural Language Workshop*, pp. 306–311.

Black, E., Garside, R. and Leech, G. (eds) (1993). *Statistically-Driven Computer Grammars of English: The IBM/Lancaster Approach*. Rodopi.

Black, E., Jelinek, F., Lafferty, J., Magerman, D., Mercer, R. and Roukos, S. (1992). Towards history-based grammars: Using richer models for probabilistic parsing. *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, pp. 31–37.

Blaheta, D. and Charniak, E. (2000). Assigning function tags to parsed text. *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 234–240.

Bloomfield, L. (1933). *Language*. The University of Chicago Press.

Bod, R. (1995). *Enriching Linguistics with Statistics: Performance Models of Natural Language*. PhD thesis, University of Amsterdam.

Bod, R. (1998). *Beyond Grammar*. CSLI Publications, University of Chicago Press.

Bod, R. (2000). Parsing with the shortest derivation. *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pp. 69–75.

Bod, R. (2001). What is the minimal set of subtrees that achieves maximal parse accuracy. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 66–73.

Bod, R. (2003). An efficient implementation of a new DOP model. *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 19–26.

Bod, R., Hay, J. and Jannedy, S. (eds) (2003). *Probabilistic Linguistics*. MIT Press.

Bod, R. and Kaplan, R. (1998). A probabilistic corpus-driven model for lexical-functional analysis. *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics (ACL-COLING)*, pp. 145–151.

Bod, R., Scha, R. and Sima'an, K. (eds) (2003). *Data-Oriented Parsing*. CSLI Publications, University of Chicago Press.

Böhmová, A., Hajič, J., Hajičová, E. and Hladká, B. (2003). The Prague Dependency Treebank: A three-level annotation scenario. In Abeillé, A. (ed.), *Treebanks: Building and Using Parsed Corpora*, Kluwer Academic Publishers, pp. 103–127.

Bohnet, B. (2003). Mapping phrase structures to dependency structures in the case of free word order languages. *Proceedings of The First International Conference on Meaning-Text Theory*, pp. 138–148.

Booth, T. and Thompson, R. (1973). Applying probability measures to abstract languages. *IEEE Transactions on Computers* **C-22**: 442–450.

Bosco, C. and Lombardo, V. (2004). Dependency and relational structure in treebank annotation. *Proceedings of the Workshop on Recent Advances in Dependency Grammar*, pp. 9–16.

Boullier, P. (2003). Guided Earley parsing. In Van Noord, G. (ed.), *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 43–54.

Brants, S., Dipper, S., Hansen, S., Lezius, W. and Smith, G. (2002). TIGER treebank. In Hinrichs, E. and Simov, K. (eds), *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 24–42.

Brants, T. (1999). Cascaded Markov models. *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 118–125.

Bresnan, J. (2000). *Lexical-Functional Syntax*. Blackwell.

Brill, E. (1993). Transformation-based error-driven parsing. *Proceedings of the Third International Workshop on Parsing Technologies (IWPT)*, pp. 13–25.

Briscoe, E. and Carroll, J. (1993). Generalised probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics* **19**: 25–59.

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

Buchholz, S., Veenstra, J. and Daelemans, W. (1999). Cascaded grammatical relation assignment. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in NLP and Very Large Corpora*, pp. 239–246.

Cahill, A., Burke, M., O'Donovan, R., Van Genabith, J. and Way, A. (2004). Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 320–327.

Campbell, R. (2004). Using linguistic principles to recover empty categories. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 646–653.

Caraballo, S. A. and Charniak, E. (1998). New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics* **24**: 275–298.

Cardie, C. (1993). Using decision trees to improve case-based learning. *Proceedings of the 10th International Conference on Machine Learning*, pp. 25–32.

Cardie, C., Daelemans, W. and Sang, E. T. K. (eds) (2000). *Proceedings of the Fourth Conference on Computational Natural Language Learning (CoNLL)*, Association for Computational Linguistics.

Carpenter, B. (1992). *The Logic of Typed Feature Structures*. Cambridge University Press.

Carpenter, B. and Manning, C. (1997). Probabilistic parsing using left corner language models. *Proceedings of the 5th International Workshop on Parsing Technologies*, pp. 147–158.

Carreras, X. and Màrquez, L. (2004). Introduction to the CoNLL-2004 shared task: Semantic role labeling. In Ng, H. T. and Riloff, E. (eds), *Proceedings of the 8th Conference of Computational Natural Language Learning (CoNLL)*, pp. 89–97.

Carroll, G. and Charniak, E. (1992). Two experiments on learning probabilistic dependency grammars from corpora, *Technical Report TR-92*, Department of Computer Science, Brown University.

Carroll, J. (1994). Relating complexity to practical performance in parsing with wide-coverage unification grammars. *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 287–294.

Carroll, J. (2000). Statistical parsing. In Dale, R., Moisl, H. and Somers, H. (eds), *Handbook of Natural Language Processing*, Marcel Dekker, pp. 525–543.

Carroll, J. (2003). Parsing. In Mitkov, R. (ed.), *The Oxford Handbook of Computational Linguistics*, Oxford University Press, pp. 233–248.

Carroll, J. and Briscoe, E. (1996). Apportioning development effort in a probabilistic LR parsing system through evaluation. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 92–100.

Carroll, J., Briscoe, E. and Sanfilippo, A. (1998). Parser evaluation: A survey and a new proposal. *Proceedings of the [First] International Conference on Language Resources and Evaluation*, pp. 447–454.

Carroll, J., Minnen, G. and Briscoe, E. (2003). Parser evaluation using a grammatical relation annotation scheme. In Abeillé, A. (ed.), *Treebanks*, Kluwer Academic Publishers, pp. 299–316.

Chanev, A. (2005). Portability of dependency parsing algorithms: An application for Italian. *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 29–40.

Chanod, J.-P. (2001). Robust parsing and beyond. In Junqua, J.-C. and Van Noord, G. (eds), *Robustness in Language and Speech Technology*, Kluwer Academic Publishers, pp. 187–204.

Charniak, E. (1996). Tree-bank grammars. *Proceedings of AAAI/IAAI*, pp. 1031–1036.

Charniak, E. (1997a). Statistical parsing with a context-free grammar and word statistics. *Proceedings of AAAI/IAAI*, pp. 598–603.

Charniak, E. (1997b). Statistical techniques for natural language parsing. *AI Magazine* **18**: 33–44.

Charniak, E. (2000). A maximum-entropy-inspired parser. *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 132–139.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 173–180.

