Acta Wexionensia
No 152/2008
Computer Science

Transition-Based Natural Language Parsing with Dependency and Constituency Representations

Johan Hall

Växjö University Press

Transition-Based Natural Language Parsing with Dependency and Constituency Representations. Thesis for the degree of Doctor of Philosophy, Växjö University, Sweden 2008.

Series editor: Kerstin Brodén
ISSN: 1404-4307
ISBN: 978-91-7636-625-7
Printed by: Intellecta Docusys, Göteborg 2008

Abstract

Hall, Johan, 2008. *Transition-Based Natural Language Parsing with Dependency and Constituency Representations*, Acta Wexionensia No 152/2008. ISSN: 1404-4307, ISBN: 978-91-7636-625-7. Written in English.

This thesis investigates different aspects of transition-based syntactic parsing of natural language text, where we view *syntactic parsing* as the process of mapping sentences in unrestricted text to their syntactic representations. Our parsing approach is data-driven, which means that it relies on machine learning from annotated linguistic corpora. Our parsing approach is also dependency-based, which means that the parsing process builds a dependency graph for each sentence consisting of lexical nodes linked by binary relations called dependencies. However, the output of the parsing process is not restricted to dependency-based representations, and the thesis presents a new method for encoding phrase structure representations as dependency representations that enable an inverse transformation without loss of information. The thesis is based on five papers, where three papers explore different ways of using machine learning to guide a transition-based dependency parser and two papers investigate the method for dependency-based phrase structure parsing.

The first paper presents our first large-scale empirical study of parsing a natural language (in this case Swedish) with labeled dependency representations using a transition-based deterministic parsing algorithm, where the dependency graph for each sentence is constructed by a sequence of transitions and memory-based learning (MBL) is used to predict the transition sequence. The second paper further investigates how machine learning can be used for guiding a transition-based dependency parser. The empirical study compares two machine learning methods with five feature models for three languages (Chinese, English and Swedish), and the study shows that support vector machines (SVM) with lexicalized feature models are better suited than MBL for guiding a transition-based dependency parser. The third paper summarizes our experience of optimizing and tuning MaltParser, our implementation of transition-based parsing, for a wide range of languages. MaltParser has been applied to over twenty languages and was one of the top-performing systems in the CoNLL shared tasks of 2006 and 2007.

The fourth paper is our first investigation of dependency-based phrase structure parsing with competitive results for parsing German. The fifth paper presents an improved encoding method for transforming phrase structure representations into dependency graphs and back. With this method it is possible to parse continuous and discontinuous phrase structure extended with grammatical functions.

Keywords: Natural Language Parsing, Syntactic Parsing, Dependency Structure, Phrase Structure, Machine Learning

Sammandrag

Denna doktorsavhandling undersöker olika aspekter av automatisk syntaktisk analys av texter på naturligt språk. En *parser* eller syntaktisk analysator, som vi definierar den i denna avhandling, har till uppgift att skapa en syntaktisk analys för varje mening i en text på naturligt språk. Vår metod är datadriven, vilket innebär att den bygger på maskininlärning från uppmärkta datamängder av naturligt språk, s.k. korpusar. Vår metod är också dependensbaserad, vilket innebär att parsning är en process som bygger en dependensgraf för varje mening, bestående av binära relationer mellan ord. Dessutom introducerar avhandlingen en ny metod för att koda frasstrukturer, en annan syntaktisk representationsform, som dependensgrafer vilka kan avkodas utan att information i frasstrukturen går förlorad. Denna metod möjliggör att en dependensbaserad parser kan användas för att syntaktiskt analysera frasstrukturer. Avhandlingen är baserad på fem artiklar, varav tre artiklar utforskar olika aspekter av maskininlärning för datadriven dependensparsning och två artiklar undersöker metoden för dependensbaserad frasstrukturparsning.

Den första artikeln presenterar vår första storskaliga empiriska studie av parsning av naturligt språk (i detta fall svenska) med dependensrepresentationer. En transitionsbaserad deterministisk parsningsalgoritm skapar en dependensgraf för varje mening genom att härleda en sekvens av transitioner, och minnesbaserad inlärning (MBL) används för att förutsäga transitionssekvensen. Den andra artikeln undersöker ytterligare hur maskininlärning kan användas för att vägleda en transitionsbaserad dependensparser. Den empiriska studien jämför två metoder för maskininlärning med fem särdragsmodeller för tre språk (kinesiska, engelska och svenska), och studien visar att supportvektormaskiner (SVM) med lexikaliserade särdragsmodeller är bättre lämpade än MBL för att vägleda en transitionsbaserad dependensparser. Den tredje artikeln sammanfattar vår erfarenhet av att optimera MaltParser, vår implementation av transitionsbaserad dependensparsning, för ett stort antal språk. MaltParser har använts för att analysera över tjugo olika språk och var bland de främsta systemen i CoNLLs utvärdering 2006 och 2007.

Den fjärde artikeln är vår första undersökning av dependensbaserad frasstrukturparsning med konkurrenskraftiga resultat för parsning av tyska. Den femte och sista artikeln introducerar en förbättrad algoritm för att transformera frasstrukturer till dependensgrafer och tillbaka, vilket gör det möjligt att parsa kontinuerliga och diskontinuerliga frasstrukturer utökade med grammatiska funktioner.

Publications

This thesis is based on the following papers, referred to throughout by their roman numerals:

I. Memory-Based Dependency Parsing
Joakim Nivre, Johan Hall and Jens Nilsson
In *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL)*, Boston, Massachusetts, USA, pp. 49–56, May 6–7, 2004.

II. Discriminative Classifiers for Deterministic Dependency Parsing
Johan Hall, Joakim Nivre and Jens Nilsson
In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 2006), Main Conference Poster Sessions*, Sydney, Australia, pp. 316–323, July 17–21, 2006.

III. Single Malt or Blended? A Study in Multilingual Parser Optimization
Johan Hall, Jens Nilsson and Joakim Nivre
To appear in Harry Bunt, Paola Merlo and Joakim Nivre (eds.) *Trends in Parsing Technology*, Springer-Verlag.

IV. A Dependency-Driven Parser for German Dependency and Constituency Representations
Johan Hall and Joakim Nivre
In *Proceedings of the ACL-08: HLT Workshop on Parsing German (PaGe-08)*, Columbus, Ohio, USA, pp. 47–54, 20 June, 2008.

V. Parsing Discontinuous Phrase Structure with Grammatical Functions
Johan Hall and Joakim Nivre
In *Proceedings of the 6th International Conference on Natural Language Processing (GoTAL 2008)*, LNAI 5221, Springer-Verlag, Gothenburg, Sweden, pp. 169–180, August 25–27, 2008.

Acknowledgments

First, I would like to thank my supervisor Joakim Nivre at Växjö University and Uppsala University for guidance and advice in my work on this thesis and for stimulating discussions and fun times when we developed MaltParser. I hope that this thesis is not the end of the development of MaltParser and that we can together explore other interesting phenomena of natural language parsing. I also want to thank my assistant supervisor Welf Löwe for interesting discussions and useful comments.

A big thank you to all my colleagues in computer science at Växjö University for making it fun to go to work every day. I especially want to thank Jens Nilsson for several useful tools, e.g., MaltEval and pseudo-projective parsing, and for fruitful discussions. Moreover, I want to thank Morgan Ericsson, Marcus Edvinsson and Niklas Brandt for all the work they have invested in keeping the Unix and Linux systems running when I loaded the systems to the maximum with experiments. Fortunately, I got access to the Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX) before we had a server melt-down in Växjö, and I want to thank all the people at UPPMAX for all extra computer power.

Thanks also to the organizers of the CoNLL shared tasks 2006 and 2007 and of the PaGe shared task on parsing German, and to all the treebank providers. A special thanks to Gülşen Eryiğit, Beáta Megyesi, Mattias Nilsson and Markus Saers for helping us with the optimization of the Single Malt parser in the CoNLL shared task 2007, and to Gülşen Eryiğit and Svetoslav Marinov for help in the CoNLL shared task 2006.

Without financial support from Växjö University, School of Mathematics and Systems Engineering, and the Swedish Research Council (Vetenskapsrådet, 621-2002-4207 and 2005-4123) this thesis would not have been written and therefore I am very grateful for this support. Thanks also to the Nordic Graduate School of Language Technology (NGSLT) and the Nordic Treebank Network for financial support of some of my trips around the world and to the Graduate School of Language Technology (GSLT) for excellent courses.

Unfortunately, my parents Gert and Karin cannot witness the completion of my PhD thesis, but I know that they would have been very proud of me. Finally, I want to express my gratitude to my love Kristina for making my world a wonderful place.

Contents

Chapter 1

Introduction

One of the challenges in natural language processing (NLP) is to transform text in natural language into representations that computers can use to perform many different tasks such as information extraction, machine translation and question answering. The transformation of natural language into formal representations usually involves several processing steps such as dividing text (or speech) into sentences and sentences into words, assigning parts of speech to words, and deriving syntactic and semantic representations for sentences. In this thesis, we will concentrate on the process of analyzing the syntactic structure of sentences. The term we will use is *syntactic parsing*, or simply *parsing*, which we will regard as the process of mapping sentences in unrestricted natural language text to their syntactic representations. Furthermore, the software component that performs this process is called a *syntactic parser*, or just *parser*.

The syntactic structure is formalized in a syntactic representation, and there exist several types of syntactic representation. In this thesis, we will use two types of syntactic representation based on the notion of *constituency* or *phrase structure* (Bloomfield, 1933; Chomsky, 1956) and the notion of *dependency* (Tesnière, 1959). Parsing a sentence with constituency representations means decomposing it into constituents or phrases, and in that way a phrase structure tree is created with relationships between words and phrases. Figure 1.1 illustrates a phrase structure tree, which contains four phrases. By contrast, with dependency representations the goal of parsing a sentence is to create a dependency graph consisting of lexical nodes linked by binary relations called *dependencies*. A dependency relation connects words with one word acting as *head* and the other as *dependent*. Figure 1.2 shows a dependency graph for an English sentence, where each word of the sentence is tagged with its part of speech and each edge labeled with a dependency type.

In this thesis, we assume that a syntactic parser should process sentences in unrestricted natural language text, which entails that the syntactic representations should be constructed regardless of whether the sentences are recognized by a formal grammar or not. In fact, the methodology we will use is not dependent on any grammar at all. Instead empirical data in the form of syntactically annotated text is used to build syntactic structures. Data-driven methods in natural language processing have been used

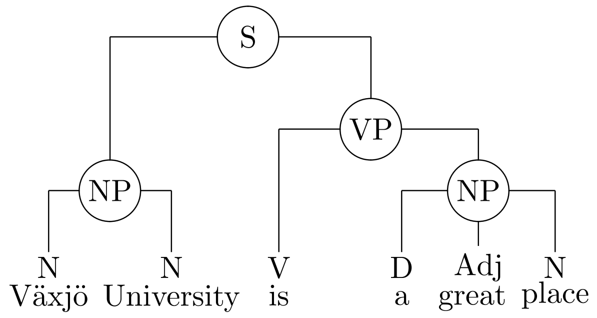


Figure 1.1: An example of a phrase structure representation for the sentence *Växjö University is a great place*, where each word is labeled with a part of speech and each phrase is labeled with a phrase category.

for many tasks in the past decade and syntactic parsing is one of the most prominent. In this thesis, we will concentrate on data-driven dependency parsing, but we will also explore ways in which a dependency-based parser can be used to derive phrase structure representations indirectly.

In data-driven dependency-based parsing, the two dominating approaches are graph-based dependency parsing and transition-based dependency parsing (McDonald and Nivre, 2007). The graph-based approach creates a parser model that assigns scores to all possible dependency graphs and then searches for the highest-scoring dependency graph (Eisner, 1996; McDonald et al., 2005; Nakagawa, 2007), whereas the transition-based approach scores transitions between parser states based on the parse history and then greedily searches for the highest-scoring transition sequence that derives a complete dependency graph (Yamada and Matsumoto, 2003; Nivre et al., 2004; Attardi, 2006). Transition-based parsers are heavily dependent on machine learning for inducing a model for predicting the transition sequence used by the parser to construct the dependency graph. We will investigate two machine learning methods for performing this task: memory-based learning (MBL) and support vector machines (SVM).

During the last two decades, the research community has built several syntactically annotated corpora, also known as *treebanks*, with large collections of syntactic examples for many languages. These treebanks are an essential component when constructing data-driven parsers and one of the potential advantages of the increasing availability of treebanks is that parsers can easily be ported to new languages. A problem is that many data-driven parsers are overfitted to a particular language, usually English. For example, Corazza et al. (2004) report increased error rates of 15–18% when using two statistical parsers developed for English to parse Italian. One of the studies in this thesis is concerned with the question of how we can adapt a dependency-based parser for several languages by starting from a baseline model and increasing accuracy by optimizing the parameters of

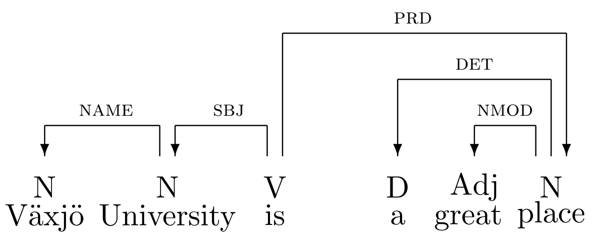


Figure 1.2: An example of a dependency representation for the sentence *Växjö University is a great place*, where each word is labeled with a part of speech and each dependency relation is labeled with a dependency category.

the parser.

In data-driven phrase structure parsing, the mainstream approach has been based on nondeterministic parsing techniques in combination with generative probabilistic models that provide an n-best ranking of the set of candidate analyses derived by the parser (Collins, 1997, 1999; Charniak, 2000). Discriminative models can be used to enhance these parsers by reranking the analyses output by the parser (Johnson et al., 1999; Collins and Duffy, 2005; Charniak and Johnson, 2005). In this thesis, we present a method for parsing phrase structure with a transition-based dependency parser that recovers both continuous and discontinuous phrases with both phrase labels and grammatical functions.

1.1 Research Questions

One of the main goals of the research presented in this thesis is to design and implement a robust and flexible system that can parse unrestricted natural language text independently of language. Given a treebank in a specific language, the system should induce a parser model that can parse unseen text in that language and output dependency graphs with reasonable accuracy. My licentiate thesis (Hall, 2006) presents a software architecture for transition-based dependency parsing that can handle different parsing algorithms, feature models and learning methods, for both learning and parsing. This architecture was first implemented in MaltParser 0.1–0.4, and has since been reimplemented in Java and further improved in MaltParser version 1.0–1.1.[1] The design and implementation of MaltParser and its optimization for different languages have been a large part of my workload. This is not directly reflected in the selected papers and this thesis, which focus on different aspects of transition-based dependency parsing, but all experiments have been performed using MaltParser.

[1] MaltParser 1.1 is distributed with an open-source license and can be downloaded from the following page: http://www.maltparser.org/.

The research questions of this doctoral thesis can be divided into two groups. The first group of questions, treated mainly in Papers I–III, concern how machine learning can be used to guide a transition-based dependency parser and how learning makes it possible to optimize such a parser for different languages using treebank data (section 1.1.1). The second group of questions, studied in Papers IV–V, concern how a transition-based dependency parser can be extended to parse continuous and discontinuous phrase structure (section 1.1.2).

1.1.1 Machine Learning

A transition-based dependency parser needs to predict the next parser action at nondeterministic choice points. The mechanism for doing this could be based on heuristics, but the most obvious and flexible solution is to use machine learning. The first research question in this group is how we can use machine learning to guide a transition-based dependency parser. The solution is discussed in Paper I and the theoretical framework of inductive dependency parsing proposed by Nivre (2006) explains the solution in detail. Furthermore, the implementation of guided transition-based dependency parsing in MaltParser 0.4 is described in Hall (2006).

The next step is to find well-suited machine learning methods for the task of guiding a transition-based dependency parser. Paper II, as well as Hall (2006), investigates this question with a systematic comparison of memory-based learning (MBL) and support vector machines (SVM). These studies also explore how we can improve learning and parsing efficiency without sacrificing accuracy. In particular, it presents a method for dividing the training instances into smaller sets and training separate classifiers, based on a method used by Yamada and Matsumoto (2003). Moreover, the division of the training instances is further improved in Paper IV and Paper V by introducing different prediction strategies, which can improve learning and parsing efficiency and in some cases also increase the accuracy of the parser.

