



Jens Nilsson

Tree Transformations in Inductive Dependency Parsing

Licentiate Thesis

School of Mathematics and System Engineering

Reports from MSI

Tree Transformations in Inductive Dependency Parsing

Jens Nilsson

Tree Transformations in Inductive Dependency Parsing

Licentiate Thesis

Computer Science

2007



A thesis for the Degree of Licentiate of Philosophy in Computer Science at Växjö University.

Tree Transformations in Inductive Dependency Parsing
Jens Nilsson

Växjö University
School of Mathematics and System Engineering
SE-351 95 Växjö, Sweden
<http://www.msi.vxu.se/~rics>

© 2007 by Jens Nilsson. All rights reserved.

Reports from MSI, no. 07002
ISSN 1650-2647
ISRN VXU-MSI-DA-R--07002--SE

Abstract

This licentiate thesis deals with automatic syntactic analysis, or parsing, of natural languages. A parser constructs the syntactic analysis, which it learns by looking at correctly analyzed sentences, known as training data. The general topic concerns manipulations of the training data in order to improve the parsing accuracy.

Several studies using constituency-based theories for natural languages in such automatic and data-driven syntactic parsing have shown that training data, annotated according to a linguistic theory, often needs to be adapted in various ways in order to achieve an adequate, automatic analysis. A linguistically sound constituent structure is not necessarily well-suited for learning and parsing using existing data-driven methods. Modifications to the constituency-based trees in the training data, and corresponding modifications to the parser output, have successfully been applied to increase the parser accuracy. The topic of this thesis is to investigate whether similar modifications in the form of tree transformations to training data, annotated with dependency-based structures, can improve accuracy for data-driven dependency parsers. In order to do this, two types of tree transformations are in focus in this thesis.

The first one concerns non-projectivity. The full potential of dependency parsing can only be realized if non-projective constructions are allowed, which pose a problem for projective dependency parsers. On the other hand, non-projective parsers tend, among other things, to be slower. In order to maintain the benefits of projective parsing, a tree transformation technique to recover non-projectivity while using a projective parser is presented here.

The second type of transformation concerns linguistic phenomena that are possible but hard for a parser to learn, given a certain choice of dependency analysis. This study has concentrated on two such phenomena, coordination and verb groups, for which tree transformations are applied in order to improve parsing accuracy, in case the original structure does not coincide with a structure that is easy to learn.

Empirical evaluations are performed using treebank data from various languages, and using more than one dependency parser. The results show that the benefit of these tree transformations used in preprocessing and post-processing to a large extent is language, treebank and parser independent.

Key-words: Inductive Dependency Parsing, Dependency Structure, Tree Transformation, Non-projectivity, Coordination, Verb Group.

Acknowledgments

First of all I want to thank my supervisor, Joakim Nivre, for help and guidance in the experimental and writing phase of this licentiate thesis. Many thanks are directed to Johan Hall and Susanne Ekeklint, the other members of our small MALT group at MSI, Växjö University. It is mainly Joakim and Johan who have implemented MaltParser, the parser used in the majority of the conducted parsing experiments. The papers that this licentiate thesis is based on were created as a result of our good teamwork. This collaboration has undoubtedly generated both better research results and, in turn, higher quality of the papers, than what otherwise would have been the case. The other research colleagues in computer science also deserve my appreciation for creating a nice working atmosphere in “our” corridor at MSI. I am of course also grateful for the financial support of MSI for my research education.

My last thanks for support in life in general are reserved for my family, especially Hanna, for making life a more pleasant experience.

Contents

Abstract	v
Acknowledgments	vi
1 Introduction	1
1.1 Research Questions and Aims	1
1.1.1 Impossible Structures	4
1.1.2 Hard Structures	4
1.2 Outline	5
2 Background	6
2.1 Dependency Structure	7
2.1.1 Dependency Theories	7
2.1.2 Dependency Graphs	9
2.1.3 Projectivity	11
2.2 Coordination	13
2.2.1 The Head of Coordination	14
2.2.2 Linguistic Theories	15
2.3 Verb Groups	17
2.4 Parsing Natural Languages	19
2.4.1 Parsing using Grammars	19
2.4.2 Parsing using Corpus Data	20
2.4.3 Dependency Parsing using Corpus Data	22
2.5 Related Work	22
2.5.1 Constituency-Based Approaches	22
2.5.2 Dependency-Based Approaches	25
3 Tree Transformations	27
3.1 Pseudo-Projective Transformations	27
3.1.1 Transformation	28
3.1.2 Encodings	31
3.1.3 Inverse Transformation	33
3.2 Coordination Transformations	35
3.2.1 Transformation	36

Contents

3.2.2	Inverse Transformation	39
3.3	Verb Group Transformations	40
3.4	Transformation Independence	42
4	Experiments	43
4.1	Experimental Setup	44
4.1.1	The Treebanks	44
4.1.2	The Parsers	47
4.1.3	Evaluation	49
4.2	Experiment I: Treebank Transformations	50
4.2.1	Impossible Structures	51
4.2.2	Hard Structures	53
4.3	Experiment II: Treebank Independence	55
4.3.1	Impossible Structures	56
4.3.2	Hard Structures	58
4.3.3	Combining Transformations	60
4.3.4	Detailed Experiments	61
4.4	Experiment III: Parser Independence	68
4.4.1	Impossible Structures	68
4.4.2	Hard Structures	69
5	Conclusion	71
5.1	Main Results	71
5.2	Future Work	74
A	Feature Models and SVM-Parameter values	76
A.1	Settings for Experiment II	76

Chapter 1

Introduction

Several different systems for natural language processing can benefit from syntactic information. These include systems for machine translation, question answering and information extraction, where syntactic information is regarded as a useful step towards a meaning-bearing, semantic representation. Automatic syntactic analysis, or *parsing*, of natural language texts can be performed using various syntactic representations. Constituency-based representations have dominated the field of natural language processing for a long time, but the experiments presented in this licentiate thesis will deal with parsing of natural language texts using a *dependency*-based structure. This is a syntactic representation which has gained renewed attention during the last decade or so.

Context-free grammar is the formalism used by constituency-based parsers with state-of-the-art accuracy on unrestricted language texts, such as Collins (1999), Charniak (2000), Petrov et al. (2006). They are trained on syntactically annotated sentences and a major part of their success can be attributed to extensive manipulations of the training data as well as the output of the parser, such as various tree transformations. The general topic of this thesis is to investigate whether corresponding manipulations in the form of tree transformations of dependency-based structures can improve accuracy for dependency parsers.

1.1 Research Questions and Aims

This licentiate thesis will deal with fully *supervised* and *data-driven* dependency parsing. This means that the parser requires a collection of texts annotated with syntactic dependency structure (a *treebank*) as training data, where the annotation must comply with the formal framework of the parser. Figure 1.1 shows an example of a dependency structure, where the arrows, or arcs, represent the syntactic relations that hold between the words. The notion of dependency structure, however, is a rather weak concept, both in terms of the formal properties and the way individual linguistic phenomena

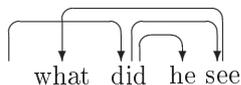


Figure 1.1: Dependency structure with crossing arcs.

are analyzed. These matters must be defined in order to create syntactic dependency parsers. The task of the parser is first to *learn* a parsing model for constructing dependency structure by looking at the training data and then use the model to construct dependency graphs for new unseen sentences. In case some of the dependency structures in the treebank do not comply with this formal framework, the parser will of course not be able to produce the correct structure. This will inevitably result in reduced parsing accuracy. One main part of this thesis is to investigate whether tree transformations partially can solve this problem.

Besides the formal framework, the parser can not make any assumptions about the structure of individual linguistic phenomena. It is simply forced to use nothing but the information in the treebank, which is the result of the design choices of the treebank annotators. These design choices are usually made on linguistic grounds, often without having automatic syntactic parsing in mind. This may make some linguistic constructions harder to parse for certain parsers.

These two aspects of syntactic dependency representations in relation to accurate syntactic dependency parsing lead to the following two research questions:

- Which linguistic dependency constructions are *impossible* to parse for certain data-driven dependency parsers, and what can be done to improve parsing accuracy by taking special care of the impossible constructions?
- Which linguistic dependency constructions are *hard* to parse for certain data-driven dependency parsers, and what can be done to improve parsing accuracy by taking special care of the hard constructions?

Dependency structures can be defined using graph theory, and the means for improving accuracy in this licentiate thesis is by applying various graph transformation techniques. More precisely:

- In order to improve accuracy, is it possible to apply graph transformations on the training data as preprocessing in combination with other graph transformations as postprocessing on the parser output?

As already mentioned, the technique of preprocessing and postprocessing has been successfully applied and more thoroughly investigated within constituency-based parsing than within dependency-based parsing. These studies reveal that preprocessing the data and postprocessing the parser output is important in order to achieve state-of-the-art accuracy for methods using constituency-based parses, such as in probabilistic context-free parsing. In other words, a linguistically sound constituency-based representation is not necessarily a good representation for data-driven parsing. The question is whether some corresponding processing of the training data can facilitate the learning task in data-driven dependency parsing.

Many of the recently proposed supervised and data-driven dependency parsers are also language-independent. As long as the training data fulfills the formal properties required by the parser, the actual language in the treebank is not important. This means that if such a treebank exists, a parser can be constructed quite rapidly. With this in mind, the transformation techniques become even more useful the more language and treebank independent they are. Additional research questions are therefore:

- Can the graph transformations be created in a language independent way?
- Can the graph transformations be created in a treebank independent way?

Language and treebank independence is preferable, but not always possible. This categorization coincides rather naturally with the distinction between structures that are impossible and hard to parse; structures being impossible to parse for one language are impossible for any language given the same formal framework, and structures being hard to parse for one treebank tend to be hard to parse for other treebanks using the same treebank annotation. Another question that is investigated is:

- Can the graph transformations be created in a parser independent way?

Different parsers with different characteristics may react differently to the graph transformations. This is an issue that will be pursued here as well.

It is important to mention that the notion of independence in the research questions above can be interpreted in at least two ways. The first and most important notion is whether any positive effect of a transformation in terms of accuracy is independent of the language, treebank and parser. The second is whether a transformation itself is language and treebank independent. In other words, must certain properties of the language or treebank be met or can they be applied to any type of language and treebank? Only if

a transformation improves accuracy, the second type of independence is of interest.

1.1.1 Impossible Structures

The dependency constructions that are considered impossible to parse in this thesis are the *non-projective* ones. Non-projectivity is formally defined in section 3.1, but a non-projective dependency structure can informally be said to contain crossing dependency relations. The dependency structure in figure 1.1 is non-projective; the arc from the root to *did* crosses the arc from *see* to *what*. The amount of non-projectivity is highly dependent on the properties of the language and on the linguistic theory. Somewhat simplified one can say that whereas most existing efficient dependency parsers are only able to create projective graphs, most linguistic dependency theories do not assume projectivity. As the vast majority of linguistic phenomena usually do not require non-projectivity, using a projective parser is sometimes an adequate approximation. However, it is theoretically unsatisfactory to know that a completely correct analysis is impossible to create for any sentence containing non-projectivity.

The issue pursued, as one part of this thesis, is whether various graph transformations can make it possible to parse non-projective constructions, while maintaining the possible advantages of projective parsers, such as efficient parsing and high accuracy for projective constructions. As stated above, we will investigate to what extent these transformations can be made independent of the language, treebank and parser as possible.

1.1.2 Hard Structures

It is difficult to define exactly what it means for a structure to be hard to parse. This may depend on several factors, such as the particular parser used, the properties of the language and the peculiarities of the treebank. There are numerous interesting linguistic phenomena to investigate. In this thesis, the focus will be on two linguistic phenomena for which the disagreement of representation in dependency theories is high, namely coordination and verb groups. For instance, in a sentence like *John and Mary must go*, the words *John and Mary* constitute a coordination and *must go* a verb group. Essentially, the relationship between the conjunction (*and*) and the conjuncts (*John, Mary*), and the relationship between the auxiliary verb (*must*) and the main verb (*go*) will be investigated here. Coordination is a linguistic phenomenon that tends to be especially hard to parse. Choosing an appropriate base representation for parsing is therefore an important task for

constituency-based parsing, which Johnson (1998) was one of the first to discover, confirmed by several subsequent studies. So the other main part of this thesis investigates whether graph transformations can increase accuracy if a well-suited base representation does not happen to coincide with the original representation of the treebank. This raises the question of how good (or bad) base representations for parsing can be identified, which is something that will be discussed in coming chapters as well.

Given that some dependency treebanks already are quite well-suited for syntactic parsing with respect to coordination and verb groups, it does not make sense to apply transformations to such treebanks. This makes these transformations less treebank-independent, but it does not exclude the possibility that they will still be language-independent. The same or very similar transformations will be applied to different treebanks for different languages adopting the same solution for coordination and verb groups.

1.2 Outline

A review of previous research related to this thesis is presented in chapter 2. The chapter starts with a presentation of some linguistic theories, especially those concerning dependency structure, and compare them with respect to how they deal with non-projectivity, coordination and verb groups. It continues with a description of parsing of natural languages in general and manipulation of training data in data-driven parsing in particular. Some additional related work ends this chapter, containing a discussion about transformations of tree structures in syntactic parsing in general.

Chapter 3 introduces the graph transformations, which are the core of this thesis. It presents the transformations for non-projectivity, coordination and verb groups, with one section for each phenomenon. For some of the phenomena, more than one transformation version have been created. They are described in detail in this chapter, together with the corresponding inverse transformations.

In chapter 4, the focus is to present the empirical experiments, including results and discussion. The goal of these experiments is to answer the research questions stated above by applying the transformations of chapter 3 on treebank data, with and without involving a parser. This is preceded by a section introducing the data resources, evaluation metrics, the used parsers and the general methodology of the conducted experiments.

This thesis ends with conclusions and some possible directions for future research in chapter 5.

Chapter 2

Background

The term *parsing* in the context of natural language processing usually refers to the task of assigning a syntactic representation to a sentence (or utterance). This rather loose description leaves room for various interpretations, where some are incompatible. One aspect is what is meant by a syntactic representation. As mentioned in the introduction, there are a large number of syntactic theories, and a broad distinction is usually made between constituency-based and dependency-based representations. Another aspect relates to the distinction between parsing using a grammar and parsing using a corpus, with or without syntactic annotation. This depends on the formal definition of parsing.

Parsing using a grammar is tied to the problem of determining whether a sentence is a member of the language defined by a formal grammar. This is a matter of recognizing the (possibly infinite) set of admissible sentences, and any algorithm constructing syntactic representations using a grammar also solves the recognition problem. Its well-defined properties have made it very useful not only in natural language processing, but especially in computer science where the grammar can be constrained in order to conduct efficient parsing.

In contrast to for example grammars for programming languages, grammars for natural languages are ambiguous. Usually only the correct parse tree is needed. Disambiguation must then be performed in all cases where a grammar produces more than one parse tree for a sentence. Parsing using a corpus defines parsing as the problem of assigning the correct syntactic representation to any sentence of *unrestricted* natural language text. This notion assumes that any sentence has exactly one syntactic representation, whereas parsing using a grammar can assign zero, one or many syntactic representations to the same sentence.

Disambiguation can be achieved by combining parsing using a grammar with corpus data. By ranking the admissible parse trees, the top-ranked parse tree can then be chosen. However, it can also be performed without any grammar at all by using the linguistic information directly. As already

mentioned, the experiments in chapter 4 are concerned with parsing using syntactically annotated corpora more directly.

The focus of this thesis will be on dependency-based representations, which are the topic of section 2.1. We will formally define the notion of dependency structure used throughout this thesis, including the notion of projectivity. Sections 2.2 and 2.3 discuss the linguistic phenomena coordination and verb groups, mainly in dependency-based structure but also in constituency-based structure. The main focus of the two final sections is on parsing natural languages, where section 2.4 talks about transformations in parsing natural languages in general and section 2.5 concentrates on transformations related to non-projectivity, coordination and verb groups.

2.1 *Dependency Structure*

The first modern theory of dependency structure as a means for syntactic representation was created by Tesnière (1959). It is almost contemporary with the corresponding formalization of constituent structure by Chomsky (1956) in the notion of phrase-structure grammar, or context-free grammar. While the notion of constituency can not be traced further back than to the beginning of the twentieth century (Percival 1976), the notion of dependency has been independently accepted several times in the grammatical history, and as early as two millenniums ago (McCawley 1973).

2.1.1 *Dependency Theories*

Since Tesnière (1959), a large number of grammatical theories based on the notion of dependency have evolved in various directions, but they share a core of common concepts and assumptions. The most basic notion is that syntactic structure consists of *binary* and *asymmetrical* relations between *lexical* elements, called *dependencies*. The asymmetrical property naturally creates a hierarchy, where one of the lexical elements acts as *head* (or *governor*) and the other as *dependent* (or *modifier*). The most obvious difference compared to constituent structure is the lack of phrasal nodes.

An important part of the creation of a dependency theory is to establish criteria for imposing dependencies between the lexical elements. A first broad distinction is often drawn between syntactic and semantic criteria. Other criteria, such as morphological ones, have also been proposed. Zwicky (1985), for instance, points out that the notion of dependency easily can be extended into the area of morphology by linking lexemes using dependency relations. Fraser (1994) enumerates the following four criteria:

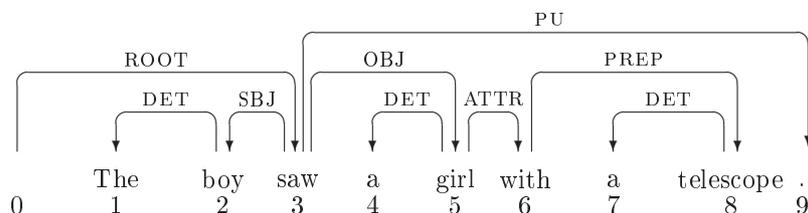


Figure 2.1: Dependency structure for English sentence

1. A head determines whether a dependent is optional or obligatory (as a modifier of the head), and not vice versa.
2. A head subcategorizes for a dependent, and not vice versa.
3. The head determines which inflectional form of a dependent occurs, and not vice versa.
4. The head identifies a meaningful object that the meaning of the dependent contributes to (specifies further), and not vice versa.

Zwicky has a similar list which is partly overlapping. Figure 2.1 is a dependency graph for an English sentence, where these criteria are applicable. The arc from the noun *boy* to the determiner *the* indicates that the noun is the head and that the determiner is a dependent. Criterion (2) is not applicable to this relation, but it is supported by (1) and (4), and in a sense also (3), (cf. *a banana - an apple*, and *an apple - many apples*).

It is however important to note that no single set of criteria provides necessary or sufficient conditions for establishing dependency relations in all cases. It is also important to keep in mind that different criteria may suggest different solutions. The arc from the verb *saw* to the noun *boy* indicates that the verb is the head and that the noun is a dependent. The label on the arc further specifies that the noun is a subject of the verb. This is in line with (2), and most syntactic theories say that the verb opens “slots” (subcategorizes), where one slot is filled by a subject. However, it has also been suggested that it is more natural to regard the subject as the entity controlling the person inflection of the verb in English, which instead according to (3) would make *boy* the head and *saw* the dependent.

Mel’čuk (1988), who advocates the multi-stratal Meaning-Text Theory (MTT), emphasizes more than Zwicky and Fraser the importance of distinguishing syntactic criteria from semantic and morphological criteria, and argues that dependency syntax first and foremost should be controlled by

syntactic criteria. MTT has three layers of dependency annotation: a morphological, a syntactic and a deep syntactic/shallow semantic. The theoretical framework Functional Generative Description (FGD) (Sgall et al. 1986) is also a multi-stratal theory with three layers, where the annotation for several phenomena, such as coordination and verb groups, on the analytical (syntactic) layer in FGD follows semantic rather than syntactic criteria. This is an issue that will be discussed further in sections 2.2 and 2.3.

In order to find the head in a group of words having a relation to each other, Nikula (1986) suggests that it is important to find out whether the construction is *endocentric* or *exocentric*. Bloomfield (1933) defines an endocentric construction as one whose distribution is identical with one or more of its constituents. That is, one of the words can readily replace the whole construction without unpredictable changes in meaning. The prototypical endocentric case is the noun in a noun phrase, where adjectives, usually located to the left of nouns in languages such as English, can be present or not without affecting the grammaticality or the semantics in an unforeseeable way. For instance, an adjective such as *happy* can readily be inserted between *the* and *boy*. An exocentric construction is the opposite to an endocentric construction, where no entity can replace the whole group of words. The mutual dependence between *saw* (the predicate) and its subject is such a case, as both need each other to compose a grammatical sentence.

How syntactic and semantic criteria, and endocentricity and exocentricity, relate to the linguistic phenomena studied here is further discussed in section 2.2 and 2.3. The following section will formally define the rather constrained form of dependency structure that the parsers conform to.

2.1.2 Dependency Graphs

A dependency graph will here be formally defined using set theory. Every sentence has been segmented into n words or tokens, which are represented by the integers $1, \dots, n$. For convenience, a special root token is located to the left, denoted 0. The dependency graph is *directed* and *labeled* with dependency types $r_0, r_1, \dots, r_m \in R$, such as SBJ, OBJ and ATTR for representing subject, objects and attributes. It is formally a triple $G = (V, E, L)$, defined as in both Nivre (2006) and Hall (2006):

Definition 2.1. $G = (V, E, L)$ is a labeled dependency graph for a sentence $x = (w_1, \dots, w_n)$, if:

1. $V = \mathbf{Z}_{n+1} = \{0, 1, 2, \dots, n\}$
2. $E \subseteq V \times V$

3. $L : E \rightarrow R$

The tokens of a sentence are assigned an integer according to the order in which they appear in the sentence, which are the integers of V including the special root token 0. The set E contains the arcs for the graph connecting two words in V . Each arc in E is assigned one dependency type (arc label) in R using the function L . To simplify the notation, a directed arc $(i, j) \in E$ will be denoted $i \rightarrow j$, and the corresponding arc labeled r will be denoted $i \xrightarrow{r} j$. For instance, the arc from *saw* to *boy* in figure 2.1 is represented by $(3, 2)$ or $3 \xrightarrow{\text{SBJ}} 2$, or even more conveniently by $\text{saw} \xrightarrow{\text{SBJ}} \text{boy}$ when the words are unambiguous. Moreover, the notation for the reflexive and transitive closure of an arc relation in E is $i \rightarrow^* j$, that is, this relation holds if and only if there is a path of zero or more arcs from i to j . Token i is said to *dominate* j .