Charniak, E., Knight, K. and Yamada, K. (2003). Syntax-based language models for machine translation. *Proceedings of MT Summit IX*, pp. 40–46.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory* **IT-2**: 113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control* **2**: 137–167.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT Press.

Chomsky, N. (1970). Remarks on nominalization. In Jacobs, R. and Rosenbaum, P. S. (eds), *Readings in English Transformational Grammar*, Ginn and Co.

Chomsky, N. (1981). *Lectures on Government and Binding*. Foris Publications.

Chomsky, N. (1995). *The Minimalist Program*. MIT Press.

Chu, Y. J. and Liu, T. J. (1965). On the shortest arborescence of a directed graph. *Science Sinica* **14**: 1396–1400.

Church, K. (1988). A stochastic parts program and noun phrase parser for unrestricted text. *Proceedings of the Second Conference on Applied Natural Language Processing*, pp. 136–143.

Civit, M., Kübler, S. and Martí, M. A. (eds) (2005). *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*.

Clark, S. and Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 104–111.

Cohn, D., Atlas, L. and Ladner, R. (1994). Improving generalization with active learning. *Machine Learning* **15**: 201–221.

Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 184–191.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 16–23.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania.

Collins, M. (2000). Discriminative reranking for natural language parsing. *Proceedings of the 17th International Conference on Machine Learning*, pp. 175–182.

Collins, M. and Brooks, J. (1995). Prepositional phrase attachment through a backed-off model. *Proceedings of the 3rd Workshop on Very Large Corpora*, pp. 27–38.

Collins, M. and Duffy, N. (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures and the voted perceptron. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 263–270.

Collins, M., Hajič, J., Ramshaw, L. and Tillmann, C. (1999). A statistical parser for Czech. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 505–512.

Collins, M. and Koo, T. (2005). Discriminative reranking for natural language parsing. *Computational Linguistics* **31**: 25–71.

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press.

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

Cover, T. M. and Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* **13**: 21–27.

Covington, M. A. (1984). *Syntactic Theory in the High Middle Ages*. Cambridge University Press.

Covington, M. A. (1990a). A dependency parser for variable-word-order languages, *Technical Report AI-1990-01*, University of Georgia.

Covington, M. A. (1990b). Parsing discontinuous constituents in dependency grammar. *Computational Linguistics* **16**: 234–236.

Covington, M. A. (1994). Discontinuous dependency parsing of free and fixed word order: Work in progress, *Technical Report AI-1994-02*, University of Georgia.

Covington, M. A. (2001). A fundamental algorithm for dependency parsing. *Proceedings of the 39th Annual ACM Southeast Conference*, pp. 95–102.

Cowie, J. and Wilks, Y. (2000). Information extraction. In Dale, R., Moisl, H. and Somers, H. (eds), *Handbook of Natural Language Processing*, Marcel Dekker, pp. 241–260.

Curran, J. R. and Clark, S. (2004). The importance of supertagging for wide-coverage CCG parsing. *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pp. 282–288.

Daelemans, W. (1999). Memory-based language processing: Introduction to the special issue. *Journal of Experimental and Theoretical Artificial Intelligence* **11**: 287–292.

Daelemans, W., Buchholz, S. and Veenstra, J. (1999). Memory-based shallow parsing. *Proceedings of the 3rd Conference on Computational Natural Language Learning (CoNLL)*, pp. 77–89.

Daelemans, W., Durieux, G. and Gillis, S. (1994). The acquisition of stress. *Computational Linguistics* **20**: 421–451.

Daelemans, W. and Hoste, V. (2002). Evaluation of machine learning methods for natural language processing tasks. *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC)*, pp. 755–760.

Daelemans, W., Hoste, V., De Meulder, F. and Naudts, B. (2003). Combined optimization of feature selection and algorithm parameter interaction in machine learning of language. *Proceedings of the 14th European Conference on Machine Learning (ECML)*, pp. 84–95.

Daelemans, W. and Van den Bosch, A. (1992). Generalisation performance of backpropagation learning on a syllabification task. In Drossaers, M. F. J.

and Nijholt, A. (eds), *Proceedings of TWLT3: Connectionism and Natural Language Processing*, Springer, pp. 77–89.

Daelemans, W. and Van den Bosch, A. (1996). A language-independent data-oriented grapheme-to-phoneme conversion. In Van Santen, J. P. H., Sproat, R. W., Olive, J. P. and Hirschberg, J. (eds), *Progress in Speech Synthesis*, Springer, pp. 77–89.

Daelemans, W. and Van den Bosch, A. (2005). *Memory-Based Language Processing*. Cambridge University Press.

Daelemans, W., Van den Bosch, A. and Zavrel, J. (2002). Forgetting exceptions is harmful in language learning. *Machine Learning* **34**: 11–43.

Daelemans, W., Zavrel, J., Berck, P. and Gillis, S. (1996). A memory-based part of speech tagger generator. In Ejerhed, E. and Dagan, I. (eds), *Proceedings of the Fourth Workshop on Very Large Corpora*, pp. 14–27.

Daelemans, W., Zavrel, J., Van der Sloot, K. and Van den Bosch, A. (2004). TiMBL: Tilburg Memory Based Learner, version 5.1, Reference Guide, *Technical Report ILK 04-02*, Tilburg University (ILK) and University of Antwerp (CNTS).

Dagan, I. and Krymolowski, Y. (2003). Compositional memory-based parsing. In Bod, R., Scha, R. and Sima'an, K. (eds), *Data-Oriented Parsing*, CSLI Publications, University of Chicago Press, pp. 169–188.

Dagan, I., Lee, L. and Pereira, F. (1999). Similarity-based models of word cooccurrence. *Machine Learning* **34**: 11–41.

De Meulder, F. and Daelemans, W. (2003). Memory-based named entity recognition using unannotated data. *Proceedings of the Seventh Conference on Computational Natural Language Learning (CoNLL)*, pp. 208–211.

De Pauw, G. (2003). An approximation of DOP through memory-based learning. In Bod, R., Scha, R. and Sima'an, K. (eds), *Data-Oriented Parsing*, CSLI Publications, University of Chicago Press, pp. 147–167.

De Saussure, F. (1916). *Cours de linguistique générale*. Payot.

Debusmann, R. (2001). *A declarative grammar formalism for dependency grammar*, Master's thesis, Computational Linguistics, Universität des Saarlandes.

Debusmann, R., Duchier, D. and Kruijff, G.-J. M. (2004). Extensible dependency grammar: A new methodology. *Proceedings of the Workshop on Recent Advances in Dependency Grammar*, pp. 78–85.

Della Pietra, S., Della Pietra, V. and Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**.