Another important research question is how we can tune a transition-based dependency parser for different languages, which involves strategies for using treebank data in a way that do not overfit the induced model to the development data and for tuning all parameters. We have gathered a lot of knowledge optimizing MaltParser for many languages over the years and to some extent this question is reflected in all five papers, but Paper III summarizes our parser optimization strategies. Especially our participation in the CoNLL shared tasks in both 2006 and 2007 (Buchholz and Marsi, 2006; Nivre et al., 2007) has been very fruitful for gaining knowledge about parser optimization. In both cases, MaltParser was one of the top-performing systems (Nivre et al., 2006; Hall and Nilsson, 2006; Hall et al., 2007). We have also performed a systematic investigation using a memory-based learner for a large collection of languages (Nivre and Hall, 2005; Nivre et al., 2007).

The research questions connected to machine learning can be summarized as follows:

Q1: How can machine learning be used to guide a transition-based dependency parser?

Q2: Which learning methods are well suited for the task of transition-based dependency parsing?

Q3: How can learning and parsing efficiency be improved without sacrificing accuracy?

Q4: How can a transition-based dependency parser be tuned for different languages?

1.1.2 Phrase Structure Parsing

Although dependency-based representations have gained more interest in recent years, the dominant kind of syntactic representation is still based on phrase structure. Therefore, it would be useful if we could find a strategy for transforming the dependency-based output to phrase structure with high accuracy, preferably with a data-driven method that is not dependent on explicit rules. Another problem that we want to address is that parsers trained on treebank data often ignore important aspects of the syntactic representation of treebank. For example, when parsers are trained on the Penn Treebank of English (Marcus et al., 1993), it is common to ignore function labels and empty categories (Collins, 1999; Charniak, 2000).[2] Another example is parsers trained on treebanks based on the Negra annotation scheme for German, which encodes both local and non-local dependencies and sometimes results in discontinuous phrases. Data-driven parsing with the Negra annotation scheme often involves a simplification of the syntactic representation, and it is common to restrict the task to deriving only the continuous phrase structure (Dubey, 2005).[3]

Paper IV presents a technique for turning a dependency parser into a phrase structure parser that recovers continuous phrases with both phrase labels and grammatical functions. Paper V investigates how this technique can be extended to parse also discontinuous phrases. The research questions connected to phrase structure parsing can be summarized as follows:

Q5: How can we transform a phrase structure tree into a dependency graph and back?

Q6: How can we turn a transition-based dependency parser into a phrase structure parser?

Q7: How do we deal with discontinuous phrases?

[2] Notable exceptions are Musillo and Merlo (2005), where the parser output is enriched with function labels, and Gabbard et al. (2007), who recover both function labels and empty categories.

[3] Notable exceptions are Plaehn (2005), who recovers both continuous and discontinuous phrases with their phrase categories, and Kübler et al. (2006), who enrich the edges with grammatical functions.

1.2 Division of Labor

This thesis is based on five papers that are joint work with other authors. The work has been divided as follows:

In Paper I, the work was divided equally among the three authors. My contribution mainly concerned implementation and experimentation.

Paper III is based on the CoNLL 2007 shared task paper with the same name (Hall et al., 2007). My contribution was approximately 50% and concerned the implementation and optimization of the Single Malt parser, as well as the writing of the paper.

Papers II, IV and V are my own work to 90%.

1.3 Outline of the Thesis

In section 1.1 I have outlined the research questions of the thesis, and in section 1.2 I have described my contributions to the papers on which the thesis is based. The remainder of the thesis is structured as follows.

Chapter 2, *Natural Language Parsing*, briefly reviews related work on natural language parsing. We define the problem of parsing unrestricted natural language text, discuss different approaches to dependency parsing, and distinguish between graph-based dependency parsing and transition-based dependency parsing.

Chapter 3, *Transition-Based Dependency Parsing*, defines the formal framework of the thesis. After a brief introduction to transition-based dependency parsing we define the syntactic representations that will be used throughout the thesis. The chapter continues with a description of two deterministic parsing algorithms and an explanation of how history-based models can be used to guide the algorithms at every nondeterministic choice point.

Chapter 4, *Machine Learning for Transition-Based Dependency Parsing*, is devoted to the first group of research questions of using machine learning to guide a transition-based dependency parser, based on Papers I–III. The chapter starts by describing support vector machines and memory-based learning, and continues with an account of the empirical studies in Papers I and II using memory-based learning and support vector machines for guided parsing. Next we investigate, based on Paper III, how we can optimize the parser to obtain a satisfying accuracy for a large variety of languages.

Chapter 5, *Dependency-Based Phrase Structure Parsing*, demonstrates how a transition-based dependency parser can be used for parsing phrase structure representations, based on Papers IV and V. First we define the transformation of a phrase structure graph into a dependency graph, where the inverse transformation is encoded in complex dependency edge labels. The chapter continues by describing how this dependency graph can be transformed back to a phrase structure graph without any loss of information. Finally, we discuss the empirical results obtained in Papers IV and V.

Chapter 6, *MaltParser*, contains a short presentation of the MaltParser system, which is the reference implementation of the framework presented in this thesis.

Chapter 7, *Conclusion*, summarizes the main contributions and results of the thesis. The chapter ends with a discussion of directions for future research.

Chapter 2

Natural Language Parsing

A *natural language* like English or Swedish is hard to define in exact terms, which is much easier for a formal language such as a programming language. Moreover, a natural language has often evolved during thousands of years and continues to evolve, which makes it impossible to state an exact definition at a given time. It is also hard to draw boundaries between natural languages, and whether a particular language is counted as an independent language is usually dependent on historical events and sociopolitics, and not only on linguistic criteria. These properties make natural language processing a challenging task but also an interesting research topic especially with the increasing use of information technology. Many computer applications that involve natural language such as machine translation, question answering and information extraction are dependent on modeling natural language in some way. Moreover, these applications usually have to deal with unrestricted text, including grammatically correct text, ungrammatical text and foreign expressions. It is desirable that such an application produces some kind of analysis. Of course, if the input is "garbage", it is likely that the system will fail to create an interesting analysis, but the system should nevertheless do its best to produce some analysis.

Natural language parsing is the process of mapping an input string or a sentence to its syntactic representation. We assume that every sentence in a text has a single correct analysis and that speakers of the language will typically agree on what this preferred analysis is, but we do not necessarily assume that there is a formal grammar defining the relation between sentences and their preferred interpretations. Nivre (2006) uses the term *text parsing* to characterize this open-ended problem that can only be evaluated with respect to empirical samples of a *text language*. The term *text language* does not exclude spoken language, but emphasizes that it is the language that occurs in real texts.

However, it is important to note that, even though, the notion of text parsing does not presuppose the notion of a grammar, many systems for text parsing do in fact include a grammar-based component. In this thesis, we will focus on methods for text parsing that are not grammar-based, but in this background chapter we will take a somewhat broader perspective and start by reviewing some basic concepts of grammar-based parsing.

2.1 Grammars and Parsing Algorithms

The study of natural language grammar dates back at least to 400 BC, when Panini described Sanskrit grammar, but the formal computational study of grammar can be said to start in the 1950s with work on *context-free grammar* (CFG) (Chomsky, 1956) and the equivalent *Backus-Naur form* (BNF) (Backus, 1959). BNF is extensively used in computer science to define the syntax of programming languages and communication protocols, and any grammar in BNF can be viewed as a context-free grammar. CFG uses productions $A \rightarrow \alpha$ to define the grammar, where A is a nonterminal symbol that denotes different phrase types in the sentence, and where α is string of terminals (symbols of the language) and/or nonterminals. Nonterminals can be replaced recursively without regarding the context. CFG was the starting point for an extensive study of formal grammars. Nowadays there are several linguistically motivated formalisms, such as Lexical Functional Grammar (LFG) (Kaplan and Bresnan, 1983), Tree-Adjoining Grammar (TAG) (Joshi, 1985), Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994) and Combinatory Categorial Grammar (CCG) (Steedman, 2000), all of which go beyond context-free grammar.

To parse with a grammar requires a parsing algorithm. One group of algorithms for grammar parsing are known as *chart parsing* algorithms (Kay, 1980) and make use of dynamic programming to store partial results in a data structure called a *chart*. For example, Earley's algorithm (Earley, 1970) and the CKY algorithm (Kasami, 1965; Younger, 1967) use this approach. While chart parsing algorithms can typically handle arbitrary CFGs, there are also deterministic parsing algorithms that can only parse limited subsets of the class of CFGs and that have been frequently used in compiler construction for programming languages. Examples of deterministic parsing algorithms are LR parsing (Knuth, 1965) and shift-reduce parsing (Aho et al., 1986). A completely different approach is *eliminative parsing*. Instead of constructively deriving an analysis these algorithms apply grammar constraints to eliminate analyses that violate the constraints. This approach is exemplified by Constraint Grammar (CG) (Karlsson, 1990; Karlsson et al., 1995) and Constraint Dependency Grammar (CDG) (Maruyama, 1990; Harper and Helzermann, 1995; Menzel and Schröder, 1998).

2.2 Text Parsing

Let us now take a closer look at the problem of text parsing. We begin by defining a text as a sequence $T = (x_1, \ldots, x_n)$ of sentences, where each sentence $x_i = (w_1, \ldots, w_m)$ is a sequence of tokens and a token w_j is a sequence of characters, usually a word form or punctuation symbol. We assume that the text T contains sentences in a text language L, which in our case is a natural language, and that the task of a parser is to derive the single correct analysis y_i for every sentence $x_i \in T$. We can formulate this in terms of three requirements on a text parsing system P (cf. Nivre

(2006)):

1. **Robustness**: P assigns at least one analysis y_i to every sentence $x_i \in T$.
2. **Disambiguation**: P assigns at most one analysis y_i to every sentence $x_i \in T$.
3. **Accuracy**: P assigns the correct analysis y_i to every sentence $x_i \in T$.

Since the text language L is not a formal language, there is no formal method for showing that a parser satisfies the third requirement, which means that text parsing is at least partly an empirical approximation problem.

Many systems that have been developed for text parsing make use of a formal grammar G, defining a formal language $L(G)$ intended to approximate the text language L. One of the problems with grammar-based methods for text parsing has to do with their limited capacity to analyze all possible sentences that can occur in natural language text, since any sentence that is not in $L(G)$ cannot be analyzed, which is a problem with respect to the robustness requirement. Hence, when using grammar-based methods for text parsing, they need to be adapted to parse sentences that are not recognized by the grammar. There has been substantial work done in this area, for example, on relaxing the grammatical constraints of the grammar (Jensen and Heidorn, 1983; Mellish, 1989) and on partial parsing (Ejerhed and Church, 1983; Lang, 1988; Hindle, 1989; Koskenniemi, 1990; Abney, 1991; Karlsson, 1990).

Another potential problem with grammar-based methods is that the grammar normally does not assign a single analysis to a given sentence, which leads to problems with respect to disambiguation. In order to overcome this problem, most grammar-based systems today incorporate a statistical component for parse selection, i.e., for selecting the single optimal analysis from the set of analyses licensed by the grammar. This is true, for example, of most broad-coverage parsers based in grammatical frameworks such as LFG (Riezler et al., 2002), HPSG (Toutanova et al., 2002; Miyao et al., 2003) and CCG (Clark and Curran, 2004). This brings us naturally to data-driven approaches, which often dispenses with the grammar completely and relies solely on statistical inference.

2.3 Data-Driven Parsing

During the last decades, there has been a great interest in data-driven methods for various natural language processing tasks, including very prominently data-driven text parsing. The essence of data-driven (or statistical) parsing is that inductive inference is used to estimate the correct analysis for a given sentence, based on a representative sample of the text language (Nivre, 2006). The text sample may consist of raw text, but usually it is taken from a treebank where the sentences are annotated with the correct analysis by human experts. A realistic approach is then to use some kind of supervised learning method that makes use of treebank data. A problem with this approach is that it restricts us to languages that have at least one treebank.

Data-driven approaches to text parsing were first developed during the 1990s for phrase structure representations. The first attempts focused on extending context-free grammars with probabilities (PCFG), where each production is augmented with a probability, and also extending the parsing algorithms so that they can make use of probabilities. This was done by Ney (1991) for the CKY algorithm and by Stolcke (1995) for Earley's algorithm.

Nowadays, the standard approaches are based on nondeterministic parsing techniques, usually involving some kind of dynamic programming, in combination with generative probabilistic models that provide an n-best ranking of the set of candidate analyzes derived by the parser. The most well-known parsers based on these techniques are the parser of Collins (Collins, 1997, 1999) and the parser of Charniak (2000). Discriminative learning methods have been used to enhance these parsers by reranking the analyses output by the parser (Johnson et al., 1999; Collins and Duffy, 2005; Charniak and Johnson, 2005).[1] However, there is often a discrepancy between the original treebank annotation and the representations used by the parser. For example, in the Penn Treebank (Marcus et al., 1993) the phrase structure annotation includes not only syntactic categories like NP, VP, etc., but also to a certain extent functional categories such as subject, predicative, etc. In addition, empty categories and co-indexation are used to capture non-local dependencies (Bies et al., 1995). Nevertheless, it is common to restrict the parsing problem to plain phrase structure with no empty categories when creating English parsers based on the Penn Treebank (Collins, 1999; Charniak, 2000). Notable exceptions, among others, are Gabbard et al. (2007), who recover both function labels and empty categories and Musillo and Merlo (2005), who enrich the parser output with function labels. Similarly for German, the Negra annotation scheme uses a combination of dependency and phrase structure representations, and encodes both local and non-local dependencies, which sometimes results in discontinuous phrases. But data-driven parsing of German often involves a simplification of the syntactic representation, and it is common to restrict the task to deriving only the continuous phrase structure and only the phrase labels (Dubey, 2005). Kübler et al. (2006) recover grammatical functions, but not discontinuities. By contrast, Plaehn (2005) parses discontinuous phrase structure using a probabilistic extension of discontinuous phrase structure grammar (DPSG) (Bunt, 1991, 1996), but evaluation is restricted to phrase labels alone.

In recent years, data-driven dependency parsing has become a popular method for parsing natural language text, and the shared tasks on multilingual dependency parsing at CoNLL 2006 (Buchholz and Marsi, 2006) and

[1] It is worth noting that these discriminative models are essentially the same as those used for parse selection by the grammar-based parsers mentioned at the end of section 2.2.

CoNLL 2007 (Nivre et al., 2007) have contributed greatly to the increase in interest. McDonald and Nivre (2007) define two dominating schools in data-driven dependency parsing: graph-based dependency parsing and transition-based dependency parsing.

In graph-based parsing, all possible arcs in the dependency graph for a given sentence are scored with a weight and parsing is performed by searching for the highest-scoring dependency graph, which is the same as finding the highest scoring directed spanning tree in a complete graph. Eisner (1996), who was one of the first to introduce data-driven methods for dependency parsing, used a graph-based probabilistic parser to assign both part-of-speech tags and an unlabeled (bare-bone) dependency structure simultaneously. McDonald et al. (2005) generalized the approach to non-projective dependency structures and showed that parsing could be performed in quadratic time. McDonald and Satta (2007) investigated algorithms for graph-based non-projective parsing with varying effects on complexity. Graph-based dependency parsing has been shown to give state of the art performance without any language specific enhancements for a wide range of languages (McDonald et al., 2006; Nakagawa, 2007).

Transition-based dependency parsing instead tries to perform parsing deterministically, using a classifier trained on gold standard derivations from a treebank to guide the parser (the technique investigated in this thesis). It was first used for unlabeled dependency parsing by Kudo and Matsumoto (2000, 2002) (for Japanese) and Yamada and Matsumoto (2003) (for English) using support vector machines. The parsing algorithm uses a variation of shift-reduce parsing with three possible parse actions: SHIFT, RIGHT and LEFT. The two latter parse actions add a dependency relation between two target nodes, which are two neighboring tokens. The parse action SHIFT moves the focus to the right in the input string, which results in two new target nodes. The worst-case time complexity of this approach is $O(n^2)$, but the worst-case rarely occurs in practice. Cheng et al. (2005) have used this methodology to parse Chinese with state of the art performance.

Nivre (2003) proposed a similar parsing algorithm with another transition system that parses a sentence in linear time. The algorithm was extended to handle labeled dependency structures by Nivre et al. (2004) for parsing Swedish. Transition-based dependency parsing has also been shown to give state of the art performance for a wide range of languages (Nivre et al., 2006; Hall et al., 2008). The transition-based parsing have also been used for phrase structure parsing. For example, Kalt (2004) used decision trees to determine the next parse action and Sagae and Lavie (2005) experimented with both support vector machines and memory-based learning to derive the transition sequence.

One possible disadvantage of the greedy and strictly deterministic approach is that there is no model that takes the global dependency graph into account. Duan et al. (2007) investigated two probabilistic parsing action models to compute the probability of the entire dependency graph and selecting the graph with the highest probability. Johansson and Nugues

(2007b) scored each parse action and uses beam search to find the best sequence of actions. Titov and Henderson (2007) combined beam search with a generative model by adding transition for generating the input words. Chapter 3 describes transition-based dependency parsing in more detail.