Despite this fairly unrestricted definition, a number of dependency theories fail to comply with this notion of dependency graph. The dependency description of Tesnière (1959), for example, distinguishes between three different type of syntactic relations (connection, junction, transfer), where only connection corresponds to directed and lexical dependency. The definition also excludes any dependency relations that are not binary, something which can be useful in relations that involve more than two words. This holds in different ways for both coordination and verb groups. However, most such constructions tend to decompose quite naturally into binary relations, as noted by Mel'čuk (1988).

The data resources used later on and the parsers follow definition 2.1. That definition together with the definition below are sufficient constraints for the data resources, but not for the parser.

Definition 2.2. A dependency graph G is a *dependency tree* if and only if:

1. **There should be one independent element:** there is no arc $i \rightarrow j$, $i \in V$, such that $j = 0$
2. **The dependency structures must be connected:** for all nodes $i, j \in V$, there is an arc $i \rightarrow^* j$ or $j \rightarrow^* i$
3. **Every dependent must have at most one head:** if $i \rightarrow j$, then there is no arc $k \rightarrow j$ and $k \neq i$

Constraint (1) states that there should be one token without a head token, which is the special token with index 0. This special token is the *root* of the dependency graph. According to (2), there is a path between any two tokens

2.1. Dependency Structure

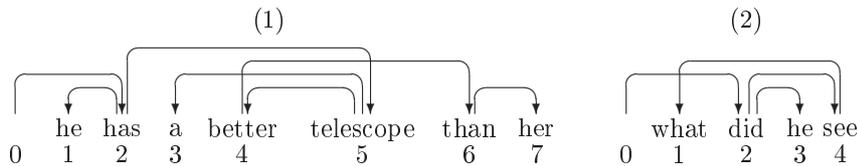


Figure 2.2: Non-projective dependency structures for English sentences.

in any dependency graph, while (3) makes sure that no token can depend on more than one other token.

The three constraints of definition 2.2 entail another important property of the type of dependency graphs that are allowed. A dependency graph is acyclic, that is, if $i \rightarrow^* j$, then there is no arc $j \rightarrow i$. Constraints (2) and (3) are not sufficient to induce acyclic graphs, but the independence of node 0, according to (1), breaks all possibilities of forming a cycle. In other words, these three transformations entail that the graph fulfills the requirements of a rooted tree in graph theoretic terms, and will be referred to as a *dependency tree*. The constraints hold for figure 2.1 which is consequently a dependency tree.

Some well-known theories based on the notion of dependency do not subscribe to definition 2.2. The EUROTRA system (Johnson et al. 1985) and Word Grammar (Hudson 1984; Hudson 1990) allow dependents to have multiple heads, which is mainly for semantic reasons such as subject sharing in *John likes walking*. In addition, Word Grammar allows cycles, which also violates definition 2.2.

2.1.3 Projectivity

An interesting constraint that can be imposed on dependency structures is *projectivity*. The notion of projectivity was discussed first by Lecerf (1960) and later also by others (e.g. Hays 1964; Solomon 1965; Robinson. 1970), and it has been defined in various ways that are not always equivalent. Very informally, non-projectivity is caused by (a) any pair of arcs that cross each other or (b) any arc that covers the root token of a sentence (if the arcs are drawn above the sentence). The dependency graphs in figure 2.2 are non-projective, where (1) and (2) contain crossing arcs. Dependency graph (2) exemplifies that condition (b) is no longer needed when there is an artificial root node located to left of the sentence. The word *did* would without it be the root, covered by the arc $see \rightarrow what$.

The corresponding notion in a constituency-based representation is a discontinuous construction. One of the often mentioned advantages of dependency structure is that non-projectivity can be explicitly and naturally described, compared to discontinuous constructions in constituency structure. Context-free grammar is unable to handle discontinuous constructions, which therefore require special treatment. Another way to deal with these kinds of phenomena is to use Slash features, proposed by GPSG (Gazdar et al. 1985) and HPSG (Pollard and Sag 1994). However, there is a discrepancy between, on the one hand dependency based syntactic theories, and on the other practical parsing systems. Any sensible dependency based syntactic theory recognizes that projectivity (and continuous constructions) can cover the vast majority of dependency structures, but realizes that there is a need for non-projectivity in certain situations. Practical parsing systems, however, tend to assume projectivity for at least two reasons: efficiency and accuracy. A projective parser is often more time-efficient, and the errors on the small number of non-projective constructions can often be neglected since accuracy for projective constructions in projective parsing is often higher than in non-projective parsing. This discrepancy is nevertheless unsatisfactory.

Word order is another issue that is tied to the notion of projectivity, since languages with freer word order tend to have a higher proportion of non-projectivity than languages with a more static word order. English is a language with a relatively static word order, at least compared to for instance Slavic languages, for which dependency-based representations have been more influential. In MTT, there is no order among the words in the syntactic layer, and any syntactic information encoded using the linear order of the words is instead captured by additional annotation associated with the tokens or dependency labels. On the other hand, such additional annotation is not used for FGD, as it preserves the word order. It is worth noting that at the tectogrammatical (the shallow semantic layer) in FGD all graphs are projective, even though both FGD and MTT allow non-projectivity at the syntactic/analytical layer.

As mentioned above, projectivity has been defined in different ways, and the one that will be adopted here is a reformulated version of the definition of Kahane et al. (1998), where projectivity is a property of individual arcs:

Definition 2.3. A dependency arc $i \rightarrow k \in E$ is *projective* if and only if for every token j occurring between i and k in the string ($i < j < k$ or $i > j > k$), $i \rightarrow^* j$.

According to this definition, all arcs in (1) of figure 2.2 are projective except $better \rightarrow than$, since *telescope*, which is located between *better* and *than*, is not dominated by *better*. It is in fact the other way around, i.e. $telescope \rightarrow^*$

better. The arc $see \rightarrow What$ in (2) is also non-projective, in this case since neither *did* nor *he* depends transitively on *see*.

Definition 2.3 can be used to define the projectivity of a dependency tree:

Definition 2.4. A dependency tree $G = (V, E, L)$ is *projective* if and only if all arcs in E are projective.

This definition entails that the dependency tree of figure 2.1 is projective, whereas (1) and (2) in figure 2.2 are non-projective.

2.2 Coordination

Although dependency structure has many advantages, some dependency-based theories can be criticized for not being expressive enough for some kinds of linguistic phenomena. Coordination and apposition,¹ and possibly also multi-word expressions,² pose a problem for the notion of dependency as such. This section will discuss how coordination is handled using dependency structure, especially given the previous definition of a dependency tree.

Coordination is a linguistic phenomenon that is troublesome for most linguistic theories, including dependency-based ones. According to Mel'čuk (1988), the criticism of dependency formalisms can be divided into three major groups:

1. **Double dependency:** a word form can depend simultaneously on two different word forms.
2. **Mutual dependency:** two word forms depend simultaneously on each other.
3. **No dependency:** some constructions have no syntactic head, e.g. coordination where there is no dependency between the conjoined items.

Points (1) and (3) are relevant for coordination, as will be discussed below. Verb groups can be said to have a mutual dependency (2), which is discussed in section 2.3. However, the criticism is not restricted to coordination and verb groups but applies to several linguistic phenomena. For instance, subject sharing and the relation between the main verb and the subject are examples of (1) and (2), respectively.

¹In *The pope John Paul II passed away*, it is hard to determine whether *The pope* is the head of *John Paul* or vice versa, as either one is optional when the other is present and either one further specifies the other (Fraser's first and fourth criteria).

²E.g., what are the internal relationships between *John*, *Paul*, and *II*?

2.2.1 The Head of Coordination

The first important issue is to determine what is the head word in a coordination construction. For a sentence like:

skilled Swedes and Danes played football

there are three proposals for *Swedes and Danes* using dependency structure, but all of them have flaws:

- One of the individual conjuncts (*Swedes* or *Danes*).
- Both the individual conjuncts (*Swedes* and *Danes*).
- The conjunction (*and*).

The first one is inadequate because one can claim that none of the nouns have higher priority than the other. Interchanging nouns (i.e. *skilled Danes and Swedes played football*) does not alter the interpretation (if *skilled* modifies both nouns), and letting one of the nouns reside lower down in the hierarchy than the other therefore seems wrong.

Bloomfield (1933) characterizes coordination as an endocentric construction due to the fact that any of the conjuncts can replace the whole coordination. Assigning both conjuncts the same hierarchical priority is supported by the second and third proposals. However, both are problematic whenever the conjuncts have a mutual dependent. The word *Skilled* can semantically depend on both *Swedes* and *Danes*, which is incompatible with the definition of a dependency tree because it would have more than one head. This is in accordance with criticism (1). In contrast to the third, the second proposal keeps the same direct relation between each conjunct and its head which would be present when the coordination instead is occupied by the conjunct alone. This holds for one of the individual conjuncts for the first proposal as well. A conjunction “and” can never by itself be the subject, like *Swedes* and *Danes*, and is e.g. not inflected according to the head of the subject (cf. Fraser’s third criterion), as conjunctions never inflect.

On the other hand, having the conjunction as the head has the advantage that the conjunction semantically acts as a *functor* and the conjuncts as its arguments (e.g. $AND(Swedes, Danes)$). In terms of dependency relations, it is then most natural to regard the functor as the head and its arguments as dependents, which is in line with the third proposal.

It is worth pointing out the complexity that coordination gives rise to for any linguistic theory, but especially for those based on dependency. Few full-fledge dependency-based theories are content with the rather simple definition of dependency trees in section 2.1 and 2.2, since it is hard to capture several

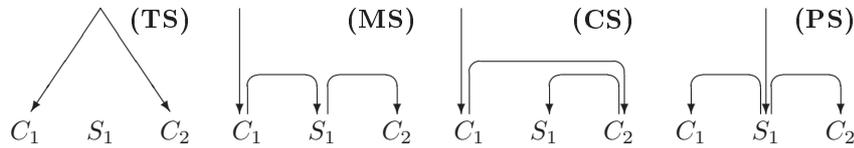


Figure 2.3: Dependency structure for coordination

types of important distinctions. One of the most basic distinctions is exemplified above, where it is important to know whether it was skilled Swedes and skilled Danes who played football, or whether it was skilled Swedes and all Danes (skilled or not) who played football. This distinction, i.e. resolving semantic scope ambiguity, will be discussed further in conjunction with the transformations presented in chapter 3.

2.2.2 Linguistic Theories

Tesnière (1959), who distinguishes between three types of syntactic relations, suggests that coordination internally is in fact not describable using ordinary dependency relations. Both conjuncts have a direct ordinary dependency relation to the same head, but the internal structure is sustained by a special junction-relation. Hudson's Word Grammar (Hudson 1990) adopts a similar analysis connecting all conjuncts to the same head. The conjunction plays no part at all in the dependency structure. The whole coordination is in a sense treated as an atomic string or group in the sentence. The leftmost picture in figure 2.3 depicts coordination in accordance with Tesnière and Hudson, denoted Tesnière-style (TS), where C_i is a conjunct and S_j a conjunction (or comma). This analysis corresponds to the second candidate above.

MTT of Mel'čuk (1988) adheres to the first proposal above, where one of the individual conjuncts is in fact the head. He argues that the apparent symmetry of coordination holds only at the semantic level and only for pure logical uses of coordination, which is a minority of all cases. In most cases, the order of the conjuncts can not be reversed without changing the interpretation:

the Swedes played football and won \neq *the Swedes won and played football*

In MTT, the leftmost conjunct is the head, which governs the succeeding group with one conjunction and one conjunct. The motivation is that the group usually can be omitted, but not the leftmost conjunct. Mel'čuk further argues that the conjunction is the head of the succeeding conjunct for passive syntactic reasons; it is in a sense the conjunction that opens a slot for the

succeeding conjunct. MTT has a more complex handling of coordination than what is described here, but the core dependency structure for coordination can be constructed using definition 2.2. The structure is shown in the second picture in figure 2.3 and will be called Mel'čuk-style (**MS**).

The other possibility that follows from the first candidate besides MS is to let the second conjunct depend directly on the first conjunct, while making either the first or second conjunct head of the conjunction. This construction resembles the relations of binary lexical relations in coordination used internally in the data-driven phrase-structure parser of Collins (1999) (see section 2.4). Its dependency structure, where the conjunction depends on the second conjunct, is shown in the third picture in figure 2.3 and is called Collins-style (**CS**).

Whereas **MS** and **CS** are syntactically grounded, the rightmost picture in figure 2.3 can be motivated by more semantic arguments. This analysis is for example adopted by FDG in the analytical layer, which is based on a linguistic tradition known as the Prague school (Sgall et al. 1986) (**PS**). The **PS**-analysis is also proposed by Nikula (1986). It is worth noting that the picture is also only schematic and hides several details. One such detail is the theories' ability to distinguish between *collective* and *disjunctive* (or *distributive*) readings of a sentence containing for example two coordinated nouns. This issue is related to the situation discussed above, i.e. whether *Skilled* modifies both succeeding nouns or only the first. For instance, did the Swedes and Danes play together (collective reading), or did the Swedes play by themselves and the Danes by themselves (disjunctive reading). This insufficiency of dependency structure could be solved in more than way, e.g. by encoding such information using the dependency labels, or by using a notion of *grouping* not too dissimilar to phrases (Mel'čuk 1988). This is however out of the scope of this thesis.

A question that arises from these various dependency representations of coordination is how they relate to a constituency-based representation. The comparison becomes more interesting when more than two coordinated conjuncts are considered, such as in *Skilled Swedes, Danes and Norwegians played football*. The standard approach when transforming dependency structure to (unlabeled) phrase structure is to create one non-terminal for each lexical token having dependents comprising all of them, and one terminal for every token (Gaifman 1965).³ **TS** corresponds to the flattest possible phrase structure with no phrase at all, i.e. $C_1 S_1 C_2 \underline{S_2} C_3$, where underscore marks the head terminal. **PS** has a similar phrase representation, with only one phrase (i.e. $[C_1 S_1 C_2 \underline{S_2} C_3]$). In the early days of transformational grammar, a

³E.g. converting figure 2.1 results in: $[[The \underline{boy}] \underline{saw} [a \underline{girl} [\underline{with} [a \underline{telescope}]]]]$

flat structure was popular (Johnson 1998), and **PS** yields a phrase structure similar to coordination in the Penn Treebank (Marcus et al. 1993).

The phrase tree for **MS** (i.e. [$\underline{C_1}$ [$\underline{S_1}$ [$\underline{C_2}$ [$\underline{C_2}$ C_3]]]]) is quite different from those of **TS** and **PS**, but more similar to that of **CS** (i.e. [$\underline{C_1}$ [$\underline{S_1}$ $\underline{C_2}$ [$\underline{S_2}$ $\underline{C_3}$]]]). Both are essentially asymmetric right-branching trees, where **MS** is purely binary. The trees contain more non-terminals and are therefore deeper. Moreover, a linguistic and grammatical motivation for using the phrase structure of either **MS** or **CS**, as opposed to **TS** and **PS** is that the set of possible grammar rules for coordination are in principle infinite for the two latter, which is not the case for the first two. In terms of dependency it corresponds to the situation where for instance a conjunction in **PS** may have arbitrary many dependents that are involved in the coordination. In **MS** and **CS**, on the other hand, the number of dependents involved in the coordination is for any token constrained to at most one for **MS**, and at most two for **CS**.

2.3 Verb Groups

The relation between the verbs in a clause containing more than one verb is in general less problematic than coordination for linguistic theories. However, the best way to represent it is still not completely settled, neither in constituency-based theories nor in dependency-based ones. The problem is also how the verbs relate to various complements and adjuncts in the clause, which is affected by the internal relation between the verbs.

Tesnière (1959) proposes that the relation between the auxiliary verb and the main verb is not a dependency relation, but instead they form a *nucleus*. All their dependents (e.g. subjects, objects and adverbials) attach to the nucleus and not directly to the word. Although this approach can be attractive for other reasons, it does not conform to definition 2.2 of a dependency tree. Moreover, in languages such as English and Swedish, it is possible to insert certain complements between the verbs (e.g. *did they play football?*), resulting in discontinuous nuclei.

Verb groups are most naturally described as exocentric constructions, since the presence of a non-finite verb without an auxiliary verb normally yields an ungrammatical sentence, as well as vice versa. However, the internal relation is still open. If one has settled that there should be a dependency relation between the verbs, there are essentially only two possibilities. First, the main verb can be the dependent of the auxiliary verb, where the latter consequently is the head of the whole clause. This analysis is advocated by Mel'čuk (1988), mainly for syntactic reasons. Most grammatically correct sentences have a finite verb whereas a non-finite verb is optional, which ac-



Figure 2.4: Dependency structure for verb groups

According to Fraser's first criterion (section 2.1) strengthens this point of view. The third criterion also favors this analysis, since the possible presence of an auxiliary verb in English controls the inflection of the main verb. The left picture of figure 2.4 illustrates this dependency analysis, denoted **MS**. Mel'čuk further suggests that all complements and adjuncts to the left and to the right attach to the auxiliary verb and main verb, respectively, but other sound solutions exist.

The right picture in the same figure shows the second possibility, which instead has the main verb as the head of the clause with a direct governing relation to the auxiliary verb. This analysis is more easily motivated by semantic criteria, where the auxiliary verb can be regarded as a modifier of the main verb, somewhat in line with Fraser's fourth criterion. Moreover, it is the main verb that determines the number of complements through its property of carrying *valency*. It subcategorizes and is in this perspective therefore in control of all complements, in compliance with the second criterion of Fraser. This approach is adopted by for instance FGD, according to the Prague school (**PS**).

It is worth noting that FGD does not admit auxiliary verbs to have dependents at all, although some linguists point out that auxiliary verbs can have their own adjuncts. Another option in **PS** is to let the subject (usually located to the left) depend on the auxiliary verb. This solution is one of many suggestions by Nikula (1986), even though he mainly supports **MS**.

As a main verb can be accompanied by more than one auxiliary verb, one may ask how they relate to each other and to the main verb. The most common solution is to form a chain of arcs between the verbs when an auxiliary verb is head ($AUXV_1 \rightarrow AUXV_2, AUXV_2 \rightarrow MAINV$), and the main verb as a direct head of all auxiliary verbs in **PS** ($MAINV \rightarrow AUXV_1, MAINV \rightarrow AUXV_2$).

The different treatments of verb groups in **MS** and **PS** has an analogous difference in constituent structure, especially for the treatment of multiple auxiliary verbs (e.g. *They have been playing football*). It is also related to the corresponding discussion for coordination. It is notable that the phrase

structure for **MS**, where each verb in a clause depends on the closest verb to the left (i.e. [AUXV [AUXV [MAINV ...]]]), is right branching, similar to the binary right branching of coordination in **MS**. Also, the flatness of **PS** for coordination is maintained for verb groups, where all auxiliary verb are dependents of the main verb (i.e. [AUXV AUXV MAINV ...]). While the number of grammar rules in **MS** can be quite low, the number of rules in **PS** is only limited by the number of possible auxiliary verb preceding main verbs. This is similar to the situation for coordination, even though the number of rules for verb groups cannot be infinite.

2.4 Parsing Natural Languages

In relation to the three previous sections, the focus will here both shift, from syntactic structure to the task of automatically deriving syntactic structure, and widen, in the sense that not only dependency-based approaches are discussed but also constituency-based ones. As already mentioned, the experiments deals with parsing including disambiguation. This approach to this problem has been further divided by Nivre (2006) into two broad types, *grammar-driven* and *data-driven* parsing. Carroll (2000) also makes this distinction, although with a slightly different meaning.

2.4.1 Parsing using Grammars

Grammar-driven parsing tries to model a formal language $L(G)$ through a grammar G . Parsing algorithms using a grammar G should be constructed so that they terminate and satisfy the requirements *soundness* and *completeness*, that is, each and every syntactic tree that eventually will be derived is correct according to G . Such algorithms have two important and intrinsic problems. The first one concerns robustness, and can be described as the problem of covering all sentences that in fact are part of a natural language. The language $L(G)$ can only be an approximation of a natural language, and it can be argued that this is only a practical problem since the grammar G can always be improved in order to cover the language better. However, extending a grammar often increases the probability of *leakage*, which happens when a grammar allows ungrammatical sentences. The second one concerns a grammar's property of assigning more than one analysis to the same sentence. The problem of choosing the right analysis is called *disambiguation*.

There is a long tradition in grammar driven parsing using frameworks such as GPSG (Gazdar et al. 1985) and HPSG (Pollard and Sag 1994), and frameworks known as mildly context-sensitive grammars (e.g. Tree Adjoining Grammar, Joshi and Schabes 1997). They are all offsprings of context-free

grammar, and thus originate in the constituency-based tradition. This holds for *probabilistic context-free grammar* (PCFG) too, an extension of context-free grammar with new properties to cope with some of the problems of parsing natural languages.

PCFG is a well-known approach within grammar-driven parsing, in which each rule has an associated probability (Charniak 1993). PCFG thus provides the possibility to rank all the permissible parse trees found by some standard parsing algorithm such as CYK or Earley’s algorithm (Jurafsky and Martin 2000). For each parse tree, the products of all probabilities for the applied grammar rules are computed. This in turn resolves the disambiguation problem by simply choosing the parse tree with the highest probability. Both the rules and their probabilities may be created by hand, or induced from real world data using a corpus, with or without the correct parse trees. Although the simplicity of PCFG is attractive, the “vanilla” PCFG have been shown to perform relatively poorly, and is heavily dependent on the *base representation* of the grammar (e.g. Johnson 1998; Klein and Manning 2003). Several of its apparent shortcomings can be compensated by applying tree-transformations on the parse trees as preprocessing and postprocessing, which will be discussed in section 2.5.