Devijver, P. A. and Kittler, J. (1982). *Pattern Recognition: A Statistical Approach*. IEEE Computer Society Press.

Dickinson, M. and Meurers, W. D. (2003). Detecting inconsistencies in treebanks. In Nivre, J. and Hinrichs, E. (eds), *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, Växjö University Press, pp. 45–56.

Diderichsen, P. (1946). *Elementær dansk grammatik*. Gyldendal.

Dienes, P. and Dubey, A. (2003). Deep syntactic processing by combining shallow methods. *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 431–438.

Dowty, D. (1989). On the semantic content of the notion of 'thematic role'. In Chierchia, G., Partee, B. H. and Turner, R. (eds), *Properties, Types and Meaning. Volume II: Semantic Issues*, Reidel, pp. 69–130.

Dubey, A. and Keller, F. (2003). Probabilistic parsing for German using sister-head dependencies. *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 96–103.

Duchier, D. (1999). Axiomatizing dependency parsing using set constraints. *Proceedings of the Sixth Meeting on Mathematics of Language*, pp. 115–126.

Duchier, D. (2003). Configuration of labeled trees under lexicalized constraints and principles. *Research on Language and Computation* **1**: 307–336.

Duchier, D. and Debusmann, R. (2001). Topological dependency trees: A constraint-based account of linear precedence. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 180–187.

Dudani, S. A. (1976). The distance-weighted k-nearest neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-6**: 325–327.

Earley, J. (1970). En efficient context-free parsing algorithm. *Communications of the ACM* **13**: 94–102.

Edmonds, J. (1967). Optimum Branchings. *Journal of Research of the National Bureau of Standards* **71B**: 233–240.

Einarsson, J. (1976a). Talbankens skriftspråkskonkordans. Lund University, Department of Scandinavian Languages.

Einarsson, J. (1976b). Talbankens talspråkskonkordans. Lund University, Department of Scandinavian Languages.

Eisner, J. M. (1996a). An empirical comparison of probability models for dependency grammar, *Technical Report IRCS-96-11*, Institute for Research in Cognitive Science, University of Pennsylvania.

Eisner, J. M. (1996b). Three new probabilistic models for dependency parsing: An exploration. *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pp. 340–345.

Eisner, J. M. (2000). Bilexical grammars and their cubic-time parsing algorithms. In Bunt, H. and Nijholt, A. (eds), *Advances in Probabilistic and Other Parsing Technologies*, Kluwer, pp. 29–62.

Ejerhed, E. (1983). Finite state parsing. In Karlsson, F. (ed.), *Papers from the Seventh Scandinavian Conference of Linguistics*, pp. 410–432.

Ejerhed, E. and Källgren, G. (1997). Stockholm Umeå Corpus. Version 1.0. Produced by Department of Linguistics, Umeå University and Department of Linguistics, Stockholm University. ISBN 91-7191-348-3.

Escudero, G., Màrquez, L. and Rigau, G. (2000). Naive bayes and exemplar-based approaches to word sense disambiguation revisited. *Proceedings of the 14th European Conference on Artificial Intelligence*, pp. 421–425.

Fillmore, C. J. (1968). The case for case. In Bach, E. W. and Harms, R. T. (eds), *Universals in Linguistic Theory*, Holt, Rinehart and Winston, pp. 1–88.

Fix, E. and Hodges, J. (1952). Discriminatory analysis: Nonparametric discrimination: Consistency properties, *Technical Report 11*, USAF School of Aviation Medicine, Randolph Field, Texas.

Forst, M., Bertomeu, N., Crysmann, B., Fouvry, F., Hansen-Schirra, S. and Kordoni, V. (2004). The TIGER dependency bank. *Proceedings of the 5th International Workshop on Linguistically Interpreted Corpora*, pp. 31–37.

Foth, K., Daum, M. and Menzel, W. (2004). A broad-coverage parser for German based on defeasible constraints. *Proceedings of KONVENS 2004*, pp. 45–52.

Fraser, N. (1989). Parsing and dependency grammar. In Carston, R. (ed.), *UCL Working Papers in Linguistics 1*, University College London, pp. 296–319.

Fraser, N. (1993). *Dependency Parsing*. PhD thesis, University of London.

Fujii, A., Inui, K., Tokunaga, T. and Tanaka, H. (1998). Selective sampling for example-based word sense disambiguation. *Computational Linguistics* **24**: 573–598.

Fusijaki, T., Jelinek, F., Cocke, J., Black, E. and Nishino, T. (1989). A probabilistic method for sentence disambiguation. *Proceedings of the 1st International Workshop on Parsing Technologies*, pp. 105–114.

Gaifman, H. (1965). Dependency systems and phrase-structure systems. *Information and Control* **8**: 304–337.

Garside, R., Leech, G. and Varadi, T. (1992). Lancaster Parsed Corpus. A machine-readable syntactically analyzed corpus of 144,000 words, available for distribution through ICAME. Bergen: The Norwegian Computing Centre for the Humanities.

Gazdar, G., Klein, E., Pullum, G. and Sag, I. (1985). *Generalized Phrase Structure Grammar*. Blackwell.

Gildea, D. (2001). Corpus variation and parser performance. *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pp. 167–202.

Gildea, D. and Palmer, M. (2002). The necessity of syntactic parsing for predicate argument recognition. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 239–246.

Goodman, J. (1996). Parsing algorithms and metrics. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 177–183.

Goodman, J. (1998). *Parsing Inside-Out*. PhD thesis, Harvard University.

Grimaldi, R. P. (2004). *Discrete and Combinatorial Mathematics*. 5th edn, Addison-Wesley.

Grishman, R., Macleod, C. and Sterling, J. (1992). Evaluating parsing strategies using standardized parse files. *Proceedings of the 3rd ACL Conference on Applied Natural Language Processing*, pp. 156–161.

Grishman, R., Thanh Nhan, N., Marsh, L. and Hirschman, L. (1984). Automated determination of sublanguage syntactic usage. *Proceedings of the 10th International Conference on Computational Linguistics (COLING) and the 22nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 96–100.

Hajič, J. (1998). Building a syntactically annotated corpus: The Prague Dependency Treebank. *Issues of Valency and Meaning*, Karolinum, pp. 106–132.

Hajič, J., Vidova Hladka, B., Panevová, J., Hajičová, E., Sgall, P. and Pajas, P. (2001). Prague Dependency Treebank 1.0. LDC, 2001T10.

Hall, J. (2003). *A probabilistic part-of-speech tagger with suffix probabilities*, Master's thesis, Växjö University.

