Modeling the Interface between Morphology and Syntax in Data-Driven Dependency Parsing

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Philosophie (Dr. phil.) genehmigte Abhandlung

vorgelegt von
Wolfgang Burkhard Werner Seeker
aus Jena

Hauptberichter:	Prof. Dr. Jonas Kuhn
1. Mitberichter:	Prof. Dr. Dr. Joakim Nivre
2. Mitberichter:	Prof. Dr. Sebastian Padó

Tag der mündlichen Prüfung: 18. Dezember 2015

Institut für Maschinelle Sprachverarbeitung der Universität Stuttgart

2016

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Stuttgart, Oktober 2015
Wolfgang Burkhard Werner Seeker

Contents

List of Figures

List of Tables

Abstract

When people formulate sentences in a language, they follow a set of rules specific to that language that defines how words must be put together in order to express the intended meaning. These rules are called the grammar of the language. Languages have essentially two ways of encoding grammatical information: word order or word form. English uses primarily word order to encode different meanings, but many other languages change the form of the words themselves to express their grammatical function in the sentence. These languages are commonly subsumed under the term *morphologically rich languages*.

Parsing is the automatic process for predicting the grammatical structure of a sentence. Since grammatical structure guides the way we understand sentences, parsing is a key component in computer programs that try to automatically understand what people say and write.

This dissertation is about parsing and specifically about parsing languages with a rich morphology, which encode grammatical information in the form of words. Today's parsing models for automatic parsing were developed for English and achieve good results on this language. However, when applied to other languages, a significant drop in performance is usually observed.

The standard model for parsing is a pipeline model that separates the parsing process into different steps, in particular it separates the morphological analysis, i.e. the analysis of word forms, from the actual parsing step. This dissertation argues that this separation is one of the reasons for the performance drop of standard parsers when applied to other languages than English. An analysis is presented that exposes the connection between the morphological system of a language and the errors of a standard parsing model. In a

second series of experiments, we show that knowledge about the syntactic structure of sentence can support the prediction of morphological information. We then argue for an alternative approach that models morphological analysis and syntactic analysis jointly instead of separating them. We support this argumentation with empirical evidence by implementing two parsers that model the relationship between morphology and syntax in two different but complementary ways.

Überblick

Wenn Menschen Sätze in ihrer Sprache bilden, dann folgen sie einem sprachspezifischen Satz an Regeln, in dem festgelegt ist, auf welche Weise Wörter miteinander kombiniert werden müssen, um eine gewünschte Bedeutung auszudrücken. Diese Regeln nennt man die Grammatik einer Sprache. Natürliche Sprachen haben im Wesentlichen zwei Möglichkeiten, grammatische Information zu kodieren: entweder durch die Reihenfolge der Wörter oder in der Form der Wörter. Englisch verwendet hauptsächlich die Reihenfolge von Wörtern um unterschiedliche Bedeutungen zu kodieren. Viele andere Sprachen markieren jedoch die grammatische Funktion von Wörtern in ihrer Form. Diese Sprachen werden gemeinhin unter dem Begriff *morphologisch reiche Sprachen* zusammengefaßt.

Parsing nennt man das automatische Verfahren zur Vorhersage der grammatischen Struktur von Sätzen. Die grammatische Struktur eines Satzes bestimmt, wie man den Satz versteht. Aus diesem Grund ist Parsing ein wichtiger Bestandteil von Programmen, die versuchen, die Bedeutung von dem, was Menschen sagen und schreiben, automatisch zu verstehen.

Diese Dissertation beschäftigt sich mit Parsing, insbesondere mit dem Parsing von Sprachen mit einer reichen Morphologie. Die aktuellen Modelle für Parsing wurden für die englische Sprache entwickelt und erzielen gute Ergebnisse für Englisch. Wenn man diese Modelle aber auf andere Sprachen anwendet, sind die Ergebnisse überlicherweise deutlich schlechter.

Das Standardmodell für Parsing ist ein sogenanntes Pipeline-Modell, welches den Parsingprozess in verschiedene Schritte aufteilt, insbesondere trennt es die morphologische Analyse, d.h. die Analyse der Wortformen, von der syntaktischen Analyse, also dem

eigentlichen Parsing. Diese Dissertation stellt die Hypothese auf, daß diese Trennung von morphologischer und syntaktischer Analyse einer der Gründe für die schlechteren Ergebnisse ist, die auf anderen Sprachen als Englisch erzielt werden. Wir zeigen in einer ersten Analyse den kausalen Zusammenhang zwischen bestimmten Eigenschaften des morphologischen System einer Sprache und den Fehlern, die das Standardmodell für Parsing begeht. In einer zweiten Reihe von Experimenten zeigen wir dann, daß Zugang zu der vollen syntaktischen Struktur eines Satzes die automatische Vorhersage von morphologischer Information verbessert. Wir diskutieren anschließend ein alternatives Parsingmodell, in dem morphologische und syntaktische Analyse gemeinsam modelliert werden. Wir belegen die Vorteile eines solchen Modells empirisch, indem wir zwei Parser entwickeln, die die Beziehung zwischen Morphologie und Syntax auf verschiedene aber komplementäre Weise nachbilden.

Acknowledgements

During the time that I worked on this thesis I had the pleasure and the honor to meet and work with some wonderful people. Learning from them and thinking together with them did not just teach me a lot about language, computer science, and science in general but made me grow as a person as well.

I am most grateful to my supervisor and Doktorvater, Jonas Kuhn. I can't remember a single time that I didn't feel motivated and full of new ideas after talking to him. He taught me to pursue my own ideas rather than waiting for someone else to tell me the next steps and he has the admirable ability to always find the positive aspects of anything. He is also someone who understands the difference between Computational Linguistics and Natural Language Processing and I have enjoyed our many small and long discussions of language phenomena very much.

I would like to thank my examiners, Joakim Nivre and Sebastian Padò, for sharing and discussing with me their thoughts on this thesis and the problems therein. It is this kind of discussion that inspires and motivates me to work on a problem and I have enjoyed this very much.

I had the honor and fortune to share my office and a lot of time with Anders Björkelund. I still sometimes have the urge to turn to the right expecting him to sit there so I can discuss some problem with him (or show him something completely unrelated to what I and he are actually supposed to do). Bouncing ideas off of him always made them come back better and more useful. I have learned many things from him but the one I am most grateful for is that one should always go to the very bottom of a question. He never stops until he knows all the answers, may it be the workings of a statistical model or the

administrative organization of a German federal state. Thanks for teaching me.

Without the dedication and passion of Özlem Çetinoğlu, some parts of this thesis wouldn't have been possible. Discussing examples in a foreign language is much more fun when it isn't foreign to the other person, and I have had a lot of fun working on Turkish with her. We missed each other in Dublin by less than a month and I am very glad that we got a second chance in Stuttgart.

While I was working with Hungarian data, I was very lucky to get help from Richárd Farkas and Veronika Vincze from Szeged University who combine linguistic expertise with the intuition of native speakers.

A lot of people have made my time at IMS unforgettable: my former Prolog tutor and Potsdam connection, Sina Zarrieß, the most awesome intern (and much more) Agnieszka Faleńska, my writing group Kerstin Eckart and Boris Haselbach, and all my other colleages, teachers, and friends: Kyle Richardson, André Blessing, Fabienne Braune, Alessandra Zarcone, Bernd Bohnet, Thomas Müller, Xiang Yu, Arndt Riester, Nils Reiter, Wiltrud Kessler, Christian Scheible, Gregor Thiele, Markus Gärtner, Nina Seemann, Daniel Quernheim, Antje and Kati Schweitzer, Michael Walsh, Ulrich Heid, Hans Kamp, and Antje Roßdeutscher. You are awesome!

I want to thank my friends and family-by-choice Marko and Janine Drotschmann, for not just the most awesome summer vacations but also our weekly retreat to another world. Thanks for coming and listening to everything I had to say.

I have to thank my parents for always being there for me even from far away and not even once asking me when I would come back. And finally, thank you, Anett, for always finding the right balance between comfort, punschrullar, and a kick in the butt.

This thesis was very generously sponsored by the Deutsche Forschungsgemeinschaft (DFG) via Sonderforschungsbereich 732 *Incremental Specification in Context*, projects D8 and B3.

Chapter 1

Introduction

In natural language, speakers express complex meaning that goes beyond the meaning of single words by combining words into phrases and sentences. The rules that specify how words must be combined in order to create well-formed sentences are defined in the grammar of the language that is used. The grammatical structure of a sentence relates the individual words to each other and thus defines how the meaning of the words must be combined to derive the meaning of the sentence. When a sentence violates rules of the grammar it may sound strange, mean something that was not intended, or become entirely uninterpretable depending on the rule that is being violated.

Language has two[1] major means for encoding the grammatical structure of a sentence: it can encode it in the order of the words or in the word forms. The grammar of English, for example, encodes grammatical structure mostly through word order. A possible sentence to express the situation in Figure 1.1a is the sentence *a dog bites someone*. The order of the words in this sentence is important for its meaning, which becomes apparent when we change it. The sentence *someone bites a dog* describes a different situation (Figure 1.1b) than the first sentence even though the sentence uses exactly the same words. Other word orders, e.g., the sentence *bites someone dog a*, may even mean nothing at all.

Other languages do not make use of word order like English but instead encode the

[1] This is of course a simplification. Spoken language, for example, can use intonation for this purpose. However, this dissertation focuses on written language.

(a) A dog bites someone.

(b) Someone bites a dog.

Figure 1.1: Changing word order in English changes meaning. Sources: (a) "Duchess and Marina" by David Burke, license CC BY-NC-ND 2.0, (b) "Man bites dog" by Jeff Noble, license CC BY 2.0.

information about the grammatical structure in the form of the words. Examples 1.1 and 1.2 show two sentences in Czech that both mean *a dog bites a man*. Changing the word order as in English does not change the meaning of the sentence. In fact, all possible orders of the three words result in a well-formed Czech sentence and basically have the same meaning (barring potential stylistic differences or differences due to what is being emphasized).

(1.1) *pes* *kouše* *člověka*
NOM ACC
dog bites man
A dog bites a man.

(1.2) *člověka* *kouše* *pes*
ACC NOM
man bites dog
A dog bites a man.

In the English sentence *a dog bites someone*, *a dog* is called the grammatical subject. In the canonical case, the subject corresponds to the acting party in a situation, here it is the biter in a biting event. In English, subjects are marked by placing them in front of the verb. Similarly, the party that is being acted upon is called the object of the sentence, in the example sentence it is the one being bitten (*someone*). The object of a sentence is marked in English by placing it after the verb. It is because subject and object are defined by their

position relative to the verb that the meaning changes when the word order is reversed.

The Czech sentences also have a subject and an object, namely *pes* and *člověka*, respectively. However, it is not their position that encodes this information but it is their form. The word *pes* stands in nominative case (marked as NOM in the examples) whereas the word *člověka* stands in accusative case (marked as *acc*). Nominative case is used to mark subjects in Czech whereas objects can be marked with accusative case (and others, but not nominative case). Changing the word order in the Czech sentence does not change the meaning because the word forms are the same regardless of their order. If we wanted to change the meaning of the sentence to something close to Figure 1.1b, we need to change the forms of the words rather than their order (Example 1.3).

(1.3) *psa* *kouše* *člověk*
ACC NOM
dog bites man
A man bites a dog.

The simple example illustrates how languages make use of word order and word forms to encode grammatical structure. Traditionally, the part of grammar that deals with word order rules is called *syntax* and the part that deals with the form of words is called *morphology*. Morphology can be further divided into the two subfields of *derivation/composition* and *inflection*. The morphological rules for derivation and composition define how new words can be formed from existing ones, whereas the rules for inflection define how a word changes its form to signal grammatically relevant information. The Czech examples show instantiations of inflection because the change in the form, i.e., changing the case feature of the words, changes the grammatical structure. All languages in the world make use of syntactic and morphological means to encode grammatical structure but they do so to different extents. Some languages like English rely heavily on word order whereas languages like Czech rely more on word forms.

1.1 Parsing

This dissertation is about models for automatically predicting the grammatical structure of sentences, in particular for languages like Czech, which encode grammatical structure by morphological means. A program that is used to predict such structures is called a *parser* and the process is called *parsing*. Since the grammatical structure defines the way we construct the meaning of sentences from the meaning of its words, predicting such structures automatically is an important component in systems that try to automatically understand what people say and write. And since people speak and write in many languages, the ultimate goal is to develop parsing models that can correctly predict the grammatical structure for any sentence from any language.

The representation of grammar that we work with in this dissertation is called *dependency grammar*. Dependency grammar represents the syntactic structure of sentences as directed trees called *dependency trees*. Examples of such trees for the sentences discussed above are shown in Figure 1.2. Dependency grammars are well suited to represent phenomena typical for languages with rich morphology and free word order.

(a) Syntactic structure of an English sentence.

(b) Syntactic Structure of a Czech sentence.

Figure 1.2: Syntactic structure visualized as dependency trees.

The models that we will use are all *data-driven* models. Data-driven models do not use rules that were hand-crafted by humans but instead learn a function for mapping input to output by looking at large amounts of input-output examples. For example, data-driven parsers are trained on treebanks, which are large collections of sentences for which humans have manually annotated the correct syntactic structure. The model thus learns to associate certain structures with certain patterns in the sentences and generalizes this knowledge to predict new structures for sentences that it has not seen before.

1.2 Claim and Contributions

The standard architecture in dependency parsing uses a pipeline setup, in which the parsing process is separated into (at least) three steps: tokenization, morphological analysis, and the actual parsing. During tokenization, a continuous input string is separated into individual tokens, which roughly correspond to words. During morphological analysis, the part-of-speech (and for other languages the morphological features) of each word are predicted. During the parsing step, words are related to each other to form the syntactic structure of the sentence. The output of each step forms the input to the next one and the output of the parsing step is the final output of the system. This architecture is very efficient because it breaks the task down into smaller ones that can be handled efficiently on their own.

Note that we will use the term parsing ambiguously to refer to the actual parsing step as well as the whole pipeline system. This is because parsing cannot reasonably be done without tokenization and at least part-of-speech tags. We therefore consider the whole process to constitute the task of parsing rather than just the last step in which words are related to each other.

The pipeline model was developed for English and state-of-the-art parsing systems achieve very good results for predicting the syntactic structure of English sentences (at least for newspaper text). However, when these parsers are applied to other languages, a significant drop in performance can be observed. It has been hypothesized that one of the main reasons for this drop is the fact that other languages make use of morphological means to encode grammatical structure and that the models that were developed for English are not suited to parse languages with rich morphology (Tsarfaty et al. 2010).

In the introductory example, we have shown that languages like Czech encode the grammatical information in word forms rather than the word order. Specifically, we have shown that particular forms are used to mark the grammatical function of a word. One could think now that one simply needs to note down for each of these forms what function they mark and parsing should work well also for these languages. However, language is much more complicated. In many languages, the same word form can be ambiguous between different interpretations and often a single interpretation is often used for more than one function in the sentence. As we said before, most languages make use of both

syntactic and morphological means to encode grammatical information. Because of this, there are intricate interactions between syntax and morphology in many languages.