McDonald and Nivre (2007) compared graph-based and transition-based dependency parsing and found that the two approaches often make different errors when deriving the dependency representations, which indicates that combining the approaches could improve the accuracy. Nivre and McDonald (2008) continued this study by exploring integrated models, where predictions from one model are used as training material for the other, in a form of classifier stacking. The results showed a significant improvement in accuracy for the integrated models on data sets from the CoNLL shared task 2006.

Another approach is to increase accuracy by combining existing parsers. Zeman and Žabokrtský (2005) improved the accuracy for parsing Czech by using a language independent ensemble method, which for each token greedily chooses a head token based on the head tokens of all single parsers. Sagae and Lavie (2006) proposed a technique that combines dependency parsers by finding the maximum directed spanning tree, where the arc weights are based on the outputs of several parser systems. Hall et al. (2007) achieved the highest accuracy in the CoNLL Shared Task 2007 by building an ensemble system that combines six transition-based dependency parsers.

2.4 Summary

In this chapter we have provided the background for the thesis by introducing the problem of parsing natural language text and discussing different approaches to this problem, including both grammar-based and data-driven approaches. From now on, we will restrict our attention to data-driven methods for dependency parsing, in particular transition-based dependency parsing.

Chapter 3

Transition-Based Dependency Parsing

In chapter 2, we introduced different techniques for parsing natural language text and from now on we will focus on transition-based parsing. We will mostly concentrate on transition-based parsing for syntactic representations based on the notion of *dependency*, but in chapter 5 we will explore how this parsing technique can also be used for parsing representations based on the notion of *constituency* or *phrase structure*. After a short introduction to transition-based parsing, we will go on in section 3.1 to define the syntactic representations used in the rest of the thesis. In section 3.2 we define two deterministic parsing algorithms and in section 3.3 we discuss the use of history-based feature models.

Transition-based parsing, as used in this thesis, is based on the theoretical framework of *inductive dependency parsing* presented by Nivre (2006) and has the following three essential components:

1. Deterministic parsing algorithms for constructing labeled dependency graphs (Kudo and Matsumoto, 2002; Yamada and Matsumoto, 2003; Nivre, 2003). Algorithms are defined in terms of a *transition system*, consisting of a set of *configurations* and a set of *transitions* between configurations. Deterministic parsing is implemented as greedy best-first search through the transition system. Section 3.2 briefly describes the Nivre parsing algorithm (Nivre, 2003) and a variant of the Covington parsing algorithm (Covington, 2001).

2. History-based models for predicting the next transition (Black et al., 1992; Magerman, 1995; Ratnaparkhi, 1997; Collins, 1999). The transition history, as encoded in the current parser configuration, is represented by a feature vector, which can be used as input to a classifier for predicting the next transition in order to guide the deterministic parser. Section 3.3 explains how history-based models are used in transition-based parsing.

3. Discriminative learning to map histories to transitions (Kudo and Matsumoto, 2002; Yamada and Matsumoto, 2003; Nivre et al., 2004). Given a set of transition sequences derived from a treebank, discriminative machine learning can be used to train a classifier. The classifier is used during parsing to discriminate between different possible transitions given a feature vector representation of the current configura-

tion. Chapter 4 discusses in more detail different discriminative learning methods for guiding a transition-based parser.

To continue the description of transition-based parsing we need to define a formal framework for representing syntactic structure and this is done in the next section.

3.1 Syntactic Representations

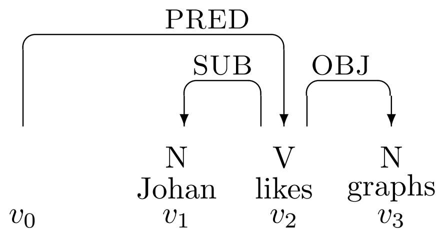
Throughout the rest of the thesis we need a formal framework for representing the syntactic structure of a sentence, which can be represented either as a *dependency graph* or as a *phrase structure graph*. These two graphs share several properties, and therefore we begin by defining a *syntax graph* that abstracts over the two more specific graphs.

Definition 1 A *syntax graph* for a sentence $x = w_1, \ldots, w_n$ is a triple $G = (V, E, F)$, where

- $V = V_R \cup V_T \cup V_{NT}$ is a set of nodes, partitioned into three disjoint subsets:
  - $V_R = \{v_0\}$, where v_0 is the designated root node,
  - $V_T = \{v_1, \ldots, v_n\}$, the set of terminal (or token) nodes, one node v_i for each token $w_i \in x$,
  - $V_{NT} = \{v_{n+1}, \ldots, v_{n+m}\}$ $(m \geq 0)$, the set of nonterminal nodes,
- $E \subseteq V \times (V - V_R)$ is a set of edges,
- $F = \{f_1, \ldots, f_k\}$ is a set of functions $f_i : D_i \rightarrow L_i$, where $D_i \in \{V_T, V_{NT}, E\}$ and L_i is a set of labels.

A *syntax graph* G is well-formed if it is a directed tree with the single root v_0.

A syntax graph G consists of a set of nodes V, which is divided into three disjoint subsets V_R, V_T and V_{NT}. The node set V_R contains only one node v_0, which is the special root node of the graph. The use of a designated root node makes it easier to design algorithms so that the well-formedness criteria of a directed tree are satisfied. The terminal nodes in V_T have a direct connection to the tokens in the sentence x (i.e., the token w_i corresponds to the terminal node v_i). Finally, the possibly empty set of nonterminal nodes V_{NT} contains m nonterminal nodes. An edge $(v_i, v_j) \in E$ connects two nodes v_i and v_j in the graph, where v_i is said to immediately dominate v_j. The last component of G is the set of labeling functions F, where each function f_i is a mapping from a domain $D_i \in \{V_T, V_{NT}, E\}$ to a finite label set L_i. In other words, a function is used to label terminal nodes, nonterminal nodes, or edges, but the specific functions that are used may vary depending on the linguistic framework or annotation scheme used. We assume that a well-formed syntax graph G is a single-rooted directed tree,



$$\begin{aligned}
G \quad & = (V_R \cup V_T \cup V_{NT}, E, F) \\
V_R \quad & = \{v_0\} \\
V_T \quad & = \{v_1, v_2, v_3\} \\
V_{NT} \quad & = \emptyset \\
E \quad & = \{(v_0, v_2), (v_2, v_1), (v_2, v_3)\} \\
F \quad & = \{f_1, f_2, f_3\} \\
f_1 : V_T \rightarrow L_W \quad & = \{(v_1, \mathit{Johan}), (v_2, \mathit{likes}), (v_3, \mathit{graphs})\} \\
f_2 : V_T \rightarrow L_P \quad & = \{(v_1, N), (v_2, V), (v_3, N)\} \\
f_3 : E \rightarrow L_R \quad & = \{((v_0, v_2), \mathrm{PRED}), ((v_2, v_1), \mathrm{SUB}), \\
& \quad ((v_2, v_3), \mathrm{OBJ})\}
\end{aligned}$$

Figure 3.1: An example dependency graph for the sentence *Johan likes graphs*, where L_W is the set of words, L_P is the set of part-of-speech tags and L_R is the set of dependency relations.

which entails that the graph G has a unique root v_0, that G is weakly connected, that each node v_i ($i \neq 0$) has exactly one incoming edge and that G is acyclic.

Given the definition of a syntax graph we define a *dependency graph* as follows:

Definition 2 A *dependency graph* for a sentence $x = w_1, \ldots, w_n$ is a syntax graph $G = (V, E, F)$, where

- $V_{NT} = \emptyset$

A dependency graph is a syntax graph with the constraint that it does not contain any nonterminal nodes, because a dependency structure is built from binary relations between tokens (or words). It follows that an edge (v_i, v_j) connects a node v_i, which is either a terminal node or the artificial root node, with a terminal node v_j. We say that v_i is the head and v_j is the dependent. Figure 3.1 exemplifies the dependency representation.

A common constraint on dependency graphs is the notion *projectivity* and we define this notion in the following way:

Definition 3 A dependency graph G is *projective* iff, for every node $v_k \in V_T$ and every edge $(v_i, v_j) \in E$ such that w_k occurs between w_i and w_j in the linear order of the sentence x (i.e., $i < k < j$ or $j < k < i$), there is a directed path from v_i to v_k (where the directed path is the transitive closure of the edge relation E).

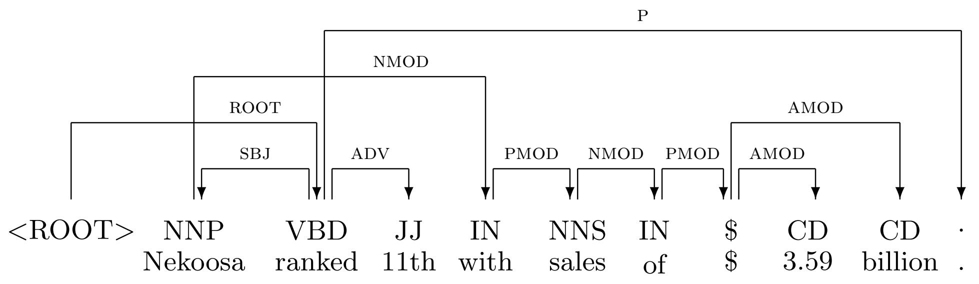


Figure 3.2: Dependency graph for an English sentence from the Wall Street Journal section of the Penn Treebank (Marcus et al., 1993).

Projectivity constraint is controversial in linguistic theory and most dependency-based frameworks allow non-projective graphs, because non-projective representations are able to capture non-local dependencies. There exist several treebanks that contain non-projective structures such as the Prague Dependency Treebank of Czech (Böhmová et al., 2003) and the Danish Dependency Treebank (Kromann, 2003). The graph shown in figure 3.2 is non-projective because the edge from *Nehoosa* to *with* span two terminal nodes (*ranked*, *11th*) that are not dominated by *Nehoosa*.

Next we define a *phrase structure graph* as follows:

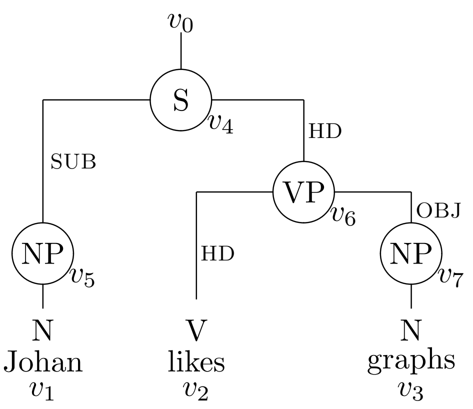
Definition 4 A *phrase structure graph* for a sentence $x = w_1, \ldots, w_n$ is a syntax graph $G = (V, E, F)$, where

- $E \subseteq (V_R \cup V_{NT}) \times (V_T \cup V_{NT})$

A phrase structure graph is a syntax graph with a restricted edge set, where only the artificial root v_0 or a nonterminal node $v_i \in V_{NT}$ can immediately dominate another nonterminal node or a terminal node v_j. Figure 3.3 exemplifies how a phrase structure graph is represented. A notion related to projectivity for dependency graphs is *continuous phrase structure*, which means that every nonterminal node $v_k \in V_{NT}$ has a leftmost descendant $v_i \in V_T$ and a rightmost descendant $v_j \in V_T$ such that v_k dominates all terminal nodes between v_i and v_j according to the linear order of x. The phrase structure graph is well-formed if it is a directed tree rooted in v_0 according to the definition of the syntax graph, which entails that v_0 dominates all terminal nodes. Phrase structure graphs will not be discussed further in this chapter, but we will return to them in chapter 5.

3.2 Deterministic Parsing

Many approaches to data-driven text parsing are based on nondeterministic parsing techniques combined with disambiguation performed on a packed forest or n-best list of complete derivations, but the disambiguation can also be performed deterministically using a greedy parsing algorithm that approximates a globally optimal solution by making a sequence of locally



$$\begin{array}{ll}
G & = (V_R, V_T, V_{NT}, E, F) \\
V_R & = \{v_0\} \\
V_T & = \{v_1, v_2, v_3\} \\
V_{NT} & = \{v_4, v_5, v_6, v_7\} \\
E & = \{(v_0, v_4), (v_4, v_5), (v_5, v_1), (v_4, v_6), (v_6, v_2), \\
 & \quad (v_6, v_7), (v_7, v_3)\} \\
F & = \{f_1, f_2, f_3, f_4\} \\
f_1 : V_T \rightarrow L_W & = \{(v_1, \mathit{Johan}), (v_2, \mathit{likes}), (v_3, \mathit{graphs})\} \\
f_2 : V_T \rightarrow L_P & = \{(v_1, N), (v_2, V), (v_3, N)\} \\
f_3 : V_{NT} \rightarrow L_{NT} & = \{(v_4, S), (v_5, NP), (v_6, VP), (v_7, NP)\} \\
f_4 : E \rightarrow L_{GF} & = \{((v_0, v_4), -), ((v_4, v_5), \text{SUB}), ((v_5, v_1), -), \\
 & \quad ((v_4, v_6), \text{HD}), ((v_6, v_2), \text{HD}), ((v_6, v_7), \text{OBJ}), \\
 & \quad ((v_7, v_3), -)\}
\end{array}$$

Figure 3.3: An example phrase structure graph for the sentence *Johan likes graphs*, where L_W is the set of words, L_P is the set of part-of-speech tags, L_{NT} is the set of phrase labels and L_{GF} is the set of grammatical functions (with $-$ as the empty label).

optimal choices. Deterministic parsing algorithms use a transition system, which consists of a set of parser configurations and transitions between configurations. Formally, we define a transition system as follows:

Definition 5 Given a set of dependency labels $L = \{l_1, \ldots, l_m\}$, a *transition system* for dependency parsing is a quadruple $S = (C, T, c^i, C^t)$, where

1. C is a set of parser configurations, each of which contains:
   (a) a sequence of $\lambda_1, \ldots, \lambda_d$ data structures,
   (b) a list τ of remaining terminal nodes,
   (c) a set E of directed edges (v_i, v_j),
   (d) a function f from E to L that assigns labels to edges,
2. T is a set of transitions, each of which is a (partial) function $t : C \rightarrow C$,

3. c^i is a function that assigns a unique initial configuration to every sentence x,
4. $C^t \subseteq C$ is a set of terminal configurations.

A parser configuration is always required to contain a list of remaining terminal nodes τ, a set E of edges and a labeling function f. In addition there will be one or more data structures $\lambda_1, \ldots, \lambda_d$, depending on the specific transition system. We use $[v_j|\tau]$ to denote a list of nodes with head v_j and tail τ and an empty list is represented by $[\ ]$, and we use τ_c to denote the current list of nodes in a configuration c. A *transition* is partial function from configurations to configurations. The set C contains all possible configurations in a given transition system, but there must also be a subset C^t of *terminal* configurations and a unique *initial* configuration c^i_x for every sentence x. For all transition systems discussed in this thesis, the subset of *terminal* configurations C^t contains any parser configuration where $\tau = [\ ]$.

Definition 6 Let $S = (C, T, c^i, C^t)$ be a transition system. A *transition sequence* for a sentence $x = w_1, \ldots, w_n$ in S is a sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ of parser configurations, such that

1. $c_0 = c^i_x$
2. $c_m \in C^t$
3. for every i $(1 \leq i \leq m), c_i = t(c_{i-1})$ for some $t \in T$.

The dependency graph assigned to x by $C_{0,m}$ is $G_{c_m} = (V, E_{c_m}, \{f_{c_m}, \ldots\})$, where E_{c_m} is the set of edges in c_m and f_{c_m} is the labeling function in c_m.

Given a terminal configuration $c_m \in C^t$ for a sentence $x = w_1, \ldots, w_n$, the dependency graph assigned to x by $c_{0,m}$ is defined to be $G_{c_m} = (V, E_{c_m}, \{f_{c_m}, \ldots\})$, where $V = V_R \cup V_T$ is the node set for x, E_{c_m} is the set of edges in c_m and f_{c_m} is the labeling function in c_m. The notation $\{f_{c_m}, \ldots\}$ is used to indicate that there may be additional labeling functions for word forms, parts of speech, etc., which are given as part of the input.

We will define several transition systems, which are all nondeterministic. Hence, there will be more than one transition applicable to a given configuration. An *oracle* for a transition system $S = (C, T, c^i, C^t)$ is a function $o : C \to T$ from configurations to transitions that can be used to overcome this nondeterminism. For each nondeterministic choice point the parsing algorithm will ask the oracle to predict the next transition. In this section we will consider the oracle as a black box, which always knows the correct transition. Later on we will see that we can approximate this oracle by history-based models and classifiers.