Hammerton, J., Osborne, M., Armstrong, S. and Daelemans, W. (2002). Introduction to special issue on machine learning approaches to shallow parsing. *Journal of Machine Learning Research* **2**: 551–558.

Han, C., Han, N. and Ko, S. (2002). Development and evaluation of a Korean treebank and its application to NLP. *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC)*, pp. 1635–1642.

Harper, M. P. and Helzerman, R. A. (1995). Extensions to constraint dependency parsing for spoken language processing. *Computer Speech and Language* **9**: 187–234.

Harper, M. P., Helzermann, R. A., Zoltowski, C. B., Yeo, B. L., Chan, Y., Steward, T. and Pellom, B. L. (1995). Implementation issues in the development of the PARSEC parser. *Software: Practice and Experience* **25**: 831–862.

Hastie, T., Tibshirani, R. and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer.

Hays, D. G. (1964). Dependency theory: A formalism and some observations. *Language* **40**: 511–525.

Hellwig, P. (1980). PLAIN – a program system for dependency analysis and for simulating natural language inference. In Bolc, L. (ed.), *Representation and Processing of Natural Language*, Hanser, pp. 195–198.

Hellwig, P. (1986). Dependency unification grammar. *Proceedings of the 11th International Conference on Computational Linguistics (COLING)*, pp. 195–198.

Hellwig, P. (2003). Dependency unification grammar. In Agel, V., Eichinger, L. M., Eroms, H.-W., Hellwig, P., Heringer, H. J. and Lobin, H. (eds), *Dependency and Valency*, Walter de Gruyter, pp. 593–635.

Henderson, J. (2004). Discriminative training of a neural network statistical parser. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 96–103.

Hendrickx, I. and Van den Bosch, A. (2003). Memory-based one-step named-entity recognition: Effects of seed list features, classifier stacking, and unannotated data. *Proceedings of the Seventh Conference on Computational Natural Language Learning (CoNLL)*, pp. 176–179.

Hindle, D. (1989). Acquiring disambiguation rules from text. *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 118–125.

Hindle, D. (1994). A parser for text corpora. In Zampolli, A. (ed.), *Computational Approaches to the Lexicon*, Oxford University Press, pp. 103–151.

Hindle, D. and Rooth, M. (1991). Structural ambiguity and lexical relations. *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 229–236.

Hinrichs, E. and Simov, K. (eds) (2002). *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*.

Hjelmslev, L. (1943). *Omkring sprogteoriens grundlæggelse*. Akademisk forlag.

Hockenmaier, J. (2003a). *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh.

Hockenmaier, J. (2003b). Parsing with generative models of predicate-argument structure. *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 359–366.

Holan, T., Kuboň, V. and Plátek, M. (1997). A prototype of a grammar checker for Czech. *Fifth Conference on Applied Natural Language Processing*, pp. 147–154.

Hopcroft, J. E., Motwani, R. and Ullman, J. D. (2001). *Introduction to Automata Theory, Languages, and Computation*. 2nd edn, Addison-Wesley.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Hudson, R. A. (1984). *Word Grammar*. Blackwell.

Hudson, R. A. (1990). *English Word Grammar*. Blackwell.

Isozaki, H., Kazawa, H. and Hirao, T. (2004). A deterministic word dependency analyzer enhanced with preference learning. *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pp. 275–281.

Jackendoff, R. (1972). *Semantic Interpretation in Generative Grammar*. MIT Press.

Jackendoff, R. S. (1977). $\overline{X}$ *Syntax: A Study of Phrase Structure*. MIT Press.

Järvinen, T. (2003). Multi-layered annotation scheme for treebank annotation. In Nivre, J. and Hinrichs, E. (eds), *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, Växjö University Press, pp. 93–104.

Järvinen, T. and Tapanainen, P. (1998). Towards an implementable dependency grammar. In Kahane, S. and Polguère, A. (eds), *Proceedings of the Workshop on Processing of Dependency-Based Grammars*, pp. 1–10.

Jebara, T. (2004). *Machine Learning: Discriminative and Generative*. Kluwer Academic Publishers.

Jelinek, F., Lafferty, J., Magerman, D., Mercer, R., Ratnaparkhi, A. and Roukos, S. (1994). Decision tree parsing using a hidden derivation model. *Proceedings of the 1994 Human Language Technology Workshop*, pp. 272–277.

Jensen, K. (1988). Why computational grammarians can be skeptical about existing linguistic theories. *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, pp. 448–449.

Jensen, K. and Heidorn, G. E. (1983). The fitted parse: 100% parsing capability in a syntactic grammar of English. *Proceedings of the [First] Conference on Applied Natural Language Processing*, pp. 93–98.

Jijkoun, V. and De Rijke, M. (2004). Enriching the output of a parser using memory-based learning. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 312–319.

Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar*. CSLI Publications, University of Chicago Press.

Johnson, M. (2001). Joint and conditional estimation of tagging and parsing models. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 314–321.

Johnson, M. (2002). A simple pattern-matching algorithm for recovering empty nodes and their antecedents. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 136–143.

Johnson, M., Geman, S., Canon, S., Chi, Z. and Riezler, S. (1999). Estimators for stochastic "unification-based" grammars. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 535–541.

Joshi, A. (1985). How much context-sensisitivity is necessary for characterizing structural descriptions – tree adjoining grammars. In Dowty, D., Karttunen, L. and Zwicky, A. (eds), *Natural Language Processing: Psycholinguistic, Computational and Theoretical Perspectives*, Cambridge University Press, pp. 206–250.

Joshi, A. K. (1997). Tree-adjoining grammars. In Rozenberg, G. and Salomaa, A. (eds), *Handbook of Formal Languages. Volume 3: Beyond Words*, Springer, pp. 69–123.

Joshi, A. and Sarkar, A. (2003). Tree adjoining grammars and their application to statistical parsing. In Bod, R., Scha, R. and Sima'an, K. (eds), *Data-Oriented Parsing*, CSLI Publications, University of Chicago Press, pp. 253–281.

Junqua, J.-C. and Van Noord, G. (eds) (2001). *Robustness in Language and Speech Technology*. Kluwer Academic Publishers.

Kahane, S., Nasr, A. and Rambow, O. (1998). Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pp. 646–652.

Kaplan, R. and Bresnan, J. (1982). Lexical-Functional Grammar: A formal system for grammatical representation. In Bresnan, J. (ed.), *The Mental Representation of Grammatical Relations*, MIT Press, pp. 173–281.

Kaplan, R. M., Riezler, S., King, T. H., Maxwell III, J. T., Vasserman, A. and Crouch, R. (2004). Speed and accuracy in shallow and deep stochastic

parsing. *Proceedings of Human Language Technology and the Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pp. 97–104.