2.4.2 Parsing using Corpus Data

The transformations presented in chapter 3 are applied before and after the application of an inductive dependency parser, which does not rely on an induced or hand-crafted grammar. It is an instance of a purely data-driven parsing approach and does not make use of a formal language L to distinguish correct sentences from incorrect. Data-driven text parsing uses corpus data, as a grammar-driven approach can do in order to induce a grammar, but it does not make use of an intermediate grammar as such. Instead it uses the data more directly to construct parse trees. To simplify the situation, one can say that parse trees are *induced* in data-driven approaches, while they are *deduced* in grammar-driven. Moreover, data-driven parsing can be either fully *supervised*, where the parser needs a syntactically annotated corpus during learning, or *unsupervised*, where the parser is created from corpus data without syntactic gold-standard trees. This thesis will deal solely with the former.

Even though data-driven and grammar-driven approaches are quite dissimilar, they are not opposites, and there exist parsers that combine both in order to combine the best of both and compensate for their drawbacks. Black et al. (1993) report about a broad-coverage parser based on the PCFG model. The parser uses a statistical optimizing technique selecting the anal-

ysis for an input string that maximizes the syntactic structure, which does not necessarily have to be a complete parse tree.

Using a data-driven approach implies that the robustness problem that grammar-driven parsers exhibit is not a problem. As noted by Nivre (2006), this can result in even more severe problems of *overgeneration*, or leakage, than in grammar-driven approaches. Disambiguation is still an issue, although less severe, since the number of induced parse trees per sentence can quite easily be restricted to one by the built-in mechanism for scoring a complete tree or a partial tree during the parse by using probabilities or some other means. These probabilities can only be extracted from a corpus, directly extracted from the data.

It has been shown for constituency-based data-driven approaches that it is important that the base representation is well-chosen. Establishing the right base representation, which can be created using various corpus transformations, was an important part of the development of the constituency-based parsers of Collins (1997, 1999) and Charniak (2000). Their parsers are not based on PCFG but rather on so called *Markov grammars*, which incorporates bilexical dependencies. This is not easily done in a PCFG, even though Markov grammars can be mapped to PCFG. The tree structure constructed by these dependencies is essentially equivalent to definition 2.1 and 2.3, apart from the fact that the arcs are unlabeled (or all arcs have the same label).

Both parsers are most famous for parsing the Wall Street Journal part of the Penn Treebank, and have for a long time performed considerably better than approaches based on standard PCFG. It is worth noting that the head-finding strategy for constructing the bilexical dependencies in the Penn Treebank, using the head-percolation rules by (Collins 1997), regards the modal or auxiliary verb as the head and the main verb as dependent in verb groups. This is in line with Mel'čuk-style in dependency structure. Collins uses some special head-finding rules to deal with coordination, which results in a dependency structure according to Collins-style.

Another case in point is the transformation-based parsing approach (Brill 1993), which applies a sequence of tree rewriting rules in order to modify the constituency-based trees. The rewriting rules can also be constructed by hand, but are often induced from a treebank. It is an instance of the corpus-driven approach. Since these transformations are the actual parsing and not performed as preprocessing or postprocessing, it has more or less only the name in common with the transformations presented here.

2.4.3 Dependency Parsing using Corpus Data

Dependency-based formalisms and parsers have sometimes been created with constituency-based formalisms and parsers in mind, such as Hays (1964). They were often originally created for English, having a relatively fix word order. A projective dependency parser is often of an adequate approximation for such a language. This may have implied that most data-driven dependency-based parsers were only able to produce projective structures. This is true for the parsers of Eisner (1996), Yamada and Matsumoto (2003), Nivre et al. (2004) (MaltParser), and McDonald et al. (2005) based on the Eisner algorithm implemented as a Maximum Spanning Tree algorithm (MSTParser).

While unable to produce non-projective dependency trees, projective dependency parsers can be motivated practically. The parsing problem for a non-projective parser becomes more complex, which usually affects efficiency and robustness negatively. MaltParser and MSTParser will be discussed in some more details among the experimental setup of chapter 4.

Using syntactically annotated corpora (e.g. treebanks) to construct robust broad-coverage parsers has gained a lot of attention for the last years. As one consequence, the CoNLL shared task of 2006 was multi-lingual dependency parsing (Buchholz and Marsi 2006). Some of the treebanks used in the experiments of chapter 4 are collected from the treebanks of the shared task.

2.5 Related Work

This section will bring up a number of studies that in one or another way relate to the topic of this thesis, restricted to data-driven approaches. Studies related to non-projectivity and tree transformations in constituency-based parsing are discussed in subsection 2.5.1, and studies in dependency-based parsing are treated in subsection 2.5.2.

2.5.1 Constituency-Based Approaches

Discontinuous constructions in constituency structure, which are related to non-projectivity, and various successfully applied tree transformations are discussed in this subsection.

Discontinuous Constructions Much research has been carried out to handle empty categories, discontinuous constructions and non-local dependencies in constituency-based parsing. Using the same annotation strategy

as the Penn Treebank, picture (2) in figure 2.2 would contain an empty NP node located to the right of *see*. The empty NP node would be located inside a VP node together with *see*, where the empty NP node has nothing but a pointer to the real object *What*. This indicates that the normal location for an object is to the right of the main verb, which in terms of dependencies means that the main verb is the head of the empty NP node. In turn, the pointer from the empty NP node to *What* entails the non-projective arc from the main verb *see* to *What*, shown in the picture.

The above example of wh-movement is just one type of usage of empty categories in the Penn Treebank, a passive verb construction another. Plain unparsed text obviously does not contain empty categories, and even though Collins's Model 3 (1999) tried to resolve nonlocal relative pronoun dependencies using trace threading like in GPSG, the parsers of Collins (1999) and Charniak (2000) are in general not able to deal with these constructions.

Several studies have been performed to recover them. Besides the *in-processing* of some non-local relations in Collins's Model 3, two other approaches to the problem can be identified. Johnson (2002) implemented an approach based on post-processing using a pattern-matching algorithm. It can be characterized as a memory-based learning procedure on constituent-based subtrees on the parser output, where the patterns are extracted from the training data. He concluded that a large proportion of the empty categories can be recovered. Dienes and Dubey (2003a, 2003b) take on the issue from the other angle by preprocessing unparsed sentences. They train a Hidden Markov Model (HMM) in order to identify empty categories before parsing. Postprocessing was performed on the parser output to attach each empty node with an antecedent. They report better results than Collins and Johnson.

The experiments done by Levy and Manning (2004) on deep dependencies extend the approach to German (NEGRA) and presents results that compare favorably with the other studies for English on the Penn Treebank. Other contemporary similar studies, such as Cahill et al. (2004), Jijkoun and de Rijke (2004) and Campbell (2004), have developed the approaches further. Jijkoun and de Rijke use an interesting approach facilitating preprocessing and postprocessing, while treating the parser as a block box. They adopt for instance dependency relations to recover among other things non-local dependencies in the Penn Treebank.

Another case in point is Schmid (2006), who performs real slash style parsing on the Penn Treebank, annotated with GPSG-style features such that link traces and fillers. He uses an unlexicalized PCFG parser, and reports generally improved parsing accuracy, as well as the best published results for the empty category prediction task and the trace-filler coindexation.

Tree Transformations Applying various transformations before and after parsing using constituency-based representations have been an important research topic for several years. This holds not only for the recovery of constructions that are impossible for existing practical parsing systems, as described above, but also to improve the accuracy for constructions that the parsers are able to produce, though not that successfully.

As mentioned, parsing based on (unlexicalized) PCFG tends not to perform well when the probabilities are induced directly from a corpus as it is. One of the weaknesses of PCFG is the lack of non-local relationships between the non-terminal nodes, and Johnson (1998) reports a significantly increased accuracy by simply augmenting the node names with the parent's node name as preprocessing for the Penn Treebank. The implicit *independence assumption* of context-free grammar is weakened by incorporating this kind of contextual information, which presumably is important to accurately resolve certain phenomena. For instance, knowing that the parent of an NP is S (subject position) and not VP (object position) considerably increases the probability that the NP will expand to a pronoun.

Johnson also elaborates with different constructions to represent PPs in a VP by applying tree transformations, such as flattening or deepening the structure, with a subsequent inverse transformation. These tree transformations do have an impact on performance, although less significantly than the parent annotation. He also points out that “the choice of tree representation can have a noticeable effect on the performance of a PCFG language model”, and notes that a tree representation that is well-chosen on linguistic grounds can be quite different from what results in good performance for the PCFG model. Later studies have extended the methods for preprocessing and postprocessing, including various tree transformations, using PCFG models. Dienes and Dubey (2003c), for example, perform related transformation experiments on German. Klein and Manning (2003), who close the gap to the lexicalized parsing models by for instance Collins (1999), iteratively combine linguistically motivated transformations.

Later studies have refined the method even further, for example Petrov et al. (2006). One of the reasons why PCFG parsing performs relatively badly for Penn Treebank is the too coarse-grained set of non-terminals. They have, among other things, automatized the process of making the set of non-terminals more fine-grained by iteratively splitting them more and more while recording how it affects parsing accuracy, with a back-off mechanism to undo bad splits. They even report that these automatically induced splits can be linguistically motivated.

Another case in point is the numerous transformations taking place for the Markov grammar in Collins's parsers for the Penn Treebank, even though

they are not described in detail in his thesis (Collins 1999). Bikel (2004) enumerates eleven applied transformations taking place in Collins's parser, including special treatment of the bilexical relations in coordination. Without exaggerating, one can conclude that various tree transformations are very important in most constituency-based state-of-the-art parsers.

2.5.2 Dependency-Based Approaches

A number of language-specific robust broad-coverage parsers producing non-projective dependency structure exist, such as the grammar-driven, or rule-based, parsers of Tapanainen and Järvinen (1997) for English, Foth et al. (2004) for German, and the two parsers of Žabokrtský and Holan respectively, for Czech (Holan and Žabokrtský 2006). It is worth noting the approach adopted by Zeman (2004) in order to improve accuracy for coordination for Czech. He uses a kind of pattern matching, based on frequencies of the parts-of-speech of conjuncts and conjunctions, which seems to improve the overall performance of the parser.

The literature contains far fewer studies on tree transformations in dependency parsing, and there are at least two possible explanations. The first reason is that parsing based on dependency is less studied compared to constituency-based parsing. The second is that dependency structure usually results in less complex tree structures, mainly due to the absence of non-terminals, which leaves less room for applying transformations. For instance, the so important parent annotation in PCFG, which distinguishes NPs in subject position (with S as parent) from NPs in object position (with VP as parent), has no correspondence in dependency structure. First, dependency structures normally have direct relations from verbs to subjects and objects. This also makes the search for these types of relations trivial, and less complicated than in constituent structure. Secondly, these relations are usually explicitly marked in the dependency structure through labels such as (*Sbj* and *Obj*).

Several of the preprocessing and postprocessing transformations in data-driven dependency parsing are actually caused by the fact that constituency-based parsers have been modified in order to output dependency structure. Collins's parser was adapted to parse Czech (Prague Dependency Treebank, see section 4.1.1), which requires that the dependency trees in the training data were converted to phrase structure (Collins et al. 1999). The internally used bilexical relations were then considered the primary output of the parser, not the phrase structure trees. Besides that important preprocessing step of converting from dependency structure to phrase structure, which is their base representation, they apply a number of linguistically motivated phrase tree

transformations. Their transformations targeted on, e.g., relative clauses, coordination and punctuation which lead to improved parsing accuracy.

Another study related to non-projectivity is the corrective modeling of Hall and Novák (2005). The motivation is that Collins et al.'s parser and Charniak's parser adapted to Czech are not able to create the non-projective arcs present in the treebank, which is unsatisfactory. They therefore aim to recover the non-projective arcs present in the parser's output. By training a MaxEnt classifier, which identifies erroneous arcs (specifically all those arcs which should be non-projective), they can pinpoint wrongly placed arcs and move them in order to recover their original (non-projective) positions. They report an improved accuracy for non-projective arcs over 50%, presumably compared to close to 0% without it.

The approach for the approximative dependency parser of McDonald and Pereira (2006) is similar to the corrective modeling of Hall and Novák in the sense that it conducts postprocessing on the output of a projective parser (the Eisner algorithm). They investigate all arcs and determine, using a model of the non-projective training data, whether the overall score of the whole parse tree can be improved by replacing arcs of the projective output with new arcs. The new arcs are presumably non-projective ones, but similar to Hall and Novák, they do not have to be that. The tree that increases the overall parse tree score the most is then selected.

Chapter 3

Tree Transformations

The full potential of dependency-based parsing cannot be realized unless non-projective dependency graphs can be produced. In section 3.1, the method for transforming any non-projective dependency graph to a projective one, as well as the corresponding inverse transformation, is described, which makes it possible for a projective dependency parser to produce non-projective graphs. Section 3.2 and 3.3 focus on the constructions that are hard to parse, i.e. coordination and verb groups, and present the transformations and inverse transformations facilitating the learning task.

While the parsing experiments will be handled in chapter 4, it is worth pointing out here how the overall methodology during parsing works. It is schematically the same for all transformations and is divided into these four steps:

1. Apply the tree transformation to the training data.
2. Train a parser model using the transformed training data.
3. Parse a text using the parser model.
4. Apply the corresponding inverse transformation to the output of the parser.

These steps do not assume that a specific parser is used in step (2) and (3). In other words, as long as the trainable dependency parser conforms to the formal dependency definition, the parser can be regarded as a black box.

3.1 *Pseudo-Projective Transformations*

As mentioned in the introduction, the non-projective structures are said to be impossible, since no projective parser is able to produce them. The idea investigated here is whether a projective dependency parser can be trained so that non-projectivity can be recovered during postprocessing. This can be

facilitated by adding information during the transformation of the training data, informally called *pseudo-projective* dependency parsing.

One may ask the question whether it is worth the effort to pay so much attention to such a rare linguistic phenomenon as non-projectivity. After all, even in languages with free word order, usually only around 2% or less of the arcs are non-projective in existing dependency treebanks (see section 4.1.1). The penalty for neglecting these is therefore usually quite low. However, the situation looks different if non-projectivity instead is measured by the proportion of non-projective dependency graphs. A treebank with approximately 2% non-projective arcs can have 25% non-projective dependency graphs. In other words, it is unsatisfactory not to be able even in theory to derive a completely correct dependency graph for that many sentences.

This problem can be solved by trying to recover non-projective arcs from the output of a projective dependency parser. One such solution, which relies only on postprocessing, has been proposed by Hall and Novák (2005). The idea of *pseudo-projective dependency parsing* (Nivre and Nilsson 2005), combines postprocessing with a preceding preprocessing step. The pseudo-projective transformations that constitute the preprocessing and postprocessing, are presented in this section.

3.1.1 Transformation

Any non-projective dependency tree can be transformed into a projective one (i.e., *projectivized*) by replacing all non-projective arcs with projective ones. As noted by Kahane et al. (1998), this can be automatized by using a lifting operation. It informally moves all non-projective arcs “upwards” in a dependency tree until the lifted arcs become projective, according to definition 2.3. However, they do not propose an inverse transformation. The idea of building a projective tree by means of lifting also appears in Kunze (1968) and is used by Hudson (1990). Kahane et al. (1998) define a lifting operation, but the one presented here is slightly different. For a dependency graph $G = (V, E, L)$ obeying definition 2.2, it is defined as:

$$\text{LIFT}(j \rightarrow k) = \begin{cases} i \rightarrow k, & \text{if } i \rightarrow j \in E \\ \text{undefined} & \text{otherwise} \end{cases}$$

The lifting operation takes an arc $j \rightarrow k$ as argument. In case j has a head token i , which according to definition 2.2 is unique, it returns the *lifted* arc $i \rightarrow k$. If no arc such as $i \rightarrow j$ exists, it means that i is the root token, and the arc $j \rightarrow k$ can consequently not be lifted.

LIFT is applied to the non-projective arc in E , which is replaced by the output of LIFT. However, the result of LIFT(e) for a non-projective arc e is

3.1. Pseudo-Projective Transformations

not necessarily projective. It may therefore be necessary to perform this more than once and preferably in a deterministic manner. The following algorithm is deterministic and the returned graph is projective:

```

PROJECTIVIZE( $G = (V, E, L)$ )
1   $E' \leftarrow E$ 
2  while some arcs in  $E'$  are non-projective
3     $e \leftarrow \text{SMALLEST-NONP-ARC}(E')$ 
4     $E' \leftarrow (E' - \{e\}) \cup \{\text{LIFT}(e)\}$ 
5  return  $(V, E', L)$ 

```

Since more than one arc can be non-projective simultaneously in a graph, `SMALLEST-NONP-ARC` at line 3 determines the lifting order. With no particular reason other than making the algorithm deterministic, it selects the arc with the smallest span (measured as the distance between the head and the dependent), breaking ties from left to right. The lifting order will only matter if two or more non-projective arcs in the same sentence interact in some way, but usually the result is independent of it. An alternative approach would be to lift the arcs in the order of increasing height. The selected approach is nevertheless likely to have very limited influence in practice, because interactive non-projective arcs are relatively rare in existing dependency treebanks.

Line 4 performs the arc replacement, which together with line 3 is repeated until the (unlabeled) graph (V, E') becomes projective. It is worth pointing out that an arc with the root as head cannot be non-projective (see the proof of theorem 3.1), which entails that `LIFT` at line 4 never can have an undefined result.

Figure 3.1 shows a non-projective Czech sentence. The arc $jedna \rightarrow Z$ violates the projectivity constraint (je is not dominated by $jedna$). This arc will consequently be removed by the algorithm and replaced by the returned arc of the lift operation, which is the arc $je \rightarrow Z$. This arc is projective because the only token it spans over ($nich$) is dominated by je . Since no more non-projective arcs remain, the algorithm returns the projectivized dependency graph shown in figure 3.2. Following the terminology of Kahane et al. (1998), the original head ($jedna$) and the new head (je) of Z are called the *syntactic head* and *linear head*, respectively.

The dependency labels of the lifted arc and the arc it was lifted over in the figure have deliberately been replaced by *LL* (lifted label) and *PL* (path label). It is linguistically questionable whether the original labels can be used in the projectivized dependency structure without distorting the meaning. In the former case, a label is tied to the properties of the head token that an arc governs. This holds in the latter case as well, since the number of tokens

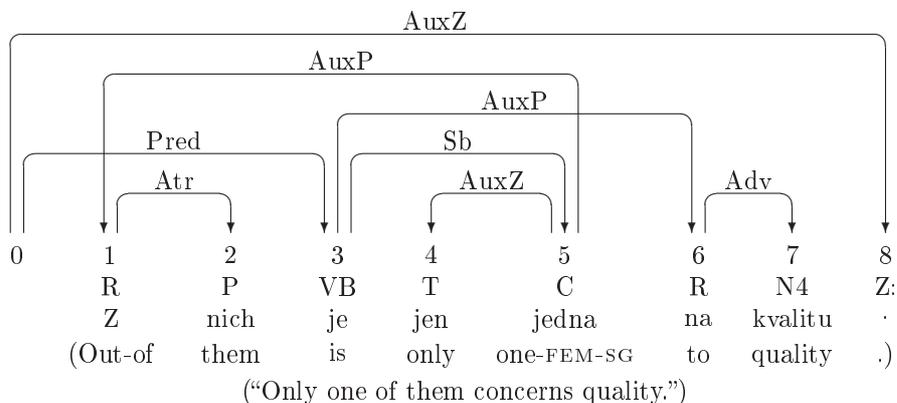


Figure 3.1: Non-projective dependency graph for Czech sentence from PDT.

and the token properties in a subtree may affect the meaning of the tokens dominating them. For instance, *jedna* in figure 3.1, with an *AuxP*-dependent, may not have the same meaning as in figure 3.2, without an *AuxP*-dependent. Moreover, all these labels are important because they will be used to encode the lifts in different ways in order to facilitate the inverse transformation. This will be discussed in subsection 3.1.2, but this subsection will conclude with a theorem regarding the above algorithm.

Theorem 3.1. *For any dependency graph $G = (V, E, L)$ satisfying definition 2.2, the algorithm $\text{PROJECTIVIZE}(G)$ will always (1) terminate and (2) return a projective dependency graph (V, E', L) satisfying definition 2.4.*

Proof. Definition 2.3 states that an arc $h \rightarrow d$ is projective iff all the tokens it spans over are dominated by the head of $h \rightarrow d$. The key consequent of definition 2.2 is that the dependency graphs form rooted trees, where token 0 is the only token that dominates all other tokens ($0 \rightarrow^* k, k \in V$). Therefore, for all arcs $h \rightarrow d$ such that $h = 0$, it follows that all tokens k (where $0 < k < d$) are also dominated by h . In other words, arcs with the root as head ($0 \rightarrow d$) are projective. This conclusion also justifies why it is unnecessary to deal with the undefined case of the LIFT-operation.

Left to prove is that any (non-projective) dependency tree after a finite number of lifts in PROJECTIVIZE forms a dependency tree with all arcs attaching to the root. As a dependency tree has a finite number of tokens $V = \{0, \dots, n\}$, it thus has a finite number of arcs E , where $|E| = |V| - 1$. The definition of a dependency tree implies that $0 \rightarrow^* i \rightarrow k$, where $i \rightarrow k$

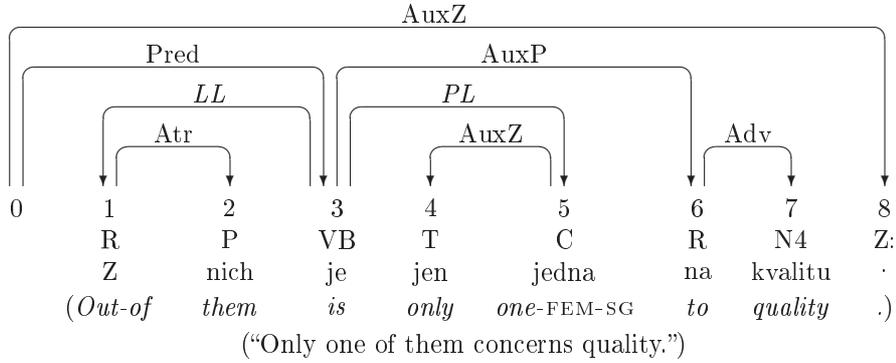


Figure 3.2: Projectivized dependency graph for Czech sentence from PDT.

to be lifted. The path distance from the root to k can in the worst case be $|E|$. If $0 \rightarrow^* j \rightarrow i$ holds, the lift operation replaces $i \rightarrow k$ with $j \rightarrow k$, where $0 \rightarrow^* j \rightarrow k$ also holds. The lift operation can be applied at most $|E| - 1$ times until $0 \rightarrow k$, which according to the previous paragraph is enough to conclude that the arc is projective.