Given a transition system with the initial configuration c^i_x for a sentence $x = w_1, \ldots, w_n$, we define a generic deterministic parsing algorithm Parse as follows:

```
PARSE($x = w_1, \ldots, w_n$)
1  $c \leftarrow c_x^i$
2  while $c \notin C^t$
3      $t \leftarrow o(c)$
4      $c \leftarrow t(c)$
5  return $G_c$
```

The algorithm starts by initializing the configuration c to the initial configuration c_x^i specific to the transition system. As long as the parser remains in a non-terminal configuration, i.e., τ is not empty, the parser applies the oracle transition $t = o(c)$ to the current configuration c. Finally, the dependency graph G_c given by E_c and f_c is returned. In this thesis we use two different parsing algorithms: Nivre and Covington. Both algorithms can be said to instantiate the generic deterministic parsing algorithm PARSE, but they differ in their parser configurations and transitions. The Nivre algorithm has one additional data structure λ_1, whereas Covington makes use of two additional data structures λ_1 and λ_2. The Nivre algorithm comes with two transition systems: arc-eager and arc-standard. The Covington algorithm can be defined in several ways, but in this thesis we only use the non-projective version of the transition system. These three transition systems are defined in the following subsections.

3.2.1 Nivre Arc-Eager

The Nivre arc-eager transition system was first proposed for unlabeled dependency parsing by Nivre (2003) and was extended to labeled dependency parsing by Nivre et al. (2004). This transition system guarantees that the parser terminates after at most $2n$ transitions, given a sentence of length n. The configuration is extended with a stack λ_1 of partially processed terminal nodes, where v_i is the top node of $[\lambda_1|v_i]$. The transition system uses four transitions, two of which are parameterized by a dependency label $l \in L$. The transition system is initialized and updates the parser configuration as follows (Nivre, 2006):

Definition 7 For every dependency label $l \in L$, the following transitions for a sentence $x = w_1, \ldots, w_n$ are possible:

$$\begin{array}{lll}
\textsc{Shift}: & (\lambda_1, [v_i|\tau], E, f) & \Rightarrow ([\lambda_1|v_i], \tau, E, f) \\
\textsc{Reduce}: & ([\lambda_1|v_i], \tau, E, f) & \Rightarrow (\lambda_1, \tau, E, f) \\
\textsc{Right-Arc}(l): & ([\lambda_1|v_i], [v_j|\tau], E, f) & \Rightarrow ([\lambda_1|v_i|v_j], \tau, E \cup \{(v_i, v_j)\}, \\
 & & \quad f \cup \{((v_i, v_j), l)\}) \\
\textsc{Left-Arc}(l): & ([\lambda_1|v_i], [v_j|\tau], E, f) & \Rightarrow (\lambda_1, [v_j|\tau], E \cup \{(v_j, v_i)\}, \\
 & & \quad f \cup \{((v_j, v_i), l)\})
\end{array}$$

Initialization: $c_x^i = ([v_0], [v_1, \ldots, v_n], \emptyset, \emptyset)$

The transition SHIFT shifts (pushes) the next input token v_i onto the stack λ_1. This is the correct action when the head of the next token is positioned to the right of the next token. The transition REDUCE reduces (pops) the token v_i on top of the stack λ_1. It is important to ensure that the parser does not pop the top token if it has not been assigned a head, since it will be left unattached.

The Right-Arc transition adds an edge from the token v_i on top of the stack λ_1 to the next input token v_j and involves pushing v_j onto the stack. Finally, the transition Left-Arc adds an edge from the next input token v_j to the token v_i on top of the stack λ_1 and involves popping v_i from the stack. This transition is only allowed when the top token v_i is not the root node. We make use of the assumption of projectivity because we know that the top token v_i cannot have any more left and right dependents and therefore it can be popped.

The configuration is initialized with the artificial root v_0 on the stack λ_1, which means that edges originating from the root can be added explicitly during parsing. This initialization allows more than one label for dependencies from the artificial root. It is also possible to have an empty initialization, the stack [ ], in which case edges from the root are only added implicitly with a default label (to any token that remains a root after parsing is completed). Empty stack initialization reduces the amount of nondeterminism, but restricts labeling to only one default label.

3.2.2 Nivre Arc-Standard

The Nivre arc-standard transition system uses a transition system similar to the arc-eager version, but has only three transitions Shift, Left-Arc and Right-Arc (no Reduce). The first two transitions, Shift and Left-Arc, are applied in exactly the same way as for the arc-eager version. This transition system also uses a stack λ_1, where the stack initialization can vary in the same way as for the arc-eager version. The Nivre arc-standard transition system is defined as follows:

Definition 8 For every dependency label $l \in L$, the following transitions for a sentence $x = w_1, \ldots, w_n$ are possible:

$$\begin{array}{lll}
\textsc{Shift}: & (\lambda_1, [v_i|\tau], E, f) & \Rightarrow ([\lambda_1|v_i], \tau, E, f) \\
\textsc{Right-Arc}(l): & ([\lambda_1|v_i], [v_j|\tau], E, f) & \Rightarrow (\lambda_1, [v_i|\tau], E \cup \{(v_i, v_j)\}, \\
 & & \quad f \cup \{((v_i, v_j), l)\}) \\
\textsc{Left-Arc}(l): & ([\lambda_1|v_i], [v_j|\tau], E, f) & \Rightarrow (\lambda_1, [v_j|\tau], E \cup \{(v_j, v_i)\}, \\
 & & \quad f \cup \{((v_j, v_i), l)\})
\end{array}$$

Initialization: $c_x^i = ([v_0], [v_1, \ldots, v_n], \emptyset, \emptyset)$

Instead of pushing the next token v_j onto the stack λ_1, Right-Arc moves the topmost token v_i on the stack back to the list of remaining input tokens τ, where it replaces the token v_j as the next token. The Nivre arc-standard transition system uses a strict bottom-up processing as in traditional shift-reduce parsing.

3.2.3 Covington Non-Projective

Covington (2001) proposes several incremental parsing strategies for dependency representations and one of these strategies can recover non-projective dependency graphs. The Covington non-projective transition system, presented in definition 9, is based on Nivre (2007) that is a variant of the non-projective parsing strategy. The Covington algorithm makes use of two additional data structures in the parser configuration. We have two stacks λ_1 and λ_2 of partially processed terminal nodes. We will have the same convention as before for the stack λ_1, where we denote the head v_i to the right and the tail λ_1 consisting of nodes in descending order (i.e. $[\lambda_1|v_i]$). For λ_2 we denote the head v_j to the left and the tail λ_2 to the right containing the nodes in ascending order. We also allow list operations on the two stacks, which means that we can move nodes from λ_2 to λ_1 all at once. We define the Covington non-projective transition system in the following way:

Definition 9 For every dependency label $l \in L$, the following transitions for a sentence $x = w_1, \ldots, w_n$ are possible:

$$\begin{array}{lll}
\textsc{Shift}: & ([\,], \lambda_2, [v_i|\tau], E, f) & \Rightarrow ([\lambda_2|v_i], [\,], \tau, E, f) \\
\textsc{No-Arc}: & ([\lambda_1|v_i], \lambda_2, [v_j|\tau], E, f) & \Rightarrow (\lambda_1, [v_i|\lambda_2], [v_j|\tau], E, f) \\
\textsc{Right-Arc}(l): & ([\lambda_1|v_i], \lambda_2, [v_j|\tau], E, f) & \Rightarrow (\lambda_1, [v_i|\lambda_2], [v_j|\tau], \\
 & & \quad E \cup \{(v_i, v_j)\}, f \cup \{((v_i, v_j), l)\}) \\
\textsc{Left-Arc}(l): & ([\lambda_1|v_i], \lambda_2, [v_j|\tau], E, f) & \Rightarrow (\lambda_1, [v_i|\lambda_2], [v_j|\tau], \\
 & & \quad E \cup \{(v_j, v_i)\}, f \cup \{((v_j, v_i), l)\})
\end{array}$$

Initialization: $c_x^i = ([v_0], [\,], [v_1, \ldots, v_n], \emptyset, \emptyset)$

The initial configuration has the root node v_0 on the stack λ_1 and an empty stack λ_2. The transition system contains four possible transitions: SHIFT, NO-ARC, RIGHT-ARC(l) and LEFT-ARC(l). The SHIFT transition is deterministic because it is the only possible transition when the stack λ_1 is empty and shifts the next token v_i together with all nodes in λ_2 to λ_1. It is possible to modify the transition system to allow a non-deterministic shift transition $(\lambda_1, \lambda_2, [v_i|\tau], E, f) \Rightarrow ([\lambda_1{+}\lambda_2|v_i], [\,], \tau, E, f)$, where all nodes in λ_2 are appended to λ_1 together with next input token v_i. The transition NO-ARC moves the node v_i to λ_2, where v_i becomes the top of the stack in λ_2. The RIGHT-ARC transition adds an edge with label l from the next input token v_j to the token v_i on top of the stack λ_1. Transition LEFT-ARC adds an edge with label l from the next input token v_j to the top token v_i. In addition, both the RIGHT-ARC transition and the LEFT-ARC transition moves the top token v_i of the stack λ_1 to λ_2. The transition is permissible if v_i is not the artificial root node v_0 and does not already have a head, and there is no path from v_i to v_j. The transition system together with the parsing algorithm has quadratic time complexity, since it proceeds by trying to link each new token to each preceding token.

3.3 History-Based Models

In section 3.2 we defined three nondeterministic transition systems. Furthermore, we introduced an oracle $o : C \rightarrow T$, which the parsing algorithm uses to get the correct transition. If it is possible to derive the correct transitions from syntactically annotated sentences, we can use these as training data to approximate such an oracle through inductive learning. In other words, we define a mapping from an input string x and a dependency graph G to a transition sequence $C_{0,m} = (c_0, \ldots, c_m)$ such that G is uniquely determined. A transition t_i from c_{i-1} to c_i is dependent on all previously made transitions $(t_1, \ldots, t_{i-1})$ and all available information about these transitions, called the *history*. The history $H_i = (t_1, \ldots, t_{i-1})$ corresponds to some partially built structure and we also include static properties that are kept constant during the parsing of a sentence, such as the word form and the part-of-speech of a token.

The basic idea is thus to train a classifier that approximates an oracle given that a treebank is available. We will call the approximated oracle a *guide* (Boullier, 2003), because the guide does not guarantee that the transition is correct. The history $H_i = (t_1, \ldots, t_{i-1})$ contains complete information about all previous transitions. All this information is intractable for training a classifier. Instead we can use history-based feature models for predicting the next transition. History-based feature models were first introduced by Black et al. (1992) and have been used extensively in data-driven parsing (Magerman, 1995; Ratnaparkhi, 1997; Collins, 1999). To make it tractable the history H_i is replaced by a feature vector defined by a feature model $\Phi = (\phi_1, \ldots, \phi_p)$, where each feature ϕ_j is a function that identifies some significant property of the current configuration c and/or the input string x. To simplify notation we will write $\Phi(c, x)$ to denote the application of the feature vector $(\phi_1, \ldots, \phi_p)$ to c and x, i.e., $\Phi(c, x) = (\phi_1(c, x), \ldots, \phi_p(c, x))$.

3.4 Summary

In this chapter we have defined a formal framework for transition-based dependency parsing. We have defined a syntax graph and its specializations: the dependency graph and the phrase structure graph. Moreover, we have seen how a generic deterministic algorithm can be used together with three different transition systems to parse dependency representations. Finally, we have briefly discussed how history-based models can be used to approximate an oracle for predicting the next transition. Chapter 4 explores different discriminative learning methods for approximating oracles with classifiers acting as guides.

Chapter 4

Machine Learning for Transition-Based Dependency Parsing

Section 3.3 describes how history-based models can be used for guiding a deterministic parsing algorithm. In this chapter we will investigate two learning methods for predicting a transition sequence used by the parsing algorithm for constructing a dependency graph.

In general, disambiguation or classification is the task of predicting the class y given a variable x, which can be accomplished by probabilistic methods and it is common to divide these methods into two classes: *generative* and *discriminative*. For generative methods, we use the Bayes rule to obtain $P(y \mid x)$ by estimating the joint distribution $P(x, y)$. By contrast, discriminative methods make no attempt to model underlying distributions and instead estimate $P(y \mid x)$ directly, or even implicitly maximize this probability without modeling it. We will investigate two discriminative methods for the learning task: support vector machines (SVM) and memory-based learning (MBL), which are described in section 4.1 and 4.2. We continue in section 4.3 by investigating, based on Papers I and II, how these two learning methods can be used for guiding a parser.

Section 4.5 explores a parser optimization strategy for porting the parser to new languages. Although the optimization of the parser is an optimization problem on the parser level it originates from the machine learning level because the disambiguation of the dependency graph is influenced by direct or indirect parameters on the machine learning level. For example, an indirect parameter can be that of different transition systems that result in different sets of training instances for the machine learner, which can be more or less difficult to approximate with a learning algorithm.

4.1 Support Vector Machines

In the last decade, there has been a growing interest in SVMs, which were proposed by Vladimir Vapnik at the end of the seventies (Vapnik, 1979). SVMs are based on the idea that two linearly separable classes, the positive and negative samples in the training data, can be separated by a hyperplane with the largest margin. It has been shown that SVMs give good generalization performance in various research areas, such as face detection

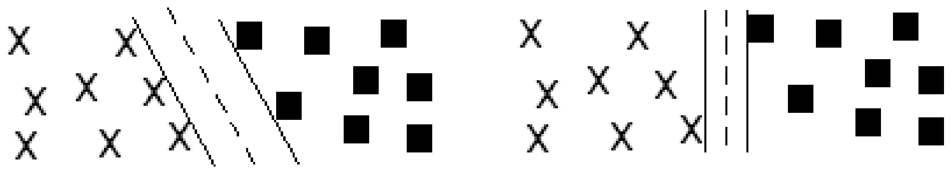


Figure 4.1: A linear support vector machine

(Osuna et al., 1997) and pedestrian detection (Oren et al., 1997). Within natural language processing they have been used extensively in, for example, text categorization (Joachims, 1998), chunking (Kudo and Matsumoto, 2001) and syntactic parsing (Yamada and Matsumoto, 2003).

Given a data set $I = \{(\overrightarrow{x_i}, y_i)\}_{i=1}^{\ell}$ of ℓ instance-label pairs, where $\overrightarrow{x_i} \in \mathcal{R}^p$ and $y_i \in \{-1, 1\}$, $\overrightarrow{x_i}$ is an p-dimensional feature vector representing the ith sample, and y_i is the class label of the ith sample which belongs to either the positive ($+1$) or the negative (-1) class. The feature vector $\overrightarrow{x_i}$ will in our case be the feature vector defined by $\Phi(c, x)$, where $\overrightarrow{x_i}[j]$ is the extracted value for the corresponding feature ϕ_j.

The class label y_i will be the transition t, but we need a method that handles multiple class labels (more about that later in this section). The idea is to estimate a vector $\overrightarrow{w}$ and a scalar b, which maximize the distance of any data point from the hyperplane defined by $\overrightarrow{w} \cdot \overrightarrow{x} + b$. The goal of SVMs are to find the solution of the following optimization (Kudo and Matsumoto, 2000; Burges, 1998):

$$\begin{array}{rl} \text{Minimize:} & L(w) = \frac{1}{2} \parallel \overrightarrow{w} \parallel^2 \\ \text{Subject to:} & y_i(\overrightarrow{w} \cdot \overrightarrow{x_i} + b) \geq 1, \forall i = 1, \ldots, \ell \end{array} \tag{4.1}$$

In other words, the SVM method tries to find the hyperplane that separates the training data into two classes with the largest margin. Figure 4.1 illustrates two possible hyperplanes, which correctly separate the training data into two classes, and the left hyperplane has the largest margin between the two classes.

The data in Figure 4.1 are easy to separate into two classes, but in practice the data may be noisy and therefore not linearly separable. One solution is to allow some misclassifications by introducing a penalty parameter C, which defines the trade-off between the training error and the magnitude of the margin.

SVMs can be extended to solve problems that are not linearly separable. The feature vector $\overrightarrow{x_i}$ is mapped to a higher dimensional space by the function φ, which makes it possible to carry out non-linear classification. The optimization problem can be rewritten into a dual form, which is done with a so called Kernel function $K(x_i, x_j) \equiv \varphi(x_i)^T \varphi(x_j)$ (Kudo and Matsumoto, 2001; Vapnik, 1998). There are many kernel functions, but the most common are:

- polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$.
- radial basis function (RBF): $K(x_i, x_j) = \exp(-\gamma \parallel x_i - x_j \parallel^2), \gamma > 0$.
- sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$.

where γ, r and d denote different kernel parameters (Hsu et al., 2004).