Karlsson, F. (1990). Constraint grammar as a framework for parsing running text. In Karlgren, H. (ed.), *Papers presented to the 13th International Conference on Computational Linguistics (COLING)*, pp. 168–173.

Karlsson, F., Voutilainen, A., Heikkilä, J. and Anttila, A. (eds) (1995). *Constraint Grammar: A language-independent system for parsing unrestricted text*. Mouton de Gruyter.

Kasami, T. (1965). An efficient recognition and syntax algorithm for context-free languages, *Technical Report AF-CRL-65-758*, Air Force Cambridge Research Laboratory.

Kay, M. (1980). Algorithm schemata and data structures in syntactic processing, *Technical Report CSL-80-12*, Xerox PARC.

Kay, M. (1989). Head-driven parsing. *Proceedings of the International Workshop on Parsing Technologies*, pp. 52–62.

Kay, M. (2000). Guides and oracles for linear-time parsing. *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT)*, pp. 6–9.

King, T. H., Crouch, R., Riezler, S., Dalrymple, M. and Kaplan, R. M. (2003). The PARC 700 dependency bank. *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora*, pp. 1–8.

Klein, D. and Manning, C. D. (2002). Conditional structure versus conditional estimation in NLP models. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9–16.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 423–430.

Klein, D. and Manning, C. D. (2004). Corpus-based induction of syntactic structure: Models of dependency and constituency. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 479–486.

Knuth, D. E. (1965). On the translation of languages from left to right. *Information and Control* **8**: 607–639.

Kokkinakis, D. and Johansson Kokkinakis, S. (1999). A cascaded finite-state parser for syntactic analysis of Swedish. *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 245–248.

Koskenniemi, K. (1990). Finite-state parsing and disambiguation. In Karlgren, H. (ed.), *Papers presented to the 13th International Conference on Computational Linguistics (COLING)*, pp. 229–232.

Koskenniemi, K. (1997). Representations and finite-state components in natural language. In Roche, E. and Schabes, Y. (eds), *Finite State Language Processing*, MIT Press, pp. 99–116.

Kouchnir, B. (2004). A memory-based approach for semantic role labeling. In Ng, H. T. and Riloff, E. (eds), *Proceedings of the 8th Conference on Computational Natural Language Learning*, pp. 118–121.

Kromann, M. T. (2003). The Danish Dependency Treebank and the DTAG treebank tool. In Nivre, J. and Hinrichs, E. (eds), *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, Växjö University Press, pp. 217–220.

Kromann, M. T. (2004). Optimality parsing and local cost functions in Discontinuous Grammar. *Electronic Notes of Theoretical Computer Science* **53**: 163–179.

Kruijff, G.-J. M. (2001). *A Categorial-Modal Logical Architecture of Informativity: Dependency Grammar Logic and Information Structure*. PhD thesis, Charles University.

Kruijff, G.-J. M. (2002). Formal and computational aspects of dependency grammar: History and development of DG, *Technical report*, ESSLLI-2002.

Krymolowski, Y. and Dagan, I. (2000). Incorporating compositional evidence in memory-based partial parsing. *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pp. 45–52.

Kübler, S. (2004). *Memory-Based Parsing*. John Benjamins.

Kübler, S. and Hinrichs, E. W. (2001). From chunks to function-argument structure: A similarity-based approach. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 346–353.

Kübler, S., Nivre, J., Hinrichs, E. and Wunsch, H. (eds) (2004). *Proceedings of the Third Workshop on Treebanks and Linguistic Theories (TLT)*, University of Tübingen.

Kübler, S. and Telljohann, H. (2002). Towards a dependency-based evaluation for partial parsing. *Proceedings of the LREC-Workshop Beyond PARSEVAL – Towards Improved Evaluation Measures for Parsing Systems*, pp. 9–16.

Kucera, H. and Francis, W. (1967). *Computational Analysis of Present-Day American English*. Brown University Press.

Kudo, T. and Matsumoto, Y. (2000). Use of support vector learning for chunk identification. In Cardie, C., Daelemans, W. and Tjong Kim Sang, E. (eds), *Proceedings of the 3rd Conference on Computational Natural Language Learning (CoNLL)*, pp. 142–144.

Kudo, T. and Matsumoto, Y. (2002). Japanese dependency analysis using cascaded chunking. *Proceedings of the Sixth Workshop on Computational Language Learning (CoNLL)*, pp. 63–69.

Kurohashi, S. and Nagao, M. (2003). Building a Japanese parsed corpus. In Abeillé, A. (ed.), *Treebanks: Building and Using Parsed Corpora*, Kluwer Academic Publishers, pp. 249–260.

Lambek, J. (1958). The mathematics of sentence structure. *American Mathematical Monthly* **65**: 154–170.

Lang, B. (1988). Parsing incomplete sentences. *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, pp. 365–371.

Lecerf, Y. (1960). Programme des conflits, modèle des conflits. *Bulletin bimestriel de l'ATALA* **1**: (4): 11–18, (5): 17–36.

Lehmann, S., Oepen, S., Regnier-Prost, S., Netter, K., Lux, V., Klein, J., Falkedal, K., Fouvry, F., Estival, D., Dauphin, E., Compagnion, H., Baur, J., Balkan, L. and Arnold, D. (1996). TSNLP – test suites for natural language processing. *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pp. 711–716.

Lehnert, W. (1987). Case-based problem solving with a large knowledge base of learned cases. *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 301–306.

Levy, R. and Manning, C. (2004). Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 328–335.

Lin, D. (1995). A dependency-based method for evaluating broad-coverage parsers. *Proceedings of IJCAI-95*, pp. 1420–1425.

Lin, D. (1996). On the structural complexity of natural language sentences. *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pp. 729–733.

Lin, D. (1998). A dependency-based method for evaluating broad-coverage parsers. *Natural Language Engineering* **4**: 97–114.

Lindgren, B. W. (1993). *Statistical Theory*. Chapman and Hall.

Ljunglöf, P. (2004). *Expressivity and Complexity of the Grammatical Framework*. PhD thesis, Chalmers University of Technology.

Lombardo, V. and Lesmo, L. (1996). An Earley-type recognizer for dependency grammar. *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pp. 723–728.

Lyons, J. (1977). *Semantics*. Cambridge University Press.

Maamouri, M. and Bies, A. (2004). Developing an Arabic treebank: Methods, guidelines, procedures, and tools. *Proceedings of the Workshop on Computational Approaches to Arabic Script-Based Languages*, pp. 2–9.