The loop spanning from line 2 to 4 will not stop iterating until all arcs are projective, which according to definition 2.4 implies that the complete graph is projective. It will terminate since the maximum number of iterations is finite. When it has stopped iterating, the new dependency graph is no longer manipulated by the algorithm and it will therefore return a projective dependency graph. \square

In the worst case, an arc can be lifted at most $l = |E| - 1$ times, which implies that the height of the dependency tree is reduced to $l - 1$. This, in turn, means that the next arc can be lifted at most $l - 1$ times, and so forth, until the height of the dependency tree is 1. In total, at most $l + (l - 1) + \dots + 1 = \sum_{i=0}^{l-1} (l - i) = (l(l - 1))/2$ lifts can be applied. Since determining whether a dependency graph is projective or not can be done in linear time in relation to the number of tokens (Havelka 2005), and line 3 and 4 take constant time, PROJECTIVIZE has in the worst case cubic running time.

3.1.2 Encodings

Whereas the projectivization algorithm is deterministic, the same does not hold for the inverse transformation. The inverse transformation performs a search for the syntactic head “downwards” among the tokens dominated by

Encoding (Symbol)	LL	PL	# labels
BASELINE (p_0)	d ($AuxP$)	h (Sb)	$ R $
HEAD (p_H)	$d\uparrow h$ ($AuxP\uparrow Sb$)	h (Sb)	$ R \cdot (R + 1)$
PATH (p_P)	$d\uparrow$ ($AuxP\uparrow$)	$h\downarrow$ ($Sb\downarrow$)	$4 R $
HEAD+PATH (p_{HP})	$d\uparrow h$ ($AuxP\uparrow Sb$)	$h\downarrow$ ($Sb\downarrow$)	$2 R \cdot (R + 1)$

Table 3.1: Projectivity encodings. **LL** = lifted label (value of LL in figure 3.2), **PL** = path label (value of PL in figure 3.2).

linear head. The linear head in figure 3.2 dominates four potential syntactic heads, the tokens 4, 5, 6, and 7. To facilitate the search, four encoding schemes, including one baseline version, have been implemented for the labels of lifted arcs and the arc(s) which have had arcs lifted over them (known as the path), illustrated by the labels LL and PL in figure 3.2.

The encoding schemes incorporate different amounts of information concerning the lifts. Needless to say, the more such information there is in the projectivized data, the easier it will be to recover the original non-projective structure. In principle, the exact syntactic head could in some way be encoded in the labels, yielding an error free inverse transformation. However, the labels should be learned by the parser and assigned to arcs of new unseen sentences, and the more information that is incorporated into the labels, the harder the learning task is for the parser. Finding the right balance of information in the new labels is therefore important, which certainly depends on properties such as size of the training data, number of distinct labels and amount of non-projectivity.

The four encoding schemes are shown in table 3.1, with the labels for LL and PL of figure 3.2. In the BASELINE-encoding, the original labels are kept unchanged. This means that the lifted arc label of Z becomes d ($AuxP$) and the path label of $jedna$ becomes h (Sb). This is the simplest possible encoding, but it does not include any information that helps in finding the syntactic head during the inverse transformation.

In the second encoding, HEAD, the lifted arc concatenates the label of its syntactic head (h) to its original label (d), separated by the symbol \uparrow .¹ The path labels will however contain the same information as BASELINE. In other words, the lifted arc signals what the label of its syntactic head is, which will guide the inverse transformation later on. Since Sb is the label of the syntactic head in figure 3.1, $LL=AuxP\uparrow Sb$ in figure 3.2.

The encoding PATH, on the other hand, does not include the label of the syntactic head in the lifted arc. It only signals that it has been lifted

¹This symbol indicates that the arc has been moved upwards in the tree.

($LL=AuxP\uparrow$), which compared to HEAD leaves the label of the syntactic head unspecified. However, the path labels are extended in such a way that they indicate that (at least) one arc has been lifted over them, attaching the symbol \downarrow to all labels along the path ($PL= Sb\downarrow$). This creates a continuous trace from the linear to the syntactic head.

The last encoding, which combines HEAD and PATH, uses the same label LL as HEAD and the same path labels as PATH. This is the most informative encoding and provides the inverse transformation with a lot of information to find the syntactic head.

The last column of table 3.1 reveals the theoretical upper bound on the number of labels that the projectivized data can contain. PATH has only a linear increase (possible labels: $r, r\uparrow, r\downarrow, r\uparrow\downarrow$). Due to the previously discussed trade-off, it is therefore the least informative encoding and should presumably at the same time be the least complicated labels for the parser to learn. The two others have quadratic increase, because any label can be combined with any label in R , probably making the assignment of labels during parsing harder. Other factors of course influence complexity of the learning task too, such as the number of relabeled arcs. This will be discussed in conjunction with the results in chapter 4.

By and large, including the label of the syntactic head in the label of the lifted arc really limits the number of possible syntactic head candidates, but if there are few distinct labels the number of candidates increases on average. This is not a problem if path information can be used, because a complete trace back to the syntactic head is provided. However, path information may be misleading whenever non-projective arcs interact during projectivization, that is, they are lifted along the same path but have different syntactic heads. Furthermore, the situation becomes even more delicate during parsing since the parser may assign incomplete or inconsistent pseudo-projective information, such as broken traces back to the syntactic head and syntactic head labels in the lifted arc labels that do not exist further down in the tree. Resolving these issues is the topic of the subsection below.

3.1.3 Inverse Transformation

The inverse transformation (i.e. deprojectivization), can rely on several different methods, including various machine learning methods. Here a completely algorithmic strategy, making use of the pseudo-projective information provided by the applied encoding scheme, has been implemented. The search for possible syntactic heads is performed among the tokens dominated by the linear head (excluding the tokens dominated by the lifted arc). The first token conforming to the provided pseudo-projective information is selected as

the new syntactic head. The new (probably non-projective) arc to the new syntactic head replaces the lifted arc.

The results presented in chapter 4 are based on the breadth-first search approach, which in preliminary experiments have been the most accurate.² The search starts by identifying each lifted arc $i \rightarrow k$ in the dependency graph, which is easily done by looking for the symbol \uparrow in the labels of all tokens (basically $i \xrightarrow{d\uparrow} k$ for PATH and $i \xrightarrow{d\uparrow h} k$ for the two others), from left to right in each sentence. For each such arc, the search is done breadth-first, left-to-right starting at i (excluding the subtree of k). For the lifted arc in figure 3.2 with the linear head je , it means that the tokens $\{5, 6, 4, 7\}$ ³ (in that order) are considered as possible syntactic heads. Then, depending on the encoding, the first candidate token m satisfying the following conditions is selected (cf. table 3.1 for the labeling):

- For HEAD: **if** m has the label h to its head, i.e., $l \xrightarrow{h} m$, **then** replace $i \xrightarrow{d\uparrow h} k$ with $m \xrightarrow{d} k$.
- For PATH: **if** all arc labels between m and i have the form $\xrightarrow{p\downarrow}$ **and** m does not have any outgoing arcs of the form $m \xrightarrow{p\downarrow} o$, **then** replace $i \xrightarrow{d\uparrow} k$ with $m \xrightarrow{d} k$.
- For HEAD+PATH: the same as PATH with the additional condition that the arc from m to its head must have the label $h\downarrow$, i.e. $l \xrightarrow{h\downarrow} m$.

For instance, when the inverse transformation for HEAD finds *jedna* (token 5, the first to be checked), it sees that the arc label to *jedna* (Sb) matches the label to the right of \uparrow in the lifted arc label ($Auxp\uparrow Sb$). Accordingly, it correctly recognizes *jedna* as the syntactic head.

The syntactic head for PATH is not allowed to have a dependent with a label containing \downarrow , as the dependent would then be a more likely syntactic head (which happens for repeated lifts). In the example, where the arc from *je* to *jedna* is labeled $Sb\downarrow$ and with no outgoing arcs labeled $p\downarrow$, *jedna* here too becomes the correct syntactic head.

Finally, when PATH and PATH+HEAD have deprojectivized a dependency tree, each arc label of the form $\xrightarrow{p\downarrow}$ is replaced by \xrightarrow{p} .

²Other investigated approaches are, for example, depth-first and closest-first (the tokens closest to the linear head are considered first).

³Depth-first would be $\{5, 4, 6, 7\}$ and closest-first $\{4, 5, 6, 7\}$.

Back-off All three encodings adopt a back-off in case none of the possible candidates can be selected given the above conditions of the chosen encoding. There are two possible explanations why the search may fail: the information is inconsistent or incomplete (which it often is when parsing), or the correct syntactic head is not among the candidates due to the interaction of several lifts (i.e. broken paths, which can happen when an arc along a path of a lifted arc is lifted itself). Therefore, if a search fails and any subsequent lifted arcs in the sentence succeed in finding their syntactic heads, the failed search is repeated, hoping that a broken path has been repaired. This will partially solve the latter problem. The process iterates until no subsequent searches succeeds any longer (which in practice happens after very few iterations, normally 1-3).

It is important to mention that the algorithm for HEAD-PATH relaxes the search condition if the search fails, namely that it performs a search equivalent to HEAD. This means that all paths are admitted, not just those of the form $p\downarrow$.

For any remaining lifted arc with a failed search, irrespective of the encoding, the algorithm's last resort is to leave the lifted arc in place. That is, it lets the linear head be the syntactic head, and the label is simply changed to d (i.e. $i \xrightarrow{d} k$).

3.2 Coordination Transformations

In contrast to the pseudo-projective transformations, the coordination transformations are more dependent on the annotation of a particular treebank. The coordination experiments will be conducted on four treebanks, of which three adopt **PS**, Prague Dependency Treebank (PDT) (Hajič 1998), Slovene Dependency Treebank (SDT) (Džeroski et al. 2006), and Prague Arabic Dependency Treebank (PADT) (Hajič et al. 2004). The dependency version of the Dutch Alpino Treebank (van der Beek et al. 2002) does in general not comply with **PS**, but treats coordination in a way very similar to **PS**, that is, the conjunction acts as the head with the conjuncts as dependents.

This section will present the transformations and inverse transformations of coordination, while the presentation of the treebanks is postponed until chapter 4. It will contain a more detailed presentation than section 3 of Nilsson et al. (2006). The **PS-MS** transformations were the only ones presented in that paper, since in preliminary experiments gave the best result compared to **PS-CS**. Also, the experiments were performed only for PDT. Later preliminary experiments revealed that **PS-MS** is better than **PS-CS** for SDT and PADT as well. In this thesis, the **PS-MS** transformations will

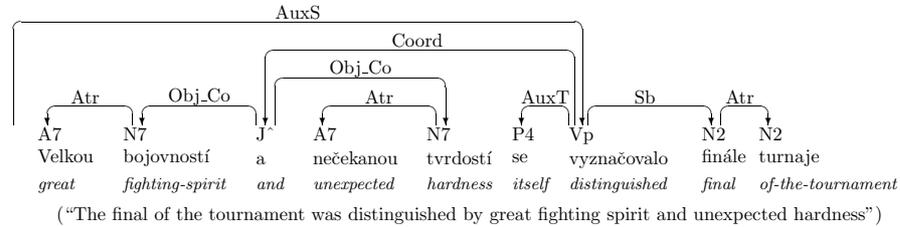


Figure 3.3: Coordination in **PS** for a Czech sentence from PDT.

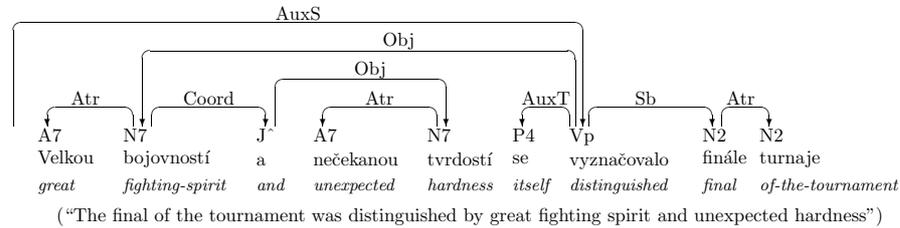


Figure 3.4: Coordination in **MS** for a Czech sentence from PDT.

therefore only be applied for PDT, SDT and PADT. For Alpino, the transformations of coordination to **CS** will also be conducted in chapter 4, which is the reason why they also will be presented below.

3.2.1 Transformation

The description below is adapted to PDT, SDT and PADT. Since the annotation of coordination in Alpino is similar but not identical, a presentation of the differences in relation to **PS** ends this subsection.

Classification of Tokens The transformation begins with the identification of a *base conjunction*, based on its dependency type and in some cases also its part-of-speech. The base conjunction has the label *Coord* in PDT, SDT and PADT. For example, the word *a* (and) in figure 3.3 is identified as base conjunction. The only tokens of interest in the transformation are the base conjunction and all its dependents. When a base conjunction has been identified, the transformation starts with a classification of the tokens into three linear ordered sets:

- Conjuncts C_1, \dots, C_n

3.2. Coordination Transformations

- Separators $S_1, \dots, S_{m_{n-1}}$, where S_{1_i}, \dots, S_{p_i} is located between C_i and C_{i+1} .
- Other dependents D_1, \dots, D_k

The base conjunction is categorized as a separator. If the coordination consists of more than two conjuncts, it normally has one or more commas (usually labeled *AuxX*) separating conjuncts, in addition to the base conjunction. They are also categorized as *S*. The coordination in figure 3.3 contains no commas, so only the word *a* will belong to *S*.

The remaining dependents of the base conjunction need to be divided into conjuncts (*C*) and other dependents (*D*). The conjuncts are easily identified in PDT since they have labels suffixed *_Co*, which is the case for the words *bojovností* and *tvrdostí* in figure 3.3. However, special care must be taken for coordinated prepositional cases (*AuxP*) and embedded clauses (*AuxC*), as the suffix is located on their dependents instead of directly on *AuxP* and *AuxC* (cf. Böhmová et al. 2003).

Unfortunately, SDT and PADT do not add *_Co* to the label of conjuncts, making the distinction harder. Heuristic rules have thus been implemented. Although the heuristic rules also consist of a number of back-off conditions, the simple and accurate main rule says that if the base conjunction has dependents on both sides with the same label, they are selected as conjuncts *C*. All other dependents of the base conjunction, not identified as *S* or *C*, are categorized as *D*. Since there are no other dependents of *a* in figure 3.3, the coordination contains no instances of category *D*.

Arc Transformations Given the classification of the words involved in a coordination, the transformation τ_c is straightforward and basically connects all the arcs in a chain:

1. Make C_1 a dependent of the original head of the base conjunction.
2. Make each S_{j_i} , located between two adjacent conjuncts C_i and C_{i+1} ($C_i < S_{1_i} < \dots < S_{p_i} < C_{i+1}$), a dependent of C_i .
3. Make each C_{i+1} ($i > 0$) a dependent of S_{p_i} , i.e. the separator closest to its left.
4. Make each $d \in D$ a dependent of the C_i closest to its left, or of C_1 if d is located to the left of C_1 .

For PDT, the dependency types of the conjuncts are truncated by removing the suffix *_Co*. Preliminary results indicated that this increases parsing

accuracy. After the transformation τ_c , every coordination forms a left-headed chain, as illustrated in figure 3.4.

This new representation creates a problem, though. For instance, the word *Velkou* in figure 3.4 is not distinguishable from a possible dependent in D . It could have been a dependent of a in figure 3.3, which is an obvious drawback when transforming back to PS. One way of distinguishing D elements is to extend the set of dependency types. The dependency type r of each $d \in D$ can be replaced by a completely new dependency type $r+$ (e.g., $Atr+$), theoretically increasing the number of dependency types to $2 \cdot |R|$. Consequently, this avoids a clash between the dependency type of the words in D and dependency types such as Atr for *Velkou*. We will denote this extended version of the transformation by τ_{c+} .

Adaptations for Alpino As mentioned above, the annotation for coordination in Alpino is similar but not identical to the annotation in PDT, SDT and PADT. A conversion to **CS** has also been done. Besides a completely different set of dependency types, there are mainly two distinguishing characteristics in Alpino:

1. The label to the head of the base conjunction and the label to the conjuncts are interchanged. In figure 3.3, this would correspond to the situation that the word a has the label *Obj* to its head and *bojovnosti* and *tvrdosti* the label *Coord* ($Coord=cnj$ in Alpino).
2. Commas acting as separators are not dependents of the base conjunction. In the normal situation, they are instead dependents of the token closest to its left, which usually is the conjunct or a token dominated by the conjunct.
3. Alpino does not distinguish between coordination dependents and conjunct dependents, i.e. the base conjunction has nothing but conjuncts as dependents.

These differences accordingly affect the classification into C , S and D . Especially the identification of commas into S is more complicated, since they now are dominated by the tokens classified as the conjuncts in C . In order to adapt the transformation with respect to the first distinguishing characteristic, their labels are interchanged before rearranging the arcs. For **PS**, this makes the labels correspond to the labels of *bojovnosti* and *tvrdosti*, and a in figure 3.3.

To convert coordination into **CS**, each C_{i+1} is basically made dependent on C_i , while all tokens in S simply are left as they are. The exception is the

base conjunction itself, which becomes a dependent of the conjunct closest to its left or a token dominated by this conjunct located to the left of the base conjunction. This means that **CS** is identical to the dependency graph in figure 3.4 with the exception that the head of *tvrdoští* is *bojovností*, not *a*.

3.2.2 Inverse Transformation

The inverse transformation for PDT, SDT and PADT again starts by identifying base conjunctions, using the same conditions as before. For each identified base conjunction, it calls a procedure that performs the inverse transformation by traversing the chain of conjuncts and separators “upwards” (right-to-left), while collecting conjuncts (C), separators (S) and *potential* conjunction dependents (D_{pot}). When this is done, the former head of the leftmost conjunct (C_1) becomes the head of the rightmost (base) conjunction (the right-most member of S). In figure 3.4, the leftmost conjunct is *bojovností*, with the head *vyznačovalo*, and the rightmost (and only) conjunction is *a*, which will then have *vyznačovalo* as its new head. All conjuncts in C become dependents of the rightmost conjunction, which means that the structure is converted to **PS**, depicted in figure 3.3.

As mentioned above, the original structure in figure 3.3 did not have any coordination dependents, but *Velkou* $\in D_{pot}$. The last step of the inverse transformation is therefore to sort out conjunction dependents from conjunct dependents, where the former will attach to the base conjunction. Four versions have been implemented, two of which take into account the fact that some dependency types occur more frequently as conjunction dependents (D) than as conjunct dependents in the training data set:

- τ_c : Do not extend arc labels in τ_c . Leave all words in D_{pot} in place.
- τ_{c^*} : Do not extend arc labels in τ_c . Attach all words with label *AuxG*, *AuxX*, *AuxY* or *Pred*⁴ to the base conjunction.
- τ_{c+} : Extend arc labels from r to $r+$ for D elements in τ_c . Attach all words with label $r+$ to the base conjunction (and change the label to r).
- τ_{c+^*} : Extend arc labels from r to $r+$ for D elements in τ_c , except for the labels *AuxG*, *AuxX*, *AuxY* and *Pred*. Attach all words with label $r+$, *AuxG*, *AuxX*, *AuxY*, or *Pred* to the base conjunction (and change the label to r if necessary).

⁴For SDT and PADT *Pred* is excluded here, as well as for τ_{c+^*}

For the particular example at hand, with the word *Velkou* labeled *Atr*, all four versions would correctly recognize it as a conjunct dependent and not attach it to the base conjunction.

The inverse transformation for Alpino with **CS** is similar, with the difference that the separators (dominated by the conjuncts) are collected in another way as the chain only contains conjuncts. Finally, the labels of conjuncts and separators are interchanged in order to restore the original annotation. The label of the rightmost conjunct is chosen as the label of the conjunction (in case the conjuncts have different labels).

The above transformation has been presented schematically. It is worth pointing out that a large number of coordination structures in all four treebanks are not as exemplary as described above, which in various ways is reflected in the experiments of chapter 4. All transformations handle the prototypical cases with high accuracy, but can be improved for special cases. So to conclude this section, the more special cases that are covered by the transformations, the better the transformation accuracy is. But this comes at the expense of generality, which is a reason why the transformations to a large extent have been kept fairly simple.

3.3 Verb Group Transformations

PDT and SDT are the only treebanks that have auxiliary verbs annotated according to **PS**, and they are therefore the two treebanks in the following experiments for which the verb group transformations have been performed. In comparison to the coordination transformation and inverse transformation from **PS** to **MS** and **CS**, and back, the corresponding transformations for verb groups are strikingly simple. The transformation algorithm, τ_v , starts by identifying all auxiliary verbs in a sentence, which will belong to the set A . A word $a \in A$ iff $m \xrightarrow{AuxV} a$, where m is the main verb. For example, the word *bude* in the Czech sentence of figure 3.5 will belong to A . Then, for each $a \in A$ processed from left to right in each sentence:

1. Replace the arc $m \rightarrow a$ with $h \rightarrow a$, where $h \rightarrow m$.
2. Replace the arc $h \rightarrow m$ with $a \rightarrow m$.
3. **For each** arc $m \rightarrow d$, $d < \text{left}(a, m)$ **do**:
replace $m \rightarrow d$ with $\text{left}(a, m) \rightarrow d$.
4. **For each** arc $m \rightarrow d$, $d > \text{right}(a, m)$ **do**:
replace $m \rightarrow d$ with $\text{right}(a, m) \rightarrow d$.