The claim that this dissertation starts from is that the separation of tokenization, morphological analysis and syntactic analysis in the pipeline model is one reason why the standard model does not perform as well for other languages as it does for English because the interaction between morphology and syntax cannot be modeled. We argue for an alternative model in which tokenization, morphology, and syntactic structure are predicted jointly. To this end, we ...

(I) ... conduct analyses and experiments that show that certain properties of the grammar of languages with rich morphology violate the independence assumption that underlies the separation of processing steps in the pipeline model. The experiments also highlight differences between languages that all have rich morphology.

(II) ... develop a simple model for integrating structural context into a model for predicting morphological information and use it to show that access to the full structural context is important for predicting morphology for languages with free word order. An additional comparison shows that the contribution of structural context is complementary to the information from large-scale lexicons.

(III) ... formulate a set of constraints that model morphosyntactic rules for three morphologically rich languages. We show that using these models to restrict the search space of a dependency parser leads to better prediction of argument structure.

(IV) ... present an efficient decoder for non-projective graph-based lattice dependency parsing and show how word segmentation, morphological analysis, and syntactic analysis can be modeled jointly. The parser is tested on Turkish and Hebrew and outperforms two state-of-the-art baseline systems based on pipeline architectures. The results on Turkish are the first published results for Turkish lattice parsing.

1.3 Publications

The contents of this dissertation have been previously reported in the following publications:

Seeker, W. and Kuhn, J. (2011). On the Role of Explicit Morphological Feature Representation in Syntactic Dependency Parsing for German. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 58–62, Dublin, Ireland. Association for Computational Linguistics

Seeker, W. and Kuhn, J. (2013a). Morphological and Syntactic Case in Statistical Dependency Parsing. *Computational Linguistics*, 39(1):23–55

Seeker, W. and Kuhn, J. (2013b). The Effects of Syntactic Features in Automatic Prediction of Morphology. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 333–344, Seattle, Washington, USA. Association for Computational Linguistics

Seeker, W. and Çetinoğlu, O. (2015). A Graph-based Lattice Dependency Parser for Joint Morphological Segmentation and Syntactic Analysis. *Transactions of the Association for Computational Linguistics*, 3:359–373

Chapter 2

Background

The purpose of this chapter is to provide the general background on the topics that this dissertation takes as given. We introduce dependency structures and dependency parsing as well as the methods that we use for training our statistical models. The intention is to give a brief overview and introduce the most important concepts. We provide pointers whenever possible to other work that describes the discussed content in more detail. Readers that are familiar with the concepts in this chapter can safely move on to the next.

2.1 Dependency Grammar and Dependency Trees

Dependency grammar is a grammar formalism that focuses on the nature of the relation between two words within their sentential context. Each word in a sentence stands in an asymmetric relation with another word, but unlike in constituency grammar no phrasal nodes are postulated, i.e., the words in the sentence are the only nodes in the structure. In a dependency relation, the word that influences the properties of the other word is called the head (or governor) and the word that is influenced by the other is called the dependent (or modifier, or governee). The dependent is said to depend linguistically[1] on the head, hence the name dependency grammar. Complex structures emerge through

[1]Modern dependency grammar formalisms strive for a description of all linguistic aspects of natural language, which encompasses phonology, morphology, syntax, semantics, and discourse.

recursive application of dependency. The nature of the relation between two words is typically classified into a set of predefined relations, e.g., for syntactic structure it would contain relations like subject or adjunct. The invention of modern dependency grammar is credited to Lucien Tesnière (Tesnière 1959, Tesnière et al. 2015) and several formalisms have been introduced since then, e.g., Functional Generative Description (Sgall et al. 1986), Meaning Text Theory (Mel'čuk 1988), and Word Grammar (Hudson 1984).

In parsing literature, a dependency structure almost always corresponds to the syntactic structure of a sentence. Figure 2.1 gives an example sentence with its syntactic structure annotated according to the Universal Dependency Grammar (Nivre 2015). By convention, we draw the dependency arcs such that the arrow points from the head to the dependent. The type of a dependency relation is indicated by a label on the arc.

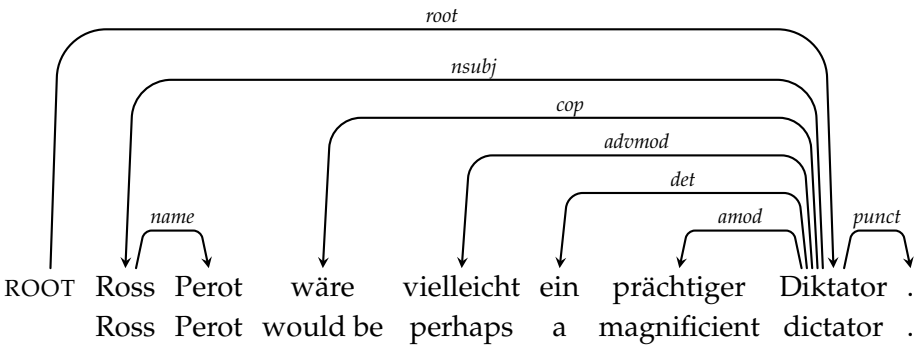


Figure 2.1: The syntactic structure of the first sentence in the TiGer treebank (Brants et al. 2002).

A well-formed (syntactic) dependency tree must fulfill three conditions:

1. There is one word that does not have a head (the root).
2. Each other word has exactly one head.
3. The tree does not contain cycles.

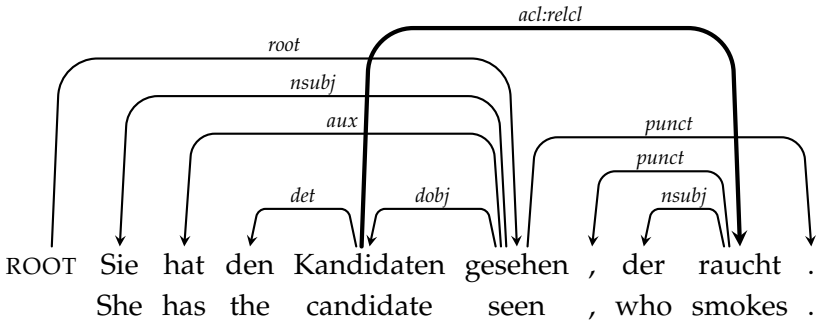
The formal graph structure that fulfills all three conditions is a directed rooted tree. Rootedness is ensured by introducing an artificial node (ROOT) that acts as the root of the tree.

We will denote dependency arcs formally as triples $\langle h, d, l \rangle$ with h being the head word, d being the dependent word, and l being the arc label denoting the dependency relation

between h and d. When we use indices, we index the artificial root node with 0 and all real words starting from 1 in the order they appear in the sentence. Furthermore, we will use the term *token* to refer to the smallest unit of parsing, which in this chapter will usually coincide with the intuitive meaning of a word in a sentence.

2.1.1 Projectivity

One recurrent problem in multilingual dependency parsing are non-projective arcs. An arc in a dependency tree is said to be non-projective if there is at least one word in the span of words covered by the arc that is not a direct or indirect descendant of the head of the arc. Figure 2.2 gives an example for a non-projective arc. The arc (acl:relcl) covers the span from *Kandidaten* to *raucht*. It is non-projective because *gesehen* is not a direct or indirect descendant of the head *Kandidaten*.



She saw the candidate who smokes.

Figure 2.2: Extraposed relative clauses in German can be represented with non-projective structures. The non-projective arc is bold-faced.

Formally, let $\rightarrow$ be the direct dominance relation defined by a dependency tree and let $\rightarrow^*$ be the transitive closure of $\rightarrow$. $h \rightarrow d$ then denotes a dependency arc in the tree and $h \rightarrow^* d$ denotes a path from h to d along the arcs of the tree. A dependency arc $h \rightarrow d$ with $h \prec d$ (the head precedes the dependent in the sentence) is defined as projective iff

$$\forall k \quad h \rightarrow^* k \quad \text{for } h \prec k \prec d \tag{2.1}$$

otherwise it is non-projective. The case for $d \prec h$ works analogously. A dependency tree is called non-projective if it has at least one non-projective arc. Otherwise, it is called projective. There are considerably more possible non-projective trees for a given sentence

than projective trees, which significantly increases the space of possible solutions that a parser needs to search through and thus the complexity of the parsing problem. While English has only few cases that are commonly analyzed with non-projective structures (e.g., wh-extraction), other languages show many such phenomena, especially languages with unrestricted word order. Parsing such languages requires parsing algorithms that can deal with non-projective edges.

Non-projectivity has also been studied with the aim to identify subclasses of non-projective structures that can be parsed efficiently. Several such classes were identified and it turns out that most of the structures (up to 99%) in today's treebanks fall into one of these subclasses (Havelka 2007, Gómez-Rodríguez et al. 2011, Kuhlmann 2013). Subsequently, parsing algorithms were developed that derive restricted sets of non-projective structures (see Section 2.2).

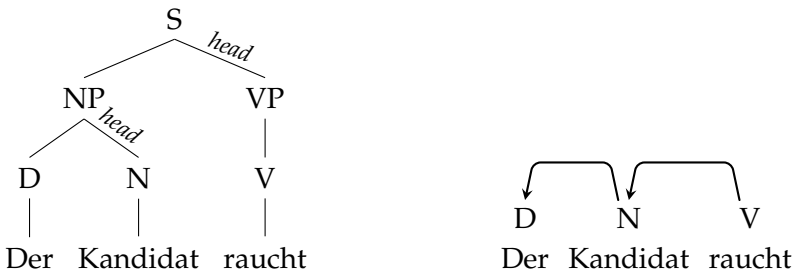
Most treebanks used in this dissertation contain non-projective structures since rich morphology and free word order often go hand in hand. The parsers that we develop as well as the baseline parsers therefore all derive non-projective structures either directly or via additional processing.

2.1.2 Treebanks

Treebanks are collections of natural language utterances (usually sentences) that have been annotated manually with their syntactic structure. The nature of the syntactic structure thereby depends on the linguistic theory that underlies the annotation guidelines. One of the oldest and perhaps the most influential treebank is the Penn Treebank (Marcus et al. 1993), which is annotated with constituency structures. With the creation of the Penn Treebank, it was possible to learn statistical models for predicting the syntactic structures of English sentences that are not in the treebank. This sparked extensive research on statistical parsing for natural language and it also led to the creation of treebanks for many other languages.

Some of the treebanks for other languages, e.g., the Prague Dependency Treebank Hajič et al. (2006), use dependency grammar as their underlying formalism. Training statistical dependency parsers on these treebanks is straight-forward. However, for many languages,

the treebanks are annotated with constituency structures and are thus not directly usable for dependency parsing. However, for these languages it is common to convert the constituency structures to dependency structures in order to have training data in the right format. This is done with head percolation rules (Magerman 1995, Collins 2003), which define for each constituent which one of their child nodes should be the head. For example, the right-most noun in a noun phrase is the head of that noun phrase. All other constituents are then attached as dependents to the selected head. The process is applied recursively until all constituents are converted. Figure 2.3 illustrates the process.



(a) Constituency tree. **(b)** Dependency tree.

Figure 2.3: Conversion by head percolation rules. For each constituent, the head is selected among its children.

There exist several automatic dependency conversions of the Penn Treebank (Yamada and Matsumoto 2003, de Marneffe et al. 2006, Johansson and Nugues 2007a, Choi and Palmer 2010). Although the automatic conversion can make mistakes, the quality of the conversion is very high and the converted treebanks are the accepted gold standard for training dependency parsers for English. Conversion procedures like this exist for many languages. In some cases, additional effort was spent to correct conversion errors by manually checking the resulting dependency structures (see for example the Hungarian dependency treebank (Vincze et al. 2010)).

In this dissertation, experiments on German are conducted on our own conversion of the TiGer Treebank (Brants et al. 2002). The conversion is performed semi-automatically by defining the head percolation rules such that they can fail (Seeker and Kuhn 2012). Failures were then inspected manually, which either led to more refined rules or to a correction of the original treebank. In a few cases, the heads were selected manually in order to treat exceptional cases that do not warrant their own rule and are not due to annotation errors in the original treebank.

Currently, there is an effort to create a unified dependency annotation for many languages. The Universal Dependency Treebank (Nivre 2015) aims at providing a common dependency analysis for many different languages in order to facilitate multi- and cross-lingual research. The example dependency trees shown in this dissertation are annotated according to these guidelines.

2.2 Dependency Parsing

Dependency parsing is the task of predicting a dependency tree for a given sentence. Statistical dependency parsers usually consist of three components. The first one is a parsing algorithm or decoder, which is a search algorithm that derives the dependency tree for a given sentence. The second component is a feature model that maps the input sentence along with potential parsing actions or dependency structures into a high-dimensional vector space. The third component is a statistical model that is trained on a treebank and uses the learned weights to score the high-dimensional vector representations.

There are currently two major paradigms in dependency parsing, transition-based parsing and graph-based parsing. The two paradigms approach the problem of dependency parsing from two different sides. Transition-based parsers uses rich feature representations but approximate search algorithms to find the best dependency tree, while graph-based parsers restrict their feature space to be able to efficiently perform global search thus having a guarantee to find the optimal structure. We discuss both of them in turn.

2.2.1 Transition-based Parsing

Transition-based parsers build the syntactic structure of a sentence incrementally by starting from an initial configuration and then repeatedly transitioning into different configurations by performing one of a set of predefined operations. The parsing process is finished when the parser ends in a defined final configuration. The sequence of operations (the *transition sequence*) that the parser performs to transition from the initial configuration to the final configuration encodes the syntactic structure of the input sentence.

As an example, Figure 2.4 shows the Arc-Standard transition system as formalized by Nivre (Yamada and Matsumoto 2003, Nivre 2004). A configuration in this system consists of three components:

1. β, an input buffer that holds the words of the input sentence, indexed from 0 to n with 0 representing the artificial root node
2. σ, a stack that serves as a memory and stores unfinished substructures
3. A, the set of arcs that form the output structure

The parser starts with the root node on the stack, all other tokens stored in the buffer, and an empty arc set. It then applies one of the three operations shown in Figure 2.4 to transition into a new state and repeats the procedure until it ends in the final configuration. The final configuration is reached when the buffer is empty and the only symbol on the stack is the artificial root node. At this point, the arc set holds all arcs that the parser has introduced between the words of the sentence.

Left-Arc$_l$ introduces an arc between the front of the buffer and the top of the stack with the front of the buffer being the head. The token on top of the stack is then discarded. Right-Arc$_l$ introduces an arc between the same items but makes the top of the stack the head. The token in front of the buffer is discarded and the top of the stack is put back onto the buffer. The third operation, Shift, simply pushes the token in front of the buffer onto the stack. Left-Arc$_l$ and Right-Arc$_l$ are additionally parameterized for the label that they introduce on the arc.

	Transition	Precondition
Left-Arc$_l$	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma, w_j\|\beta, A \cup \{\langle w_j, w_i, l\rangle\})$	$i \neq 0$
Right-Arc$_l$	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma, w_i\|\beta, A \cup \{\langle w_i, w_j, l\rangle\})$	
Shift	$(\sigma, w_i\|\beta, A) \Rightarrow (\sigma\|w_i, \beta, A)$	

Figure 2.4: The Arc-Standard transition system adapted from Kübler et al. (2009: fig. 3.1).