An SVM in its basic form is a binary classifier, but many learning problems have to deal with more than two classes. To make SVMs handle multi-classification, many binary classifiers are used. For multi-class classification, we can choose between the methods one-against-all and all-against-all. Given that we have n classes, the one-against-all method trains n classifiers to separate each class from the rest and the all-against-all method trains $n(n-1)/2$ classifiers, one for each pair of classes (Vural and Dy, 2004). A voting mechanism or some other measure is used to discriminate across all these classifiers to classify a new instance.

4.2 Memory-Based Learning

Memory-based learning and classification is based on the assumption that a cognitive learning task to a high degree depends on direct experience and memory, rather than extraction of an abstract representation. MBL has been used for many language learning tasks, such as part-of-speech tagging (Cardie, 1993; Daelemans et al., 1996), semantic role labeling (den Bosch et al., 2004; Kouchnir, 2004) and syntactic parsing (Kübler, 2004; Nivre et al., 2004).

MBL is a lazy method and is based on two fundamental principles: learning is storing experiences in memory, and solving a new problem is achieved by reusing solutions from previously solved problems that are similar to the new problem. The idea during training for MBL is to collect the values of different features from the training data together with the correct class (Daelemans and den Bosch, 2005). MBL generalizes by applying a similarity metric without abstracting or eliminating low-frequency events. This similarity metric can be seen as an implicit smoothing mechanism for rare events. Daelemans and colleagues have shown that it may be harmful to eliminate rare events in the training data for language learning tasks (Daelemans et al., 2002), because it is very difficult to discriminate noise from valid exceptions.

The p feature-values are mapped into a p-dimensional space, where each feature vector from the training data with its corresponding class is a point in this space. The task at decision time is to find the nearest neighbor(s) in this p-dimensional space and return a category based on the k nearest neighbor(s). The way this search is performed can be varied in many different ways.

The *overlap metric* is one of the most basic metrics and uses the distance $\Delta(\overrightarrow{x_i}, \overrightarrow{z_i})$ between two feature vectors $\overrightarrow{x_i}$ and $\overrightarrow{z_i}$, which are represented as

p features:

$$\Delta(\overrightarrow{x_i}, \overrightarrow{z_i}) = \sum_{j=1}^{p} w_i \delta(\overrightarrow{x}[j], \overrightarrow{z}[j]) \tag{4.2}$$

where w_j is a weight for feature j, and the function $\delta(\overrightarrow{x}[j], \overrightarrow{z}[j])$ is the distance per feature and will be 0 if $\overrightarrow{x}[j] = \overrightarrow{z}[j]$, otherwise 1. The weight w_j can be calculated by a variety of methods, e.g., *information gain* (IG), which measures each feature's contribution to our knowledge with respect to the target class.

A variation of the overlap metrics is the more sophisticated *modified value difference metric* (MVDM), introduced by Cost and Salzberg (1993), which estimates the distance between two values of a feature by considering their cooccurrence with the target classes. However, this metric is more sensitive to sparse data.

4.3 Learning for Parsing

In section 3.3 we introduced a *guide* that approximates an oracle, and in this section we will investigate how discriminative learning can be used to disambiguate transition sequences for constructing the dependency graph. It is convenient to distinguish between two phases when describing transition-based parsing: *the learning phase* and *the parsing phase.*

In the learning phase, the parser derives the correct transition by using an oracle function o applied to a gold standard treebank. For each transition it provides the learner with a training instance $(\overrightarrow{x_i}, t)$, where $\overrightarrow{x_i}$ is a current vector of feature values defined by a feature model $\Phi(c, x)$ and t is the correct transition. The learning problem is to induce a parser model (a classifier) from a set of training instances by using a learning algorithm. In this way, 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.

In the parsing phase, the parser uses the parser model, as a guide, to predict the next transition and now the vector of feature values $\overrightarrow{x_i}$ is the input and the transition t is the output of the guide. The feature extraction is performed in exactly the same way as in the learning phase.

The feature extraction uses the feature model, which is defined in terms of a feature vector $(\phi_1, \ldots, \phi_p)$, where each feature ϕ_j is a function, can be defined in terms of three simpler functions: an *address function* a_{ϕ_j}, which identifies a specific token in a given parser configuration, an *attribute function* f_{ϕ_j}, which picks out a specific attribute of the token, and a *mapping function* m_{ϕ_j}, which defines equivalence classes of attribute values.

1. For every i and j, $1 \leq i \leq d$ and $j \geq 0$, $\lambda_i[j]$, $\tau[j]$ are address functions defined on the input list τ or the ith data structure of the remaining data structures in the parse configuration (cf. definition 5). The index j identifies a particular token in the addressed data structure. For

example, if the Nivre parsing algorithm is used, then $\lambda_1[j]$ is the j+1th token from the top of the stack λ_1, and $\tau[j]$ is the $j+1$th token from the start of the input list τ, respectively. (Hence, $\lambda_1[0]$ is the top of the stack, and $\tau[0]$ is the next input token.)

2. If α is an address function, then $l(\alpha)$ and $r(\alpha)$ are address functions, identifying the left and right string neighbors, respectively, of the token identified by α.
3. If α is an address function, then $h(\alpha)$, $lc(\alpha)$, $rc(\alpha)$, $ls(\alpha)$ and $rs(\alpha)$ are address functions, identifying the head (h), the leftmost child (lc), the rightmost child (rc), the next left sibling (ls) and the next right sibling (rs), respectively, of the token identified by α (according to the partially built dependency graph).
4. If α is an address function and f is an attribute function, then $f(\alpha)$ is a feature function, identifying a particular attribute of the token identified by α. Papers I and II make use of three attribute functions: part-of-speech (p), word form (w), and edge label or dependency type (d) (where the edge label or the dependency type, if any, is given by the partially built dependency graph). Paper III, which uses the CoNLL dependency data format (Buchholz and Marsi, 2006), has three additional attribute functions: coarse-grained part-of-speech (cp), lemma (le) and unordered set of syntactic and/or morphological features (sm). It is possible to add more attribute functions if there are more attributes available in the data.
5. If $f(\alpha)$ is a feature function, then $m(f(\alpha))$ is a feature function if m is a mapping from the range of f to some new value set. For example, a mapping function can be used to restrict the value of a lexical feature to the last n characters of the word form.

A set of training instances, extracted according to the feature model is used by a learner to induce a parser model (a classifier). The set of possible transitions is discrete and finite, and therefore we can view this as a *classification* problem. A classifier is an approximation of the true oracle o, and is used to predict a transition t given a feature vector $\overrightarrow{x_i}$, which is the extracted values of using the same feature model $\Phi(c, x)$ as during learning. The feature models presented in Paper II are shown in table 4.1; the feature model Φ_2 is the non-lexical model and Φ_3 the lexical model in Paper I.

In practice, a learner will normally be an interface to a standard machine learning package. The experiments presented in all papers are based on two different software packages:

- TiMBL (Tilburg Memory-Based Learner) is a software package for memory-based learning and classification (Daelemans and den Bosch, 2005), which directly handles multi-valued categorical features. A training instance contains a feature vector $\overrightarrow{x_i}$ of categorical values and a transition t and can be used directly by TiMBL, because the input format of TiMBL is a column-based format with categorical values

Feature	Φ_1	Φ_2	Φ_3	Φ_4	Φ_5	Φ_6
$p(\lambda_1[0])$	+	+	+	+	+	+
$p(\tau[0])$	+	+	+	+	+	+
$p(\tau[1])$	+	+	+	+	+	+
$p(\tau[2])$				+	+	+
$p(\tau[3])$				+	+	+
$p(\lambda_1[1])$					+	
$d(\lambda_1[0])$		+	+	+	+	+
$d(lc(\lambda_1[0]))$		+	+	+	+	+
$d(rc(\lambda_1[0]))$		+	+	+	+	+
$d(lc(\tau[0]))$		+	+	+	+	+
$w(\lambda_1[0])$			+	+	+	+
$w(\tau[0])$			+	+	+	+
$w(\tau[1])$					+	+
$w(h(\lambda_1[0]))$					+	+

Table 4.1: Five different feature models used in Papers I and II; feature model Φ_6 is the model used by Nivre (2006).

separated by tabulation. A null-value is used to represent feature values that cannot be extracted, for example when the lookhead is out of range or the stack is empty.

- LIBSVM library (Chang and Lin, 2001) is a software library for SVM classification, which is able to handle multi-class classification. All categorical features are converted to numerical features using the standard technique of binarization.

Both TiMBL and LIBSVM have a wide variety of parameters, which can be tuned for a specific learning task. Paper I uses only TiMBL and Paper II uses both TiMBL and LIBSVM; Papers III–V only make use of LIBSVM.

4.4 Experimental Results

The leading evaluation metrics used in Papers I–III are the unlabeled and labeled attachment score. The *unlabeled attachment score* (UAS) is the proportion of tokens that are assigned the correct head (regardless of dependency label) and *labeled attachment score* (LAS) is the proportion of tokens that are assigned the correct head and the correct dependency label. Paper I presents the attachment score both as a macro-average (per sentence) and as a micro-average (per word). The former (per sentence) was abandoned because the latter (per word) has become standard practice in the research community for evaluating parsers with dependency representations. Another evaluation metric is the *exact match score*, which is the proportion of sentences that are assigned a completely correct dependency graph and also comes in an unlabeled and a labeled version.

Paper I presents our first large scale attempt to parse unrestricted natural language text in Swedish with a transition-based dependency parser

TS	**S**	**Swedish**		**English**		**Chinese**	
		UAS	*LAS*	*UAS*	*LAS*	*UAS*	*LAS*
NE	No	**86.8**	**83.1**	**89.2**	**87.7**	83.8	82.3
	Yes	86.4	82.6	88.9	87.4	83.9	82.4
NS	No	85.6	81.6	88.6	87.1	**84.8**	**83.2**
	Yes	84.6	80.5	88.6	87.1	84.5	83.0
		LT	*PT*	*LT*	*PT*	*LT*	*PT*
NE	No	42 min	5 min	46 h	2 h	11 h	1 h
	Yes	**2 min**	**22 s**	**3 h**	**9 min**	2 h	12 min
NS	No	1 h	5 min	62 h	2 h	12 h	1 h
	Yes	3 min	28 s	5 h	9 min	**1.5 h**	**11 min**

Table 4.2: Parsing accuracy and learning and parsing times when splitting the training instances into smaller sets (S = Yes/No) used by the SVM implementation of MaltParser 0.4; two versions of Nivre's parsing algorithms (TS), where arc-eager is denoted NE and arc-standard NS; the accuracy is measured by the attachment score (AS) metrics; U: unlabeled, L: labeled; LT: learning time, PT: parsing time.

using MBL.[1] Nivre and Scholz (2004) present similar experiments for English using the Wall Street Journal section of Penn Treebank II converted into dependency representations. Nivre (2006) investigates different feature models and MBL settings for both Swedish and English using the same approach. Paper II further investigates how machine learning can be used as a guide for three languages (Chinese, English and Swedish) by optimizing five different feature models (cf. table 4.1) and using two learning methods.

A study that was conducted in conjunction with Paper II, but not presented there, concerned how the set of training instances for the SVM learner could be divided into smaller sets according to the part-of-speech of the next token in order to reduce the learning and parsing times. Similar techniques have been used by Yamada and Matsumoto (2003), among others. The results are presented in Hall (2006), and are summarized in table 4.2 with feature model Φ_5. We can see that both learning and parsing times are reduced by a factor of 15 for English and Swedish and approximately by a factor of 6 for Chinese with a drop in accuracy of only a few tenths of a percentage point. The small decrease in accuracy is usually compensated by the fact that we can use more iterations to optimize the parser compared to the non-split instance set.

Table 4.3 shows a summary of the best results reported in Paper I, Paper II and Nivre (2006) using both MBL and SVM.[2] The results for Swedish and English in Paper I and II are based on part-of-speech tags, which are automatically assigned in a preprocessing phase by an HMM part-of-speech

[1] Nivre (2004) presents a pilot experiment corresponding to the MCLE-model in Paper I.

[2] The results for Paper II in table 4.3 are slightly higher than the results in table 4.2 for Swedish and English because of further optimization and the results for Chinese are slightly lower because Nivre arc-eager (NE) is used for the results for Paper II in table 4.3.

Paper	**FM**	**LM**	**Swedish**		**English**		**Chinese**	
			UAS	**LAS**	**UAS**	**LAS**	**UAS**	**LAS**
Paper I	Φ_3	MBL	84.7	80.6				
Nivre (2006)	Φ_6	MBL	86.3	82.0	88.1	86.3		
Paper II	Φ_5	MBL	86.6	82.3	88.0	86.2	81.1	79.2
		SVM	86.9	83.2	89.4	87.9	84.3	82.7

Table 4.3: Parsing accuracy; FM: feature model; LM: learning method; UAS: unlabeled attachment score; LAS: labeled attachment score.

tagger (Hall, 2003). Paper I shows that lexicalized models outperforms non-lexicalized models and one of the important results in Paper II is that SVM outperforms MBL when using more complex feature models with lexical features. The investigation presented in Paper II was a turning point in that we used SVM instead of MBL for guiding the parser. The results for English and Chinese were close to the state of the art at the time, while the results for Swedish were the best reported so far.

4.5 Parser Optimization

In this section we discuss how a transition-based dependency parser can be used for a large variety of languages and how we can optimize the parser to obtain a satisfying accuracy. Nivre and Hall (2005) shows that a transition-based dependency parser with MBL can be tuned for five languages using treebank data from Swedish, English, Czech, Danish and Bulgarian. Nivre et al. (2007) explore in an extensive empirical study how a transition-based dependency parser with MBL can be ported to a wide range of languages (Bulgarian, Chinese, Czech, Danish, Dutch, English, German, Italian, Swedish and Turkish) with an unlabeled attachment score over 80%. In the 2006 CoNLL-X shared task (Buchholz and Marsi, 2006), MaltParser using an SVM classifier was tuned for thirteen languages, and the highest labeled attachment score over all languages was obtained by MaltParser (Nivre et al., 2006; Hall and Nilsson, 2006) and MSTParser (McDonald et al., 2006) (both over 80%). Finally, we participated with two systems (both based on MaltParser) in the multilingual track of the CoNLL shared task 2007 (Nivre et al., 2007), where the task was to parse ten languages with dependency representations (Hall et al., 2007). The Single Malt system achieved an overall labeled attachment score close to 80%, which was the third best score in the shared task together with two other systems. The Blended system had the highest score (over 81%), based on six Single Malt parsers using the parser combination technique proposed by Sagae and Lavie (2006).

Paper III, based on the work done for the CoNLL shared task 2007 (Hall et al., 2007), summarizes all the experience that we have gained over the last six years of tuning MaltParser for a wide range of languages and it also presents the results of the CoNLL shared task 2007. Parser optimization in this context can be viewed as a machine learning optimization because almost all disambiguation is done by the machine learner. The optimization parameters can be divided into three main groups:

- *Learning algorithm parameters* are parameters that affect the learner directly. For example, both TiMBL and LIBSVM are equipped with several parameters such as kernel parameters for LIBSVM and the k-nearest neighbor parameter for TiMBL. A subgroup within this group of parameters is the *learning interface parameters*, which are parameters that affect the interface between the parser and one or more learners. For example, parameters concerned with the splitting of the training set into smaller subsets belong to this group.
- *Feature model parameters* are parameters that affect the learner indirectly. Adding or removing a feature will modify each training instance, which will also slightly modify the learning problem.
- *Parsing algorithm parameters* are parameters that affect the learner indirectly by altering the set of training instances and in that way the learning problem is slightly modified. The transition system of the parsing algorithm is one example of such a parameter. A subgroup within this group of parameters is the *pre/post-processing parameters*, which are parameters that modify the input during learning and the output during parsing. Pseudo-projective parsing proposed by Nivre and Nilsson (2005) is an example of pre/post-processing that allows a projective parsing algorithm to recover non-projective dependencies.

It is essential to optimize these parameters for the language of interest to get a good overall result. It is intractable to try out all combinations because there is an infinite number of combinations of these parameters. We have identified an optimization strategy that has been shown to give good accuracy for over twenty languages:

1. Define a good baseline system that have performed well for other languages and annotation schemes with similar properties.
2. Tune the parsing algorithm and the pre/post-processing parsing parameters once (with default settings for the feature model and learning algorithm parameters).
3. Optimize the feature model and the learning algorithm parameters in an interleaved fashion.

Here is an example of a good baseline system that is often used as the starting point for optimization:

- Nivre arc-eager transition system.
- Pseudo-projective parsing if the treebank contains a large proportion of non-projective sentences; otherwise there is no point in using this pre/post processing parameter.