3.3. Verb Group Transformations

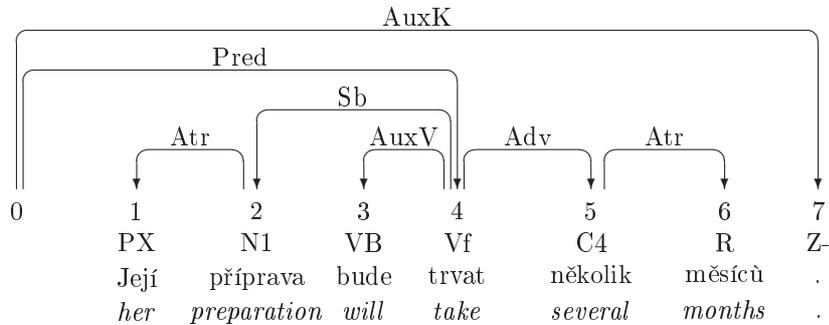


Figure 3.5: Verb group in **PS** for a Czech sentence from PDT.

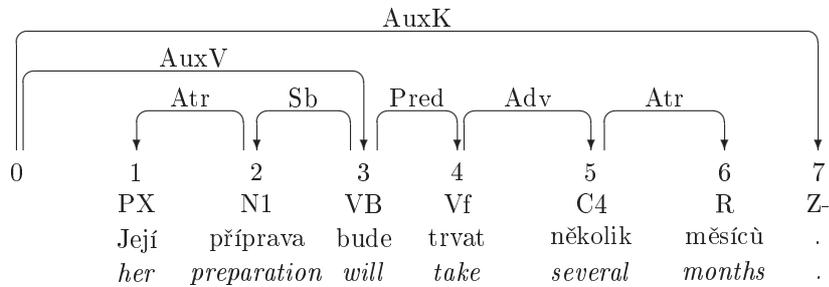


Figure 3.6: Verb group in **MS** for a Czech sentence from PDT.

Figure 3.6 depicts the dependency graph of the same Czech sentence in **MS**. Line 1 and 2 are not complicated. The transformation into **MS** reverses the relation between the verbs, i.e., $a \rightarrow m$, and the former head of m becomes the new head of a .

The main verb can be located on either side of the auxiliary verb and can have other dependents, whereas auxiliary verbs never have dependents. Line (3) states that all common dependents of the two verbs located to the left of the right-most verb after the transformation will be dependents of the left verb. The only such arc in the example is the subject *příprava*, which becomes a dependent of the auxiliary verb after the transformation, i.e. the left-most verb ($\text{left}(a, m)$). Line (4) takes care of all other dependents located to the right of the right-most verb (such as *několik*), which will depend on the right-most verb ($\text{right}(a, m)$).

Without line (3) and (4), the dependency graph may become non-projective, which happens in the case where the main verb is the left-most verb having dependents to the right of the auxiliary verb (that is $m < a < d$,

where $m \rightarrow d$). The arc $m \rightarrow d$ would be non-projective in **MS**.

Performing the inverse transformation for verb groups is quite simple and essentially the same procedure inverted. Each sentence is traversed from right to left looking for arcs of the type $h \xrightarrow{AuxV} a$, where $a \rightarrow m$. For every such arc, h will be the new head of m , and m the new head of a . Furthermore, since auxiliary verbs do not have dependents in **PS**, all dependents of a in **MS** will become dependents of m in **PS**.

3.4 Transformation Independence

As mentioned in the introduction, language, treebank and parser independence can be interpreted in at least two ways. Foremost, one may ask whether the transformations improve accuracy independent of language, treebank and parser. This cannot yet be determined, since this requires parsing experiments, presented in chapter 4.

In case the transformations result in an improved accuracy, one may ask another question regarding independence, that is, can the transformations be constructed independent of language, treebank and parser? If accuracy cannot be improved, this type of independence becomes rather irrelevant, but the question can nevertheless be answered without parsing experiments.

It seems clear that the pseudo-projective transformation is independent of all three. The definition of projectivity is only based on definition 2.2, which says nothing about any particular language, treebank or parser. In other words, as long as a particular (projective or non-projective) dependency treebank and a particular projective dependency parser conform to definition 2.2, the pseudo-projective transformation can be applied to the treebank and the projectivized treebank can be used as training data by the parser.

For coordination and verb groups, it is clear that parser independence holds given the methodology of the parsing experiments presented in the beginning of this chapter. It is just as clear that they are not treebank independent, since these phenomena must have the particular dependency structure for coordination and verb groups. Moreover, Arabic seems to be a language that lacks auxiliary verbs as individual tokens. This makes even the language independence questionable, at least for verb groups.

Chapter 4

Experiments

The graph transformations introduced in chapter 3 will in this chapter be put into action. Before presenting the results of the three conducted experiments in sections 4.2 – 4.4, the first section 4.1 discusses the experimental setup and some methodological issues. The main research question that this chapter is focused on is whether the transformations can improve parsing accuracy. The secondary research question is to investigate the transformation’s independence of language, treebank and parser with respect to improved parsing accuracy. Three experiments have been performed:

- **Experiment I: Treebank Transformations:** *To what extent do the transformations themselves distort the treebank data?* All transformations are devised to facilitate the learning task, but they make errors. To estimate how much the change in accuracy can be attributed to a transformation’s peculiarity to introduce errors, the transformation and inverse transformation are applied to the treebanks without using a parser. In contrast to the two following experiments, this one does not relate directly to any of the research questions stated in the introduction.
- **Experiment II: Treebank Independence:** *To what extent are the transformations independent of the language and treebank?* The transformations were constructed to increase parsing accuracy while keeping them as language and treebank independent as possible. This will be investigated by applying the transformations to treebanks for different languages with various annotation schemes using an inductive dependency parser.
- **Experiment III: Parser Independence:** *To what extent are the transformations parser independent?* A data-driven dependency parser may have characteristics that make its learning and parsing behavior more or less sensitive to the treebank annotation. This experiment will therefore evaluate the effect of the transformations using the same data, but another data-driven dependency parser, compared to experiment II.

	Slovene SDT	Arabic PADT	Dutch Alpino	German Tiger	Czech PDT
# T	29	54	195	700	1249
# S	1.5	1.5	13.3	39.2	72.7
NPS	22.2	11.2	36.4	27.8	23.2
%-NPA	1.8	0.4	5.4	2.3	1.9
%-C	9.3	8.5	4.0	-	8.5
%-S	5.1	4.6	2.2	-	4.4
%-VG	8.8	-	-	-	1.3

Table 4.1: Overview of the data sets (ordered by size), where # S * 1000 = number of sentences, # T * 1000 = number of tokens, %-NPS = percentage of non-projective sentences, %-NPA = percentage of non-projective arcs, %-C = percentage of conjuncts, %-S = percentage of separators, %-VG = percentage of verb groups.

4.1 Experimental Setup

The utilized parsers and treebank resources are introduced here, followed by a subsection concerning the evaluation details.

4.1.1 The Treebanks

Five different treebanks representing five different languages (Arabic, Czech, Dutch, German and Slovene) are in focus in the coming experiments. The treebanks have been selected since they have annotation properties suitable for the transformations. They are all dependency treebanks, or have been converted to dependency treebanks. Experiment II uses the treebanks in the CoNLL format (Buchholz and Marsi 2006), designed by the organizers of the CoNLL shared task 2006, because all five treebanks have been converted to this format, and because this will enable comparison with the results of the shared task.¹ Experiment III is restricted to two treebanks, the Alpino treebank and PDT. Table 4.1 contains an overview of the five treebanks.

Prague Dependency Treebank As other Slavic languages, Czech is characterized by rich inflection and relatively free word order. Prague Dependency Treebank contains newspaper text and is the largest manually annotated treebank (Hajič 1998, Hajič et al. 2001), with almost 1.3 million words of training data. The annotation is based on the linguistic theories of the

¹See Buchholz and Marsi (2006) for details about the conversion of the original treebanks to the CoNLL format.

Prague-school (Functional Generative Description) and consists of tree layers of annotation (Böhmová et al. 2003), but it is only the analytical (syntactic) layer that will be used here. All dependency relations of the syntactic layer are labeled with one of 28 dependency types.

Experiment I and II are based on the version of PDT used in the CoNLL shared task, whereas experiment III is based on the original version of PDT. This is motivated by the fact that it will be easier to compare the results for PDT in experiment I with the other treebanks in the shared task, and that the results of experiment II will be compared to the results of the shared task. Experiment III will enable comparison to published results of parsing PDT, using the established division into training and testing sets of the original data. Moreover, the parser in experiment III is not able to handle the CoNLL format.

The following description applies only to the Czech data of experiment III (as mentioned above, see Buchholz and Marsi (2006) concerning the treebank conversion of the shared task). The original Czech training and test sets of experiment III are part-of-speech tagged with an HMM-tagger distributed with the treebank. There are over 1500 distinct tags in the original tag set, which is far too many in order to perform efficient and accurate parsing for the parsers of Experiments III. The HMM-tagger has an accuracy of 94.1%, and the number of tags has been reduced to a manageable 61 tags according to Collins et al. (1999), which has become the standard when parsing PDT. All figures in Experiment III are reported for the evaluation test set (informally known as *etest*), while the development test set was used during the experimental phase.

As the last five rows of table 4.1 reveals, PDT contains a quite high proportion of non-projectivity, since almost every fourth dependency graph contains at least one non-projective arc. The table also shows that coordination is more common than verb groups in PDT. Only 1.3% of the tokens in the training data are identified as auxiliary verbs, whereas 12.9% of the tokens (identified as conjuncts and separators) are involved in coordination. Judging from these figures, it is likely that transformations for coordination will have a greater impact on the accuracy than the corresponding transformations for verb groups.

Slovene Dependency Treebank Slovene is just as Czech a Slavic language, with similar properties in terms of rich morphology and free word order. The creation of SDT (Džeroski et al. 2006) has been highly influenced by PDT. Therefore, SDT uses an annotation that is very similar to PDT, including the set of syntactic tags labeling the dependency relations. SDT contains 2,000 sentences from the first part of the Slovene translation of Or-

well's 1984, of which 1,500 are extracted as training data in the conversion to the CoNLL format. It is compared to PDT a very small treebank, and the authors characterize the treebank as a proto-released treebank, meaning that the treebank is still under development.

The proportions of non-projectivity, conjuncts and separators is in fact quite similar to the proportions in PDT. The big difference is the proportion of auxiliary verbs, with many more auxiliary verbs in SDT than in PDT. It is therefore plausible that the transformations for verb groups in SDT are more influential on the parser accuracy.

Prague Arabic Dependency Treebank As the name implies, PADT is also an offspring of PDT (Hajič et al. 2004), and is also influenced by the Prague-School tradition. PADT contains newswire text of Modern Standard Arabic with both morphological and analytical annotation. It has the same number of sentences as SDT although with almost twice as many tokens, but is still small compared to PDT.

Arabic is not a Slavic languages such as Czech and Slovene, and the annotation has therefore to a larger extent than for SDT been altered. One such example is that Arabic does not have auxiliary verbs, resulting in the absence of the dependency label *AuxV*. Table 4.1 consequently does not present the amount of verb groups. The amount of coordination is on the other hand comparable to both PDT and SDT. The table also reveals that the amount of non-projective arcs is less than 25% in relation to PDT and SDT, although the amount of non-projective sentences is still as much as half compared to PDT and SDT (due to long sentences in PADT²). As mentioned before, the CoNLL version of PADT is used in the experiments.

Dutch Alpino Treebank Similar to the three above treebanks, Dutch Alpino Treebank (van der Beek et al. 2002) contains text collected from newspaper articles. The treebank has grown in the last years and the number of tokens in the CoNLL version is over 13 000 sentences. The original Alpino Treebank uses a mixture of the annotation schemes and guidelines applied to the *CGN Corpus* of spoken Dutch and the TIGER Treebank (Kakkonen 2005).

One of the characteristics of Alpino is the high amount of non-projectivity. A projective parser without special treatment of non-projectivity would not even in theory be able to produce a completely correct dependency structure for more than one third of all sentences. Another way to express this is to

²According to the organizers of the shared task: in many cases, the unit in PADT is not a sentence but a paragraph.

say that every non-projective arc results in at least one erroneous arc in the output of a projective parser, either the arc itself or a nearby arc if it actually has the correct head. In the Alpino treebank, neglecting non-projective arcs would result in at least 5.4 percentage points lower accuracy as the asymptotic goal (compared to 100% correct accuracy) in an otherwise maximally correct dependency structure of a projective parser. Moreover, table 4.1 also shows that coordination is less frequent in the CoNLL version of Alpino than in the three previously presented treebanks, which probably will lower the effect of the Dutch coordination transformations.

German TIGER Treebank The German TIGER Treebank (Brants et al. 2002) is the data set that so far has received the least attention in this thesis, simply because it can only be used in the pseudo-projective experiments. The original treebank has emerged from the NEGRA Corpus project (Skut et al. 1998) and its annotation combines both phrase structure and dependency structure. It is a collection of German newspaper text on various topics.

It is a relatively large dependency treebank, the second largest in the CoNLL shared task with almost 40,000 sentences. Coordination and verb groups in the treebank are not annotated in a way similar to the Prague-School style. However, its size and a high proportion of non-projectivity compared to the treebanks based on the Prague-School style make the treebank interesting for this study, as the pseudo-projective transformations are applicable. As table 4.1 shows, the CoNLL version of the treebank contains 2.3% non-projective arcs.

4.1.2 The Parsers

Experiment II uses version 0.4 of MaltParser, which has been further developed since the paper by Nivre et al. (2004), and experiment III uses version 0.1 of MSTParser (McDonald, Crammer, and Pereira 2005), using first-order edge features. There is currently a version 0.2 of MSTParser, using second-order edge features (McDonald and Pereira 2006), but it was not available during the experimental phase.

MaltParser is according to Nivre et al. (2006) essentially based on three components:

- Dependency graphs are built using deterministic parsing algorithms.
- History-based feature models for predicting the next parser action.
- Histories are mapped to parser actions using discriminative machine learning.

MaltParser can be characterized as a data-driven parser generator, since a parser can be created given that there exists a treebank containing dependency trees. It is modularized in the sense that different parsing algorithms, machine learning methods and feature sets can be selected fairly independently of each other. Various versions of Covington's (2001) and Nivre's (2003) parsing algorithms are implemented in MaltParser, both projective and non-projective. All versions are essentially incremental, that is, they process sentences from left to right. However, there is a fundamental difference, namely that while Nivre's algorithms have linear time complexity ($O(n)$) with respect to sentence length, Covington's have quadratic ($O(n^2)$). Any parsing algorithm can be combined with either Memory-Based Learning (MBL) or Support Vector Machines (SVM) using the implementations TIMBL (Daelemans and Van den Bosch 2005) and LIBSVM (Chang and Lin 2001), which are the two currently incorporated machine learners in MaltParser.

MaltParser can be executed in two different modes: training and parsing. In training mode, Nivre's algorithms require a treebank as input conforming to the constraints imposed by definition 2.2 and 2.4. For a selected parsing algorithm, machine learning method and feature set, a model is learned. The model is utilized in parsing mode for constructing dependency trees according to the treebank annotation and the output follows definition 2.2 and 2.4. As we will see in chapter 4, the projectivity constraint can actually be dropped during training, but it results in a situation where it is not completely clear what the parser actually learns.

Version 0.1 of MSTParser can also be executed in training and parsing modes. As MaltParser, it uses a supervised training method. Both the Eisner and the Chu-Liu-Edmonds parsing algorithms are implemented as (directed) maximum spanning tree algorithms, which requires that scores, or weights, are assigned to all permissible arcs of a sentence to parse. During training, the basic idea is to extend the Margin Infused Relaxed Algorithm (MIRA), which is an algorithm for learning structural outputs. This case, it is applied to dependency structure. The general idea is to learn the weights that are assigned to arcs during parsing. MIRA considers a single training instance at a time, in this case a dependency structure of a training sentence, where each training sentence is parsed with the current weights. With a loss function such as the proportion of words with incorrect head, the weights are tuned. This updating is done iteratively where the training data in principle can be processed a number of times. MIRA attempts to keep the new weights as close as possible to the old weights.

It is worth noting that in contrast to MSTParser, which considers more than one possible dependency tree during parsing, MaltParser is completely

deterministic. On the other hand, while MaltParser can assign labels to dependency relations, this is usually done in a second phase in order to achieve high labeled accuracy for MSTParser. Unlike MSTParser, MaltParser can easily condition on the labels of the previously inserted arcs. As mentioned above, Nivre’s algorithms have linear time complexity, while the algorithm for finding the maximum spanning tree has quadratic running time in relation to the number of words.

4.1.3 Evaluation

In all three experiments, the main evaluation metric is *attachment score* (AS). This metric comes in two versions, *unlabeled* attachment score (AS_U) and *labeled* attachment score (AS_L), where the former is the proportion of tokens with the correct head and the latter the proportion of tokens with the correct head and dependency type.

Another adopted metric is *exact match*, which also comes in an unlabeled (EM_U) and labeled (EM_L) version. Unlabeled exact match means the proportion of sentences where all tokens in the sentence have the correct head, and for labeled exact match the label for each token must also be correct. It is much harder to score well on exact match than on attachment score.

The accuracy for tokens directly involved in the various transformations, such as the accuracy for auxiliary verbs or tokens with a non-projective arc to its head, will also be evaluated. This accuracy is divided into two metrics:

- Recall: the proportion of tokens with the correct head, that in the gold-standard trees were identified as e.g. auxiliary verbs, with or without the correct dependency type. This is known as labeled (R_L) and unlabeled (R_U) recall.
- Precision: the proportion of tokens with the correct head, that in the dependency trees produced by the parser were identified as e.g. auxiliary verbs, with or without the correct dependency type. This is known as labeled (P_L) and unlabeled (P_U) precision.

Different languages, or treebanks, are for various reasons not equally hard to parse. In order to even out these differences in accuracy, *error reduction* can be used to facilitate comparison of the same transformation on different treebanks. Error reduction is defined as: $ER = (m_x - m_y)/(1.0 - m_y)$, where m_x and m_y could be any labeled or unlabeled measurement (AS, EM, P, R) for two different transformations x and y . Moreover, McNemar’s test is applied in order to verify whether a difference in accuracy between different

transformations of the same data set is statistically significant. This can also be carried out for any labeled or unlabeled metrics.³

The less training and test data, the less reliable the comparison. Therefore, ten-fold cross-validation was conducted during the development phase in order to obtain a more reliable evaluation for the smaller treebanks in the CoNLL data. They have been divided into ten equally large sets in a pseudo-randomized way. Every sentence i with the same value for $i \bmod 10$ will belong to the same set. For instance, sentences 1, 11, 21, ... constitute the first set. The parser is trained ten times, where all ten sets are excluded from the training data once while the excluded set is the test set. The various metrics are presented by computing the mean and standard deviation of the ten test sets. Ordinary training was used in Experiment II for the larger treebanks during the development, 20% (set 0 and 9) of the CoNLL data as test data and the other 80% (1 to 8) as training data.

4.2 *Experiment I: Treebank Transformations*

The transformations for non-projectivity, coordination and verb groups are algorithmic and fairly simple. They can certainly be improved by incorporating more code for dealing with more special cases, which there are lots of especially for coordination. Another possibility would be to rely on some kind of machine learning for the inverse transformation in order to improve the quality of the inverse transformation, something that is worth investigating especially for non-projectivity.

The implemented transformations and inverse transformations are not perfect, they always distort the data to a higher or lower degree. The question pursued in this experiment is how much the various transformations themselves introduce errors, which is of interest in the evaluation of the parsing experiments. All five previously introduced treebanks from CoNLL shared task are used. The experimental procedure is as follows:

1. Apply a transformation to a complete training data set of a treebank.
2. Apply the corresponding inverse transformation to the output of step 1.
3. Compute AS by comparing the original untransformed treebank with the output of step 2.

³The evaluation script, provided by the organizers of the CoNLL shared task, excludes punctuation tokens. It does not explicitly reveal which tokens are excluded, which unfortunately disables the use of McNemar's test for AS in experiment II.

The figures presented below are constrained to AS_U , mainly because the corresponding AS_L figures do not contribute any new interesting results (besides being slightly lower than the AS_U figures). Since all five treebanks contain non-projectivity, the AS_U for all of them will be presented below. German is excluded below for the coordination transformation **PS-to-MS** and back, and only Czech and Slovene is of interest for the verb group transformations **PS-to-MS** and back.

4.2.1 Impossible Structures

Table 4.2 contains the AS_U figures for the four pseudo-projective transformations **BASELINE** (p_0), **HEAD** (p_H), **PATH** (p_P), **HEAD+PATH** (p_{HP}), with the AS_U for the non-projective arcs alone in parenthesis. One thing to note is that 100% minus p_0 equals **%-NPA** in table 4.1 for the five treebanks. Since p_0 simply projectivizes the data without any attempt to perform an inverse transformation, each non-projective arc counts for one AS_U -error. For the other three transformations, it can be concluded that the most informative transformation, p_{HP} , is also the most accurate one. The distortion introduced is for all treebanks virtually negligible, with no result lower than 99.98% overall AS_U (for Alpino), or 99.3% AS_U for non-projective arcs alone (for Czech). It is worth noting that despite the fact that there is a only linear increase in the number of pseudo-projective dependency labels for p_P , it has a comparable (SDT and PDT) or higher (PADT, Alpino and Tiger) AS_U than p_H .