A statistical multiclass classifier is used to decide at each step, given the current configuration, which of the three operations the parser should apply. The classifier is trained on oracle transition sequences. Oracle transition sequences are derived from manually annotated treebanks by running the transition system on a sentence such that it derives

the treebank tree for this sentence. It is possible that there is more than one oracle transition sequence for a given sentence. A canonical sequence can be defined by ranking the operations in the transition system, preferring higher-ranked ones in case multiple are allowed. Ranking the operations as in Figure 2.4 (Left-Arc $>$ Right-Arc $>$ Shift), the canonical oracle transition sequence for the example sentence in Figure 2.1 would be Shift, Right-Arc$_{name}$, Shift, Shift, Shift, Shift, Shift, Left-Arc$_{amod}$, Left-Arc$_{det}$, Left-Arc$_{advmod}$, Left-Arc$_{cop}$, Left-Arc$_{nsubj}$, Shift, Right-Arc$_{punct}$, Right-Arc$_{root}$, Shift.

In order to predict the next transition, the parser extracts features from its current configuration. These features include information about the next items in the buffer and the partially processed items on the stack. By accessing the stack, the feature model has access to the entire structure that has been build so far. As the parser advances, more structure is build and becomes available to the feature model.

There are several different flavors of transition-based parsers, all of which have in common that they build the output structure incrementally at each step predicting the next one based on the current configuration. Nivre (2008) makes a distinction between stack-based and list-based algorithms. Stack-based algorithms are e.g., the Arc-Standard algorithm shown above (Yamada and Matsumoto 2003, Nivre 2004) and the Arc-Eager algorithm (Nivre 2003). List-based algorithms are proposed in Covington (2001). Instead of a stack, they use one or more lists to store partially processed tokens. Non-directional parsers (Shen and Joshi 2008, Goldberg and Elhadad 2010b) abandon the strict left-to-right processing and instead allow introducing arcs between neighbouring tokens anywhere in the sentence. The statistical models guiding these parsers not only learn which tokens to connect but also which tokens should be processed before others.

Most of the transition-based algorithms derive projective trees only (with the exception of some of the list-based algorithms in Covington (2001) and the parser in Shen and Joshi (2008)). Modifications in different directions have been proposed to deal with non-projectivity: Nivre and Nilsson (2005) propose a preprocessing step that projectivizes trees prior to parsing and reintroduces non-projective edges afterwards. Nivre (2009), Nivre et al. (2009) add a swap operation to the transition-system that reorders tokens during parsing in order to create a projective parsing order. The swap operation was applied in non-directional parsing by Tratz and Hovy (2011). With the swap operation transition-based algorithms are able to derive any possible non-projective structure for a given sentence, but it comes with an increased time complexity. Other approaches

avoid the increase in complexity by restricting the set of non-projective structures that can be derived (Attardi 2006, Gómez-Rodríguez and Nivre 2010, Pitler and McDonald 2015). These subsets can be parsed efficiently and are shown to include almost all of the non-projective arcs that can be found in the treebanks.

Transition-based parsers are fast due to their low time complexity. The stack-based variants find a tree in linear time with respect to the length of the input sentence and the list-based algorithms have quadratic complexity (for proofs, see e.g. Nivre 2008). The complexity of non-directional parser by Goldberg and Elhadad (2010b) is $\mathcal{O}(n \log(n))$. Introducing the swap operation increases the complexity of the stack-based parsers to $\mathcal{O}(n^2)$.

Another aspect that makes transition-based parsers fast and efficient is that they search for the best tree greedily, i.e., they always go with the locally best decision under the assumption that this will usually also lead to the globally best output. This is one of the fundamental differences to graph-based parsers, which perform global optimization to find the best tree. However, greedy search suffers from error propagation because once the parser has made an incorrect attachment it cannot correct it anymore. For this reason, transition-based parsers used to perform worse than graph-based parsers in terms of parsing accuracy. Since then, techniques like beam-search (Johansson and Nugues 2007b, Zhang and Clark 2008a) and dynamic programming (Huang and Sagae 2010), which allow the parser to pursue several derivations in parallel, closed that gap while increasing runtime by a constant factor only. At the same time, error propagation in greedy parsers has been mitigated with the help of dynamic oracles, which allow the parser to learn how to recover from past mistakes (Goldberg and Nivre 2012, 2013).

2.2.2 Graph-based Parsing

In contrast to transition-based parsers, graph-based parsers do not build dependency trees incrementally. Given a statistical model that assigns scores to dependency trees, graph-based parsers search through the space of all possible dependency trees for a sentence returning the one to which the statistical model assigns the highest score. This approach guarantees that the returned tree is indeed the optimal given the statistical model. However, since there are exponentially many trees for a given sentence, one cannot simply look at each possible tree individually and compare their scores. Instead,

algorithms were developed that efficiently search through the space without having to create every single tree.

Eisner (1997, 2000) develops a dynamic programming algorithm that finds the optimal dependency tree for a given sentence in cubic time, but is restricted to projective trees. The algorithm works like the chart parsers known from constituency parsing and uses the fact that subtrees of a dependency tree are also dependency trees. Alternatively, McDonald et al. (2005) propose to use spanning tree algorithms to find the optimal dependency tree for a given sentence. The particular algorithm used in McDonald et al. (2005) is the Chu-Liu-Edmonds algorithm. It is not restricted to projective trees and with clever implementation it runs in quadratic time with respect to the length of the input sentence.

The Chu-Liu-Edmonds Algorithm

The Chu-Liu-Edmonds algorithm (Chu and Liu 1965, Edmonds 1967) is a maximum spanning tree algorithm. Given a graph with arc weights, it searches for the spanning tree that connects all vertices in the graph and has the maximum sum of arc scores. We make use of this algorithm in several places in this dissertation, e.g., to enforce tree properties (Chapter 7). For this reason, we present the algorithm in this section, following the description in Kübler et al. (2009: 48). We demonstrate the algorithm by using an example. Proper pseudocode and a longer discussion can be found in McDonald et al. (2005) and Kübler et al. (2009: 47).

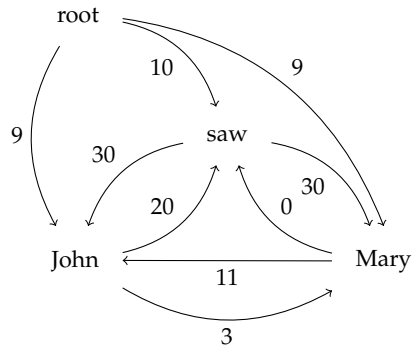
Figure 2.5 shows an example run of the algorithm on a small graph with four nodes. The nodes represent the sentence ⟨ROOT *John saw Mary*⟩. The algorithm starts from a fully connected graph (Figure 2.5a) whose arcs are weighted, e.g., by a statistical parsing model. The first step is to make a greedy selection by choosing the highest-scoring incoming arc for each word (Figure 2.5b). The algorithm would stop at this point if the resulting structure is a proper tree since this tree would be the maximum spanning tree over the original graph. However, greedy selection can create cycles as shown in the example between *John* and *saw*. In such a case, the algorithm contracts the cycle into a single node and recomputes the arc scores for each arc that enters or leaves the contracted cycle (Figure 2.5c). The new arc scores are computed depending on the score of the cycle and the scores of the incoming/outgoing arcs (see Kübler et al. 2009: 47 for the exact procedure).

Once the cycle is contracted and the new arc scores are computed, the algorithm calls itself recursively on the new smaller graph starting again with a greedy selection. If another cycle is found, the contraction procedure starts again and the recursion continues. Since each recursive call reduces the number of nodes in the graph due to the contraction, the algorithm eventually finds a tree structure without a cycle. This is shown in Figure 2.5d. From there, the contractions are resolved until an acyclic tree structure is obtained for the original graph (Figure 2.5e).

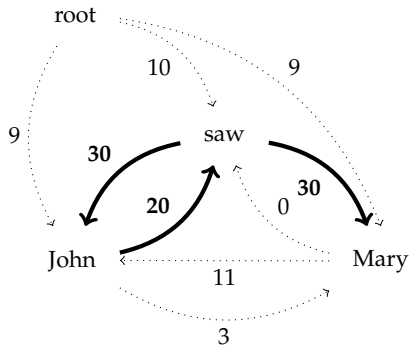
Arc-factored Model and Higher-order Factors

Eisner's algorithm and the Chu-Liu-Edmonds algorithm both rest on the assumption that the individual arcs in the tree are independent of each other. The underlying statistical model assigns a score to each arc and the score of a tree is the sum of all arc scores. The score of an arc is thereby independent of any other arc in the tree, i.e., it does not change if the other arcs in the tree change. This model is called the arc-factored model, since the parameters of the model factor over single arcs.

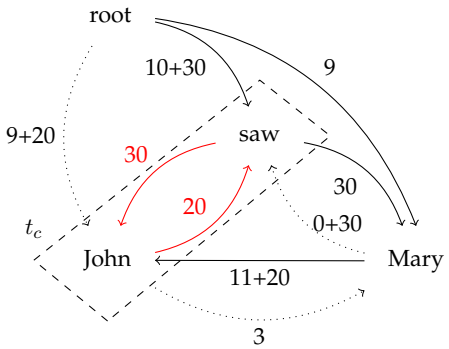
Arc-factorization makes the search for the best dependency tree tractable but it is an unrealistic model from a linguistic point of view. To see its limitations, consider the two German sentences in Figure 2.6. Both sentences contain a dependent clause, the upper one contains a relative clause, the lower one contains a subordinate clause expressing a clausal relationship. The crucial difference here is that the relative clause depends on the object of the matrix clause whereas the subordinate clause depends on the verb (red arcs in both trees). The head of both dependent clauses is the same verb *bellt*. An arc-factored model now has to decide the attachment of this word without knowing any of the other arcs in the tree. In particular it does not know the arcs marked in blue. In the upper sentence, the blue arc attaches the relative pronoun to *bellt*, in the lower sentence, it attaches the subordinating conjunction. Linguistically, this information makes the attachment decision trivial, but the arc-factored model cannot access it. In short sentences like the examples in Figure 2.6, this information can be approximated by looking at the surrounding words. The relative pronoun in the upper sentence is the immediate left neighbor of *bellt*. However, German syntax allows for any number of other words to occur between *bellt* and the relative pronoun/subordinating conjunction, which makes surrounding context a rather unreliable source of information for this purpose.



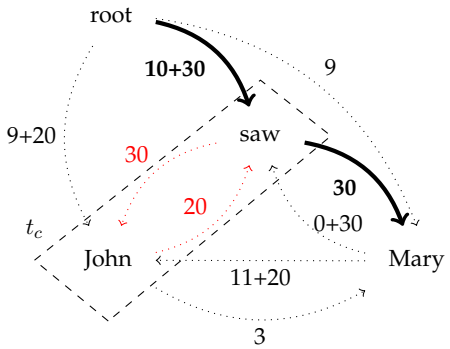
(a) Start from a fully connected graph.



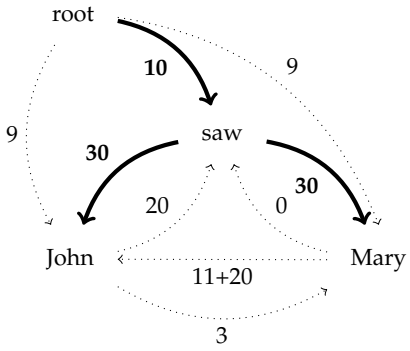
(b) Select greedily the highest-scoring incoming arc for each node.



(c) Contract cycles and recompute scores.



(d) Run Chu-Liu-Edmonds on the contracted graph.



(e) Resolve contractions.

Figure 2.5: The Chu-Liu-Edmonds algorithm in five steps (see also Kübler et al. 2009: 48).

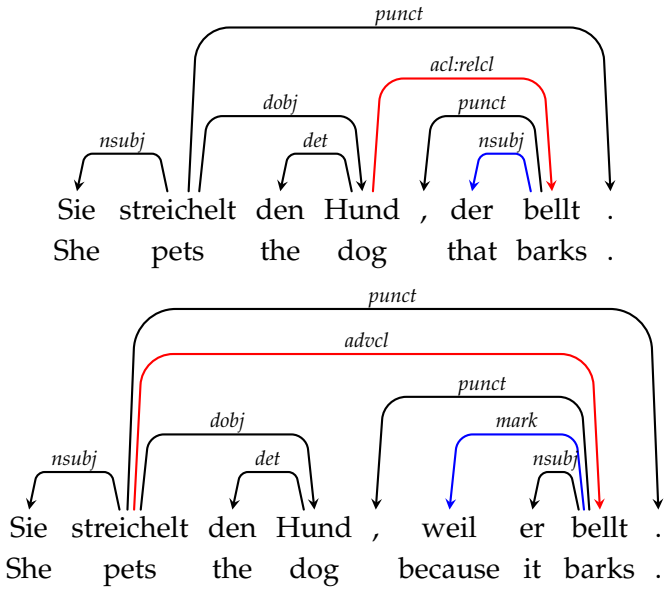
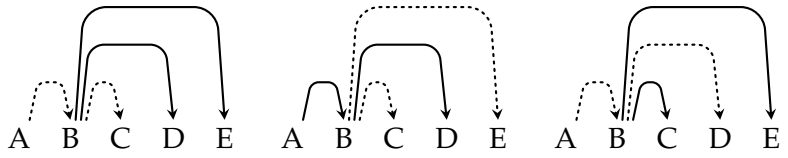


Figure 2.6: Attachment of dependent clauses to illustrate second-order dependencies.

Figure 2.6 shows that modeling dependencies between arcs opens access to relevant information that is hidden from the arc-factored model. Unfortunately, higher-order models[2] come with increased complexity. Full non-projective parsing with more than first-order models was proven to be NP-hard (for the proof, see McDonald and Pereira 2006), but Eisner's algorithm, which derives projective trees only, can be extended to higher order models while keeping its polynomial complexity. McDonald and Pereira (2006) propose a variant of Eisner's algorithm that uses factors of consecutive siblings (Figure 2.7a), to which Carreras (2007) adds factors over grandchildren (Figure 2.7b). While the variant with consecutive siblings retains its cubic time complexity, Carrera's decoder already runs with $\mathcal{O}(n^4)$ with n being the length of the input sentence. Koo and Collins (2010) go one more step and introduce third-order features while at the same time keeping the runtime at $\mathcal{O}(n^4)$. Zhang and McDonald (2012) generalize Eisner's decoder to any-order models, reporting a time complexity of $\mathcal{O}(n^{3+2x})$ where x is the number of free variables in their parsing rules.

[2]Models that consider dependencies between two arcs are called second-order models. If they model dependencies between three arcs they are called third-order models. The arc-factored model is hence called a first-order model.



(a) Consecutive siblings. **(b)** Grandchildren. **(c)** Arbitrary siblings.

Figure 2.7: Second-order factors. Solid lines show one factor.

Factorization restricts the type of features that the feature model has access to. In the arc-factored case, features can only be extracted from single arcs, in higher-order models, sibling or grandchild features can be added. However, structural features can never go beyond the information retained in a factor. This is one of the fundamental differences between graph-based and transition-based parsers as the latter have access to their entire parse history, can extract structural features of any complexity, and are restricted only be the fact that some structure may not have been built yet.