Magerman, D. M. (1995). Statistical decision-tree models for parsing. *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 276–283.

Malouf, R., Carroll, J. and Copestake, A. (2000). Efficient feature structure operations without compilation. *Natural Language Engineering* **6**: 29–46.

Manning, C. D. and Schütze, H. (2000). *Foundations of Statistical Natural Language Processing*. MIT Press.

Marcus, M. P. (1980). *A Theory of Syntactic Recognition for Natural Language*. MIT Press.

Marcus, M. P., Santorini, B. and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* **19**: 313–330.

Marcus, M. P., Santorini, B., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K. and Schasberger, B. (1994). The Penn Treebank: Annotating predicate-argument structure. *Proceedings of the ARPA Human Language Technology Workshop*, pp. 114–119.

Marcus, S. (1965). Sur la notion de projectivité. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* **11**: 181–192.

Marinov, S. and Nivre, J. (2005). A data-driven parser for Bulgarian. *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 89–100.

Maruyama, H. (1990). Structural disambiguation with constraint propagation. *Proceedings of the 28th Meeting of the Association for Computational Linguistics (ACL)*, pp. 31–38.

Masand, B., Linoff, G. and Waltz, D. (1992). Classifying news stories using memory-based reasoning. *Proceedings of SIGIR*, pp. 59–65.

Matthews, P. H. (1981). *Syntax*. Cambridge University Press.

McDonald, R., Crammer, K. and Pereira, F. (2005). Online large-margin training of dependency parsers. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 91–98.

McDonald, R., Pereira, F., Ribarov, K. and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pp. 523–530.

Megyesi, B. (2002). *Data-Driven Syntactic Analysis: Methods and Applications for Swedish*. PhD thesis, KTH: Department of Speech, Music and Hearing.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press.

Mellish, C. S. (1989). Some chart-based techniques for parsing ill-formed input. *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 102–109.

Menzel, W. (1995). Robust processing of natural language. *Proceedings of the 19th Annual German Conference on Artificial Intelligence*, pp. 19–34.

Menzel, W. and Schröder, I. (1998). Decision procedures for dependency parsing using graded constraints. In Kahane, S. and Polguère, A. (eds), *Proceedings of the Workshop on Processing of Dependency-Based Grammars*, pp. 78–87.

Milward, D. (1994). Dynamic dependency grammar. *Linguistics and Philosophy* **17**: 561–605.

Misra, V. N. (1966). *The Descriptive Technique of Panini*. Mouton.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Miyao, Y., Ninomiya, T. and Tsujii, J. (2003). Probabilistic modeling of argument structures including non-local dependencies. *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP)*, pp. 285–291.

Mohri, M. and Nederhof, M.-J. (2001). Regular approximation of context-free grammars. In Junqua, J.-C. and Van Noord, G. (eds), *Robustness in Language and Speech Technology*, Kluwer Academic Publishers, pp. 153–163.

Montague, R. (1970). Universal grammar. *Theoria* **36**: 373–398.

Montague, R. (1973). The proper treatment of quantification in ordinary English. In Hintikka, J., Moravcsik, J. M. E. and Suppes, P. (eds), *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*, Reidel, pp. 221–242.

Moreno, A., López, S., Sánchez, F. and Grishman, R. (2003). Developing a Spanish treebank. In Abeillé, A. (ed.), *Treebanks: Building and Using Parsed Corpora*, Kluwer Academic Publishers, pp. 149–163.

Morrill, G. (1994). *Type-Logical Grammar*. Kluwer Academic Publishers.

Morrill, G. (2000). Incremental processing and acceptability. *Computational Linguistics* **26**: 319–338.

Nakagawa, T., Kudoh, T. and Matsumoto, Y. (2002). Revision learning and its application to part-of-speech tagging. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 497–504.

Nasr, A. and Rambow, O. (2004). A simple string-rewriting formalism for dependency grammar. *Proceedings of the Workshop on Recent Advances in Dependency Grammar*, pp. 25–32.

Nederhof, M.-J. (1998). Context-free parsing through regular approximation. *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pp. 13–24.

Nederhof, M.-J. (2000). Practical experiments with regular approximations of context-free languages. *Computational Linguistics* **26**: 17–44.

Nelson, G., Wallis, S. and Aarts, B. (2002). *Exploring Natural Language: Working with the British Component of the International Corpus of English*. John Benjamins.

Ney, H. (1991). Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing* **39**: 336–340.

Ng, H. and Lee, H. (1996). Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 40–47.

Nikula, H. (1986). *Dependensgrammatik*. Liber.

Nilsson, J., Hall, J. and Nivre, J. (2005). MAMBA meets TIGER: Reconstructing a Swedish treebank from Antiquity. In Henrichsen, P. J. (ed.), *Proceedings of the NODALIDA Special Session on Treebanks*.

Nivre, J. (2002). What kinds of trees grow in Swedish soil? A comparison of four annotation standards for Swedish. In Hinrichs, E. and Simov, K. (eds), *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 123–138.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In Van Noord, G. (ed.), *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 149–160.

Nivre, J. (2004a). Incrementality in deterministic dependency parsing. In Keller, F., Clark, S., Crocker, M. and Steedman, M. (eds), *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pp. 50–57.

Nivre, J. (2004b). Inductive dependency parsing, *Technical Report 04070*, Växjö University, School of Mathematics and Systems Engineering.

Nivre, J. (2005). Bootstrapping lexical models for memory-based dependency parsing. *Proceedings of Promote IT 2005*, Studentlitteratur, pp. 327–336.

Nivre, J. (forthcoming). Treebanks. In Kytö, M., Lüdeling, A. and McEnery, T. (eds), *Handbook of Corpus Linguistics*, Mouton de Gruyter.

Nivre, J. and Hall, J. (2005). MaltParser: A language-independent system for data-driven dependency parsing. *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 137–148.

Nivre, J., Hall, J. and Nilsson, J. (2004). Memory-based dependency parsing. In Ng, H. T. and Riloff, E. (eds), *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pp. 49–56.

Nivre, J. and Hinrichs, E. (eds) (2003). *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, Växjö University Press.

Nivre, J. and Nilsson, J. (2004). Multiword units in syntactic parsing. *Proceedings of the Workshop on Methodologies and Evaluation of Multiword Units in Real-World Applications (LREC)*, pp. 39–46.

Nivre, J. and Nilsson, J. (2005). Pseudo-projective dependency parsing. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 99–106.

Nivre, J. and Scholz, M. (2004). Deterministic dependency parsing of English text. *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pp. 64–70.