- Feature model Φ_5 as the starting point for feature selection. If the treebank data is annotated with more attributes such as lemma and syntactic and/or morphological features then add these attributes for the top token of the stack and the next input token.
- LIBSVM as the classifier with the polynomial kernel $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$ of degree 2 ($d = 2$), and use $\gamma = 0.2$ and $r = 0$ for the kernel parameters, $C = 0.5$ for the penalty parameter, and $\epsilon = 1.0$ for the termination criterion.
- Split training instances into smaller sets according to the part of speech of the next input token $p(\tau[0])$ to reduce the learning and parsing times, which will make parser optimization more feasible.

The next step is to tune the parsing algorithm parameters, which can involve replacing the Nivre arc-eager transition system by the Covington non-projective transition system. It is important to try out all parser algorithm specific parameters in an interleaved fashion with all the pseudo-projective parsing parameters (if the treebank contains a large amount of non-projective sentences). If the parsing algorithm and the transition system are exchanged it is also important to modify the feature model so that it is adapted to the algorithm and transition system. MaltParser 1.0 or later is equipped with several default feature models depending on the algorithm and the transition system.

After the first round of optimization we will have one or at most a handful of candidates, which we want optimize in more detail. It is almost impossible to do an exhaustive search for the best feature model, so instead we use two different strategies:

1. Investigating properties of the languages and using features defined to capture these properties.
2. Batch testing of new features by forward and backward selection.

This can be further improved by a search for better learner specific settings with a grid search in an interleaved fashion with feature selection, but an exhaustive search is intractable.

Finally, it is important to point out that the optimization strategy presented above and exemplified in Paper III has not been evaluated in comparison to other strategies. Hence, the only way in which we can conclude that it is a good optimization strategy is that it has led to competitive results for a wide range of languages in several evaluations. There is no way of proving that it is an optimal strategy.

4.6 Summary

We have investigated how machine learning can be used to approximate the oracle to guide deterministic transition-based parsing algorithms. Moreover, we have empirical evidence that SVMs with a quadratic kernel outperforms

MBL for two languages, English and Chinese. We have only investigated two learning methods in Paper I and II, which makes it impossible to conclude that SVM is the best learning method for the task, but by comparing our results to the state of the art we can at least conclude that it seems to be well suited for the task. Finally, we have identified a strategy based on experience for optimizing a transition-based parser, which has been successfully applied in the CoNLL shared tasks in 2006 and 2007.

Chapter 5

Dependency-Based Phrase Structure Parsing

Despite the fact that dependency representations have received more attention in the parsing community lately, especially for free word order languages such as Czech, the dominant syntactic representation is still based on phrase structure. The standard method for using treebanks that are annotated with phrase structure representations as training material for dependency parsing is to convert phrase structures into dependency graphs using head-finding rules. There are several disadvantages compared to using treebanks annotated with genuine dependency representations. When converting the phrase structure treebank there has usually not been any way to do the inverse conversion, which has restricted parser evaluation to the converted version rather than the original treebank. Another problem is the derivation of meaningful dependency labels for the dependency edge labels. These disadvantages have been unsatisfactory when parsing treebanks based on phrase structure representations as in the experiments in Paper II.

Unfortunately, the simplification of the parsing problem has not been limited to parsing dependency versions of phrase structure treebanks. For example, English parsers trained on the Wall Street Journal section of the Penn Treebank (Marcus et al., 1993) often restrict the parsing problem to recovering nonterminals with their phrase labels. Besides phrase labels some nodes are annotated with function labels (such as the subject relation) and non-local dependencies are represented using empty categories and co-indexation (Bies et al., 1995), aspects of the annotation that are often ignored. Another example is German parsers trained on treebanks based on the Negra annotation scheme (Skut et al., 1997), where often the grammatical functions and the discontinuous phrases are not recovered by the parser. One explanation why parser developers ignore some aspects of the representation is that it is difficult to integrate these concepts into the parser model without sacrificing time and memory efficiency.

Paper IV presents, in the context of the PaGe 2008 shared task on parsing German (Kübler, 2008), a method for turning a data-driven dependency parser into a parser that parses phrase structure representations with grammatical functions. This method is evaluated without any complex head-finding rules on two German treebanks with continuous phrase structure representations extended with grammatical functions.

Paper V defines the method in more detail, which involves algorithms for transforming a phrase structure graph into a dependency graph and back. The method is capable of handling arbitrary phrase structures including discontinuous phrases and grammatical functions. Two experiments are based on the Swedish treebank Talbanken05 (Nivre et al., 2006) and the German TIGER treebank (Brants et al., 2002). The first experiment verifies empirically that the method can transform the phrase structure graphs into dependency graphs and back without loss of information. The second experiment evaluates parsers on the two treebanks with competitive results. The method for parsing phrase structure with a transition-based dependency parser consists of four steps:

1. Transform phrase structures into dependency graphs that encode the inverse transformation in complex edge labels.
2. Train a transition-based dependency parser on the dependency graphs derived in step 1.
3. Parse new sentences using the parser model induced in step 2.
4. Apply the inverse transformation to the dependency graphs produced in step 3, using the information encoded in the complex edge labels.

Paper V describes the algorithms used for the transformation from phrase structure to dependency (step 1) and back (step 4). Below we provide a formal characterization of the two mappings. We begin by defining a class of *mappable* phrase structure graphs, which is a subclass of the set of phrase structure graphs.

Definition 10 Given a set L_{NT} of phrase labels and a set L_E of edge labels a *mappable* phrase structure graph is a phrase structure graph $G = (V, E, F)$ such that

- $F = \{f_T, f_{NT}, f_E\}$, where
  - $f_T(v_i) = w_i$, for every $v_i \in V_T$.
  - $f_{NT}(v_i) \in L_{NT}$, for every $v_i \in V_{NT}$.
  - $f_E(e_i) \in L_E$, for every $e_i \in E$

The definition of a mappable phrase structure graph imposes special conditions on the labeling functions in F to ensure that each terminal node v_i is labeled with the corresponding word w_i, and that nonterminal nodes and edges are labeled with phrase labels and edge labels, respectively.

The rest of this chapter is structured as follows. Section 5.1 defines the transformation from a mappable phrase structure graph to an *encoding* dependency graph, and section 5.2 defines the inverse transformation. To make this more understandable, we go through an example in section 5.3. In section 5.4 we discuss the results in relation to Paper IV and V. Finally, section 5.5 summaries the main contributions and results of the chapter.

5.1 Phrase Structure to Dependency

The transformation of a mappable phrase structure graph G_P into an encoding dependency graph G_D is presented in Paper V as a two-step approach. First the heads of each node in G_P are identified, which is a standard approach for converting a phrase structure to a dependency graph. The algorithm IDENTIFY-HEAD in Paper V traverses the phrase structure graph from the root node and identifies the head-child v_c and the lexical head v_h for all nonterminal nodes V_{NT} in a recursive depth-first search. To identify the head-child v_c for each nonterminal v_m the algorithm uses heuristics called *head-finding rules*. The head-finding rules can be very simple, such as: Take the leftmost terminal child as the head-child if it exists; otherwise take the leftmost nonterminal child. But the head rules can also be more complex with a priority list for each phrase type. The second step in Paper V is an algorithm CREATE-DEPENDENCYGRAPH that creates a dependency graph according to the identified lexical heads and labels the edges with complex edge labels that makes it possible to perform the inverse transformation.

We formalize both steps with the following definition:

Definition 11 Given a mappable phrase structure graph $G_P = (V_R \cup V_T \cup V_{NT}, E_P, \{f_T, f_{NT}, f_E\})$ for a sentence $x = w_1, \ldots, w_n$, the encoding dependency graph is the graph $G_D = (V_R \cup V_T, E_D, \{f_T, f_D\})$ such that, for every edge $(v_m, v_c) \in E_P$ such that $h(v_m) = v_h, h(v_c) = v_d$ and $v_h \neq v_d$, it holds that:

- $(v_h, v_d) \in E_D$
- $f_D(v_h, v_d) = (f_E(v_m, v_c), \textsc{e-path}(v_d, v_c), \textsc{nt-path}(v_d, v_c), \textsc{dist}(v_h, v_m))$

and E_D and f_D are the smallest sets that satisfy these conditions.

In other words, for every edge (v_m, v_c) in the phrase structure graph G_P, we identify the lexical heads $h(v_m) = v_h$ and $h(v_c) = v_d$ using some function h typically defined by heuristic head finding rules, and add the edge (v_h, v_d) to the encoding dependency graph G_D. The function h must be constructed so that, for any non-terminal node $v_i \in V_{NT}$, $h(v_i)$ is a terminal node dominated by v_i. However, for the special root node v_0, we assume that $h(v_0) = v_0$. The edge (v_h, v_d) is labeled by a complex label consisting of four sublabels:

1. $f_E(v_m, v_c) = l$ is the grammatical function of the edge between the mother nonterminal node v_m and the child node v_c.
2. $\textsc{e-path}(v_d, v_c) = l_1^E | \ldots | l_p^E$ is the path of edge labels (or grammatical functions) from v_d to v_c in G_P.
3. $\textsc{nt-path}(v_d, v_c) = l_1^{NT} | \ldots | l_p^{NT}$ is the path of nonterminal labels (or phrase labels) from v_d to v_c in G_P.
4. $\textsc{dist}(v_h, v_m) = k$ is the number of non-terminal nodes between v_h and v_m.

5.2 Dependency to Phrase Structure

The inverse transformation is presented in Paper V as a bottom-up and top-down process on the dependency graph. First, the algorithm visits every node in the dependency graph and restores the spine of nonterminal nodes with phrase labels and grammatical functions for each terminal node v_d using the information in the sublabels $l_1^E|\ldots|l_p^E$ and $l_1^{NT}|\ldots|l_p^{NT}$ of the incoming edge. By the *spine* of v_d, we mean the path (sequence of edges) connecting v_d to the highest nonterminal node v_c in the phrase structure graph such that $h(v_c) = v_d$. Thus, the bottom-up process results in a spine of zero or more edges from each terminal to the highest nonterminal node of which the terminal is the lexical head. Secondly, the spines are weaved together according to the edges of the dependency graph. This is achieved by traversing the dependency graph recursively from the root using a pre-order depth-first search, where the dependent spine is attached to its head spine or to the root of the phrase structure graph G_P. The edge label is given by the first sublabel in the dependency graph, and the attachment point is given by the fourth sublabel, which specifies the number of nonterminal nodes between the (lexical) head and the attachment node.

We define the inverse transformation with the following definition:

Definition 12 If $G_D = (V_R \cup V_T, E_D, \{f_T, f_D\})$ is an encoding dependency graph, then for every edge $(v_h, v_d) \in E_D$ such that $f_D(v_h, v_d) = (l, l_1^E|\ldots|l_p^E, l_1^{NT}|\ldots|l_p^{NT}, k)$ let

- $V_{NT}^d = \{v_{d_1}, \ldots, v_{d_m}\}$
- $E_P^d = \{(\textsc{Ancestor}(k, v_h), v_{d_m}), (v_{d_m}, v_{d_{m-1}}), \ldots, (v_{d_1}, v_d)\}$
- $f_{NT}(v_{d_i}) = l_i^{NT}$
- $f_E(v_{d_i}, v_{d_{i-1}}) = l_i^E (v_{d_0} = v_d)$

The corresponding phrase structure graph is then the graph $G_P = (V_R \cup V_T \cup V_{NT}, E_P, \{f_T, f_{NT}, f_E\})$ such that

- $V_{NT} = \cup_{i=1}^n V_{NT}^i$
- $E_P = \cup_{i=1}^n E_P^i$

Each edge (v_h, v_d) in the dependency graph G_D, where v_h is the head and v_d is the dependent, is used for reconstructing the phrase structure graph G_P. The dependent spine of nonterminal nodes corresponding to v_d is a subset of nonterminal nodes V_{NT}^d identified by the path of nonterminal labels $l_1^{NT}|\ldots|l_p^{NT}$. The edges of the dependent spine is a subset of edges E_P^d identified by the path of edge labels $l_1^E|\ldots|l_p^E$ and the edge connecting the dependent spine to the head spine is calculated by the function Ancestor. The function Ancestor takes sublabel k and the lexical head v_h of the head spine as arguments and returns a nonterminal node of G_P in V_{NT} given by traversing the head spine $k+1$ steps up. Moreover, if $k+1 > |V_{NT}^h|$ then the

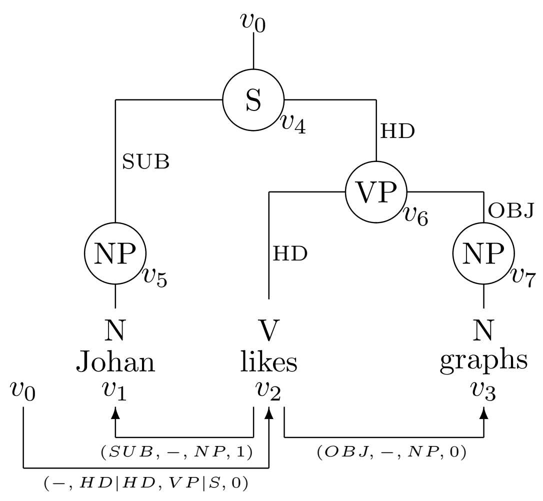


Figure 5.1: An example of encoding a mappable phrase structure graph G_P (illustrated above the sentence) as an encoding dependency graph G_D (below the sentence) for the sentence *Johan likes graphs.*

function ANCESTOR returns the root node v_0 of G_P.[1] The label functions of G_P associate the labels to the nonterminal nodes and the edges using the information in the edge labels.

5.3 Encoding Example

Figure 5.1 illustrates the encoding of a mappable phrase structure graph into an encoding dependency graph with complex edge labels for an English sentence. The token *likes* is the lexical head v_h (i.e., $h = 2$) of the nonterminal S because the nonterminal VP is the head-child of the nonterminal S and *likes* is the lexical head of the nonterminal VP. This means that *likes* is the lexical head of both S and VP. The mother node of S is the artificial root node of the phrase structure graph, which means that *likes* is the dependent of the artificial root node in the encoding dependency graph (illustrated by an edge from node v_0 to v_2 below the sentence).[2] The edge is labeled with a complex label (−, HD|HD, VP|S, 0), which consists of four sublabels. The first sublabel $l = -$ indicates an empty edge label from the artificial root node v_0 to S in the phrase structure graph. The second and third sublabels encode the path of edge labels HD|HD and nonterminal labels VP|S in the dependent spine in the phrase structure graph. Finally, the fourth sublabel is $k = 0$, which means that the dependent spine should be attached directly to the lowest possible node of the head spine, in this case the artificial root node v_0.

The left NP nonterminal has the lexical head v_1 (i.e., the token *Johan*), which is a dependent of the node v_2 in the encoding dependency graph with

[1] This takes care both of the special case where $v_h = v_0$ and of cases where the parser predicts an impossible attachment level.

[2] Remember that v_0 is its own head.

an edge labeled (SUB, −, NP, 1). The first sublabel SUB comes from the edge between the S nonterminal and the left NP nonterminal, which is the edge that connects the dependent spine and the head spine. The sublabels − and NP encode the empty edge label and the single nonterminal label NP of the dependent spine. The fourth sublabel is $k = 1$, which means that the dependent spine is connected one step up in the head spine (i.e., the left NP is connected to S, not VP).

Finally, the dependent spine with the right NP nonterminal is represented in the encoding dependency graph as an edge from v_2 to v_3 with a complex label (OBJ, −, NP, 0). The sublabels − and NP encode the dependent spine going from the node v_3, and the attachment level 0 indicates that the dependent spine should be attached to the first nonterminal node of the head spine (i.e., VP) with the label OBJ.

5.4 Experimental Results

Paper IV presents the results of parsing both dependency and phrase structure representations for two German treebanks in the context of the PaGe shared task on parsing German. The treebanks prepared by the organizers were based on the German TIGER (Brants et al., 2002) and TüBa-D/Z (Telljohann et al., 2005) treebanks. The dependency and phrase structure versions contained the same set of sentences, approximately 23000 sentences. Both treebanks were converted into dependency representations and continuous phrase structure representations by the organizers.

The dependency track was straightforward for MaltParser, using the parser optimization strategies presented in Paper III and briefly described in section 4.5. We achieved a labeled attachment score close to 90% for both the treebanks, 90.80 for TIGER and 88.64 for TüBa-D/Z, but we were unfortunately the only participants in this track.