Eisner's algorithm allows for efficient higher-order models, but (and because) it can derive projective trees only. In order to derive non-projective trees, different modifications have been proposed. McDonald and Pereira (2006) propose a hill-climbing algorithm for postprocessing that starts from the highest-scoring projective tree and reattaches edges until the overall score does not increase anymore. Pitler (2014) extends Eisner's algorithm to directly derive 1-Endpoint-Crossing trees,[3] a subset of all possible non-projective structures (see also Pitler et al. 2013).

While clever factorization keeps the parsing algorithms tractable, they are still rather slow, especially compared to greedy transition-based parsers. Many graph-based parsers therefore run a pruning step first that uses a simple method for cutting of arcs that are unlikely to be chosen by the parser. For example, a sentence with 100 words requires the parser to consider 99 heads for each word (in the non-projective case). Cutting this number down to 10 heads per word makes the combinatory problem considerably smaller and leads to faster parsing time. Of course, if the pruning step cuts off a correct arc, it cannot be recovered by the parser. A popular method to decide which arcs to keep is to

[3] In 1-Endpoint-Crossing trees, a non-projective arc is crossed by arcs that all have the same endpoint.

use the marginals of a probabilistic arc-factored model (McDonald and Satta 2007, Smith and Smith 2007, Koo et al. 2007). Rush and Petrov (2012) use cascades of models with increasing complexity so that the simpler models narrow down the search space for the more complex models. Zhang and McDonald (2012) show how any kind of higher-order model can be kept tractable by using cube pruning to restrict the number of arcs that are considered between two words.

Parsing as an Integer Linear Program

Parallel to the effort of extending Eisner's decoder, other methods of finding the optimal tree were investigated, for example dependency parsing as solving an integer linear program. Integer linear programming is a mathematical tool to describe constrained optimization problems. It consists of an objective function that is being optimized and a set of constraints over the variables in the objective function that need to hold in the optimal solution.

The general idea is the following: each possible arc between the tokens of a given sentence is represented by a binary variable that marks the presence or absence of this arc in the output tree (recall the representation of dependency trees as indicator vectors from above). Each of these binary variables is weighted by an arc score from a statistical model. The objective function of the integer linear program is then to optimize the overall score of the tree by finding the combination of binary variables whose weights sum up to the highest score.

Without any constraints, the solution to this optimization problem is to set all variables to 1 that have a positive arc weight. However, this will most likely not result in a well-formed dependency tree. Therefore, some constraints are added to the integer linear program that only allow solutions which are well-formed dependency trees. The conditions that need to be met are listed in the beginning of this section (root has no head, one head for each token, no cycles). Riedel and Clarke (2006), who proposed the first parser based on integer linear programming, defined constraints for the first two conditions, but had to resort to an iterative method to enforce acyclicity. They first compute a solution, and if the solution contains a cycle, they add a constraint that explicitly excludes this particular cycle. They then run the solving process again, possibly ending up with another cycle, until finally

an acyclic solution is found. Obviously, this iterative process makes the approach very inefficient. Martins et al. (2009) find a concise formulation that directly enforces acyclicity without the need for an iterative process (see also Martins et al. 2010b). Their parser ensures cycle-freeness by employing a single-commodity flow formulation (Magnanti and Wolsey 1995). It models a flow from the root to each of the tokens in the sentence along the arcs of the tree. Together with the single-head constraint, only acyclic trees can fulfill this constraint.

Solving the integer linear program with the described constraints outputs a well-formed dependency tree that is optimal with respect to the scoring function. Any general-purpose constraint solver for (integer) linear programs can be used to find the optimal solution. This parser is attractive from a modeling point of view, because any kind of other constraints can be added to the formulation. We make use of this property in this dissertation to model dependencies between morphology and syntax. A formal definition of the parser is therefore deferred to Chapter 6. Higher-order features can be added through the definition of additional variables that are linked to the arc variables via constraints. The higher-order dependencies are scored by the statistical model and their corresponding variables are included into the objective function. Martins et al. (2009) propose several second-order features, e.g., the already mentioned consecutive siblings, grandchildren, or arbitrary siblings Figure 2.7c.

As with all graph-based parsers, the disadvantage of integer linear programming parsers is their complexity. Dropping the integer constraint, i.e., allowing for the variables to take any real value, creates a linear program, for which solvers exist that run in polynomial time. However, relaxing the problem in this way forfeits the guarantee to get a well-formed dependency tree since some of the variables in the solution may end up with fractional values. Martins et al. (2009) postprocess such fractional solutions by projecting them to the nearest integer solution. This is done by running the Chu-Liu-Edmonds algorithm on the first-order output graph with the fractional assignments as arc weights. However, they find that in the vast majority of cases the projection is unnecessary since the original solution already only contains integers.

Lagrangian Relaxation and Dual Decomposition

Lagrangian relaxation is another method for solving constrained optimization problems that trades the guarantee for an exact solution for more efficient decoding. Rush et al. (2010) introduced this method to perform efficient inference in complex models for natural language processing. The idea of Lagrangian relaxation is to solve a hard constrained optimization problem by moving some or all of the constraints into the objective function and then searching for the solution that maximizes the original function while at the same time violating the constraints as little as possible. The following part is based on Rush and Collins (2012).

Assume that we have a problem where we wish to maximize a set of variables $\boldsymbol{x}$ given a set of parameters $\boldsymbol{\theta}$. Additionally, we have a set of constraints on the values of $\boldsymbol{x}$.

$$\begin{aligned} &\arg\max_{\boldsymbol{x}} \boldsymbol{x} \cdot \boldsymbol{\theta} \\ &\text{subject to} \quad A\boldsymbol{x} = \boldsymbol{b} \end{aligned} \tag{2.2}$$

We assume now that we can solve the unconstrained problem efficiently, but with the constraints it is very difficult to do so. The idea of Lagrangian relaxation is to circumvent the constraints that make the problem difficult to solve by moving them into the objective function together with a set of Lagrangian multipliers ($\boldsymbol{\lambda}$).

$$L(\boldsymbol{\lambda}, \boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{\theta} + \boldsymbol{\lambda} \cdot (A\boldsymbol{x} - \boldsymbol{b}) \tag{2.3}$$

The dual objective of the original problem is still to find the maximum values of $\boldsymbol{x}$

$$L(\boldsymbol{\lambda}) = \arg\max_{\boldsymbol{x}} L(\boldsymbol{\lambda}, \boldsymbol{x}) \tag{2.4}$$

and the dual problem is to minimize the value of the Lagrangian multipliers

$$\arg\min_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda}) \tag{2.5}$$

Solutions to the dual objective are upper bounds to the solution of the original function and

by minimizing the dual problem, the upper bound is moved closer to the optimal solution. Rush and Collins (2012) show how to use subgradient descent for the optimization. The solution that is output in the end is not guaranteed to be identical to the optimal solution of the original problem. However, if at any point in the optimization the constraints are not violated, i.e., $A\boldsymbol{x} - \boldsymbol{b} = \mathbf{0}$, then the upper bound and the optimal solution coincide and the $\boldsymbol{x}$ at this point is guaranteed to be optimal.

Dual Decomposition is a special case of Lagrangian relaxation in which the constrained optimization problem can be decomposed into two or more sub-problems and the constraints connect the sub-problems in some way. As before, it is assumed that the sub-problems can be solved efficiently when the constraints are ignored.

Let the original problem be

$$\begin{aligned} \arg\max_{\boldsymbol{x},\boldsymbol{z}} &= \boldsymbol{x} \cdot \boldsymbol{\theta^1} + \boldsymbol{z} \cdot \boldsymbol{\theta^2} \\ \text{subject to} \quad & A\boldsymbol{x} + B\boldsymbol{z} = \boldsymbol{c} \end{aligned} \tag{2.6}$$

where $\boldsymbol{x}$ and $\boldsymbol{z}$ are the variables for the two sub-problems and $\boldsymbol{\theta^1}$ and $\boldsymbol{\theta^2}$ are the corresponding sets of parameters. A, B, and $\boldsymbol{c}$ define the constraints over $\boldsymbol{x}$ and $\boldsymbol{z}$.

The constraints are integrated into the objective function as before

$$L(\boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x} \cdot \boldsymbol{\theta^1} + \boldsymbol{z} \cdot \boldsymbol{\theta^2} + \boldsymbol{\lambda} \cdot (A\boldsymbol{x} + B\boldsymbol{z} - \boldsymbol{c}) \tag{2.7}$$

and the dual objective is to maximize $\boldsymbol{x}$ and $\boldsymbol{z}$.

$$L(\boldsymbol{\lambda}) = \arg\max_{\boldsymbol{x},\boldsymbol{z}} L(\boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{z}) \tag{2.8}$$

The dual problem is as before to minimize the value of the Lagrangian multipliers.

$$\arg\min_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda}) \tag{2.9}$$

Rush et al. (2010) illustrate the use of dual decomposition with two problems. In the first, they show a model for joint phrase-structure parsing and part-of-speech tagging, in the second they combine a phrase-structure parser with a dependency parser. They

derive simple subgradient algorithms to optimize the complex models. In the case of joint phrase-structure parsing and part-of-speech tagging, the complex problem is decomposed into two problems, namely phrase-structure parsing and part-of-speech tagging. Both tasks alone are well-studied and can be solved efficiently with known algorithms. The difficult problem of the joint task is to enforce the equality constraints that postulate that the part-of-speech tags assigned by the parser should be the same as the ones assigned by the tagger. They can solve this problem efficiently with the described dual decomposition approach.

Rush et al. (2010) show that their algorithms solve a linear programming relaxation of the joint problem, making the method equivalent to the relaxed version of the parser based on integer linear programming described in the previous section. The same method can also be used not to solve a joint problem but rather to keep a complex problem tractable. Koo et al. (2010) develop a dependency parser that models second-order features with head automata. The head automata are used to model the right and left dependent of each individual word in the input sentence. However, since all head automata are optimized independently of each other, it is likely that different automata contradict each other and thus their union will not lead to a well-formed dependency tree. To make the automata agree on a common dependency tree, Koo et al. (2010) use a dual decomposition algorithm that alternatingly optimizes the head automata and the Chu-Liu-Edmonds algorithm. The purpose of the latter is to enforce a tree structure since it always outputs a spanning tree over the input sentence.

We make use of head automata in Chapter 7 to model second order features in our lattice parsing model. The lattice parser uses dual decomposition to find the optimal tree and segmentation for a given morphological lattice. However, we do not use a subgradient algorithm like Rush et al. (2010) to find the optimal solution. Instead, we use Alternating Directions Dual Decomposition or AD^3 (Martins et al. 2010a, 2011a,b, 2015). AD^3 is a form of augmented Lagrangian relaxation, where a regularization term is added to each subproblem in order to facilitate faster convergence to a common solution. In essence, this method does the same as the subgradient algorithms of Rush et al. (2010) only more efficiently. It is, however, also better suited to deal with situations where there are many sub-problems. For example, we use multiple logic constraints in Chapter 7 to enforce well-formed output structures.These logic constraints are treated as additional sub-problems in the problem formulation increasing the number of sub-problems considerably.

2.2.3 Transition-based or Graph-based Parsing?

Both parsing paradigms have different strengths and weaknesses and are often even complementary to each other (see e.g. Nivre and McDonald 2008). Transition-based parsers are fast and can tap into rich feature representations but rely on inexact search and may suffer from error propagation. Graph-based parsers are guaranteed to find the globally optimal structure, but do so with restricted feature sets and complex (and therefore slow) decoding algorithms. While global optimization initially gave an edge to the graph-based parsers, transition-based parsers have since caught up and both types are reaching state-of-the-art performance.

Zhang et al. (2014b) have recently shown with their sampling-based parser that greedy decoding can be as good as global optimization. One of their motivations comes from the observation that relaxation methods (e.g. Koo et al. 2010) most of the time still arrive at an exact solution when used for dependency parsing. The search space in dependency parsing might be structured in a way that it is essentially unnecessary to search the full space in order to find the best tree. This hypothesis seems even more plausible if one thinks about the simple models that are commonly used for pruning the search space before parsing. Given these developments, a rich set of features that model the underlying data well seems to be more important than a search algorithm that searches the entire space of possible dependency trees to give a guarantee for optimality.

2.3 Training the Statistical Models

In this dissertation, we use supervised learning to learn the statistical models. Supervised learning means that we train the model on data for which we already know the correct solution. In Chapter 5, we predict the morphological features of words. The training data is therefore a corpus for which the morphological features have been manually annotated. Analogously, when we train parsers, we train them on treebanks. In this case, the model is used to predict dependency trees given an input sentence.

Let X be the set of inputs and Y be the set of outputs. For example, $x \in X$ could be a word in morphology prediction or a sentence in parsing, and $y \in Y$ could be a

morphological feature label or a dependency tree. A training set for supervised learning then consists of elements from the input set labeled with elements from the output set, i.e., $T = \{\langle x, y\rangle, \ldots\}$ is a set of input-output pairs. The y's in T are also called the gold standard and are assumed to be correct.

Throughout the dissertation we use linear models to obtain a score for an input-output pair. In prediction, the score is used to find the optimal y for a given x. The input is first mapped into a high-dimensional feature space using a feature function ϕ. Most of the features that we work with are binary, i.e., the value of a single feature can be either 0 or 1. The statistical model is a function that takes the features as input and computes a score. It is called linear because the score depends linearly on the feature values. Training the model means to learn a weight for each feature.

$$score(x, y) = \boldsymbol{w} \cdot \phi(x, y) = \sum_{i=0}^{d} w_d * \phi_d(x, y) \tag{2.10}$$

Equation (2.10) shows the general scoring function. Given a pair $\langle x, y\rangle$ and a weight vector $\boldsymbol{w}$, the score is computed by taking the dot product of $\boldsymbol{w}$ and the feature vector produced by the feature function $\phi(x, y)$. The dot product is the sum of the pairwise multiplication of each weight with its corresponding feature. d is the number of different features, i.e., $\boldsymbol{w} \in \mathbb{R}^d$ and $\phi(x, y) \in \mathbb{R}^d$.

The models in this dissertation are trained with a general online learning algorithm as shown in Algorithm 1. It takes a labeled training set and a predefined number of iterations (usually set to 10). The algorithm goes through the training data one instance after the other, each time making a prediction (Line 6). The weight vector then updated with respect to the prediction (Line 7). We furthermore use averaging to prevent overfitting and to obtain models that generalize better to unseen data (Freund and Schapire 1999, Collins 2002).