Obrębski, T. (2003). Dependency parsing using dependency graph. In Van Noord, G. (ed.), *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 217–218.

Oepen, S. and Carroll, J. (2000). Parser engineering and performance profiling. *Natural Language Engineering* **6**: 81–97.

Oepen, S. and Flickinger, D. (1998). Towards systematic grammar profiling: Test suite technology ten years after. *Computer Speech and Language* **12**: 411–435.

Oflazer, K. (2003). Dependency parsing with an extended finite-state approach. *Computational Linguistics* **29**: 515–544.

Oflazer, K., Say, B., Hakkani-Tür, D. Z. and Tür, G. (2003). Building a Turkish treebank. In Abeillé, A. (ed.), *Treebanks: Building and Using Parsed Corpora*, Kluwer Academic Publishers, pp. 261–277.

Palmer, D. M. (2000). Tokenisation and sentence segmentation. In Dale, R., Moisl, H. and Somers, H. (eds), *Handbook of Natural Language Processing*, Marcel Dekker, pp. 11–35.

Pereira, F. C. N. and Wright, R. N. (1997). Finite-state approximation of phrase-structure grammars. In Roche, E. and Schabes, Y. (eds), *Finite-State Language Processing*, MIT Press, pp. 149–174.

Pereira, F. C. and Schabes, Y. (1992). Inside-outside reestimation from partially bracketed corpora. *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 128–135.

Pollard, C. and Sag, I. A. (1987). *Information-Based Syntax and Semantics*. CSLI Publications, University of Chicago Press.

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. CSLI Publications, University of Chicago Press.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning* **1**: 81–206.

Ramshaw, L. A. and Marcus, M. P. (1995). Text chunking using transformation-based learning. *Proceedings of the Third Annual Workshop on Very Large Corpora*, pp. 82–94.

Ranta, A. (2004). Grammatical Framework, a type-theoretical grammar formalism. *Journal of Functional Programming* **14**: 145–189.

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1–10.

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning* **34**: 151–175.

Resnik, P. (1992). Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, pp. 418–424.

Riezler, S., King, M., Kaplan, R., Crouch, R., Maxwell, J. and Johnson, M. (2002). Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 271–278.

Riloff, E. and Lehnert, W. (1994). Information extraction as a basis for high-precision text classification. *ACM Transactions on Information Systems* **12**: 296–333.

Robins, R. H. (1967). *A Short History of Linguistics*. Longman.

Robinson, J. J. (1970). Dependency structures and transformational rules. *Language* **46**: 259–285.

Roche, E. (1997). Parsing with finite state transducers. In Roche, E. and Schabes, Y. (eds), *Finite-State Language Processing*, MIT Press, pp. 241–281.

Rosenfeld, R. (2000). Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE* **88**: 1270–1278.

Rosenkrantz, D. J. and Lewis, P. M. (1970). Deterministic left corner parsing. *Proceedings of the 11th Symposium on Switching and Automata Theory*, pp. 139–152.

Sagae, K. and Lavie, A. (2005). A classifier-based parser with linear runtime complexity. *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pp. 125–132.

Sågvall Hein, A. (1982). An experimental parser. *Proceedings of the Ninth International Conference on Computational Linguistics (COLING)*, pp. 121–126.

Sampson, G. (1995). *English for the Computer: The SUSANNE Corpus and Analytic Scheme*. Oxford University Press.

Sampson, G. (2003). Thoughts on two decades of drawing trees. *Treebanks: Building and Using Parsed Corpora*, Kluwer Academic Publishers, pp. 23–41.

Samuelsson, C. (2000). A statistical theory of dependency syntax. *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*.

Samuelsson, C. and Rayner, M. (1991). Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. *Proceedings of the 12th International Joint Conference on Artificial Intelligence*.

Samuelsson, C. and Wirén, M. (2000). Parsing techniques. In Dale, R., Moisl, H. and Somers, H. (eds), *Handbook of Natural Language Processing*, Marcel Dekker, pp. 59–91.

Sapir, E. (1921). *Language: An Introduction to the Study of Speech*. Harcourt Brace.

Schabes, Y. (1992). Stochastic lexicalized tree adjoining grammars. *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, pp. 426–432.

Schabes, Y., Abeillé, A. and Joshi, A. (1988). Parsing strategies with "lexicalized" grammars: Applications to tree adjoining grammars. *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, pp. 578–583.

Schröder, I. (2002). *Natural Language Parsing with Graded Constraints*. PhD thesis, Hamburg University.

Sgall, P., Hajičová, E. and Panevová, J. (1986). *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel.

Shieber, S. M. (1983). Sentence disambiguation by a shift-reduce parsing technique. *Proceedings of the 21st Conference on Association for Computational Linguistics (ACL)*, pp. 113–118.

Shieber, S. M. (1986). *An Introduction to Unification-Based Approaches to Grammar*. CSLI Publications, University of Chicago Press.

Shieber, S. M., Schabes, Y. and Pereira, F. C. N. (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming* **24**: 3–36.

Sima'an, K. (1996a). Computational complexity of probabilistic disambiguation by means of tree grammar. *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pp. 1175–1180.

Sima'an, K. (1996b). An optimized algorithm for data-oriented parsing. In Mitkov, R. and Nicolov, N. (eds), *Recent Advances in Natural Language Processing. Selected Papers from RANLP '95*, John Benjamins, pp. 35–47.

Sima'an, K. (1999). *Learning Efficient Disambiguation*. PhD thesis, University of Amsterdam.

Sima'an, K. (2003). On maximizing metrics for syntactic disambiguation. In Van Noord, G. (ed.), *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 183–194.

Skousen, R. (1989). *Analogical Modeling of Language*. Kluwer.

Skousen, R. (1992). *Analogy and Structure*. Kluwer.

Sleator, D. and Temperley, D. (1991). Parsing English with a link grammar, *Technical Report CMU-CS-91-196*, Carnegie Mellon University, Computer Science.

Sleator, D. and Temperley, D. (1993). Parsing English with a link grammar. *Third International Workshop on Parsing Technologies (IWPT)*, pp. 277–292.

Stanfill, C. and Waltz, D. (1986). Toward memory-based reasoning. *Communications of the ACM* **29**: 1213–1228.

Starosta, S. (1988). *The Case for Lexicase: An Outline of Lexicase Grammatical Theory*. Pinter Publishers.

Steedman, M. (2000). *The Syntactic Process*. MIT Press.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics* **21**: 165–202.

Streiter, O. (2001a). Memory-based parsing: Enhancing recursive top-down fuzzy match with bottom-up chunking. *Proceedings of the 19th International Conference on Computer Processing of Oriental Languages (ICCPOL)*, pp. 219–224.