We used a version of the method presented in section 5.1 and 5.2 to parse the phrase structure versions of the treebanks with MaltParser. More precisely, we used the Nivre parsing algorithm with the arc-eager transition system. One difference from the approach described in chapters 3 and 4 is that the prediction of the transition t is not combined with the prediction of the edge label into one complex prediction with one feature model. Instead the prediction of the parser action is divided into several predictions. The reason for dividing the prediction is that the number of distinct complex edge labels can be quite large. The prediction strategy presented in Paper IV instead predicts the transition as follows: if the transition is SHIFT or REDUCE the nondeterminism is resolved, but if the predicted transition is RIGHT-ARC or LEFT-ARC the parser continues to predict the sublabels of the complex edge label separately with separate feature models for RIGHT-ARC and LEFT-ARC. With this prediction strategy there will be nine feature models and the parser makes use of nine separately trained classifiers. We used very simple head rules for transforming the phrase

		TIGER			TüBa-D/Z		
System	**Gold**	**recall**	**precision**	**F_1-score**	**recall**	**precision**	**F_1-score**
Berkeley	TG	69.23	70.41	69.81	83.91	84.04	83.97
Stanford	TG	58.52	57.63	58.07	79.26	79.22	79.24
Stanford				49.03			73.36
Växjö	T	67.06	63.40	65.18	76.44	74.79	75.60

Table 5.1: The results of the phrase structure track of the PaGe shared task on parsing German, taken from Kübler (2008) and Rafferty and Manning (2008). TG means that the gold part of speech tags including grammatical functions going from nonterminal to terminal is used and T means that the gold part of speech tags are used.

structure representation into dependency representations. For the TIGER treebank we perform a left-to-right search to find the leftmost lexical child. If no lexical child can be found, the head-child of the nonterminal will be the leftmost nonterminal child and the lexical head will be the lexical child of the head child recursively. For the TüBa-D/Z treebank we used similar techniques but varied the direction of search according to the label of the target nonterminal.

The results for the three participants are shown in table 5.1. It is difficult to compare the results because of different inputs to the systems. The labeled F_1-score was the main evaluation metric, and labeled in this context means that both the phrase labels and the grammatical functions should agree with the gold-standard, but grammatical functions labeling the edge between a nonterminal and a terminal were not included in the evaluation. The Berkeley parser (Petrov and Klein, 2008) based on learning latent variable grammars has the best scores for both treebanks, but uses gold part of speech tags including grammatical functions going from nonterminals to terminals during the evaluation. MaltParser (Växjö in table 5.1) makes use of the gold part of speech tags, but not the incoming grammatical functions to the terminal. For TüBa-D/Z, the Stanford parser (Rafferty and Manning, 2008) outperforms MaltParser when using gold part of speech tags including grammatical functions, but not when tagged data is used. For TIGER, MaltParser outperforms the Stanford parser regardless of whether the gold tags are used or not. From our point of view, an interesting comparison is to compare the results of the dependency track and the phrase structure track, where MaltParser obtains an accuracy close to 90% in the dependency track, but only 65% for TIGER and close to 76% for TüBa-D/Z in the phrase structure track. There seems to be room for improvements, especially if we can find more suitable head rules. However, it is also clear that the phrase structure parsing problem is a more complex task.

With these promising results we continued developing the method and adjusted the algorithms so that also discontinuous phrases could be handled. Besides the algorithms for transforming phrase structure graphs to dependency graphs and back, Paper V contains an empirical study of parsing both continuous and discontinuous phrase structure. The Swedish treebank

Talbanken05 (Nivre et al., 2006) and the German TIGER treebank (Brants et al., 2002) are used for evaluating the parser. Both treebanks are encoded in the TIGER-XML format, but use different annotation schemes. Note that the TIGER treebank used in the evaluation in Paper IV is based on the original TIGER treebank, but has been adapted for the PaGe shared task both with respect to size and by eliminating all discontinuous phrases. The evaluation in Paper V uses the original treebank. For Swedish, we used Talbanken05, which is a modernized version of Talbanken (Einarsson, 1976a,b).

In Paper V, we used the Covington algorithm to be able to recover non-projective edges (corresponding to discontinuous phrases) with the same prediction strategy described above for dividing the prediction of the transition into several steps. More complex head rules, compared to Paper IV, were used for transforming phrase structure graphs to dependency graphs. Actually, we can regard the search of appropriate head rules as an optimization of the parser, which will fall into the subgroup of *pre/post processing parameters* presented in section 4.5. The evaluation shows that the parser can recover both continuous and discontinuous phrases labeled with both phrase labels and grammatical functions with an F_1-score close to 70% for TIGER using the gold part of speech tags and close to 65% with tagged data. To compare the results with those in Paper IV is difficult because of different data sets. The accuracy increased by almost 5 percentage points, and taking into account that 22.5% out of all phrases are discontinuous phrases these results are more impressive. On the other hand, the training set used in Paper IV is half the size of the data set used in Paper V. For Talbanken05, the F_1-score is over 65% using the gold part of speech tags and over 58% with tagged data. Comparison to the state of the art is also difficult, because of difference in input and output representations, but Paper V includes a few relevant comparisons, which indicate that our results are competitive.

5.5 Summary

In this chapter we have defined a transformation from phrase structure graphs into dependency graphs and back in a loss-less fashion. Moreover, we have presented empirical studies, based on Papers IV and V, showing that it is possible to obtain competitive results for two languages when parsing phrase structure representations with a transition-based dependency parser. Furthermore, we have shown that the original annotation of both TIGER treebank and Talbanken05 can be recovered by the parser without simplifications including both discontinuous phrases and grammatical functions.

Chapter 6

MaltParser

One of the major contributions of this thesis is the MaltParser system, which is a language-independent system for transition-based dependency parsing of both dependency and phrase structure representations. The development of MaltParser has been joint work with especially Joakim Nivre, but also Jens Nilsson. MaltParser implements the approach described in this thesis and the theoretical framework of inductive dependency parsing proposed by Nivre (2006). The underlying architecture uses a strict modularization of parsing algorithms, feature models and learning methods, thereby giving maximum flexibility in the way these components can be varied independently of each other. MaltParser can be seen as a data-driven parser-generator framework given that a treebank is available (Nivre et al., 2006). Whereas a traditional parser-generator maps a grammar to a parser, a data-driven parser-generator maps a treebank to a parser. With different settings for parsing algorithms, feature models and learners the parser-generator will construct different parsers. The idea is to give the user the flexibility to experiment with the components in a more convenient way, although there are still dependencies between components, in the sense that not all combinations will perform well with respect to accuracy and efficiency.

MaltParser (version 0.x) was first implemented in C, and the design and the implementation are presented in more detail in Hall (2006). After some consideration we decided to completely reimplement MaltParser in Java (version 1.x) to make the system more flexible. The current release of MaltParser 1.x contains several extensible interfaces, including:

- *A deterministic parsing algorithm interface*, currently implemented by the Nivre parsing algorithm (arc-eager and arc-standard) and the Covington parsing algorithm (projective and non-projective).
- *A feature extraction interface*, where several feature types can be defined using the following types of auxiliary functions:
  - *Address functions*, which are used for addressing a particular token in the current parser configuration. For example, the top token of the stack or the leftmost dependent of the next input token.
  - *Attribute functions*, which are used for extracting a feature value given a token (identified by a address function), e.g., the part of

speech or the word form.

  - *Mapping functions*, which are used for mapping a feature value onto a new set of values, e.g., extract the suffix or the prefix of a feature value.

  It is easy to add new address, attribute and mapping functions.

- *A reader and writer interface*, which allows developers to add new types of readers and writers. There are currently readers and writers for tab-separated input and for the Negra export format.
- *An input and output format interface*, which allows the user to adapt the format in an XML file. For example, the CoNLL data format is included in the distribution.
- *A learner interface*, where the LIBSVM package (Chang and Lin, 2001) is included in the distribution.

Moreover, MaltParser is equipped with a plug-in management system that allows developers to extend interfaces and add control options in a self-contained plug-in, which is loaded when the system starts.

MaltParser has been used by several researchers (excluding the developers) for various kind of research. For example, Johansson and Nugues (2006) describe an implementation of a Swedish semantic role labeling system that uses MaltParser. Johansson and Nugues (2007a) compared MaltParser and MSTParser for parsing English with a new conversion approach to convert the Penn Treebank into dependency representations that allow non-projective graphs. Goldberg and Elhadad (2008) used MaltParser to evaluate a fast, space efficient and non-heuristic method for calculating the decision function of polynomial kernel classifiers. Jorgensen (2007) used MaltParser and Dan Bikel's parser to measure the effects of disfluency detection in parsing spoken language. Mitamura et al. (2007) describe the JAVELIN system to perform cross-lingual question answering from Japanese and Chinese documents, where MaltParser is used for syntactic parsing. Marsi et al. (2006) use MaltParser to explore the usefulness of normalized alignment of dependency trees for entailment prediction. Herrera et al. (2007) presents a Spanish dependency parser using MaltParser trained on a dependency version of the Cast3LB corpus. Ciaramita et al. (2008) describe the DeSRL system for semantic role labeling in linear-time, which uses MaltParser as one of three parser for the parsing sub-task. Samuelsson et al. (2008) uses a blend of eight MaltParser models to perform the parsing sub-task of the semantic role labeling. Øvrelid (2008) investigates how linguistically motivated features can improve the parsing accuracy for data-driven dependency parsing of Swedish using MaltParser.

MaltParser is distributed with an open source license and is freely available from *maltparser.org*, where also a user guide and several pre-trained models, among other things, can be found.

Chapter 7

Conclusion

In this final chapter, we summarize the main contributions of the thesis and point to promising directions for future work.

7.1 Main Contributions

This thesis presents several empirical studies for parsing dependency and phrase structure representations using MaltParser to highlight different aspects of transition-based parsing.

Papers I–III investigate the possibilities of using machine learning to resolve nondeterminism in the transitions between parser configurations. Machine learning makes it easier to port a parser to other languages, given that a treebank is available for the language in question, compared to applying heuristics for resolving the nondeterminism.

Paper I presents the idea of using machine learning to guide a deterministic parsing algorithm and an empirical study of parsing Swedish with dependency representations using MBL. The promising results of this study led to a more extensive evaluation in Paper II, with an empirical comparison of MBL and SVM for deterministic dependency parsing, using treebank data from Swedish, English and Chinese and five feature models of varying complexity. The outcome shows that SVM gives higher parsing accuracy for complex and lexicalized feature models for English and Chinese, whereas for Swedish there is no significant improvement. The parsing efficiency is slightly better for SVM when using the most complex model, but for the other models it is the opposite. As expected, MBL has the fastest learning times for all models since it only stores the training data in a suitable representation, whereas SVM generalizes the data into a model. The drawback of using learning methods like SVM is that they are computationally expensive. We have examined a method proposed by Yamada and Matsumoto (2003) that makes the parsing more efficient with only a tiny drop in accuracy by dividing the set of training instances into smaller sets according to the part of speech of the next input token, which makes it possible to create separate classifiers for each set. Another method for speeding up the parser is the prediction strategy introduced in Paper IV and V, where the prediction of the transition is divided into the prediction of parser actions and the prediction of the individual sublabels of the complex edge label.

Paper III demonstrates, in the context of the CoNLL shared task 2007, that transition-based dependency parsing, with careful optimization, can give highly accurate dependency parsing for a wide range of languages. Moreover, the paper presents an optimization strategy based on experience of optimizing MaltParser for over twenty languages with different properties. The strategy proceeds from a good baseline model that can be optimized using knowledge about the language in question in combination with batch testing.

Papers IV–V go beyond dependency-based representations and explore a method for parsing phrase structure representations. Paper IV shows with rather competitive results that a transition-based dependency parser can be used for parsing German continuous phrase structures with grammatical functions. The phrase structures are transformed into dependency representations using naive head-finding rules during learning time and information is encoded in complex edge labels of the dependency graphs, which makes it possible to apply the inverse transformation during parsing. Paper V demonstrates how the method can be refined so that discontinuous phrases can be recovered by the parser using a deterministic non-projective parsing algorithm. The evaluation on Swedish and German shows state of the art results for parsing phrase structure with both phrase labels and grammatical functions. Paper V also illustrates that parsers can be used for parsing treebanks with the original annotation without simplifying the representation. A secondary contribution of the two papers is the loss-less transformation of phrase structures to dependency graphs and back.

Beside the findings and results in Papers I–V, an underlying goal has been to construct a flexible system for transition-based dependency parsing that allows us to experiment with different interesting features in the area of data-driven dependency parsing. Without overstating we can conclude that this underlying goal has been fulfilled with the MaltParser system. In fact, the system has performed quite well in comparison to other parser systems in several evaluations and it is used by other researchers to perform various kinds of NLP applications.

7.2 Future Directions

Despite having presented a flexible and quite efficient system for transition-based parsing of both dependency and phrase structure representations and empirical studies showing that parsing can be performed with high accuracy there is still room for improvements. We believe that a syntactic parser needs to be robust, flexible, accurate and very efficient to become a natural component of large-scale NLP applications. We have seen a lot of improvement during the last decades on syntactic parsing, but more research is needed. We have compiled a list of four major categories of future research directions for transition-based parsing:

1. **Efficiency**: One interesting factor for data-driven transition-based

parsing is the efficiency both during learning and parsing. This is very important, if a parser is to be useful in NLP applications. One way to increase the efficiency would be to run more in parallel. For instance, we could investigate new approaches of dividing the set of training instances into smaller sets and train separate classifiers, which can be based on different machine learning methods. Another interesting idea would be to combine heuristics and machine learning. Moreover, we could experiment with other machine learning techniques such as online learning.

2. **Optimization**: Performing a complete feature and parameter optimization is practically impossible. If we could find a good approach that automatically finds the best-suited feature model together with near optimal parameters in reasonable time, it could be implemented in an automatic feature and parameter optimization tool.
3. **Nondeterminism**: Loosening the strictly deterministic approach to parsing may lead to higher accuracy since we could explore several analyses at a more global level. There has been some work in this direction, for example, the work by Duan et al. (2007), Johansson and Nugues (2007b), Titov and Henderson (2007) and Sagae and Lavie (2005), but hopefully more research in this area can improve the results even more.
4. **Phrase structure**: We have seen that a transition-based dependency parser can be used successfully for parsing German and Swedish phrase structure representations. It would be interesting to explore other strategies for encoding the inverse transformation into complex edge labels of the dependency graph. Recently, Carreras et al. (2008) presented a technique for phrase structure parsing, where the phrase structure is decomposed into spines in a way very similar to the method presented here and achieve state of art results for English. Instead of enriching edge labels, as we do, they label nodes with spines and only use edges for combining the spines.

   Another research task is to experiment with more languages in combination with different encoding strategies. One of the top priorities is to parse the Penn Treebank in the original format, which means that we need to introduce empty categories at parsing time. One way to do this would be to adapt the parsing algorithm to include one or more new parse actions that insert empty categories. Another way would be to incorporate the ideas of Johansson and Nugues (2007a) to convert the phrase structure representation of the Penn Treebank into non-projective dependency graphs and find the inverse conversion.

Bibliography

Abney, S. (1991). Parsing by chunks. In R. Berwick, S. Abney, and C. Tenny (Eds.), *Principle-Based Parsing*, pp. 257–278. Kluwer Academic Publishers.

Aho, A. V., R. Sethi, and J. D. Ullman (1986). *Compilers: Principles, Techniques, and Tools.* Addison-Wesley Longman Publishing Co.

Attardi, G. (2006). Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pp. 166–170.

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM. In *Proceedings of the International Conference on Information Processing*, pp. 125–135.

Bies, A., M. Ferguson, K. Katz, and R. MacIntyre (1995). Bracketing guidelines for Treebank II style, Penn Treebank project. University of Pennsylvania, Philadelphia.

Black, E., F. Jelinek, J. D. Lafferty, D. M. Magerman, R. L. Mercer, and S. Roukos (1992). Towards history-based grammars: using richer models for probabilistic parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, pp. 31–37.

Bloomfield, L. (1933). *Language.* The University of Chicago Press.

Böhmová, A., J. Hajič, E. Hajičová, and B. Hladká (2003). The PDT: a 3-level annotation scenario. In A. Abeillé (Ed.), *Treebanks: Building and using parsed corpora*, Chapter 7, pp. 103–127. Kluwer Academic Publishers.

Boullier, P. (2003). Guided Earley parsing. In G. van Noord (Ed.), *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 43–54.

Brants, S., S. Dipper, S. Hansen, W. Lezius, and G. Smith (2002). The TIGER Treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories Sozopol*, pp. 1–18.

Buchholz, S. and E. Marsi (2006). CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pp. 149–164.

Bunt, H. (1991). Parsing with discontinuous phrase structure grammar. In M. Tomita (Ed.), *Current Issues in Parsing Technology*, pp. 49–63. Kluwer Academic Publishers.

Bunt, H. (1996). Formal tools for describing and processing discontinuous constituency structure. In H. Bunt and A. van Horck (Eds.), *Discontinuous Constituency Natural Language Processing 6*, pp. 63–83. Mouton de Gruyter.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery 2*(2), 121–167.