A general and quite expected observation is that the more non-projectivity, the lower overall AS_U . Alpino has the lowest overall AS_U for all pseudo-projective encodings, probably due to the highest proportion of non-projectivity, whereas PADT with the lowest proportion of non-projectivity has the highest AS_U . However, for the AS_U -figures for the non-projective arcs alone, the situation is in fact reversed. The inverse transformations for τ_{p_H} , τ_{p_P} and $\tau_{p_{HP}}$ manage to correctly recover most non-projective arcs for Alpino. PADT, on the other hand, has a very low AS_U of 73.3% for p_H , but has AS_U -figures for p_P and p_{HP} that are comparable to the other treebanks. A possible reason for this result for PADT is that it has the most skewed distribution of dependency labels. As much as 35% of all arcs have the label *Atr*, which is more than for any other treebank.⁴ This is more problematic for p_H , since the number of possible syntactic heads increases for the numerous non-projective arcs with a syntactic head labeled *Atr*. The two other transformations are less sensitive to this, as the path back to the correct syntactic head is clearly marked.

⁴Also, there are actually 36% non-projective arcs with a (syntactic) head labeled *Atr*.

	p_0	p_H	p_P	p_{HP}
SDT	98.24	99.82 (90.1)	99.82 (90.0)	99.99 (99.4)
PADT	99.61	99.89 (73.3)	99.96 (90.3)	100.00 (99.5)
Alpino	94.68	99.47 (90.2)	99.74 (95.2)	99.98 (99.7)
Tiger	97.74	99.76 (89.5)	99.83 (92.7)	99.99 (99.5)
PDT	98.11	99.86 (92.5)	99.86 (92.7)	99.99 (99.3)

Table 4.2: Distortion results for the pseudo-projective transformations with figures for AS_U ; in parenthesis: AS_U for non-projective arcs.

					# Lifts			
	p_0	p_H	p_P	p_{HP}	1	2	3	>3
SDT	27	129	66	149	88.4	9.1	1.7	0.84
PADT	28	82	61	99	66.5	14.4	5.2	13.9
Alpino	26	112	70	148	84.6	13.8	1.5	0.07
Tiger	46	268	117	329	83.6	14.8	1.4	0.16
PDT	83	527	206	643	93.8	5.6	0.5	0.1

Table 4.3: Number of distinct dependency relations for pseudo-projective parsing.

Two other variables that may affect the amount of distortion, are presented in table 4.3. One is the distribution of the number of lifts that non-projective arcs have to undergo in order to find their linear heads. One can expect that the more deeply nested a non-projective arc is, the harder it will be to correctly find its syntactic head. The figures reveal some interesting differences between the treebanks. For example, 93.8% of all arcs in PDT require only one lift before they become projective, whereas the corresponding figure for PADT is as low as 66.5%. PADT also has a very high proportion of very deeply nested non-projective arcs (>3) in comparison to the other treebanks. These facts are problematic for the breadth-first approach of the inverse transformation for PADT, since the search for the syntactic head investigates depth 1 first before moving on to depth 2 followed by depth 3, and so on. It is worth noting that the Danish Dependency Treebank (Kromann 2003), for which an improvement in accuracy was not observed in Nivre and Nilsson (2005), is both larger and less deeply nested than PADT. One can therefore anticipate that a positive effect of the pseudo-projective parsing will be hard to achieve.

The other variable is the number of distinct dependency labels, which

of course is related to the distribution of dependency labels discussed above for PADT. In this particular experiment, a large set of dependency labels, evenly distributed over all arcs, should provide the best conditions to achieve as little distortion as possible. For instance, a more fine-grained set of dependency labels for *Atr* in PADT is likely to decrease distortion. However, this is usually not a beneficial property during parsing. The task of assigning the correct dependency label to arcs becomes harder as the number of dependency labels increases, and as the number of training instances for each dependency label decreases. This holds irrespectively of whether parsing is pseudo-projective or not.

The first column of table 4.3 contains the number of distinct dependency labels for p_0 . These figures are always the same as before the transformation, since p_0 does not extend the set of dependency labels. However, one interesting thing to note is the number of distinct pseudo-projective dependency labels for the other encodings. The theoretical upper bound on the number of dependency labels for the two transformations with a quadratic increase (p_H and p_{HP}) is far higher than what the figures are in practice. For instance, 643 pseudo-projective dependency labels for p_{HP} and PDT is a very large set, but it is far less than 13944, the theoretical upper bound given 83 original dependency labels (cf. table 3.1). Fortunately for the parser, there is a huge number of combinations that for linguistic reasons, or data sparseness, never occur.

4.2.2 Hard Structures

With some exceptions, the results above show that the pseudo-projective transformations tend to be quite free from distortion. The corresponding overall AS_U figures for the coordination and verb group transformations are presented in the left and right table of table 4.4. As mentioned in chapter 3, preliminary experiments indicated that the **PS-to-CS** transformation was less successful compared to the **PS-to-MS** transformation for PDT, which is the reason why only results for the latter are presented in the table and throughout this chapter.

A comparison between the four versions for coordination transformations reveals that the simplest version (τ_c), which does not extend the set of dependency types or uses predefined lifts for coordination dependents, has with a quite large margin the lowest AS_U .⁵ In other words, a substantial ER is obtained in this experiment by the two types of modifications designed to deal with dependents of coordinations and conjuncts. For instance, extend-

⁵Alpino does not distinguish between coordination dependents and conjunct dependents (cf. section 3.2), making all except τ_c unnecessary.

	τ_c	τ_{c^*}	τ_{c+}	τ_{c+^*}		τ_v
SDT	97.24	98.03	99.10	98.89	SDT	99.82
PADT	97.25	97.61	98.23	98.13		
Alpino (MS)	99.40	-	-	-		
Alpino (CS)	99.71	-	-	-		
PDT	97.77	98.50	99.23	99.26		
					PDT	100.00

Table 4.4: Distortion results for coordination (left) and verb groups (right), with figures for AS_U .

ing the set of dependency labels for PDT result in 69% fewer errors, 67% for SDT, and 35% for PADT. Whether this has a positive impact during parsing is investigated in experiment II.

The figures also indicate that the less sophisticated solution, the predefined lifts τ_{c^*} , is less accurate compared to extending the set of dependency labels. It does, however, have the advantage that no additional burden is laid on the parser to make the distinction between dependents of coordination and conjuncts. Its influence is also investigated in experiment II. Moreover, from the coordination figures, these observations hold as well:

1. **CS** introduces less distortion than **MS** for Alpino,
2. which, in turn, introduces less distortion than for PDT,
3. which, in turn, introduces less distortion than for SDT and PADT.

The transformation back and forth for **CS** is less complicated than for **MS**, due to the annotation of commas in coordination in Alpino. As they are not part of the actual coordination, it conforms better with the status of commas in **CS** compared to **MS**. Observation (2) can probably be attributed to the fact that coordination is annotated in a simpler fashion for Alpino than in the three other treebanks using the Prague-school annotation. Empirical observations during the implementation of the transformations indicated that the number of special cases is smaller in Alpino than in the others. A possible explanation can be that Alpino has been converted to dependency structure. This may imply that different coordination types with different structures in the original data, receive the same structure in the converted data. The other treebanks, on the other hand, were designed to be dependency treebanks from the beginning, which eliminates this risk. A plausible explanation for (3) is that the coordination transformations were originally designed for PDT, making them more biased to the intrinsic properties of coordination in PDT.

4.3. Experiment II: Treebank Independence

As mentioned in section 3.2, conjuncts in SDT and PADT are not explicitly marked with the suffix *_Co*, as in PDT. To investigate the expected increased distortion due to this, a simple experiment was conducted on PDT using τ_c . The same heuristic rules for identifying the conjuncts in SDT and PADT were applied to PDT after removing all occurrences of *_Co*. The accuracy for this transformation was 97.69%, which is very close to the corresponding figure of 97.77% in the table, using the suffix information.

The left table reveals that the distortion of the verb group transformations for SDT and PDT is very close to negligible, especially for PDT. The complexity and number of special cases for this transformation is much lower compared to the coordination transformations, which can explain the difference in accuracy. The difference between the two treebanks can probably again be attributed to the fact that the transformations are slightly biased towards PDT, which they originally were designed for. Finally, the results for PDT here are in line with the corresponding results in Nilsson et al. (2006), for both coordination and verb groups, even though it is not exactly the same data set.

4.3 Experiment II: Treebank Independence

The course of action in this experiment was presented in the beginning of chapter 3. It is duplicated here for convenience:

1. Apply the tree-transformation to the training data.
2. Train a parser model using the transformed training data.
3. Parse a text using the parser model.
4. Apply the corresponding inverse transformation to the output of the parser.

The parser of step 3 will here be MaltParser. As mentioned in subsection 4.1.2, MaltParser is a history-based parser, and requires features of the derivation history to predict the next parser action. Experiment II will be constrained to one set of features per treebank. The experiments will also be constrained to SVM, as it in previous parsing studies (e.g. Hall et al. 2006) have delivered the best results, and to one set of SVM parameter values per treebank. It should be noted that modifications in the training data are likely to change the optimal combination of feature set and SVM parameter values, but this has been an issue of lower priority in the experiments below. The feature sets and SVM parameter values are listed in Appendix A.

		None	p_0	p_H	p_P	p_{HP}
SDT	AS_U	77.31	76.63	77.11	77.07	77.04
	σ_{AS_U}	0.42	0.36	0.37	0.33	0.37
	AS_L	68.76	68.39	68.90	68.83	68.82
	σ_{AS_L}	0.50	0.39	0.41	0.40	0.43
PADT	AS_U	76.96	77.07	77.07	76.94	77.06
	σ_{AS_U}	0.15	0.22	0.22	0.21	0.16
	AS_L	65.78	66.01	65.95	65.88	66.02
	σ_{AS_L}	0.27	0.34	0.32	0.31	0.27
Alpino	AS_U	82.75	83.28	86.77	87.03	87.08
	σ_{AS_U}	0.09	0.13	0.10	0.08	0.08
	AS_L	79.87	80.28	83.25	83.39	83.50
	σ_{AS_L}	0.10	0.14	0.13	0.10	0.10
Tiger	AS_U	88.23	88.54	89.58	89.61	89.74
	AS_L	85.63	85.81	86.87	86.81	86.97
PDT	AS_U	83.41	83.32	84.28	84.42	84.38
	AS_L	76.98	76.94	77.84	78.00	77.94
Average	AS_U	81.73	81.77	82.96	83.01	83.06
Average	AS_L	75.40	75.49	76.56	76.58	76.65

Table 4.5: Parsing results on development sets, or as a result of cross-validation, for the pseudo-projective transformations using MaltParser.

Ten-fold cross-validation was performed during the development phase for the three smallest treebanks (SDT, PADT and Alpino), which is why standard deviation (denoted σ below) is computed in addition to the mean.

The first two subsections cover impossible structures (4.3.1), followed by the hard structures (4.3.2). This section ends with final tests for the combinations of the best transformations for each treebank (4.3.3), followed by some more detailed experiments (4.3.4).

4.3.1 Impossible Structures

The question in focus here is, whether any difference in accuracy between the various transformation versions can be detected, and whether the effect of the pseudo-projective transformations are treebank and language independent.

Table 4.5 presents the overall attachment score results for all five treebanks. The column marked **None** contains the result when the training data sets have not been projectivized and with no inverse transformation of the output of the parser. The bold figures mark the best accuracy for each row.

4.3. Experiment II: Treebank Independence

The first thing to note is that the pseudo-projective baseline (p_0) which projectivizes the training data without trying to conduct an inverse transformation, compared to **None** results in the highest improvement for Alpino. There are corresponding improvements for PADT and Tiger, although less prominent, which can be a result of the higher proportion of non-projectivity in Alpino. The impact of the baseline transformation for SDT and PDT is therefore somewhat contradictory, especially since it gave an improvement in AS_L of 0.7%-points for PDT in Nivre and Nilsson (2005). It is, however, important to remember that there are at least four things that are different compared to that study, (1) the machine learning method, (2) division of the training and testing data, (3) the test data contains gold standard tags in the CoNLL shared task but not in the earlier study, and (4) the information that the feature set looks at is not the same due to different data representations (the original format versus the CoNLL format).

A comparison between p_0 on the one hand, and p_H , p_P and p_{HP} on the other, reveals that both AS_U and AS_L for all treebanks increase, with PADT as an exception. The highest improvement is obtained for Alpino (22.7% ER between p_0 and p_{HP}), which again can be explained by its highest proportion of non-projectivity. Even AS_L for SDT increases compared to **None**, although AS_U is still lower. The absence of positive result for PADT is understandable given the characteristics of PADT presented in the previous section (4.2), for instance the deeply nested non-projectivity. Nevertheless, the overall results clearly show that the parser successfully can learn to correctly identify some lifted dependency relations and/or the lifted path, and that the inverse transformation to a large extent correctly can recover non-projectivity, yielding an overall improved accuracy.⁶

When p_H , p_P and p_{HP} are compared with each other, no clear winner can be chosen. If a difference would emerge, a conceivable hypothesis is that the more training data, the better p_{HP} works, since it is the most informative encoding. However, this cannot be verified, as the largest treebank (PDT) has the highest accuracy for p_P , whereas p_{HP} shows the best results for at least Alpino and Tiger. Of course, the differences are for all languages too small to be statistically significant. It is worth noting that it was p_H which had the best accuracy in (Nivre and Nilsson 2005), but the difference was again too small to be statistically significant.

Another possibility can be that a high proportion of non-projectivity is more beneficial for the most informative encoding. This will remain a hypothesis until further experiments have been conducted.

⁶The pseudo-projective parsing approach also improves accuracy to a higher a lower degree for other languages, such as Danish, Portuguese and Turkish, as reported by Hall and Nilsson (2006).

		None	τ_c	τ_{c+}	τ_{c^*}	τ_{c+^*}	τ_v
SDT	AS _U	77.27	79.33	79.24	79.05	78.99	77.92
	σ_{AS_U}	0.38	0.31	0.33	0.32	0.32	0.49
	AS _L	69.19	70.45	70.26	70.18	70.10	69.36
	σ_{AS_L}	0.42	0.38	0.39	0.39	0.39	0.46
PADT	AS _U	76.96	78.81	78.70	79.05	78.73	-
	σ_{AS_U}	0.15	0.24	0.23	0.25	0.29	-
	AS _L	65.78	67.33	67.23	67.61	67.39	-
	σ_{AS_L}	0.27	0.32	0.30	0.34	0.38	-
Alpino (MS)	AS _U	82.75	83.15	-	-	-	-
	σ_{AS_U}	0.09	0.11	-	-	-	-
	AS _L	79.87	80.21	-	-	-	-
	σ_{AS_L}	0.10	0.11	-	-	-	-
Alpino (CS)	AS _U	82.75	83.38	-	-	-	-
	σ_{AS_U}	0.09	0.14	-	-	-	-
	AS _L	79.87	80.49	-	-	-	-
	σ_{AS_L}	0.10	0.13	-	-	-	-
PDT	AS _U	83.41	85.38	85.51	85.34	85.46	83.58
	AS _L	76.98	77.15	77.22	77.12	77.18	77.11

Table 4.6: Parsing results of development sets for coordination and verb group transformations using MaltParser, excluding punctuation.

Before ending this part, it is interesting to compare the figures for standard deviation between the three treebanks for which cross-validation was performed. They confirm the well-known fact that the more available data, the more reliable results. SDT, the smallest treebank, exhibits a large variation in accuracy between the ten parts of the cross-validation, which is reflected in the standard deviation for both AS_U and AS_L. As the amount of data grows, the standard deviation decreases, which happens for PADT and Alpino. One can therefore expect that corresponding cross-validation experiments for Tiger and PDT have even lower standard deviation than Alpino.

4.3.2 Hard Structures

As the results above show, pseudo-projective parsing can be beneficial for improving parsing accuracy. The question that will be answered here is whether the transformations for the hard structures, coordination and verb groups, can be beneficial for parsing accuracy. Table 4.6 contains the overall results of the experiments, where the **None**-column has the same values as in table 4.5.

A brief look at the figures indicates that the improvement in accuracy in

4.3. Experiment II: Treebank Independence

Nilsson et al. (2006) is confirmed. All versions of the coordination transformations from **PS** to **MS** and back during parsing increase accuracy, not only for PDT but for the other four treebanks as well. Since SDT, PADT and PDT have comparable proportions of coordination and since the coordination transformations for PDT are more accurate than for SDT and PADT (table 4.4), it is not that surprising that PDT exhibits the highest ER of 15.0% for AS_U , compared to 9.1% for SDT and 10.0% for PADT. The result is the opposite for AS_L , but this can probably be attributed to the fact that PDT has the suffix *_Co* on the conjuncts' dependency labels. The inverse transformation has to augment the dependency labels with the suffix, which is not a trivial issue (although linguistically more informative).

The same coordination transformation for Alpino, **Alpino (MS)**, results in a higher accuracy as well, although less prominent than for the others (2.4% ER for AS_U). This is quite expected, since the proportion of coordination is lower than for the others. However, it is interesting to note that the **CS**-style transformation for Alpino yields an even higher accuracy than **MS**. It has an ER for AS_U of 3.8%, which is comparable to SDT and PADT, as Alpino has slightly less than half the proportion of coordination.

A comparison between the extended (τ_{c+} , τ_{c+^*}) and unextended (τ_c , τ_{c^*}) versions gives no clear answers. The extended ones have slightly lower accuracy than their corresponding unextended ones for SDT and PADT, whereas the opposite holds for PDT. This may indicate that the amount of training data may be of importance for the extended versions, but the differences in accuracies are too small to say anything conclusive.

A conclusive answer is also hard to give when comparing the versions with predefined lifts (τ_{c^*} , τ_{c+^*}) with those without (τ_c , τ_{c+}). They are beneficial for PADT but not for the others, which may appear contradictory given the clear advantage of τ_{c^*} and τ_{c+^*} in experiment I. It is, however, important to note that punctuation is included in that experiment, which is not the case here (to enable comparison with the results of the CoNLL shared task). The increase in accuracy does not take place because the majority of the tokens that contribute to the higher accuracy in experiment I for τ_{c^*} and τ_{c+^*} are in fact punctuation tokens. The results in Nilsson et al. (2006) on the other hand do include punctuation for PDT, which clearly boosts accuracy for the predefined lifts. Nevertheless, one can argue that the coordination transformations are treebank and language independent, resulting in an improved accuracy which cannot be neglected.

The language independence holds for the verb group transformation too, even though it gives less improvement compared to the coordination transformation. The results confirm the boosted accuracy for PDT in Nilsson et al. (2006). The ER is actually higher for SDT than PDT (2.9% and 1.0% for

AS_U , respectively), despite the fact that the transformation originally was designed for PDT, and thus distorts the PDT-data less. But since SDT contains more than six times as many arcs in verb groups (table 4.1), the result is not that surprising.

A comparison between the two types of transformations for the hard structures indicates that the gain in overall ER is higher for coordination than for verb groups, even if the figures are normalized according to frequency. Furthermore, since the linguistic arguments for coordination and verb groups are easier to motivate syntactically in **MS** than in **PS** (see sections 2.2 and 2.3), the experiments indicate that data-driven syntactic parsing should be based on syntactically annotated treebanks, at least for MaltParser.

4.3.3 Combining Transformations

The best of each type of transformation for each language will here be combined in order to check whether the transformations give an accumulative improvement. The best figures for each type of transformation are in bold-face in table 4.2 and 4.6. The outcome of this is shown in table 4.7. The experiment was performed on the development sets (**Dev**),⁷ as well as one parse for each treebank on the held out test sets used in the final tests in the shared task (**Eval**). The next column (**N. et. al.**) contains the results reported in Nivre et al. (2006) for the same held out test sets as in the **Eval**-column, which all used only the pseudo-projective transformation p_H . The last column shows the best individual results for each treebanks reported in the CoNLL shared task.

A look at the figures in the **Dev**-column shows, for example, that the combination of coordination and verb group transformations is higher than when they are applied individually for both SDT and PDT. For PADT, Alpino and PDT, the sum of the increases of the individually applied transformations is slightly lower than the increase of the combination, but actually vice versa for SDT, having a marginally better result. This indicates quite clearly that these types of transformations are not mutually harmful, but in fact can be combined with good result.

The final test using the held out test sets of the shared task further strengthens all previously conducted experiments during the development phase. AS_U and AS_L for all treebanks are higher in the **Eval**-column than in the **N. et. al.**-column.

⁷The figures for Tiger in the **Dev**-column are the same as in table 4.2.

4.3. Experiment II: Treebank Independence

	Trans.		Dev	Eval	N. et.al.	Best
SDT	$\tau_v \circ \tau_c$	AS _U	80.40	82.01	78.72	83.17
	$\tau_v \circ \tau_c$	AS _L	71.06	72.44	70.30	73.44
PADT	$\tau_{c^*} \circ p_0$	AS _U	78.97	78.56	77.52	79.34
	$\tau_{c^*} \circ p_0$	AS _L	67.63	67.58	66.71	66.91
Alpino (CS)	$\tau_c \circ p_{HP}$	AS _U	87.63	82.85	81.35	83.57
	$\tau_c \circ p_{HP}$	AS _L	84.02	79.73	78.59	79.19
Tiger	p_{HP}	AS _U	89.74	89.08	88.76	90.38
	p_{HP}	AS _L	86.97	86.20	85.82	87.34
PDT	$\tau_v \circ \tau_{c^+} \circ p_P$	AS _U	85.72	85.98	84.80	87.30
	$\tau_v \circ \tau_{c^+} \circ p_P$	AS _L	78.56	78.80	78.42	80.18

Table 4.7: Parsing results when combining transformations, where **Dev** = development test, **Eval** = CoNLL test set., **N. et al.** = results of Nivre et al. (2006), **Best** = best result in the CoNLL shared task.

	Precision			None	Recall			
	p_H	p_P	p_{HP}		p_0	p_H	p_P	p_{HP}
SDT	71.6	71.2	60.3	8.9	7.0	32.1	32.5	32.3
PADT	17.3	14.2	13.7	14.4	14.5	15.6	15.3	14.3
Alpino	71.6	74.8	77.2	2.5	0.7	60.8	63.8	66.0
Tiger	67.4	66.8	74.1	17.3	5.6	50.3	51.4	54.7
PDT	69.7	61.6	72.7	11.5	6.4	57.3	54.4	59.4

Table 4.8: Precision (P_U) and recall (R_U) for arcs marked as lifted.