Streiter, O. (2001b). Recursive top-down fuzzy match: New perspectives for memory-based parsing. *Proceedings of the 15th Pacific Asia Conference on Language, Information and Computation (PACLIC)*, pp. 345–356.

Tang, M., Luo, X. and Roukos, S. (2002). Active learning for statistical natural language parsing. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 406–414.

Tapanainen, P. and Järvinen, T. (1997). A non-projective dependency parser. *Proceedings of the 5th Conference on Applied Natural Language Processing*, pp. 64–71.

Teleman, U. (1974). *Manual för grammatisk beskrivning av talad och skriven svenska*. Studentlitteratur.

Tesnière, L. (1959). *Éléments de syntaxe structurale*. Editions Klincksieck.

Thompson, C. A., Califf, M. E. and Mooney, R. J. (1999). Active learning for natural language parsing and information extraction. *Proceedings of*

the Sixteenth International Conference on Machine Learning (ICML), pp. 406–414.

Thompson, H. S. (1981). Chart parsing and rule schemata in GPSG. *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 167–172.

Tjong Kim Sang, E. F. and Veenstra, J. (1999). Representing text chunks. *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 173–179.

Tjong Kim Sang, E. F. and Veenstra, J. (2001). Transforming a chunker to a parser. In Daelemans, W., Sima'an, K., Veenstra, J. and Zavrel, J. (eds), *Computational Linguistics in the Netherlands 2000*, pp. 177–188.

Tomita, M. (1987). An efficient augmented-context-free parsing algorithm. *Computational Linguistics* **13**: 31–46.

Torisawa, K., Nishida, K., Miyao, Y. and Tsujii, J. (2000). An HPSG parser with CFG filtering. *Natural Language Engineering* **6**: 63–80.

Toutanova, K., Manning, C. D., Shieber, S. M., Flickinger, D. and Oepen, S. (2002). Parse disambiguation for a rich HPSG grammar. In Hinrichs, E. and Simov, K. (eds), *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 253–263.

Tzoukermann, E., Klavans, J. L. and Strzalkowski, T. (2003). Information retrieval. In Mitkov, R. (ed.), *The Oxford Handbook of Computational Linguistics*, Oxford University Press, pp. 529–544.

Ule, T. and Kübler, S. (2004). From phrase structure to dependencies, and back. *Proceedings of the International Conference on Linguistic Evidence*, pp. 169–170.

Uszkoreit, H. (1986). Categorial unification grammars. *Proceedings of the 5th International Conference on Computational Linguistics (COLING)*, pp. 187–194.

Van den Bosch, A. and Buchholz, S. (2002). Shallow parsing on the basis of words only: A case study. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 443–440.

Van den Bosch, A., Canisius, S., Daelemans, W., Hendrickx, I. and Tjong Kim Sang, E. (2004). Memory-based semantic role labeling: Optimizing features, algorithm, and output. In Ng, H. T. and Riloff, E. (eds), *Proceedings of the 8th Conference on Computational Natural Language Learning*, pp. 102–105.

Van den Bosch, A. and Daelemans, W. (1999). Memory-based morphological analysis. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 285–292.

Van der Beek, L., Bouma, G., Malouf, R. and Van Noord, G. (2002). The Alpino dependency treebank. In Theune, M., Nijholt, A. and Hondorp, H. (eds), *Language and Computers, Computational Linguistics in the Netherlands 2001. Selected Papers from the Twelfth CLIN Meeting*, Rodopi, pp. 8–22.

Van Noord, G. (1997). An efficient implementation of the head-corner parser. *Computational Linguistics* **23**: 425–456.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer.

Veenstra, J. (1998). Fast NP chunking using memory-based learning techniques. *Proceedings of BENELEARN-98*, pp. 71–79.

Veenstra, J. and Daelemans, W. (2000). A memory-based alternative for connectionist shift-reduce parsing, *Technical Report ILK-0012*, Tilburg University.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* **IT-13**: 1260–1269.

Voutilainen, A. (2001). Parsing Swedish. Extended Abstract for the 13th Nordic Conference of Computational Linguistics, Uppsala University, May, 20–22, 2001.

Wang, W. and Harper, M. P. (2004). A statistical constraint dependency grammar (CDG) parser. In Keller, F., Clark, S., Crocker, M. and Steedman, M. (eds), *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pp. 29–42.

Weijters, A. J. M. M. (1991). A simple look-up procedure superior to NETtalk? *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, pp. 1645–1648.

Wettschereck, D. (1994). *A Study of Distance-Based Machine Learning Algorithms*. PhD thesis, Oregon State University.

White, A. P. and Liu, W. Z. (1994). Bias in information-based measures in decision tree induction. *Machine Learning* **15**: 321–329.

Wu, T.-F., Lin, C.-J. and Weng, R. C. (2004). Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research* **5**: 975–1005.

Xia, F. (2001). *Automatic Grammar Generation from Two Different Perspectives*. PhD thesis, University of Pennsylvania.

Xia, F. and Palmer, M. (2001). Converting dependency structures to phrase structures. In Allan, J. (ed.), *Proceedings of HLT 2001, First International Conference on Human Language Technology Research*.

Xue, N., Xia, F., Chiou, F.-D. and Palmer, M. (2004). The Penn Chinese Treebank: Phase structure annotation of a large corpus. *Natural Language Engineering* **11**: 207–238.

Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In Van Noord, G. (ed.), *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 195–206.

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 523–530.

Yang, Y. and Chute, C. (1994). An example-based mapping method for text classification and retrieval. *ACM Transactions on Information Systems* **12**: 252–295.

Yli-Jyrä, A. (2003). Multiplanarity – a model for dependency structures in treebanks. In Nivre, J. and Hinrichs, E. (eds), *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, Växjö University Press, pp. 189–200.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control* **10**: 189–208.

Yuret, D. (1998). *Discovery of Linguistic Relations Using Lexical Attraction*. PhD thesis, Massachusetts Institute of Technology.

Zavrel, J. and Daelemans, W. (1997). Memory-based learning: Using similarity for smoothing. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 436–442.

Zavrel, J., Daelemans, W. and Veenstra, J. (1997). Resolving PP attachment ambiguities with memory-based learning. *Proceedings of the Workshop on Computational Natural Language Learning*, pp. 136–144.

Zeevat, H., Klein, E. and Calder, J. (1991). Unification categorial grammar. *Lingua e Stile* **26**: 499–527.

Zwicky, A. M. (1985). Heads. *Journal of Linguistics* **21**: 1–29.

Index