We train models for two purposes: multiclass classification and structured prediction (Collins 2002). The first case is used for example in Chapter 5 to predict morphological feature values for words. The input are words in their context and the output is a label from a set of morphological feature descriptions. For each word, the statistical model takes a feature representation of the word in its sentential context and outputs the feature

Algorithm 1 Online Learning with Averaging

Require: $T = \{\langle x_0, y_0\rangle, \ldots, \langle x_t, y_t\rangle\}$ ▷ The labeled training set
Require: number of iterations I
1: $\boldsymbol{w} = \mathbf{0}$ ▷ Initialize the weight vector
2: $\boldsymbol{w_a} = \mathbf{0}$ ▷ Keep a second vector for averaging
3: **for** $i = 1$ to I **do**
4: SHUFFLE(T) ▷ Shuffle the training data
5: **for all** $\langle x, y\rangle \in T$ **do**
6: $\hat{y} = \text{PREDICT}(\boldsymbol{w}, x)$ ▷ Make a prediction
7: UPDATE($\boldsymbol{w}, x, \hat{y}, y$) ▷ Update the weights according to prediction
8: $\boldsymbol{w_a} = \boldsymbol{w_a} + \boldsymbol{w}$ ▷ Store the current weight vector
9: **end for**
10: **end for**
11: $\bar{\boldsymbol{w}} = \boldsymbol{w_a}/(T * I)$ ▷ Average the weights
12: **return** $\bar{\boldsymbol{w}}$

label with the highest score.

$$\hat{y} = \arg\max_{y \in Y} \boldsymbol{w} \cdot \phi(x, y) \tag{2.11}$$

The parsers developed in this dissertation belong to the graph-based paradigm and are trained with structured prediction. Unlike in the multiclass prediction case, the output values Y have an internal structure, i.e., they are dependency trees. As there are exponentially many dependency trees for a given sentence, it is not feasible to simply do an argmax over all possible output values as in Equation (2.11). However, since we are only interested in the highest-scoring dependency tree, we can use one of the parsing algorithms from above, say Chu-Liu-Edmonds, to find the highest-scoring dependency tree efficiently without having to go through exponentially many trees one after the other. Recall that this is efficient because the amount of information to which the statistical model has access is limited.

$$\hat{y} = \text{Chu-Liu-Edmonds}(\boldsymbol{w}, \phi, x) \tag{2.12}$$

Equations (2.11) and (2.12) are the instantiations of Line 6 in Algorithm 1 for multiclass prediction and structured prediction, respectively. For adjusting the weights of a model during training (Line 7 in Algorithm 1), we use the passive-aggressive update rule by Crammer et al. (2003, 2006) which is shown in Equations (2.13) and (2.14). The model is

updated only if the prediction is incorrect with respect to the gold standard. The update changes the weight vector just as much as it needs to make a correct prediction for the current input, but not more because the model should stay good on the examples where it made correct predictions. The name passive-aggressive describes this behaviour: it is passive when the model made a correct prediction, but in the other case it aggressively changes the weights to get a correct prediction next time.

$$\delta = \frac{\boldsymbol{w} \cdot \phi(x, \hat{y}) - \boldsymbol{w} \cdot \phi(x, y) + \text{LOSS}(\hat{y}, y)}{\|\phi(x, y) - \phi(x, \hat{y})\|^2} \tag{2.13}$$

$$\boldsymbol{w} = \boldsymbol{w} + \delta(\phi(x, y) - \phi(x, \hat{y})) \tag{2.14}$$

In Equations (2.13) and (2.14), $\phi(x, y)$ is the feature vector of the gold standard and $\phi(x, \hat{y})$ is the feature vector of the best prediction. Note that in the multiclass case, $\hat{y}$ is a single label whereas in the structured prediction case, it is a complete dependency tree. Equation (2.13) computes the amount δ by which the weight of each feature in the prediction and the gold standard is changed. In Equation (2.14), the weights for features in the gold standard are increased whereas the feature weights of the best prediction are decreased. After the update, the gold standard should get a higher score than the best prediction. Additionally, the passive-aggressive update enforces a margin between the score of the best prediction and the score of the gold standard that must be at least as big as the loss between the two. We use a zero-one loss in the multiclass prediction meaning that the loss is one if the predicted label is incorrect and otherwise 0. For parsing, the loss is a function of the number of tokens that did not get the correct head. Note that in structured prediction, the feature vectors in Equations (2.13) and (2.14) are the sum of the feature vectors for each factor in the structure, for example in the arc-factored model, it would be the sum of the feature vectors for each arc in the tree.

Chapter 3

Motivation

In this chapter, we develop the hypotheses and research questions of this dissertation. We first present morphological and syntactic phenomena of morphologically rich languages. We then examine the models that are commonly used in parsing and show that some of the assumptions that are built into these model do not hold for languages with rich morphology. This chapter sets the scene for the following chapters, in which we test the developed hypotheses empirically.

3.1 Morphology and Syntax

In English, the syntactic structure of a sentence is mainly expressed by the order in which the words appear in the sentence. Consider the example[1] by Bresnan (2001) in Figure 3.1. The fact that *children* comes before *are chasing* in Figure 3.1 determines the subjecthood of the word. In the same way, *dog* is the direct object because it follows the verb. Switching the positions of these words would result in a change of meaning, as now, *dog* would be subject and *children* would be the direct object.

Consider now the example in Figure 3.2, also by Bresnan (2001). This sentence is from the

[1]The original examples come with a phrase structure analysis which we changed to dependencies since the argument we want to make does not depend on the syntactic theory.

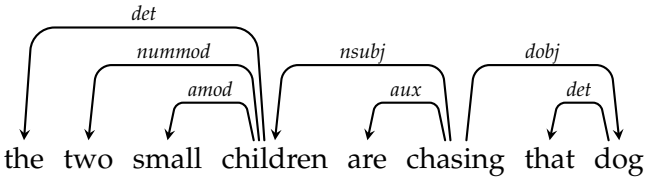


Figure 3.1: English syntactic structure is mostly expressed through word order. Example taken from Bresnan (2001: 5).

Australian language Warlpiri and expresses the same semantic concept as the sentence in Figure 3.1. According to Bresnan, any other order of the words in this sentence are also acceptable to express the same meaning as long as the auxiliary occupies the second position in the sentence. In Warlpiri, word order therefore cannot serve to determine the syntactic relationships between the individual words. Instead, the morphology of the words overtly marks the roles that they play in the syntactic structure of this sentence. The subject of the sentence, *witajarrarlu kurdujarrarlu*, is in ergative case, whereas the direct object, *yalumpu maliki*, is in absolutive case. Furthermore, the words for *small* and *children* do not need to be adjacent since their identical inflection relates them to each other.

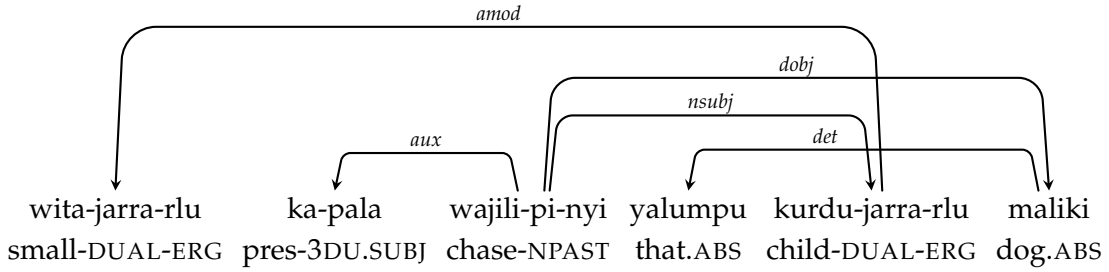


Figure 3.2: In Warlpiri, syntactic structure is expressed by morphology. Example taken from Bresnan (2001: 6).

These two sentences exemplify two opposite points on the scale of options that languages have to express syntactic structure. Bresnan uses them to illustrate a phenomenon commonly observed by language typologists: languages with rich morphology usually allow for free word order whereas languages with rather poor morphology often have very strict word order rules. Bresnan summarizes this observation with the slogan: *Morphology competes with syntax* (Bresnan 2001: 6).

Figure 3.2 makes it clear that without information about the morphology of the individual

words, automatic syntactic analysis (i.e., syntactic parsing) would have a hard time discovering the correct syntactic structure in Warlpiri. While not all languages in the world are as extreme as Warlpiri, most of them employ morphology to some extent to express syntactic structure. Morphological analysis is therefore an important part of syntactic parsing in a multilingual setting. However, automatic morphological analysis in itself is not a simple task either, since—as always in natural language processing—it needs to deal with ambiguity.

3.2 Syncretism

Consider the two German sentences presented in Examples 3.1 and 3.2. Both sentences contain the word *fahren* (to drive). In the first example, *fahren* is a finite word form in third person singular present tense, in the second example, *fahren* is the present tense infinitive. Although both word forms look the same, they have different morphological features and are indeed different inflections of the lemma *fahren*. The phenomenon when two (or more) morphological forms of a lemma have the same surface form is called *syncretism*.

(3.1) *Peter sagte, daß sie morgen nach Berlin* ***fahren***.
Peter said that.CONJ they tomorrow to Berlin drive.3-SG-PRES
Peter said that they will drive to Berlin tomorrow.

(3.2) *Peter will morgen nach Berlin* ***fahren***.
Peter want.MODAL tomorrow to Berlin drive.INFINITIVE
Peter wants to drive to Berlin tomorrow.

To correctly predict the morphological features of *fahren* in each sentence, one needs to know if the verb is the main verb of a subordinate clause introduced by *daß* (Example 3.1) or if the verb is embedded by the modal verb *will* (Example 3.2). Thus, the resolution of the syncretism must rely on syntactic information about the sentential context of the syncretic word.

Baerman et al. (2005: 2) characterize syncretism as a *mismatch between morphology and syntax*: A syntactically relevant distinction is not made by the morphology. In the examples above,

it is the distinction between finite and non-finite verb forms. Syncretism is a common phenomenon in many languages and occurs in verbal and nominal morphological paradigms. The World Atlas of Language Structures Online[2] lists 60 languages with syncretism in verbal PERSON and NUMBER marking out of the 141 languages listed that mark PERSON or NUMBER at all (Baerman and Brown 2013b). Similarly, it lists 40 languages with a syncretic case system out of the 75 languages listed that have a case system (Baerman and Brown 2013a).

Formally, syncretism can be characterized as an *Identity in form between two grammatically different inflections* (Trask 1997, as cited in Baerman et al. 2005: 2). Syncretism occurs when one surface form of a single word occupies more than one cell in this word's inflection paradigm. To illustrate this, Table 3.1 shows the declension paradigms of two Czech nouns, *bratr* (brother) and *město* (city). Syncretic forms are marked by different colors. The two examples show mostly syncretism with respect to CASE with the exception of the form *města* additionally being ambiguous with respect to NUMBER. Case syncretism is a typical property of Indo-European languages. Among these, Slavonic languages show the highest degree of variation and complexity (Baerman et al. 2005: 38).

MASC ANI	SG	PL
NOM	bratr	bratři
ACC	bratra	bratry
DAT	bratrovi/u	bratrům
GEN	bratra	bratrů
VOC	bratře	–
LOC	bratrovi/u	bratrech
INS	bratrem	bratry

(a) Czech, masculine animate: *brother*

NEUT	SG	PL
NOM	město	města
ACC	město	města
DAT	městu	městům
GEN	města	měst
VOC	město	–
LOC	městě/u	městech
INS	městem	městy

(b) Czech, neuter: *city*

Table 3.1: Syncretism in two Czech nominal inflection paradigms (masculine animate and neuter).

3.2.1 Disambiguation in Context

Given in isolation, neither machine nor human are able to fully disambiguate a word form like *bratra* in Table 3.1a. By the word form alone, we can disambiguate it to masculine

[2] www.wals.info

singular but cannot decide on the case value; for a word like *města* (Table 3.1b), we would not even be able to decide for the number value. However, we rarely encounter words in isolation but rather in sentences and texts. In a sentence, a word is embedded into a syntactic context that can be used to disambiguate the ambiguous word form.[3]

Examples 3.1 and 3.2 already show that the syntactic context (presence of a subordinating conjunction versus embedding under a modal verb) can serve to disambiguate a syncretic word form. Another way of disambiguating a given word form in a sentence is to use the morphosyntactic rules in the grammar of a language, e.g., rules of *government* and *agreement*. Both describe a particular relationship between words in a sentence:

- *Government* describes the situation when a word imposes certain morphological values onto another word, e.g., when a verb imposes specific case values on its nominal dependents. For example, German subjects have to be in nominative case.
- *Agreement* can be defined as a systematic co-variance of a semantic or formal feature between two words (Corbett 2006: 4). For example, the number feature of a subject and a predicate co-vary in English.

Figure 3.3 illustrates government and agreement with a simple German sentence. Government is shown in the arc above the sentence, agreement with the arcs below. The predicate *schläft* governs the nominative case of the noun *Schnecke*. The verb and the noun agree with respect to NUMBER and the noun agrees with the adjective and the determiner with respect to CASE, NUMBER, and GENDER.

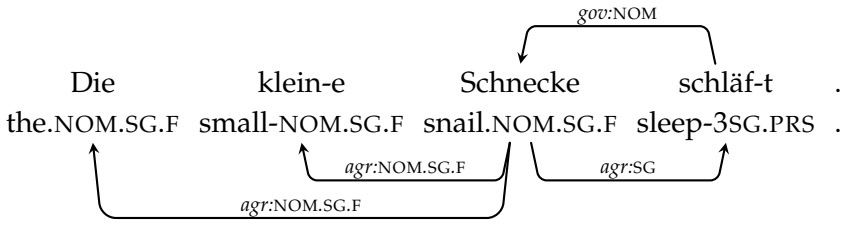


Figure 3.3: Government and Agreement examples in German.

[3]Humans use many different linguistic and non-linguistic sources to disambiguate a sentence. In this work, we focus on the syntactic context and its relation to morphological ambiguity. It is clear however that even though a system that perfectly models all interactions between syntax and morphology in a language would get us a big step forward, there would still be a long road ahead of us.

The government and agreement rules shown in Figure 3.3 are part of the grammar of German and need to be obeyed in German sentences. To see how they help in dealing with syncretism, consider the inflection paradigm of the German definite article and the noun *Schnecke* in Table 3.2. The noun in Table 3.2b is morphologically marked only for NUMBER, there are no forms that distinguish case values. German noun inflection generally distinguishes NUMBER and only in some cases marks CASE. The definite determiner in Table 3.2a, however, is generally better marked for CASE than for NUMBER. Both words therefore carry different loads of morphological information within a noun phrase, a situation that has been dubbed *Funktionsteilung* (*function sharing*) by Eisenberg (2006: 142).

	M.SG	N.SG	F.SG	MFN.PL
NOM	der	das	die	die
ACC	den	das	die	die
DAT	dem	dem	der	den
GEN	des	des	der	der

(a) German, definite determiner: *the*

Schnecke.F	SG	PL
NOM	Schnecke	Schnecke-n
ACC	Schnecke	Schnecke-n
DAT	Schnecke	Schnecke-n
GEN	Schnecke	Schnecke-n

(b) German, feminine: *snail*

Table 3.2: Syncretism in the definite determiner and a noun in feminine gender in German.

By relating the two words by government and agreement, as shown previously in Figure 3.3, both words are able to disambiguate each other. The nominative case is imposed on the noun by the governing verb and thus excludes all other case values. At the same time, the determiner agrees with it with respect to GENDER, NUMBER, and CASE. The fully specified noun therefore fully disambiguates the determiner by virtue of the Agreement relation.

We can thus see that access to syntactic structure is an important source for disambiguating morphological information in the cases where a word form is ambiguous. A word form that is ambiguous between nominative and accusative case in a language where subjects are marked by nominative case fails at marking subjecthood. The syntax must then rely on other means to determine the subject, e.g., NUMBER agreement.