Cardie, C. (1993). Using decision trees to improve case-based learning. In *Proceedings of the 10th International Conference on Machine Learning*, pp. 25–32.

Carreras, X., M. Collins, and T. Koo (2008). TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning (CoNLL)*, pp. 9–16.

Chang, C.-C. and C.-J. Lin (2001). LIBSVM: A library for support vector machines.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *Proceedings of the First Annual Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 132–139.

Charniak, E. and M. Johnson (2005). Coarse-to-fine *n*-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 173–180.

Cheng, Y., M. Asahara, and Y. Matsumoto (2005). Chinese deterministic dependency analyzer: Examining effects of global features and root node finder. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, pp. 17–24.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory IT-2*, 113–124.

Ciaramita, M., G. Attardi, F. Dell'Orletta, and M. Surdeanu (2008). DeSRL: A linear-time semantic role labeling system. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pp. 258–262.

Clark, S. and J. R. Curran (2004). Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL)*, pp. 104–111.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *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.* Ph. D. thesis, University of Pennsylvania.

Collins, M. and N. Duffy (2005). Discriminative reranking for natural language parsing. *Computational Linguistics 31*, 25–70.

Corazza, A., A. Lavelli, G. Satta, and R. Zanoli (2004). Analyzing an Italian treebank with state-of-the-art statistical parsers. In S. Kübler, J. Nivre, E. Hinrichs, and H. Wunsch (Eds.), *Proceedings of the Third Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 39–50.

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

Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pp. 95–102.

Daelemans, W. and A. V. den Bosch (2005). *Memory-Based Language Processing.* Cambridge University Press.

Daelemans, W., A. V. den Bosch, and J. Zavrel (2002). Forgetting exceptions is harmful in language learning. *Machine Learning 34*, 11–43.

Daelemans, W., J. Zavrel, P. Berck, and S. Gillis (1996). A memory-based part of speech tagger generator. In E. Ejerhed and I. Dagan (Eds.), *Proceedings of the Fourth Workshop on Very Large Corpora*, pp. 14–27.

den Bosch, A. V., S. Canisius, W. Daelemans, I. Hendrickx, and E. T. K. Sang (2004). Memory-based semantic role labeling: Optimizing features, algorithm, and output. In *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL)*, pp. 102–105.

Duan, X., J. Zhao, and B. Xu (2007). Probabilistic parsing action models for multi-lingual dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pp. 940–946.

Dubey, A. (2005). What to do when lexicalization fails: Parsing German with suffix analysis and smoothing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 314–321.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM 13*, 94–102.

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. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pp. 340–345.

Ejerhed, E. and K. Church (1983). Finite state parsing. In F. Karlsson (Ed.), *Papers from Seventh Scandinavian Conference of Linguistics*, pp. 410–432.

Gabbard, R., M. Marcus, and S. Kulick (2007). Fully parsing the Penn Treebank. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, pp. 184–191.

Goldberg, Y. and M. Elhadad (2008). splitsvm: Fast, space-efficient, non-heuristic, polynomial kernel computation for NLP applications. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Short Papers (Companion Volume)*, pp. 237–240,.

Hall, J. (2003). A probabilistic part-of-speech tagger with suffix probabilities. Master's thesis, MSI report 03015. Växjö University: School of Mathematics and Systems Engineering.

Hall, J. (2006). MaltParser – An architecture for inductive labeled dependency parsing. Licentiate's thesis, MSI report 06050. Växjö University: School of Mathematics and Systems Engineering.

Hall, J. and J. Nilsson (2006). CoNLL-X shared task: Multi-lingual dependency parsing. Technical report, MSI report 06060. Växjö University: School of Mathematics and Systems Engineering.

Hall, J., J. Nilsson, and J. Nivre (2008). Single malt or blended? A study in multilingual parser optimization. In H. Bunt, P. Merlo, and J. Nivre (Eds.), *Trends in Parsing Technology.* Springer.

Hall, J., J. Nilsson, J. Nivre, G. Eryiğit, B. Megyesi, M. Nilsson, and M. Saers (2007). Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pp. 933–939.

Harper, M. P. and R. A. Helzermann (1995). Extensions to constraint dependency parsing for spoken language processing. *Computer Speech and Language 9*, 187–234.

Herrera, J., P. Gervàs, P. J. Moriano, A. Muñoz, and L. Romero (2007). Building corpora for the development of a dependency parser for Spanish using MaltParser. Procesamiento del Lenguaje Natural.

Hindle, D. (1989). Acquiring disambiguation rules from text. In *Proceedings of the 27th annual meeting on Association for Computational Linguistics*, pp. 118–125.

Hsu, C.-W., C.-C. Chang, and C.-J. Lin (2004). A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University.

Jensen, K. and G. E. Heidorn (1983). The fitted parse: 100% parsing capability in a syntactic grammar of English. In *Proceedings of the first conference on Applied Natural Language Processing*, pp. 93–98.

Joachims, T. (1998). Text categorization with support vector machines. In *Proceedings of the 10th European Conference on Machine Learning (ECML)*, pp. 137–142.

Johansson, R. and P. Nugues (2006). Construction of a FrameNet labeler for Swedish text. In *Proceedings of the fifth International Conference on Language Resources and Evaluation (LREC)*.

Johansson, R. and P. Nugues (2007a). Extended constituent-to-dependency conversion for English. In *Proceedings of the 16th Nordic Conference on Computational Linguistics*, pp. 105–112.

Johansson, R. and P. Nugues (2007b). Incremental dependency parsing using online learning. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pp. 1134–1138.

Johnson, M., S. Geman, S. Canon, Z. Chi, and S. Riezler (1999). Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 535–541.

Jorgensen, F. (2007). The effects of disfluency detection in parsing spoken language. In J. Nivre, H.-J. Kaalep, K. Muischnek, and M. Koit (Eds.), *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA)*, pp. 240–244.

Joshi, A. K. (1985). Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions. In D. R. Dowty, L. Karttunen, and A. M. Zwicky (Eds.), *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*, pp. 206–250. Cambridge University Press.

Kalt, T. (2004). Induction of greedy controllers for deterministic treebank parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 17–24.

Kaplan, R. M. and J. Bresnan (1983). Lexical-functional grammar: a formal system for grammatical representation. In J. Bresnan (Ed.), *The Mental Representation of Grammatical Relations*, pp. 173–281. The MIT Press.

Karlsson, F. (1990). Constraint grammar as a framework for parsing running text. In *Proceedings of the 13th conference on International Conference on Computational Linguistics (COLING)*, pp. 168–173.

Karlsson, F., A. Voutilainen, J. Heikkil, and A. Anttila (1995). *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Tex.* Mouton de Gruyter.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Technical report, Air Force Cambridge Research Laboratory, Bedford, MA.

Kay, M. (1980). Algorithm schemata and data structures in syntactic processing. Technical report, Xerox Palo Alto Research Center.

Knuth, D. E. (1965). On the translation of languages from left to right. *Information and Control 8*, 607–639.

Koskenniemi, K. (1990). Finite-state parsing and disambiguation. In *Proceedings of the 13th conference on Computational linguistics - Volume 2*, pp. 229–232.

Kouchnir, B. (2004). A memory-based approach for semantic role labeling. In *Proceedings of the Eighth Conference on Computational Natural Language Learning*, pp. 118–121.

Kromann, M. T. (2003). The Danish Dependency Treebank and the DTAG treebank tool. In J. Nivre and E. Hinrichs (Eds.), *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 217–220. Växjö University Press.

Kübler, S. (2004). *Memory-Based Parsing*. Amsterdam: John Benjamins.

Kübler, S. (2008). The PaGe 2008 shared task on parsing German. In *Proceedings of the ACL-08: HLT Workshop on Parsing German (PaGe)*, pp. 55–63.

Kübler, S., E. W. Hinrichs, and W. Maier (2006). Is it really that difficult to parse German. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 111–119.

Kudo, T. and Y. Matsumoto (2000). Japanese dependency structure analysis based on support vector machines. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC)*, pp. 18–25.

Kudo, T. and Y. Matsumoto (2001). Chunking with support vector machines. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Kudo, T. and Y. Matsumoto (2002). Japanese dependency analysis using cascaded chunking. In *Proceedings of the Sixth Workshop on Computational Language Learning (CoNLL)*, pp. 63–69.

Lang, B. (1988). Parsing incomplete sentences. In *Proceedings of the 12th conference on Computational linguistics - Volume 1*, pp. 365–371.

Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 276–283.

Marcus, M., B. Santorini, and M. Marcinkiewicz (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics 19*(2), 313–330.

Marsi, E., E. Krahmer, and M. T. Wauter Bosma (2006). Normalized alignment of dependency trees for detecting textual entailment. In *Proceedings of the Second PASCAL Recognising Textual Entailment Challenge (RTE-2)*.

Maruyama, H. (1990). Structural disambiguation with constraint propagation. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 31–38.

McDonald, R., K. Crammer, and F. Pereira (2005). Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 91–98.

McDonald, R., K. Lerman, and F. Pereira (2006). Multilingual dependency analysis with a two stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pp. 216–220.

McDonald, R. and J. Nivre (2007). Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 122–131.

McDonald, R. and G. Satta (2007). On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*, pp. 122–131.

Mellish, C. S. (1989). Some chart-based techniques for parsing ill-formed input. In *Proceedings of the 27th annual meeting on Association for Computational Linguistics*, pp. 102–109.

Menzel, W. and I. Schröder (1998). Decision procedures for dependency parsing using graded constraints. In *Proceedings of the Workshop on Dependency-Based Grammars*, pp. 78–87.

Mitamura, T., F. Lin, H. Shima, M. Wang, J. Ko, J. Betteridge, M. Bilotti, A. Schlaikjer, and E. Nyberg (2007). JAVELIN III: Cross-lingual question answering from Japanese and Chinese documents. In *Proceedings of NTCIR-6 Workshop Meeting*, pp. 202–209.

Miyao, Y., T. Ninomiya, and J. Tsujii (2003). Probabilistic modeling of argument structures including non-local dependencies. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP)*, pp. 285–291.

Musillo, G. and P. Merlo (2005). Lexical and structural biases for function parsing. In *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT)*, pp. 83–92.

Nakagawa, T. (2007). Multilingual dependency parsing using global features. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pp. 952–956.

Ney, H. (1991). Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing 39*, 336–340.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 149–160.

Nivre, J. (2004). Inductive dependency parsing. Technical report, School of Mathematics and Systems Engineering (MSI).

Nivre, J. (2006). *Inductive Dependency Parsing*. Springer.

Nivre, J. (2007). Incremental non-projective dependency parsing. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pp. 396–403.

Nivre, J. and J. Hall (2005). MaltParser: A language-independent system for data-driven dependency parsing. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 137–148.

Nivre, J., J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret (2007). The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pp. 915–932.

Nivre, J., J. Hall, and J. Nilsson (2004). Memory-based dependency parsing. In H. T. Ng and E. Riloff (Eds.), *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL)*, pp. 49–56.

Nivre, J., J. Hall, and J. Nilsson (2006). MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC)*, pp. 2216–2219.

Nivre, J., J. Hall, J. Nilsson, A. Chanev, G. Eryiğit, S. Kübler, S. Marinov, and E. Marsi (2007). MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering 13*(2), 95–135.

Nivre, J., J. Hall, J. Nilsson, G. Eryiğit, and S. Marinov (2006). Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pp. 221–225.

Nivre, J. and R. McDonald (2008). Integrating graph-based and transition-based dependency parsers. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 950–958.

Nivre, J. and J. Nilsson (2005). Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 99–106.

Nivre, J., J. Nilsson, and J. Hall (2006). Talbanken05: A Swedish treebank with phrase structure and dependency annotation. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC)*, pp. 1392–1395.

Nivre, J. and M. Scholz (2004). Deterministic dependency parsing of English text. In *Proceedings of the 20st International Conference on Computational Linguistics (COLING-2004)*, pp. 64–70.

Oren, M., C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio (1997). Pedestrian detection using wavelet templates. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, pp. 193–199.

Osuna, E., R. Freund, and F. Girosi (1997). Training support vector machines: An application to face detection. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, pp. 130–136.

Øvrelid, L. (2008). Linguistic features in data-driven dependency parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pp. 25–32.

Petrov, S. and D. Klein (2008). Parsing German with latent variable grammars. In *Proceedings of the ACL-08: HLT Workshop on Parsing German (PaGe)*, pp. 33–39.

Plaehn, O. (2005). Computing the most probable parse for a discontinuous phrase structure grammar. In H. Bunt (Ed.), *New Technologies in Parsing Technology*, pp. 283–298. Kluwer Academic Publishers.

Pollard, C. J. and I. A. Sag (1994). *Head-Driven Phrase Structure Grammar.* CSLI Publications, University of Chicago Press.

Rafferty, A. and C. D. Manning (2008). Parsing three German treebanks: Lexicalized and unlexicalized baselines. In *Proceedings of the ACL-08: HLT Workshop on Parsing German (PaGe)*, pp. 40–46.

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1–10.

Riezler, S., T. H. King, R. M. Kaplan, R. Crouch, J. T. M. III, and M. Johnson (2002). Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 271–278.

Sagae, K. and A. Lavie (2005). A classifier-based parser with linear runtime complexity. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pp. 125–132.

Sagae, K. and A. Lavie (2006). Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference and the Annual Meeting of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pp. 129–132.

Samuelsson, Y., O. Täckström, S. Velupillai, J. Eklund, M. Fishel, and M. Saers (2008). Mixing and blending syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pp. 248–252.

Skut, W., B. Krenn, T. Brants, and H. Uszkoreit (1997). An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)*, pp. 314–321.

Steedman, M. (2000). *The Syntactic Process.* The MIT Press.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics 21*(2), 165–201.

Telljohann, H., E. W. Hinrichs, S. Kübler, and H. Zinsmeister (2005). Stylebook for the Tübingen Treebank of written German (TüBa-D/Z). Seminar für Sprachwissenschaft, Universität Tübingen, Germany.

Tesnière, L. (1959). *Éléments de syntaxe structurale.* Editions Klincksieck.

Titov, I. and J. Henderson (2007). Fast and robust multilingual dependency parsing with a generative latent variable model. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pp. 947–951.

Toutanova, K., C. D. Manning, S. M. Shieber, D. Flickinger, and S. Oepen (2002). Parse disambiguation for a rich HPSG grammar. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 253–263.

Vapnik, V. (1979). Estimation of dependences based on empirical data. Technical report, Nauka, Moscow.

Vapnik, V. (1998). *Statistical Learning Theory.* John Wiley and Sons, New York.

Vural, V. and J. G. Dy (2004). A hierarchical method for multi-class support vector machines. *ACM International Conference Proceeding Series 69*, 105–113.

Yamada, H. and Y. Matsumoto (2003). Statistical dependency analysis with support vector machines. In G. van Noord (Ed.), *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pp. 195–206.

Younger, D. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control 10*, 189–208.

Zeman, D. and Z. Žabokrtský (2005). Improving parsing accuracy by combining diverse dependency parsers. In *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT)*, pp. 171–178.

Paper I

Memory-Based Dependency Parsing

Joakim Nivre
Växjö University

Johan Hall
Växjö University

Jens Nilsson
Växjö University

Published in *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL)*, Boston, Massachusetts, pp. 49–56, May 6–7, 2004.

Paper II

Discriminative Classifiers for Deterministic Dependency Parsing

Johan Hall
Växjö University

Joakim Nivre
Växjö University

Jens Nilsson
Växjö University

Published in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 2006), Main Conference Poster Sessions*, Sydney, Australia, pp. 316–323, July 17–21, 2006.

Paper III

Single Malt or Blended? A Study in Multilingual Parser Optimization

Johan Hall
Växjö University

Jens Nilsson
Växjö University

Joakim Nivre
Växjö University and Uppsala University

To appear in Harry Bunt, Paula Merlo and Joakim Nivre (eds.) *Trends in Parsing Technology*, Springer-Verlag

Paper IV

A Dependency-Driven Parser for German Dependency and Constituency Representations

Johan Hall
Växjö University

Joakim Nivre
Växjö University and Uppsala University

Published in *Proceedings of the ACL-08: HLT Workshop on Parsing German (PaGe-08)*, Columbus, Ohio, USA, pp. 47–54, 20 June, 2008.

Paper V

Parsing Discontinuous Phrase Structure with Grammatical Functions

Johan Hall
Växjö University

Joakim Nivre
Växjö University and Uppsala University

Published in *Proceedings of the 6th International Conference on Natural Language Processing (GoTAL 2008)*, LNAI 5221, Springer-Verlag, Gothenburg, Sweden, pp. 169–180, August 25–27, 2008.