4.3.4 Detailed Experiments

The main goal of this thesis is to investigate whether the transformations can improve parsing accuracy. The above experiments confirm that this goal has been reached for MaltParser, even though the positive effect of the transformations vary. This subsection will study the previous results in more details for a somewhat more in-depth analysis of these variations. For example, how is the accuracy for the arcs directly involved in the transformations affected, such as the non-projective arcs, or the auxiliary verbs? Another question that already has been touched upon is why the pseudo-projective transformations are more beneficial for Alpino, Tiger and PDT compared to SDT and PADT. To begin, precision and recall for non-projective arcs for the parsing experiment of table 4.5 are shown in table 4.8, which sheds some light on these issues for the pseudo-projective transformations.

Precision and Recall Any arc that is identified as lifted (i.e. if the arc label contains \uparrow) in the parser output is part of the set of arcs used for computing precision. As the various encodings identify different arcs as lifted, precision is difficult to compare for the encodings. Because **None** and p_0 do not identify lifted arcs in the parser output, precision for them has deliberately been excluded from the table. This is not a problem for recall, as the set of lifted arcs of the gold-standard trees is identical for all encodings.

The figures for recall reveal that the parser trained on non-projective data (**None**) actually manages to assign the correct head to a quite high proportion of non-projective tokens, for instance as much as 17% for Tiger. Whether this is a good or bad result for Tiger is actually hard to determine, since it must inevitably mean that these non-projective tokens have (several) erroneous neighboring arcs.

The differences in recall between **None** and p_0 are for all treebanks except PADT in favor of the former. This may seem contradictory for SDT, Alpino and Tiger, having higher AS_U and AS_L for p_0 than **None**. However, p_0 simply distorts the dependency trees by replacing linguistically correct (non-projective) arcs with linguistically incorrect (projective) arcs. The non-projective arcs are likely to have several similar projective arcs in the same data, i.e. with the similar types of heads and dependents. The projective ones certainly can help the parser assign a correct arc for the similar non-projective ones (at the expense of the surrounding arcs), and vice versa. In combination with the fact that the parser does not necessarily end up in strange states during training for the non-projective arcs, but instead for the surrounding (projective) arcs, this may imply that **None** is less forgiving for the many surrounding arcs than p_0 . On the other hand, due to the absence of arcs between the real heads and dependents for non-projective arcs, p_0 is more merciful to (the many) surrounding arcs than for (the fewer) non-projective arcs.

The recall results for the three other encodings exhibit higher values compared to **None** and p_0 for all treebanks, again with PADT as the expected exception. For the others, Alpino has the highest increase and SDT the lowest, which correlates quite well with the corresponding figures in table 4.2. This again indicates that the amount of training data, both the total amount of data and the amount of non-projectivity, has an impact on the parser's ability to conduct pseudo-projective parsing (as well as how deeply nested non-projectivity is). Moreover, the differences in recall for the three encodings p_0 , p_H and p_P are statistically significant compared to both **None** and p_0 , but not when comparing p_0 , p_H , and p_P with each other (McNemar's test, $p < 0.01$).

Whereas recall for the treebanks ranges from less than 35% to above 65%,

4.3. Experiment II: Treebank Independence

it is interesting to note that the precision figures are much closer to each other (again excluding PADT). Alpino still has the highest values, but it seems that precision is less sensitive to the total amount of data and the proportion of non-projectivity. Moreover, all precision values are slightly higher than the corresponding value for recall. All three parsers of the three encodings for all treebanks mark fewer arcs as lifted than there are non-projective arcs present in the treebank. Together with the low precision for PADT, this indicates that it is mostly the simpler (shallowly nested) non-projective arcs that are correctly marked as lifted by the parsers, which usually is followed by a subsequent successful search for the syntactic heads.

The corresponding values for coordination and verb groups are presented in table 4.9, with precision and recall for conjuncts and auxiliary verbs. The relationship between precision and recall is the same as for non-projective arcs, that is, higher precision than recall. The number of tokens marked as conjuncts and auxiliary verbs by the parsers are lower than the actual number of conjuncts and auxiliary verbs. As for non-projectivity, this again may indicate that the parsers can handle the “simpler” cases better than without transformations, while it still fails to find even the more complicated ones. This is, however, less clear for verb groups, at least for SDT since its precision and recall are virtually unaffected by the transformation.

Although the increase in accuracy is lower for conjuncts than for non-projective arcs, for obvious reasons, the better precision and recall with the transformations are relatively good. The differences between the values for **None** and their corresponding values for **Tr** are statistically significant (marked with *) for all cases except SDT and recall for Alpino, using McNemar’s test ($p < 0.01$).

Learning Curves To test the above presented hypothesis that more data gives better parsing results for pseudo-projective transformations, a learning curve experiment was performed, constrained to Alpino and Tiger. The training set for Alpino was divided into the same ten parts as during the cross-validation, with 0 and 9 for testing and $1 \dots n, 1 \leq n \leq 8$ for training. To normalize the values, compensating for the increasingly higher accuracy with more data, the AS_U results are presented as error reduction (ER). Table 4.10 contains all the values for the learning curve experiments, and is illustrated in the diagram of figure 4.1.

The dotted curves represent Alpino and unbroken curves Tiger, where ER for the four encodings p_0, p_H, p_P and p_{HP} is computed in relation to no transformation (**None**). For both Alpino and Tiger, the ER for p_0 is unaffected by the amount of data. While the ER varies slightly, it turns out that the ER is virtually the same for 10% as training data as for 80%, which

Set			Precision		Recall	
			None	Tr	None	Tr
<i>C</i>	SDT	τ_c	57.3	68.7*	46.2	60.3*
	PADT	τ_{c^*}	59.6	68.5*	48.2	64.8*
	Alpino	τ_c	77.9	84.1*	74.0	74.5
	PDT	τ_{c+}	66.8	79.0*	62.9	77.7*
<i>A</i>	SDT	τ_v	94.2	94.9	94.0	93.3
	PDT	τ_v	88.1	92.7*	86.4	89.0*

Table 4.9: Precision (P_U) and recall (R_U) for conjuncts *C*, and auxiliary verbs *A*, where **None** means that no transformation has been applied, and **Tr** that the transformation shown for each tree-bank is applied. The symbol * marks a statistically significant difference between **None** and **Tr** (McNemar's test, $p < 0.01$).

		10%	20%	30%	40%	50%	60%	70%	80%
Alpino	p_0	3.9	4.5	5.1	3.1	5.1	4.0	3.9	4.1
	p_H	16.5	19.2	20.2	19.3	21.0	21.9	22.0	23.1
	p_P	17.2	19.7	22.0	20.5	23.7	22.3	23.6	24.9
	p_{HP}	17.7	20.3	22.4	21.2	22.9	23.0	24.4	24.8
Tiger	p_0	2.3	2.7	1.7	3.6	2.2	2.0	2.0	2.6
	p_H	5.8	8.1	8.5	10.2	9.8	10.6	10.4	11.5
	p_P	5.4	7.8	8.6	10.5	10.7	10.7	11.4	11.7
	p_{HP}	5.8	7.9	9.2	10.7	10.7	11.5	12.6	12.8

Table 4.10: Learning curve figures for Alpino and Tiger measured as ER for AS_U .

is about 4% and 2.5% for Alpino and Tiger, respectively.

However, for p_H , p_P and p_{HP} , which actively try to recover non-projectivity, the learning curves clearly indicate that the amount of data matters. Tiger, starting at approximately 5.5% ER, has an increasing performance, with a more than doubled ER for all data. Alpino, with 36% non-projective arcs, starts at about 17% ER and has a climbing curve up to almost 25% ER.

Although this experiment shows that there is a correlation between the amount of data and the accuracy for pseudo-projective parsing, it does probably not tell the whole story. If it did, one could expect that the ER for p_H , p_P and p_{HP} would be much closer to p_0 when the amount of data is low (to the left in the figure) than they apparently are. Of course, the difference is likely to diminish with even less data, but it should be noted that 10% of Alpino has about half the size of PADT, for which the positive impact of pseudo-projective parsing is absent.

4.3. Experiment II: Treebank Independence

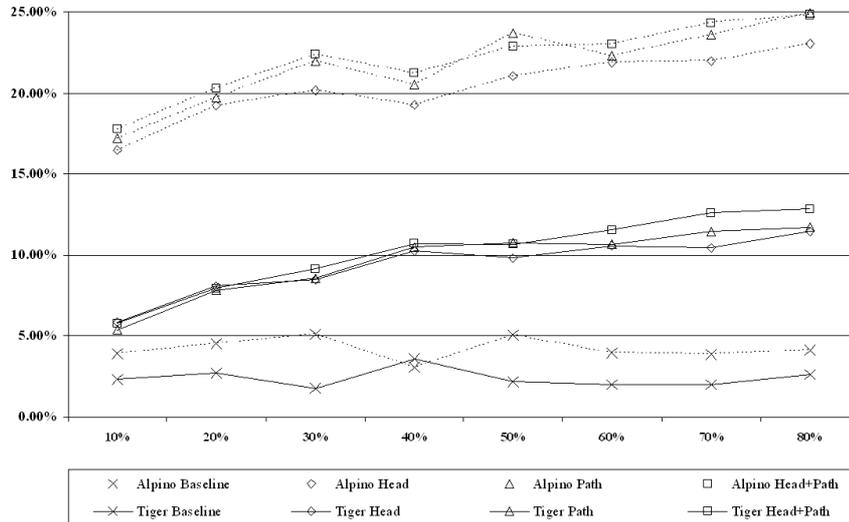


Figure 4.1: Learning curves for Alpino and Tiger measured as ER for ASU.

Why do the Transformations Help? The pseudo-projective transformations increase parsing accuracy because they enable the projective Malt-Parser to correctly parse all arcs, not just the projective ones. But why do the transformations for coordination and verb groups help? The list below contains three possible explanations:

- The average arc length is reduced.
- The average branching factor is reduced.
- The consistency is increased.

Increased consistency is hard to define and measure. It can be exemplified for coordination, for instance by comparing the dependency structures for one sentence, where the predicate has two coordinated subjects, with another one, where the second subject and the conjunction are absent. In **PS**, the first subject attaches to the conjunction in the first sentence, but to the predicate in the second one. On the other hand, in both **MS** and **CS**, the first subject will always be the only token with a direct dependency relation to the predicate, which in a sense increases consistency.

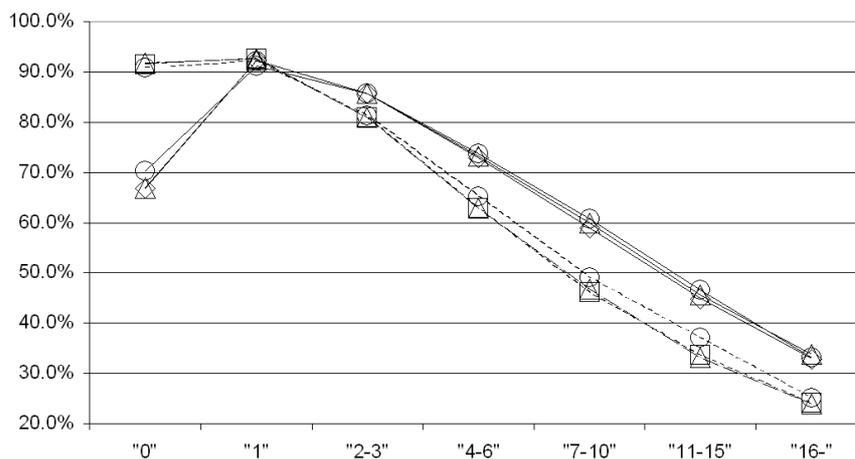


Figure 4.2: Unlabeled precision (unbroken lines) and recall (dotted lines) per arc length for PDT with no (\square), verb group (\triangle) and coordination (\circ) transformations for PDT.

Compared to the third possible explanation, the first two are easy to measure. There is also a correlation between one and two, which will be discussed briefly at the end of this section. The focus here will be on the average arc length, even though the third one probably is an important factor as well. The diagram of figure 4.2 illustrates how AS_U precision and recall vary as the arc length increases for PDT⁸ with no transformation, verb group transformation and the best coordination transformations, where the x-value "0" represents all arcs attaching to the root.

The most important observation here is that both precision and recall drop with increasing arc length. For the curves for untransformed training data, it is in fact only precision for arcs of length 1-3 and recall for arcs of length 0-2 that are above the AS_U of 78.97%. It is likely that all parsers exhibit less accuracy for longer relations, but the quite rapidly decreasing accuracy is probably a consequence of the incremental one-pass parsing strategy of MaltParser, as well as the low precision for root tokens. One can also observe that precision and recall are virtually unaffected by the coordination and verb group transformations, as the curves are very similar. In other words, long relations are roughly equally hard to parse irrespectively

⁸A similar pattern holds for the other treebanks too.

4.3. Experiment II: Treebank Independence

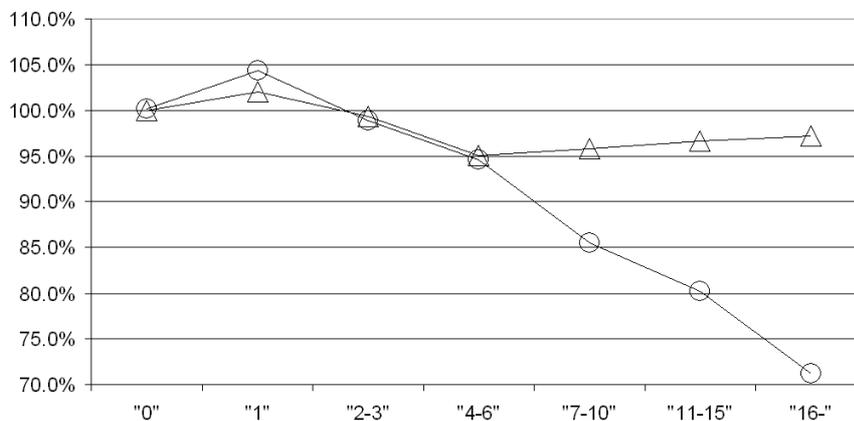


Figure 4.3: Proportions of arcs per arc length for verb group (Δ) and coordination (\circ) transformations for PDT.

of whether the treebanks have been transformed or not.

This property may imply that if one can decrease the average arc length in the training data, the learning task becomes easier. In principle, the average arc length can be reduced to 1, by forming a chain where each arc attaches two neighboring tokens. The parsing task would then be trivial, but the inverse transformation extremely difficult.

The diagram of figure 4.3 plots the proportion of arcs in the transformed training data of PDT compared to the untransformed training data for coordination and verb groups, divided into the same groups of arc lengths as in figure 4.2. It illustrates that the proportion of long arcs have decreased for both coordination and verb groups, whereas arcs of length 1 have increased. This entails that the average arc length is lower for the coordination transformation and verb group transformation. In figures, the average arc length is 2.3 (100.0%) for the untransformed training data, 2.12 (92.2% of 2.3) for the coordination transformation, and 2.25 (98.0% of 2.3) for the verb group transformation. The trend is most prominent for coordination, which partly can be explained by the fact that PDT has more instances of coordination than verb groups, and may indicate that the average arc length has an impact on the difficulty of the learning task.

Before ending this section, it is worth noting that the arc length correlates with the branching factor, at least in any sensible dependency graph. This was discussed for coordination and verb groups in section 2.2 and 2.3, where the lower branching factor of **MS** and **CS** compared to **TS** and **PS** was

emphasized. A low branching factor could be of importance for MaltParser, as it is hard (although possible) to represent more than one child on each side of the topmost stack token in the feature model.

4.4 Experiment III: Parser Independence

The previous experiment shows that an increased accuracy for MaltParser often is the result of the various transformations that have been applied. The question that will be pursued here is whether accuracy can be increased for another parser, MSTParser, in order to investigate the transformations' degree of parser independence.

Since MSTParser version 0.1, presented in McDonald et al. (2005) and McDonald and Pereira (2006), can only handle word forms and one part-of-speech per token, in addition to one arc (head and dependency relation) per token, the full CoNLL format is no longer suitable. PDT in the original format, on the other hand, conforms with both MSTParser and MaltParser and will therefore be used for the coordination and verb group transformations. It is in principle also suitable for the pseudo-projective transformations. However, MSTParser's very high demand of RAM and temporary hard drive space, both of which increase with the distinct number of dependency relations and with the amount of training data, could not be met. The pseudo-projective transformations are therefore constrained to the Alpino CoNLL data, using only the word form and coarse-grained parts-of-speech.

4.4.1 Impossible Structures

MSTParser used in this experiment is of the first order, that is, it does not condition its decision on any surrounding arcs. It comes in two versions, a projective one (the Eisner algorithm) and a non-projective one (Chu-Liu-Edmonds algorithm). The figures for these pseudo-projective experiments are shown in table 4.11.⁹

The first three rows contain the result for the Eisner algorithm using no transformation (**None**), p_0 and p_{PH} . The figures show a very similar pattern to MaltParser, with a boost in accuracy for p_0 compared to **None**, with a significantly higher accuracy for p_{HP} over p_0 . Another thing to note is that MaltParser has higher AS_L and slightly higher AS_U compared to MSTParser (see table 4.5). It is however important to note that the MSTParser

⁹All figures in the table use the following training setting: "training-k:1 loss-type:punc". Also, the figures are unfortunately not completely comparable to the previously presented Dutch results for MaltParser, since MaltParser's features model has access to more information than MSTParser has in this experiment.

4.4. Experiment III: Parser Independence

		AS_U	σ_{AS_U}	AS_L	σ_{AS_L}
Eisner	None	81.79	0.07	77.06	0.09
	p_0	83.23	0.08	78.74	0.11
	p_{HP}	86.45	0.08	81.30	0.09
CLE	None	86.39	0.07	81.64	0.10

Table 4.11: Pseudo-projective parsing for Alpino with cross-validation using MSTParser.

experiments have not been optimized to the same extent as the MaltParser experiments. The relatively better AS_L compared to AS_U for MaltParser compared to MSTParser is in line with the results of the CoNLL shared task, even though that MSTParser was of the second order, looking at contextual arcs (one sibling) (McDonald et al. 2006). MaltParser uses a history-based parsing approach with access to previously assigned dependency labels, which tends to be important for later labeling decisions.

The last row contains the result for Chu-Liu-Edmonds' non-projective algorithm, which accordingly does not require pseudo-projective parsing. A comparison between the Eisner algorithm with p_{HP} and Chu-Liu-Edmonds' non-projective algorithm reveals that pseudo-projective parsing is better than this non-projective parsing approach for AS_U , whereas AS_L is only slightly lower for pseudo-projective parsing, but the differences are not statistically significant.

4.4.2 Hard Structures

This is the subsection for the hard structures for MSTParser. Table 4.12 contains the figures for the conducted parsing experiments. Unfortunately, the same positive effect of the transformations is not recorded for MSTParser. The first row shows the AS_U and EM_U for the Chu-Liu-Edmonds algorithm with no transformation, which are identical to the figures presented by McDonald et al. (2005). The following two rows show the result for the coordination transformation with predefined lifts and the verb group transformation. Whereas the verb group transformation has no influence on the parser accuracy at all, the coordination transformation decreases the accuracy.

A more detailed investigation is required to explain the outcome of the experiment, but one can speculate about the reasons. A plausible explanation is that MaltParser's one-pass incremental parsing approach makes it very aware of the partially constructed dependency structure to the left of the current parsing position, but has no knowledge of the dependency structure

	Trans.	AS_U	EM_U
CLE	None	84.5	32.9
CLE	τ_{c^*}	83.5	32.6
CLE	τ_v	84.5	32.9

Table 4.12: Coordination and verb group transformations for PDT using MSTParser.

to the right. The chains going from left to right for both coordination and verb groups in **MS** therefore postpone crucial decisions. In coordination, for example, the words that are located to the right of the conjunction are often important for the words to the left, and vice versa. The coordination chains help to postpone such decisions. For MSTParser, on the other hand, the same type of problem does not exist. It has in a way the ability to compare the scores of several dependency trees, which seems to make it less sensitive to whether coordination forms chain or not.

Even though the transformations for coordination and verb groups did not result in higher accuracy, it is still possible that there are tree-transformations that are beneficial for MSTParser and not for MaltParser. This, however, is a problem that has not been studied here.

Chapter 5

Conclusion

In this final chapter, the main results and conclusions of chapters 3 and 4 and are presented in the first section, which is followed by some proposals for direction of future work.

5.1 Main Results

In contrast to experiments II and III, experiment I is only indirectly connected to the research questions stated in the introduction. It was performed in order to estimate the amount of distortion that the transformations give rise to. The experiment shows that all transformations introduce errors in the dependency trees. The amount of errors vary with the type of transformation and the properties of the treebanks. For instance, PADT has much more deeply nested non-projective arcs, which negatively affects PADT's accuracy in experiment I. This partly can explain why its parsing accuracy does not increase in the subsequent experiments. Experiment I also reveals that the complexity of coordination is reflected in more distortion for the coordination transformations compared to the transformation verb groups.

As mentioned in the introduction, the primary goal was to investigate how the pseudo-projective transformation influences parsing accuracy for different languages, treebanks and parsers. In case parsing accuracy increased, it is of interest assessing the two notions of independence with respect to language, treebank and parser.

Given the pseudo-projective transformation, the impossible non-projective structures in dependency structure are not impossible to parse for a projective parser. Experiments II and III show that parsing accuracy can be improved for several languages and treebanks, and for more than one parser. To say that the pseudo-projective transformation is completely language and treebank independent is, however, a too strong statement. There are languages and treebanks for which it does not help due to properties like deeply nested non-projectivity and small treebanks, such as for PADT and to some extent SDT. PADT is also a treebank with small amounts of non-projectivity.

However, a conclusion one can draw from experiment III is that the benefit of the pseudo-projective transformations is not restricted to MaltParser, but is at least as great for MSTParser using Eisner's algorithm. As MSTParser is based on a completely different parsing strategy, it is at least likely that the pseudo-projective transformations will work for other inductive dependency parsers as well. This makes the pseudo-projective transformation parser independent with respect to parsing accuracy.