A third source of information that interacts with government and agreement is the *valency*, or the *subcategorization*, of a verb . Consider the example in Figure 3.4. The word form *města* can be nominative or accusative plural, or genitive singular (see Table 3.1b). But the fact that the verb already has a subject (*bratr*) restricts the choice of functions for *města*, because verbs cannot have more than one subject. If *města* turns out to be direct object, its

correct case value must be accusative case as this is the case assigned to the direct object of the verb. The valency of the verb thus helps in disambiguating the morphological features of the ambiguous word form.

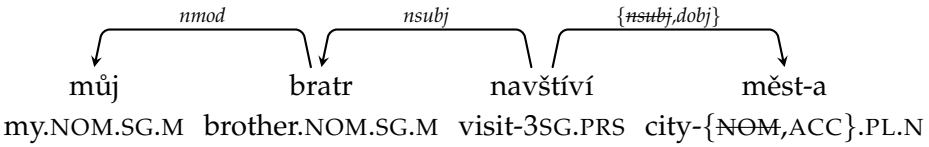


Figure 3.4: Verb valency helps disambiguating morphologically ambiguous word forms.

3.3 Ambiguity in Word Segmentation

Syncretism is a particular form of morphological ambiguity that is restricted to the word forms within the inflection paradigm of a single lemma. But word forms are often ambiguous across lemmas as well, take for example the English verb *bear* (to carry/support) and the English noun *bear* (an animal, e.g., a polar bear). The two words are *homonyms*, i.e., they are pronounced and written the same way but have different (and unrelated) meanings.[4]

In languages with rich morphology, homonymity occurs frequently also due to composition and derivation processes. Take for example the Turkish word *çekti* in Examples 3.3 and 3.4. The word can be segmented into different combinations of basic morphemes such that each distinct segmentation gives rise to entirely different interpretations of the word. Note that this ambiguity is orthogonal to the ambiguity introduced by syncretism. Each of these forms can potentially also be (and often are in Turkish) syncretic within their inflection paradigm.

(3.3) *çekti*
pull.3SG.PAST
'it pulled'

[4]In text-based natural language processing, homography, i.e., being written the same way, is usually enough to create problems even if the words are pronounced differently (cmp. *access* as a verb and *access* as a noun).

(3.4) *çek* *-ti*
cheque.NOM exist.3SG.PAST
'it was a cheque'

Turkish has a very productive morphology and often forms complex words that involve multiple steps of derivation interlaced with syntactic structure. In order to make the underlying syntactic relations visible, the Turkish treebank (Oflazer et al. 2003a) annotates dependency structure not over words but sub-units of words. Figure 3.5 shows an example from Eryiğit et al. (2008) that demonstrates the syntactic annotation in the Turkish treebank. Words are shown within solid frames. The dependency arcs in the example connect sub-units of words rather than the words themselves. Oflazer et al. (2003a) call these sub-units of a word *Inflectional Groups* (IGs), which are separated by *Derivational Boundaries*. The semantic root and derivational morphemes in a word are represented by different IGs, but an IG can contain additional inflection morphemes.

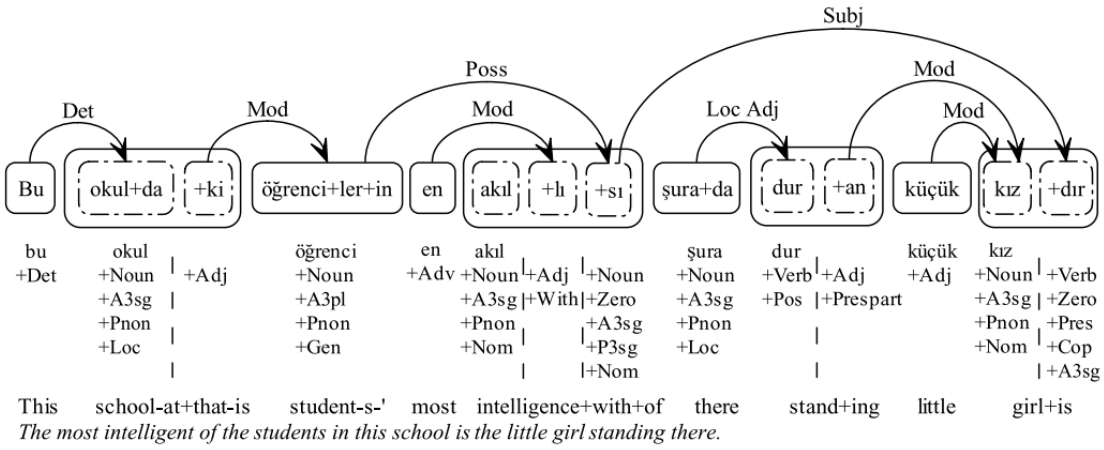


Figure 3.5: A Turkish sentence annotated in the style of the Turkish Dependency Treebank. Links are annotated on a sub-lexical level. The example is taken from Eryiğit et al. (2008). Note that this graphic follows a different convention than ours and draws dependency arcs pointing from the dependent to the head.

As an example, consider the second word in Figure 3.5, which contains two IGs. The first one, *okul+da*, is made of a stem *okul* and an inflectional suffix *da*, whereas the second IG, *ki*, is a derivational suffix that turns the word into an adjective. The first word *Bu*, a determiner, depends syntactically on the first IG of the second word, which is a noun root (the entire word is an adjective though). As another example, take the last word in the sentence: it is a verb that was formed from a noun meaning *girl*. The suffix *dır* is the

copula suffix that turns the noun into a verb. However, the word before the last means *little* and modifies the noun root of the verb rather than the verb itself.

Splitting words into IGs and annotating syntax between them would not be remarkable if the decision of how to split a word would be straight-forward all the time. But Turkish words can be highly ambiguous and context is often needed to resolve a segmentation ambiguity. Figure 3.6 shows the word *çekti* from above in a sentence. The word *açık* is an adjective and means *blank*. In the Turkish treebank, adjectives normally modify nouns but not verbs. The presence of *açık* in the sentence therefore makes it more likely to assume the interpretation in Example 3.4 than the one in Example 3.3. The syntactic context thus helps to arrive at the correct segmentation of a word. However, the correct syntactic structure can be found only if the segmentation of the words is correct.

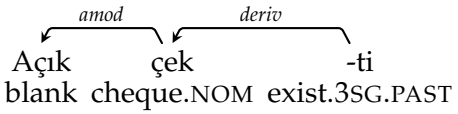


Figure 3.6: Syntactic context disambiguates the segmentation of *çekti*.

Since the basic units in the Turkish treebank are separated by derivational boundaries, ambiguous segmentation of a given word means that there is more than one derivational structure for this word. But ambiguity in word segmentation can also arise from different sources, e.g., from orthography. In Modern Hebrew, written words can be ambiguous with respect to their segmentation into meaningful units because there are eight common prepositions, articles, and conjunctions that are always attached to the following word (Goldberg and Elhadad 2013). This process is recursive so that several of such affixes can be attached in sequence. However, it is not always immediately obvious from a word form whether there is such affixation or not. Goldberg and Elhadad (2013) give the example in Example 3.5 to demonstrate the ambiguity. The displayed word can either mean *onion* or it can mean *in the shadow*, if the first character (read from right to left) is interpreted as a preposition affix.

(3.5) **בצל** *or* **ב-צל** (Goldberg and Elhadad 2013: 123)
onion or *in the shadow*

To illustrate the interaction between segmentation, morphology and syntax in Hebrew, Cohen and Smith (2007: 2) give an example of a Hebrew sentence that can be interpreted in

different ways depending on the segmentation of a specific word. The two interpretations of this sentence are repeated in Examples 3.6 and 3.7. The word in question is the sixth word of the sentence (counted from right to left). While the segmentation of the word is decided locally, the morphological and syntactic interpretation of the first and the last word depend on this decision.

(3.6)

יפה	שם	ש+רועה	ו+ה+מרוחק	ה+גדול	ה+ירוק	ב+ה+אחו	ה+רועה
is-beautiful	there	shepherds that	distant the and	big the	green the	meadow the in	shepherd the
ADJ+MASC		VB+MASC					MASC

'The shepherd in the big green distant meadow who shepherds there is beautiful.'

(Cohen and Smith 2007: 2)

(3.7)

יפה	שם	שרועה	ו+ה+מרוחק	ה+גדול	ה+ירוק	ב+ה+אחו	ה+רועה
nicely	there	is-lying	distant the and	big the	green the	meadow the in	shepherd the
ADV		VB+FEM					FEM

'The shepherdess in the big green distant meadow is lying there nicely.'

(Cohen and Smith 2007: 2)

The difference between Turkish and Hebrew with respect to segmentation ambiguity is that in Turkish, the segments of a word still all belong to the same syntactic unit, whereas in Hebrew the different segments of a word may belong to entirely different syntactic contexts. In Hebrew, the eight affixes mentioned above are always attached to the following word regardless of what this word is and whether they belong together syntactically. For example, the attached affix could be the subordinating conjunction of a subordinate clause, but may be written as a part of a word outside of the clause. In Turkish, an inflectional group morphologically and syntactically always belongs to the word it is part of, because segmentation in the Turkish treebank represents the derivational genesis of this word. However, in both cases there exists an interdependency between the morphological and syntactic interpretation of the words and its parts.

3.4 Modeling Choices

In the previous sections, we have motivated the necessity of morphological analysis as part of syntactic parsing and we have given examples of the interaction between morphology

and syntax, demonstrating that not only is information about the morphology of words important for discovering their syntactic structure, but syntactic context is often crucial in determining the morphological features of a word in the first place. We now take a look at the architecture that is commonly being used to model this relationship.

Traditionally, parsing systems use a pipeline model as shown in Figure 3.7. The parsing process is separated into at least three steps: *tokenization*, *morphological analysis*, and *syntactic analysis*. During morphological analysis, the tokens are annotated with their lexical and morphological information, i.e., they are assigned to a lemma and a part-of-speech, and their morphological features are determined. Morphological analysis is sometimes further divided by performing the three sub-tasks successively after each other instead of performing them jointly. During syntactic analysis, tokens are related to each other by grouping them into constituents or connecting pairs of words with dependency relations in order to expose the syntactic structure of the input sentence.

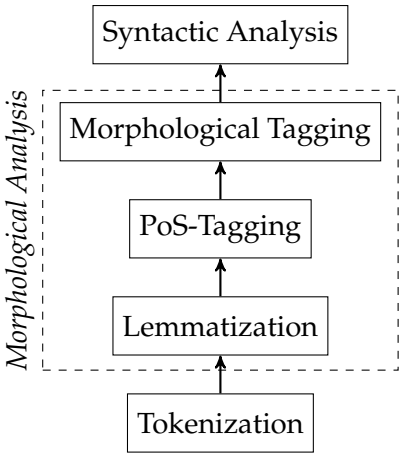


Figure 3.7: The pipeline model for syntactic parsing.

For languages like Hebrew or Turkish, i.e., languages where the segmentation of words needs to be predicted as well, tokenization and morphological analysis are often combined (Bar-Haim et al. 2005, Sak et al. 2008) into one system that predicts a consistent segmentation and morphological analysis of the input sentence. The parser is then run on the predicted segmentation.

Pipelines are designed for efficiency. By splitting the problem of sentence processing into several sub-tasks that can be performed efficiently on their own the entire system stays tractable and fast. The underlying assumption in a pipeline is that it is possible to

correctly perform the individual steps independent of the output of the later stages in the pipeline. For the parsing system in Figure 3.7, this independence assumption states, for example, that one can solve the problem of tokenization without access to the output of the morphological or syntactic analysis step. Analogously, it states that one can solve the problem of morphological analysis without access to the output of the parser.

3.4.1 Sequence Models for Morphological Analysis

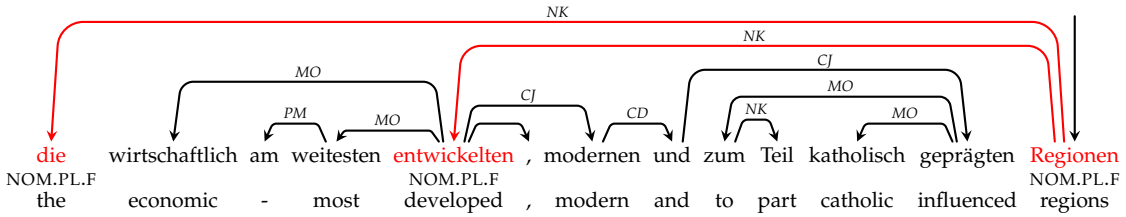
The standard models for performing morphological analysis are sequence models. The task is to predict a sequence of tags (think for example of a sequence of part-of-speech tags) given a sequence of tokens (from the tokenization step) such that the first tag in the output sequence corresponds to the first word in the input sequence, the second tag corresponds to the second word, and so on. In this model, the decision for the best tag is conditioned on the word itself, its surrounding words, and the previous tags in the output sequence. The optimal sequence given the input sentence is found via dynamic programming.

As we saw previously, morphological analysis is difficult because word forms can be ambiguous and syntactic context is necessary for disambiguation. This necessary syntactic context is modeled in a sequence model by conditioning the best tag on surrounding words and preceding tags. In other words, the syntactic structure of the sentence is approximated by looking at the words to the left and the right of the word in question. This approximation works well for a language like English because of its strict word order: Knowing that the last two words were a determiner and an adjective is a reliable indicator in English for predicting a noun next.

However, such an approximation becomes considerably less reliable if the word order is not as strict as in English because words can occur in more diverse contexts and they share these contexts with many other words. We have seen in the beginning of this chapter that morphologically rich languages tend to have a rather free word order since syntactic information is encoded overtly in the form of the words (recall Figure 3.2). If it was the case that the word forms encode the morphological information unambiguously, the model would still work because it would not need the context at all. But we also saw that phenomena like syncretism introduce ambiguity that cannot be resolved without

context. Syntactic context in these languages is therefore as important as in English to resolve morphological ambiguity, but the relevant context cannot reliably be found in the immediate vicinity of a word anymore.

To give a rather extreme example, consider the German noun phrase in Figure 3.8. The words marked in red are highly ambiguous with respect to their morphology, but they also stand in an agreement relation to each other that excludes most of the potential analyses. A sequence model would need a context of 13 words in order to relate the determiner in the first position to the noun in the last. Such long distance dependencies cannot be captured by sequence models. In the dependency tree, on the other hand, these words are direct neighbors of each other as indicated by the red arcs.



'the regions that are economically most developed, modern, and partly catholic'

Figure 3.8: A long German noun phrase. GENDER, NUMBER, and CASE agreement marked in red.

But the syntactic structure of a sentence as output by a parser is hidden from the morphological analysis in a pipeline model. Even more, it is intentional design of the pipeline that output of later processing stages should not be available. With decreased reliability of the available context and no access to the explicit syntactic structure, a morphological analyzer is therefore bound to make mistakes. This in turn feeds into another problem of pipelines which is called error propagation.

3.4.2 Error Propagation and Jackknifing

Error propagation describes the propagation and escalation of errors made by early steps in a pipeline through to the later systems. Error propagation occurs in pipelines because mistakes once made cannot be corrected by later processing steps and may lead to follow-up errors. For example, a mistake in predicting the case value of a word, e.g., predicting accusative case instead of nominative case, may cause the parser to predict an object

relation instead of a subject relation. Mistakes in the tokenization/segmentation are even worse since this step defines the tokens on which all subsequent systems operate. Incorrectly segmenting a word in a Hebrew sentence forces all subsequent processing steps to go along with this interpretation.

A common way to alleviate error propagation is to train the individual pipeline systems via n-fold jackknifing. The idea behind training via jackknifing is the following: assume that we want to train a dependency parser that uses part-of-speech tags as part of the input information. By training the parsing model with gold-standard part-of-speech tags, which we can get from manually annotated treebanks, the model learns to predict dependency trees using the gold-standard part-of-speech tags. When we run the parser on unseen data, there will be no manually annotated part-of-speech tags available, so we are going to use a part-of-speech tagger to automatically predict the part-of-speech tags. But this tagger is going to make mistakes. These mistakes will propagate to the parser and likely cause parsing mistakes because the parser learned to fully trust the part-of-speech tags. Now, if we had trained the parsing model on part-of-speech tags that have the same quality as the ones predicted for the unseen data, then the parser could learn when to trust them and when not to, thus potentially making fewer follow-up mistakes caused by incorrect part-of-speech tags.

However, simply running the part-of-speech tagger over the training data to get automatically predicted part-of-speech tags does not work because the tagger is usually trained on the same data and will thus be much better on this data than on data it has not seen before. Jackknifing is a technique that circumvents this problem and makes it possible to annotate the training data with automatically predicted part-of-speech tags of the same quality as one would find on unseen data (of the same domain). Technically, jackknifing works like cross-validation: the training data is partitioned into n parts, say $n = 10$, and each part is annotated by a part-of-speech tagger that was trained on the remaining $n - 1$ parts (in the case of $n = 10$, it would be trained on the remaining 9). Afterwards, the n parts that were annotated are merged together to form the original training set annotated with automatically predicted part-of-speech tags.

Jackknifing alleviates error propagation because it allows the models at the different steps in the pipeline to learn how reliable information coming from the previous steps is going to be. Pipelines trained like this perform better compared to pipelines where each model is trained on gold-standard data. But jackknifing cannot be applied in all

situations. When dealing with a segmentation problem as described above for Turkish and Hebrew, the segmentation step is not available to jackknifing. Parsers cannot be trained on automatically predicted tokens/segmentations because training a parsing model on a treebank requires gold-standard trees to train on, and gold-standard trees necessarily presuppose correct segmentation of the words in the tree. However, when running the parser on unseen data, both segmentation and syntax are predicted automatically, and errors in the segmentation will then always also lead to errors in the parser output.

3.4.3 Joint Modeling of Morphology and Syntax

As we have seen, the division of tokenization, morphological and syntactic analysis into separate steps creates a number of problems when applied to languages with a rich morphological system. Error propagation in the pipeline is aggravated by the lower performance of the early steps like tokenization and morphological analysis, which in turn is caused by a lack of syntactic information. A natural consequence of this observation is then to abandon this separation and to develop models that allow an exchange of information between the different levels. Joint models can resolve the mutual dependency between morphology and syntax elegantly by allowing both to influence the outcome of the other.

The hypothesis that we will be exploring and arguing for in the following chapters is that models that jointly predict morphology and syntax are better than models that separate them. To qualify this, we argue that the availability of the syntactic structure and the possibility to directly model interaction between morphology and syntax, e.g., agreement, allows a joint model to avoid errors caused by syncretism, unreliable sequential context due to free word order, and segmentation ambiguity. Since these problems occur frequently in languages with rich morphology, this hypothesis entails that joint models are better suited for parsing such languages.

Joint models for parsing have been explored before both for constituency parsing and dependency parsing (see Chapter 7 for a detailed account). These parsers predict the morphological features of words (and e.g., the segmentation into tokens) while simultaneously predicting the syntactic structure of the sentence. The typical challenge is to provide the additional information in an efficient way since simply combining the two tasks directly

quickly results in an intractable model.

In the remainder of this dissertation, we approach the topic of joint models for morphologically rich languages in four steps. We start by having a closer look at parsing with a pipeline model. In Chapter 4, we experiment with a state-of-the-art dependency parser in a pipeline setup analyzing the parser's output for three morphologically rich languages, Czech, German, and Hungarian. The analysis demonstrates that syncretism in the morphological system of a language directly causes parsing errors in a pipeline setup. In Chapter 5, we continue by showing that direct access to the syntactic structure of a sentence can improve the prediction of morphological features. Furthermore, we find that the syntactic information from the parser complements the information that is provided by language-specific lexicons. In Chapter 6, we then design a joint parser that models interaction between morphology and syntax explicitly by imposing constraints on the syntactic structure. The constraints implement morphosyntactic rules and act as a filter on the search space of the parser. The constrained model outperforms its unconstrained baseline as well as a state-of-the-art pipeline parser. In Chapter 7, we address the segmentation problem by designing an efficient graph-based dependency parser for morphological lattices that performs segmentation, morphological analysis, and parsing jointly. We test the parser on Turkish and Hebrew and show that it outperforms three state-of-the-art pipeline systems.

Chapter 4

Error Propagation between Morphology and Syntax

The standard architecture for parsing systems are pipelines, where parsing is broken down into tokenization, morphological analysis, and the actual parsing. The individual steps are applied one after the other with each step feeding its output as information to the following. Pipelines are efficient but may suffer from error propagation, because introduced errors cannot be corrected later on, though the problem might be alleviated by using jackknifing.

In Chapter 3, we motivate joint models by claiming that error propagation in pipelines becomes a more serious problem when parsing morphologically rich languages. This is because the morphological analysis makes more mistakes due to syncretism and insufficient information and these mistakes then cause follow-up mistakes in the parsing step. In this chapter, we support this claim empirically by showing that certain mistakes of the morphological analysis correlate with parsing errors and that these mistakes originate in the syncretism in the morphological system of the language that is being parsed.[1]

For the analysis, we run a state-of-the-art pipeline system on data from three different languages: Czech, German, and Hungarian. All three languages belong to the broad category

[1]The content of this chapter is published in Seeker and Kuhn (2013a).

of morphologically rich languages. Czech and German are both Indo-European languages, Czech from the Slavonic branch and German from the Germanic branch. Hungarian on the other hand is a Finno-Ugric language of the Ugric branch. Syntactically, they all use a case system to mark the function of verbal arguments. However, the morphological realization of case systems in the three languages shows important differences. Both Czech and German are *fusional* languages, where multiple morphological categories are fused into one inflection suffix. For example, nominal inflection suffixes in Czech and German signal gender, number, and case values simultaneously. Hungarian, on the other hand, is an *agglutinative* language. Every morphological feature is signaled by its own morpheme and the different morphemes are chained at the end of the word.

For us, however, the important difference between the morphological systems of these languages is that Czech and German show wide-spread syncretism in their inflection paradigms whereas Hungarian inflection suffixes are to the most extent unambiguous. With German and Czech on one side and Hungarian on the other, we can observe the effect of syncretism on the parsing system on Czech and German and use Hungarian as a control experiment. Being the prime example of a morphosyntactic feature, the analysis is focused on case and the grammatical functions that are marked by it.

The chapter starts in Section 4.1 by describing the experimental setup. In Sections 4.2 and 4.3 we then analyze the quality of the morphological and syntactic annotation and demonstrate the error propagation in the parser. Section 4.4 concludes with a discussion of the results of the analysis.

4.1 Experimental Setup

The experimental setup is straightforward. We train three parsing models for each language varying the quality of the morphological information: the first model uses gold-standard morphology, the second one uses automatically predicted morphology, and the third one provides no morphological information at all. In the analysis, we compare these models with each other and across languages. Other information, i.e., lemmas and part-of-speech tags, is predicted automatically and stays fixed for each language.

Comparing the performance of the three models to each other shows the effect that the morphological information has on parsing performance. The model using gold morphology serves as an upper bound where we can observe the behavior of the parser when it is not disturbed by errors coming from the automatic morphological analyzers. Note that this model is unrealistic since syncretisms are fully resolved. The model using predicted morphology serves as a realistic scenario where one can observe the problems introduced by mistakes in the morphological prediction. And finally, the model using no morphology shows how much non-morphological information contributes to the parsing performance. In comparison with the other two models, it shows the contribution of morphological information[2] to the parsing process.

The analysis is performed on the training sets for Czech and German and the full data set for Hungarian. For this, the data sets are parsed via 5-fold jackknifing. We use the training sets rather than the development sets in order to have access to more data for analysis. The rest of this section gives the technical details of the data sets, preprocessing, and the parser. The analysis of the models is presented in the next two sections.

4.1.1 Data

For Czech, we use the CoNLL 2009 Shared Task data set (Hajič et al. 2009), which consists of sentences from the Prague Dependency Treebank (Böhmová et al. 2000, Hajič et al. 2006). The German data set is a subset of the TiGer treebank (Brants et al. 2002) with the dependency conversion described in Seeker and Kuhn (2012). The subset recreates the set of sentences used in the CoNLL 2009 Shared Task for German.[3] The Hungarian data set consists of the general newspaper subcorpus of the dependency version of the Szeged Treebank (Csendes et al. 2004, Vincze et al. 2010).

Table 4.1 shows the size of each of the three data sets. For the experiments in this chapter,

[2]By morphological information, we always mean the complete annotation available in the treebanks. Although we concentrate in the analysis on gender, number, and case, the models using morphological information always use the whole set, including e. g. verbal morphology.

[3]Except for three sentences that for some reason were missing in the 2006 version of the TiGer treebank, from which this corpus was derived. The original data set in the CoNLL 2009 Shared Task was derived from the 2005 version, which still contains these three sentences. The 2005 version also contained spelling errors in the raw data that had been removed in the 2006 version. These errors were manually reintroduced in order to recreate the data set as exact as possible.

	#sentences
Czech	38,727
German	36,017
Hungarian	10,188

Table 4.1: Data set sizes for the three treebanks.

they correspond to the training set sizes in the CoNLL 2009 Shared Task for Czech and German. For Hungarian, we use all sentences of the general newspaper section.

For the Czech and the Hungarian data, we keep the predicted information for lemmata, part-of-speech tags, and morphology that is provided with the data. For both languages, this information has been predicted in a two-step process where a morphological dictionary produces a set of possible annotations for a given word form, which is then disambiguated by a statistical model trained on gold-standard data (for Czech, see Spoustová et al. 2009; for Hungarian, see Zsibrita et al. 2010). The German data was annotated with mate tools[4] via 10-fold jackknifing. Contrary to Czech and Hungarian, lemma, part-of-speech tag, and morphological information are annotated in three steps, each building upon the preceding one.

We make three changes to the annotation in the Czech and the Hungarian treebanks in order to allow for a more fine-grained analysis. First, we copy the SubPOS feature value[5] over to the part-of-speech column. This leads to a much more fine-grained part-of-speech tag set and thus supports a more detailed evaluation based on part-of-speech tags. The German part-of-speech tag set (STTS, Schiller et al. 1999) is already fine-grained enough for our purposes. For the same reason, we also change the object labels (*Obj*) in the Czech data set by combining it with the case value in the gold standard morphology, creating *Obj1-7*. This introduces a more fine-grained object distinction for the analysis and it also separates the case-marked objects from the clausal objects, which do not have a case feature and therefore keep the original *Obj* label.[6] Finally, we remove the empty nodes from the Hungarian data set by attaching all dependents of an empty node to its head

[4] http://code.google.com/p/mate-tools

[5] The SubPOS feature distinguishes subcategories inside the main part-of-speech categories and is part of the morphological description.

[6] Prepositional objects headed by prepositions (pos: RR, RF, RV) are also excluded.

node while changing their label to *ExD* as in the Czech data set. In the German data set, we use the version without explicit empty nodes (cf. Seeker and Kuhn 2012). Table 4.2 shows the encoding of grammatical functions in the versions of the three treebanks that we use in the experiments.

	CZECH	GERMAN	HUNGARIAN
subject	Sb	SB	SUBJ
nominal predicate	Pnom	PD	PRED
object	Obj1-7	OA, OA2, DA, OG	OBJ, DAT

Table 4.2: Core argument functions and their encoding in the different treebanks. The different object labels for Czech were introduced by us. The original function is Obj.

4.1.2 The Parser

For the parsing step in the analysis, we use mate parser[7] (Bohnet 2009, 2010), a state-of-the-art second-order graph-based dependency parser that uses Carreras' decoder (Carreras 2007) and outputs non-projective structures by using the non-projective approximation algorithm described in McDonald and Pereira (2006). The underlying statistical model is trained via passive-aggressive online training (Crammer et al. 2003, 2006). In all experiments, the model is trained for 10 iterations (default setting).

Since we are interested in the way the parser handles morphological information, we briefly discuss the inclusion of morphological features (see also Bohnet 2009: 3). The parser extracts features about morphological information by combining the part-of-speech tags of the head and the dependent of an arc with the cross-product of their morphological feature values. For this, the morphological information is split and every single morphological feature value is treated as one morphological feature in the statistical model. The cross-product then pairs the single feature values of dependent and head, creating all combinations thereof. One single feature computed for the edge between an adjective and a noun in Czech may then look like (A,N,acc,acc), which states the information that both words have the accusative case. However, other features are created as well that might look like (A,N,sg,masc), which states that the adjective has singular number and the noun has masculine gender. The algorithm therefore does not pay attention to the main syntactic category of a word. Furthermore, the cross-product is computed for every edge

[7] http://code.google.com/p/mate-tools

in the tree. All features are additionally combined with the label on the arc such that a morphological feature like case is directly combined with the label with which it appears together in the treebank. Because of this, the parser has direct access to the information about which case value signals a particular grammatical function. The statistical model should therefore be able to learn that certain dependent-head configurations often occur with certain morphological feature combinations. For example, a subject edge between a noun and a verb should very often occur together with morphological features involving nominative case while a dative object edge should often occur with a dative feature.

4.1.3 Evaluation

In this and the following chapters we evaluate the quality of morphological and syntactic annotation by measuring either accuracy or precision and recall. These metrics are used throughout the dissertation except in Chapter 7 where evaluation cannot be done with standard metrics.

Accuracy. We evaluate the quality of automatically predicted morphology by measuring the accuracy of the prediction, which is the percentage of tokens in a sentence that received the correct annotation. In the same way, we evaluate the quality of the syntactic annotation, i.e., the predicted dependency tree, by computing the percentage of tokens that were assigned the correct head (and dependency label). In order to know whether a prediction is correct, we compare them to a *gold standard*. In our case, the gold standard is a treebank that was manually annotated for morphology and syntax.