The main outcome of experiment II is that MaltParser has the ability to learn to distinguish arcs that have been lifted from arcs that have not been lifted. The experiments also show that the success of the pseudo-projective transformation during parsing is tied to the proportion of non-projectivity, as the error reduction for Alpino is higher than for Tiger and PDT. The learning curve experiments for Alpino and Tiger imply that it correlates with the total amount of data as well. In other words, the more non-projectivity and data, the better the result. The learning curve experiment together with the precision and recall figures for PADT show that the more complex the non-projective arcs, i.e. the more deeply the arcs are nested, the lower the probability is to accurately recover them. On the other hand, it is likely that any inductive dependency parser will have difficulties dealing with the non-projectivity in PADT.

Since the pseudo-projective transformation improves accuracy under several conditions, the other notion of independence is of interest. That is, can the transformation be applied on any language, treebank and parser conforming to definition 2.2? This was concluded already in chapter 3:

- The pseudo-projective transformation itself is language independent.
- The pseudo-projective transformations itself is treebank independent.
- The pseudo-projective transformations itself is parser independent.

These properties make them very useful, since they can be applied as preprocessing and postprocessing to any parser conforming to definition 2.2. It is likely that parsing accuracy increases if the conditions are good (such as relatively high amount of training data and non-projectivity), and seldom decreases accuracy under less beneficial circumstances.

Moving over to the hard structures, a number of statements can now be established. Since the coordination and verb group transformations do not increase accuracy for MSTParser, the question whether the transformations themselves are parser independent (which they by the way are) becomes irrelevant. Furthermore, as concluded in section 3.4, they are not treebank independent since they are closely tied to the type of annotation in the treebank. Neither are they language independent, at least not the verb group

transformation, because Arabic is a language that does not treat auxiliary verbs as individual tokens.

However, the parsing results for coordination and verb groups in experiment II show that parsing accuracy improves and that:

- The transformations from **PS** annotation to (primarily) **MS** for coordination and verb groups are language independent with respect to parsing accuracy.
- The transformations from **PS** annotation to (primarily) **MS** for coordination and verb groups are treebank (or annotation type) independent with respect to parsing accuracy.

Whereas the pseudo-projective transformations have very little or no positive effect on SDT and PADT, significantly improved accuracies were recorded for the coordination transformations. This implies that the transformations are less dependent on the amount of training data compared to pseudo-projective parsing. The same conclusion holds for the verb group transformation, since it exhibits comparably higher accuracies for both SDT and PDT. The impact for the verb group transformation is less prominent than the coordination transformation, which can be the result of higher accuracy in **PS** for the words involved in verb groups than in coordination.

While experiment III indicates that the transformations for coordination and verb groups are not as parser independent as the pseudo-projective transformations with respect to parsing accuracy, it is still fair to claim that the coordination transformations are treebank independent with respect to parsing accuracy. The **PS**-annotation of coordination in the dependency version of Alpino, which is based on a completely different type of annotation than SDT, PADT and PDT, can be transformed into **MS** or **CS**, both with increased accuracy for MaltParser as a result. In other words, while **MS** is not necessarily the best type of structure for coordination to perform inductive dependency parsing, it is clear that **PS** is not optimal. As mentioned in the introduction, they are not treebank independent in the sense that they can be applied to any treebank, simply because all treebanks do not annotate coordination and verb groups in the same way.

As no transformation is flawless, it is likely that if one devotes more time to their optimization, improved accuracy can be expected, especially for coordination. One thing that could be of interest is to find the reason why AS_U in most experiments increases more than AS_L . Is it an inherent property of the transformations, or is it simply the case that too little attention has been given to the quality of labels in the transformations? In any case, this is an issue that could need some more work to sort out.

As the transformations and inverse transformations inevitably introduce distortion in the data, these results may also be of interest for treebank creators. For instance, these issues were taken into consideration in the reconstruction of the Swedish treebank Talbanken into dependency structure (Nilsson et al. 2005; Nivre et al. 2006). Without compromising linguistic correctness and expressiveness, parsing accuracy was one factor taken into consideration before the final dependency structure, for phenomena such as coordination and verb groups, was determined. Constructing new dependency treebanks, or constituency-based treebanks for that matter, in a way that learning is facilitated is of interest, which consequently reduces the need for explicit tree transformations.

To summarize this licentiate thesis, one can claim that preprocessing and postprocessing is important not only in constituent-based parsing but also for inductive dependency parsing. As stated by Johnson (1998), choosing the right base representation is an important task in constituency-based parsing, which he and several subsequent studies confirm. This is an assertion that can be generalized to data-driven inductive dependency-based parsing as well.

5.2 Future Work

Even though this licentiate thesis is completed, this does not mean that there is no room for further research on this topic. Here is a list of possible research tracks to explore, given the outcome of the presented experiments:

- Continue with a more detailed investigation of experiment III by looking at the differences between projective parsing with pseudo-projective transformations and direct non-projective parsing. Which strategy is most sensitive to the amount of training data and the proportion of non-projectivity? Which one performs best for non-projective arcs? Why does it seem that AS_L is higher for direct non-projective parsing but not for AS_U ?
- Explore other encodings for pseudo-projective parsing, while maintaining the black-box approach. In particular, investigate whether preprocessing in combination with machine learning can improve the accuracy of pseudo-projective parsing, while maintaining the black-box principle.
- Perform corresponding parsing experiments as in experiment III for the projective second-order MSTParser (McDonald and Pereira 2006; McDonald et al. 2006), which uses an approximative postprocessing method to cope with non-projectivity. The interesting question is

whether the combination of preprocessing and post-processing of the pseudo-projective transformations can compete with or even increase the accuracy of the approximative, pure postprocessing approach.

- Incorporate the decision of whether an arc is lifted or not as a separate classifying task inside the parser. The classification is then presumably done after the arc has been assigned a dependency type. This is in contrast to the current approach, where the information about the lift is part of the dependency type of the lifted arc and/or the arcs along the lifting path, and consequently classified together. As a result, the set of dependency types does not need to be extended, which may be beneficial to avoid sparse data problems. However, the disadvantage is that the black-box approach is lost, making it more parser dependent.
- Develop methods to automatically find dependency structures for linguistic phenomena that are hard to parse. This has been done for constituency structure, where Petrov et al. (2006) takes this one step further by also automatizing the transformations (by making the non-terminals more fine-grained). The fact that dependency structure is less complex, for instance because of the lack of non-terminals, may result in less room for improvements for dependency parsers. Nevertheless, methods that can predict dependency structures that are difficult to parse for any or a specific inductive dependency parser may be possible to create, at least as a first step.
- Another possible research area that touch upon the work of Petrov et al. is to find the right level of granularity in the dependency labels in order to achieve the highest AS_U . One can expect that AS_L usually will increase as the number of distinct dependency types decreases, but the question is whether the same holds for AS_U and whether this can be done in an automatized manner. A treebank with fine-grained dependency types would serve as good material for such an experiment.

This list of suggestions for future work is just some possibilities, and other research tracks within this topic may evolve eventually. I therefore believe that I will not find myself in a situation where I am out of ideas in the coming years as a doctoral student.

Appendix A

Feature Models and SVM-Parameter values

This appendix contains the feature models and the values for the SVM parameters for Experiment II. They are based on the setting used in Nivre et al. (2006) and Hall and Nilsson (2006), but are not identical.

A.1 Settings for Experiment II

In the experiment presented in section 4.3, the polynomial kernel function of degree 2 is used for SVM, with the additional SVM parameter values listed in table A.1. The three rightmost columns are MaltParser options indicating whether the training data has been divided into smaller parts in order to reduce training time. Details about the splitting strategy are described in Hall et al. (2006), and the list of features below uses the same notation as Hall and Nilsson (2006).

	γ	C	r	ϵ	S	F	T
SDT	.2	.1	.8	0.1	N		
PADT	.16	.3	0	1.0	N		
Alpino	.20	1.0	0	1.0	D	$p(\tau_0)$	1000
Tiger	.2	.5	0	1.0	D	$p(\tau_0)$	1000
PDT	.2	.5	0	1.0	D	$p(\tau_0)$	200

Table A.1: The final parameter settings for the SVM learner.

In the final experiment for Alpino in table 4.7 (**Eval**-column), the parameter S was changed to no splitting of the training data, i.e. “-S 0”. The features sets below were used together with the SVM parameter values above in experiment II:

A.1. Settings for Experiment II

- SDT:
 - $p(\sigma_0), p(\tau_0), p(\tau_1), p(\tau_2), p(\sigma_1), p(r(\sigma_0)), p(l(\tau_0));$
 - $d(\sigma_0), d(lc(\sigma_0)), d(rc(\sigma_0)), d(lc(\tau_0)), d(rs(lc(\sigma_0)));$
 - $w(\sigma_0), w(\tau_0), w(\tau_1), w(h(\sigma_0));$
 - $fea(\sigma_0), fea(\tau_0), fea(\tau_2);$
 - $c(\sigma_0), c(\tau_0), c(\sigma_2);$

- PADT:
 - $p(\sigma_0), p(\tau_0), p(\tau_1), p(\tau_2), p(\tau_3), p(\sigma_1), p(rs(lc(\sigma_0))), p(l(\tau_0));$
 - $c(\sigma_0), c(\tau_0), c(h(lc(\sigma_0)));$
 - $d(\sigma_0), d(lc(\sigma_0)), d(rc(\sigma_1));$
 - $w(\sigma_0), w(\tau_0), w(\tau_1), w(h(\sigma_0)), w(ls(rc(\sigma_0))); fea(\sigma_0), fea(\tau_0);$
 - $lem(\sigma_0), lem(\tau_0);$

- Alpino:
 - $p(\sigma_0), p(\tau_0), p(\tau_1), p(\tau_2), p(\sigma_1);$
 - $c(\sigma_0), c(\tau_0);$
 - $fea(\sigma_0), fea(\tau_0);$
 - $d(h(\sigma_0)), d(\sigma_0), d(rc(\sigma_0)), d(lc(\tau_0));$
 - $w(h(\sigma_0)), w(\tau_0), w(\tau_1), w(h(\sigma_0));$
 - $lem(\sigma_0), lem(\tau_0), lem(l(\tau_0)), lem(lc(\sigma_0));$

- Tiger:
 - $p(\sigma_0), p(\tau_0), p(\tau_1), p(\tau_2), p(\tau_3), p(\sigma_1), p(h(\sigma_1)), p(l(\sigma_0)), p(l(\tau_0));$
 - $d(\sigma_0), d(lc(\sigma_0)), d(rc(\sigma_0)), d(lc(\tau_0));$
 - $w(\sigma_0), w(\tau_0), w(\tau_1), w(\tau_2), w(h(\sigma_0)), w(l(\sigma_0));$

- PDT:
 - $p(\sigma_0), p(\tau_0), p(\tau_1), p(\tau_2), p(\tau_3), p(\tau_4), p(lc(\tau_0)), p(\sigma_1)$
 - $d(\sigma_0), d(lc(\sigma_0)), d(rc(\sigma_0)), d(lc(\tau_0));$
 - $w(\sigma_0), w(\tau_0), w(\tau_1), w(h(\sigma_0));$
 - $c(\sigma_0), c(\tau_0);$
 - $fea(\sigma_0), fea(\tau_0), fea(\tau_1), fea(\tau_2), fea(\tau_3)$
 - $lem(\sigma_0), lem(\tau_0)$

References

- Bikel, D. M. (2004). Intricacies of Collins' Parsing Model. *Computational Linguistics* 30(4), 479–511.
- Black, E., R. Garside, and G. Leech (1993). *Statistically driven computer grammars of English: the IBM/Lancaster approach*. Rodopi.
- Bloomfield, L. (1933). *Language*. The University of Chicago Press.
- Böhmová, A., J. Hajič, E. Hajičová, and B. Hladká (2003). The Prague Dependency Treebank: A three-level annotation scenario. In A. Abeillé (Ed.), *Treebanks: Building and Using Syntactically Annotated Corpora*. Kluwer Academic Publishers.
- Brants, S., S. Dipper, S. Hansen, W. Lezius, and G. Smith (2002). TIGER treebank. In E. Hinrichs and K. Simov (Eds.), *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 24–42.
- Brill, E. (1993). Transformation-Based Error-Driven Parsing. In *Proceedings of the Third International Workshop on Parsing Technologies (IWPT)*, pp. 13–25.
- 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)*, pp. 1–17.
- Cahill, A., M. Burke, R. O'Donovan, J. Van Genabith, and A. Way (2004). Long-Distance Dependency Resolution in Automatically Acquired Wide-Coverage PCFG-Based LFG approximations. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pp. 319–326.
- Campbell, R. (2004). Using Linguistic Principles to Recover Empty Categories. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pp. 645–652.
- Carroll, J. (2000). Statistical parsing. In R. Dale, H. Moisl, and H. Somers (Eds.), *Handbook of Natural Language Processing*. New York: Dekker.
- Chang, C.-C. and C.-J. Lin (2001). LIBSVM: A library for support vector machines.

- Charniak, E. (1993). *Statistical Language Learning*. MIT Press, Cambridge.
- Charniak, E. (2000). A Maximum-Entropy-Inspired Parser. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 132–139.
- Chomsky, N. (1956). Three models for the description of language. *I.R.E. Transactions on information theory IT-2*, 113–124.
- Collins, M. (1997). Three generative, lexicalized 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., J. Hajič, L. Ramshaw, and C. Tillmann (1999). A statistical parser for Czech. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL 99)*, pp. 100–110.
- Covington, M. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pp. 95–102.
- Daelemans, W. and A. Van den Bosch (2005). *Memory-Based Language Processing*. Cambridge University Press.
- Dienes, P. and A. Dubey (2003a). Antecedent Recovery: Experiments with a Trace Tagger. In *Conference on Empirical Methods in Natural Language Processing, (EMNLP 2003)*, Sapporo, Japan.
- Dienes, P. and A. Dubey (2003b). Deep Syntactic Processing by Combining Shallow Methods. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*.
- Dienes, P. and A. Dubey (2003c). Probabilistic Parsing for German using Sister-Head Dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pp. 96–103.
- Džeroski, S., T. Erjavec, N. Ledinek, P. Pajas, Z. Žabokrtsky, and A. Žele (2006). Towards a Slovene Dependency Treebank. In *LREC2006*.
- Eisner, J. (1996). Three New Probabilistic Models for Dependency Parsing: An Exploration. In *16th International Conference on Computational Linguistics, Proceedings of the Conference (COLING)*, pp. 340–345.
- Foth, K., M. Daum, and W. Menzel (2004). A Broad-Coverage Parser for German Based on Defeasible Constraints. In *Proceedings of KONVENS-2004*, Wien, Austria.

References

- Fraser, N. M. (1994). Dependency grammar. In R. Asher (Ed.), *The Encyclopedia of Language and Linguistics*, pp. 860–864. Pergamon Press, Oxford, United Kingdom.
- Gaifman, H. (1965). Dependency systems and phrase-structure systems. *Information and Control* 8, 304–337.
- Gazdar, G., E. Klein, G. Pullum, and I. Sag (1985). *Generalized Phrase Structure Grammar*. Blackwell: Oxford.
- Hajič, J. (1998). Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In *Issues of Valency and Meaning*, pp. 12–19. Prague Karolinum, Charles University Press.
- Hajič, J., B. V. Hladka, J. Panevová, E. Hajičová, P. Sgall, and P. Pajas (2001). Prague Dependency Treebank 1.0. LDC, 2001T10.
- Hajič, J., O. Smrž, P. Zemánek, J. Šnaidauf, and E. Beška (2004). Prague Arabic Dependency Treebank: Development in Data and Tools. In *NEMLAR*, pp. 110–117.
- Hall, J. (2006). Maltparser - An Architecture for Inductive Labeled Dependency Parsing. Licentiate’s thesis, School of Mathematics and System Engineering, Växjö University.
- 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. Nivre, and J. Nilsson (2006). Discriminative Classifiers for Deterministic Dependency Parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics Main Conference Poster Sessions*, Sydney, Australia, pp. 316–323.
- Hall, K. and V. Novák (2005). Corrective modeling for non-projective dependency parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pp. 42–52.
- Havelka, J. (2005). Projectivity in totally ordered rooted trees. In *Prague Bulletin of Mathematical Linguistics*, Volume 84, pp. 13–30. Univerzita Karlova, Praha.
- Hays, D. G. (1964). Dependency theory: A formalism and some observations. *Language* 40, 511–525.
- Holan, T. and Z. Žabokrtský (2006). Combining Czech Dependency Parsers. In *Proceedings of the 9th International Conference on Text, Speech and Dialogue*.

- Hudson, R. (1984). *Word Grammar*. Basil Blackwell.
- Hudson, R. (1990). *English Word Grammar*. Basil Blackwell.
- Jijkoun, V. and M. de Rijke (2004). Enriching the Output of a Parser Using Memory-based Learning. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pp. 311–318.
- Johnson, M. (1998). PCFG Models of Linguistic Tree Representations. *Computational Linguistics 24*, 613–632.
- Johnson, M. (2002). A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 136–143.
- Johnson, R., M. King, and L. des Tombe (1985). EUROTRA: A multilingual system under development. *Computational Linguistics 11*, 155–169.
- Joshi, A. K. and Y. Schabes (1997). Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Volume 3, pp. 69–124. Berlin, New York: Springer.
- Jurafsky, D. and J. H. Martin (2000). *Speech and Language Processing*. Prentice Hall.
- Kahane, S., A. Nasr, and O. Rambow (1998). Pseudo-Projectivity: A Polynomially Parsable Non-Projective Dependency Grammar. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pp. 646–652.
- Kakkonen, T. (2005). Dependency Treebanks: Methods, Annotation Schemes and Tools. In *Proceedings of the 15th NODALIDA conference, Joensuu 2005*, pp. 94–104.
- Klein, D. and C. Manning (2003). Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 423–430.
- Kromann, M. T. (2003). The Danish Dependency Treebank and the DTAG Treebank Tool. In *The Second Workshop on Treebanks and Linguistic Theories*.
- Kunze, J. (1968). The treatment of nonprojective structures in the syntactic analysis and synthesis of English and German. *Computational Linguistics 7*, 67–77.

References

- Lecerf, Y. (1960). Programme des conflits, module des conflits. *Bulletin bimestriel de l'ATALA* 1(4): 11–18, 1(5): 17–36.
- Levy, R. and C. Manning (2004). Deep Dependencies from Context-Free Statistical Parsers: Correcting the Surface Dependency Approximation. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pp. 327–334.
- Marcus, M. P., B. Santorini, and M. A. Marcinkiewicz (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19, 313–330.
- McCawley, J. D. (1973). *Grammar and Meaning: Papers on Syntactic and Semantic Topics*. Tokyo: Taishukan.
- 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'05)*, Ann Arbor, Michigan, pp. 91–98.
- McDonald, R., K. Lerman, and F. Pereira (2006). Multilingual Dependency Parsing with a Two-Stage Discriminative Parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pp. 216–220.
- McDonald, R. and F. Pereira (2006). Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 81–88.
- Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press.
- Nikula, H. (1986). *Dependensgrammatik*. Liber Förlag.
- Nilsson, J., J. Hall, and J. Nivre (2005). MAMBA Meets TIGER: Reconstructing a Swedish Treebank from Antiquity. In *Proceedings of NODALIDA 2005 Special Session on Treebanks for Spoken and Discourse*, Volume 32, pp. 119–132. Copenhagen Studies in Language.
- Nilsson, J., J. Nivre, and J. Hall (2006). Graph Transformations in Data-Driven Dependency Parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia, pp. 257–264.
- Nivre, J. (2003). An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, Nancy, France, 23–25 April 2003, pp. 149–160.

- Nivre, J. (2006). *Inductive Dependency Parsing*. Springer.
- Nivre, J., J. Hall, and J. Nilsson (2004). Memory-based Dependency Parsing. In H. T. Ng and E. Riloff (Eds.), *Proceedings of the 8th 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 06)*.
- 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)*, pp. 221–225.
- Nivre, J. and J. Nilsson (2005). Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, Ann Arbor, Michigan, 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 (LREC2006)*.
- Percival, W. K. (1976). Notes from the linguistic underground (syntax and semantics). In J. McCawley (Ed.), *On the Historical Source of Immediate Constituent Analysis 7*, pp. 229–42. London: Academic Press.
- Petrov, S., L. Barrett, R. Thibaux, and D. Klein (2006). Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia, pp. 433–440.
- Pollard, C. and I. A. Sag (1994). *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press.
- Robinson, J. J. (1970). Dependency structures and transformational rules. *Language* 46(2), 259–285.
- Schmid, H. (2006). Trace Prediction and Recovery with Unlexicalized PCFGs and Slash Features. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia, pp. 177–184.
- Sgall, P., E. Hajičová, and J. Panevová (1986). *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel.

References

- Skut, W., T. Brants, B. Krenn, and H. Uszkoreit (1998). A linguistically interpreted corpus of German newspaper text. In *Proceedings of the 10th European Summer School in Logic, Language and Information*.
- Solomon, M. (1965). Sur la notion de projectivité. *Zeitschr. f. math. Logik und Grundlagen d. Math.* 11, 181–192.
- Tapanainen, P. and T. Järvinen (1997). A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pp. 64–71. Association for Computational Linguistics.
- Tesnière, L. (1959). *Eléments de Syntaxe Structurale*. Paris: Librairie C. Klincksieck.
- van der Beek, L., G. Bouma, R. Malouf, and G. van Noord (2002). The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*.
- 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 03)*, pp. 195–206.
- Zeman, D. (2004). *Parsing with a Statistical Dependency Model*. Ph. D. thesis, Univerzita Karlova, Praha.
- Zwicky, A. M. (1985). Heads. *Journal of Linguistics* 21, 1–29.



School of Mathematics and System Engineering

ISSN 1650-2647 ISRN VXU-MSI-DA-R--07002--SE