For dependency parsing, the accuracy is called the *attachment score*. There are two versions of the attachment score, one that takes the dependency labels into account (labeled attachment score or short LAS) and one that ignores it (unlabeled attachment score or short UAS). When a parser is evaluated on more than one sentence, the micro average over all sentences is computed. Let G_l and G_u be the set of labeled and unlabeled arcs in the gold-standard tree, respectively, and let P_l and P_u be the corresponding arc sets of a tree predicted by a parser for the same sentence. Labeled and unlabeled attachment score are defined as the percentage of overlap between the respective sets of the gold standard

and the prediction.

$$\text{LAS} = \frac{|G_l \cap P_l|}{|G_l|} \qquad \text{UAS} = \frac{|G_u \cap P_u|}{|G_u|} \tag{4.1}$$

Precision and Recall. We measure precision and recall when evaluating a system with respect to specific morphological tags, e.g., NOM.SG.MASC (nominative singular masculine) or specific dependency arcs, e.g., arcs labeled with *nsubj* (nominal subject). Precision gives the percentage of correctly labeled instances of the particular tag/arc in the prediction, while recall gives the percentage of correctly labeled instances of the particular tag/arc in the gold standard. For the example of nominal subjects, let G_{nsubj} be the set of arcs in the gold-standard tree that are labeled with *nsubj* and let P_{nsubj} bet its counterpart of a tree predicted by a parser for the same sentence.

$$\text{precision} = \frac{|G_{nsubj} \cap P_{nsubj}|}{|P_{nsubj}|} \qquad \text{recall} = \frac{|G_{nsubj} \cap P_{nsubj}|}{|G_{nsubj}|} \tag{4.2}$$

4.2 Quality of the Morphological Information

We start the analysis by looking more closely at the quality of the predicted morphology. We will see that already here there is a significant difference between Czech and German on one side and Hungarian on the other.

Table 4.3 shows the quality of the automatically predicted morphological information in the three data sets. On the left hand side, precision and recall are shown for case, number, and gender on all words, on the right hand side, only those words were evaluated where the predicted part-of-speech tag matched the gold standard one. Because of our focus on case, the figures in Table 4.3 consider case, number, and gender features only. We include number and gender because in German and Czech, these features are fused together with case into one inflection suffix. Hungarian as an agglutinating language does not fuse different morphological categories into one inflection suffix, but we show number for comparison. There is no gender category in Hungarian.

Czech and Hungarian achieve high scores on all three categories, with Czech achieving

over 95% for each feature, and Hungarian over 94% recall and almost 98% precision. In contrast, we find a rather mediocre quality in the German data set, where only the number feature can be predicted with comparable quality (there are only two values to predict though) while the prediction quality for gender and case is rather low. To a certain extent, the lower performance for German compared to Czech can be explained by the more informed annotation tool for Czech. The German data set was annotated by purely statistical tools while the Czech annotation tool uses a dictionary to support the statistical disambiguator.

		all		correct POS	
		prec	rec	prec	rec
CZECH	case	95.73	95.63	96.06	96.06
	gender	97.59	97.45	98.03	98.03
	number	98.18	98.08	98.47	98.47
GERMAN	case	88.69	88.51	89.26	89.06
	gender	90.16	89.99	90.95	90.74
	number	96.18	95.63	96.92	96.61
HUNGARIAN	case	97.83	94.11	99.22	99.22
	number	98.64	95.91	99.88	99.88

Table 4.3: Annotation quality of case, number, and gender, for all words and for those words with a correctly predicted part-of-speech tag.

Hungarian shows a big gap between precision and recall (97.83% and 94.11% for case) when evaluating all words, but the performance on the words with the correct part-of-speech tag is almost perfect (99.22% for case). The reason for this drop seems to be the part-of-speech tagger. The part-of-speech tag set in the Hungarian treebank uses a category *X* as a catch-all category where annotators would put tokens they could not assign anywhere else. The precision for this class is below 10%, because the tool is assigning a considerable amount of proper nouns to this class. The class *X* however does not get a morphological specification so that about 3,500 out of 12,500 proper nouns do not receive a case and a number value at all. The reason for the lower precision and recall in Hungarian is thus not due to the difficult morphology of Hungarian, it is a coverage problem of the part-of-speech tagger that is propagated through.

The big gap between all words and words with correct part-of-speech tags that we find in

Hungarian does not show in Czech and German.[8] The results are in fact only a bit better when the correct part-of-speech is known. While it seems that in Hungarian, knowing the part-of-speech almost guarantees to get case and number values correct, the same cannot be said about Czech and German. Predicting morphology in Czech and German seems to be more difficult than in Hungarian. One explanation for this is that Czech and German have a lot of syncretism in their inflection paradigms, which makes the prediction of morphological features dependent on additional information like syntactic or semantic knowledge. Hungarian on the other hand is surprisingly unambiguous with respect to marking morphological information. For example, there is only one regular syncretism in the case system of Hungarian (genitive vs dative case) out of about 20 different case values.

4.3 Error Propagation

We now turn to the actual parsing output and show how the errors in the morphological annotation propagate through to the syntax. We do the analysis in three steps, with every step zooming in a bit closer. In every step, we compare the three models that were trained for each language, namely a model using gold standard morphology, one using predicted morphology, and one using no morphological information. In the tables, we call these models GOLD-M, PRED-M, and NO-M, respectively. In the first step, we evaluate the performance of the models on the entire data to see the general trend. In the second step, we focus on the grammatical functions that are marked by case values, i.e., subject, object, etc. In the last step, we zoom in once more and compare the three models with respect to the most frequent confusion errors they make.

[8] For German, precision and recall give different numbers when evaluated on words with correct part-of-speech tag. This is due to the independence of the morphological analyzer from the part-of-speech tagger. The morphological analyzer is not bound to a particular feature template determined by the part-of-speech of the word, such that, in principle, it can assign case to verbs and tense to nouns. This is not the case for the Czech and Hungarian analyzers, where precision and recall collapse into simple accuracy.

4.3.1 Overall Parsing Quality

The overall parsing performance of the different models is shown in Table 4.4. Attachment scores are given in percent. The German and the Hungarian scores exclude punctuation while the Czech ones include them because punctuation in the Czech treebank is sometimes used as the head in coordination.

	Czech		German		Hungarian	
	LAS	UAS	LAS	UAS	LAS	UAS
GOLD-M	82.49	88.61	91.26	93.20	86.70	89.70
PRED-M	81.41	88.13	89.61	92.18	84.33	88.02
NO-M	79.00	86.89	89.18	91.97	78.04	86.02

Table 4.4: Overall performance of mate parser for every language and different kind of morphological annotation. Results for German and Hungarian are without punctuation.

Comparing across languages, Table 4.4 gives the usual picture that has been observed in several shared tasks on dependency parsing for multiple languages (e.g., Buchholz and Marsi 2006, Hajič et al. 2009): The performance on German is pretty high although not as high as it would be for English while the performance on Czech is considerably lower.[9] For Hungarian, the performance is comparable to Czech in terms of UAS but the LAS for Hungarian is better.

The results furthermore show the expected ordering in performance for the models using different kinds of morphological information. The gold models always outperform the models using predicted morphology, which in turn outperform the models using no morphological information. It is noteworthy that while the performance on German does not degrade very much when using no morphological information, it is very harmful for Hungarian to do so (78.04% LAS for NO-M in comparison to 84.33% LAS for PRED-M). The Czech results lie in between. Obviously, Hungarian relies much more on morphological means to encode syntactic information than German, which can also be seen in the higher complexity of Hungarian morphology or in the entirely free word order of Hungarian compared to the partially free word order in German. In an imaginary continuum of languages, German is still much more similar to English than Hungarian.

[9] The extreme divergence between LAS and UAS for Czech is due to the way, the Czech treebank labels certain phenomena, which makes it difficult for the parser to decide on the correct label. See Boyd et al. (2008: 8–9) for examples.

4.3.2 Quality of Grammatical Functions

While the scores in Table 4.4 reflect the overall quality of the parser, we do not expect case morphology to influence all of the decisions that the parser has to make. We therefore go into more detail and concentrate on nominal elements (nouns, pronouns, adjectives, etc.)[10] and core grammatical functions (subjects, objects, nominal predicates, etc., see also Table 4.2.) because in all three languages, nominal elements carry case morphology to mark their syntactic function. Core grammatical functions are vital to the interpretation of a sentence since they mark the participants of a situation. We exclude clausal and prepositional arguments, which can fill the argument slot of a verb but would not be marked by case morphology. Table 4.5 shows the performance of the three parsing models for each language on the core grammatical functions. As described in Section 4.1.1, we split the object function for Czech according to their associated case value.

NO-M VS. PRED-M. Comparing the NO-M models to their respective PRED-M counterparts, we observe what we already saw in Table 4.4, namely that morphological features are much more important in parsing for Czech and Hungarian than they are for German. The performance on all grammatical functions except the rather rare genitive object is generally higher for German, indicating that the parser is able to use information from lexicalization and configurational information to a large extent (see also Seeker and Kuhn 2011). Results for Czech and Hungarian are lower in the NO-M model but improve by large margins when switching to predicted morphology. Czech accusative objects improve from 72.71% f-score to 84.12% f-score in the PRED-M model. In Hungarian, the f-scores for dative objects improve by over 33 percentage points to 73.49% f-score when switching to the PRED-M model. Even though we saw this effect already in Table 4.4, it is much more pronounced when we concentrate on grammatical functions because they are marked by morphological means directly.

[10] We determine a nominal element by its gold standard part-of-speech tag:
Czech: AA, AG, AM, AU, C?, Ca, Cd, Ch, Cl, Cn, Cr, Cw, Cy, NN, P1, P4, P5, P6, P7, P8, P9, PD, PE, PH, PJ, PK, PL, PP, PQ, PS, PW, PZ
German: ADJA, ART, NE, NN, PDAT, PDS, PIAT, PIS, PPER, PPOSAT, PPOSS, PRELAT, PRELS, PRF, PWS, PWAT
Hungarian: Oe, Oi, Md, Py, Oh, Ps, On, Px, Pq, Mf, Pp, Pg, Mo, Pi, Pr, Pd, Mc, Np, Af, Nc.

		GOLD-M			PRED-M			NO-M		
	freq	prec	rec	f	prec	rec	f	prec	rec	f
subject	38,742	89.29	91.18	90.22	83.96	87.01	85.46	74.10	78.82	76.39
obj (acc)	21,137	92.50	93.35	92.93	85.25	83.01	84.12	73.42	72.02	72.71
predicate	6,478	89.07	87.14	88.09	88.24	86.00	87.11	82.34	78.19	80.21
obj (dat)	3,896	83.18	85.68	84.41	80.21	78.88	79.54	74.29	48.05	58.35
obj (instr)	1,579	71.38	66.50	68.85	67.74	62.51	65.02	58.93	35.53	44.33
obj (gen)	1,053	86.69	77.30	81.73	80.42	62.39	70.26	74.60	48.81	59.01
obj (nom)	167	57.63	40.72	47.72	56.97	29.34	38.74	48.67	32.93	39.29

(a) Czech.

		GOLD-M			PRED-M			NO-M		
	freq	prec	rec	f	prec	rec	f	prec	rec	f
subject	45,670	95.11	96.05	95.58	89.95	91.23	90.59	88.32	89.86	89.08
obj (acc)	23,830	93.93	94.80	94.36	84.83	84.89	84.86	82.20	83.35	82.77
obj (dat)	3,864	89.56	87.73	88.64	79.17	64.44	71.05	77.09	50.78	61.23
predicate	2,732	78.07	73.35	75.64	75.80	72.91	74.33	76.20	71.01	73.51
obj (gen)	155	80.25	41.93	55.08	60.66	23.87	34.26	52.94	17.42	26.21

(b) German.

		GOLD-M			PRED-M			NO-M		
	freq	prec	rec	f	prec	rec	f	prec	rec	f
subject	11,816	88.34	91.57	89.93	84.96	88.15	86.53	64.58	66.44	65.50
obj (acc)	9,326	93.63	94.22	93.92	92.36	92.70	92.53	66.23	63.86	65.03
obj (dat)	1,254	80.55	76.95	78.71	75.57	71.53	73.49	58.36	30.62	40.17
predicate	941	81.05	75.45	78.15	77.39	72.37	74.79	72.49	71.41	71.95

(c) Hungarian.

Table 4.5: Precision, recall, and f-score for core grammatical functions marked by case, sorted by frequency. Locative objects in Czech and second accusative objects in German are omitted due to their low frequency.

PRED-M VS. GOLD-M. Turning to the GOLD-M models, we see that in general, German and Czech benefit more from the gold standard morphological annotation than Hungarian. There is however still a gain of information in Hungarian as the error propagation due to wrong part-of-speech tags in Hungarian is eliminated in the GOLD-M model. One effect

that shows very clearly is the improvement for subjects and accusative objects in Czech and German when moving from predicted to gold morphology. A typical syncretism in Indo-European languages exists between nominative and accusative in the neuter gender (Blake 2001), which is correctly disambiguated in the gold-standard morphology: comparing the performance on subjects (marked by nominative case) and accusative objects, we see a considerable improvement between 5 percentage points for Czech subjects and almost 10 percentage points for German accusative objects when switching to gold morphology. Hungarian, which does not have this syncretism, does not show improvements on this scale.

Finally, the statistical nature of the parser can be seen in the accuracies with respect to the frequency of the functions. For all languages, predictions are less accurate for the less frequent functions. The general order for all three languages from most frequent to least frequent is *subjects* > *accusative objects* > *predicates/dative objects* > *instrumental/genitive objects*. In case of doubt, the parser resorts to the more frequent function. A clear sign for this is that for infrequent functions, the precision is always higher than the recall. As an example, note the performance of the parsing models on dative and genitive objects. The parser annotates genitive objects if it has strong evidence, hence the high precision, but it frequently fails to find it in the first place, hence the low recall.

4.3.3 Analysis of Confusion Errors

In the previous section, we have seen that better morphological information leads to improved parsing accuracy for case-marked functions. Clearly, the parser utilizes the morphological information to better model the syntactic structures annotated in its training data. We now go one step deeper and look at confusion errors that the parser commits during parsing. Our hypothesis states that the mistakes that the parser makes are (among other reasons) due to errors in the morphological information which are caused by syncretic word forms that the morphological analyzer cannot resolve properly. If this is true, then one should be able to predict the errors based on the ambiguity in the morphosyntactic systems of the language. For example, with automatically predicted case values, we would expect the parser to confuse subjects and accusative objects when parsing Czech and German, since there exist regular syncretisms between these functions. When using gold-standard information, we would expect the parser only to confuse functions that

are marked by the same case value, e.g., subjects and nominal predicates. Conversely, we would expect the models without morphological information to commit confusion errors all over the place since they do not have the necessary information.

Subjects. To start with the last expectation, we examine the confusion errors with subjects made when using no morphological information (NO-M). Subjects are marked by nominative case in all three languages, with Czech allowing for dative and genitive subjects under special circumstances. The NO-M models do not have access to morphological information and should therefore mix up functions regardless of the case value that would usually distinguish them. Table 4.6 shows the top five confusion errors made by the NO-M models on the subject function. The values are split for correct and incorrect head selection to tease apart simple label classification errors from errors involving label classification and attachment.

	correct head		wrong head	
rank	label	freq	label	freq
1	Obj4	4996	Atr	2644
2	Pnom	1261	Obj4	981
3	Adv	811	Sb_M	948
4	Obj3	752	Adv	273
5	Obj7	380	Obj_M	245

(a) Czech.

	correct head		wrong head	
rank	label	freq	label	freq
1	OA	2680	OA	1498
2	PD	776	NK	906
3	DA	458	DA	431
4	EP	301	AG	313
5	MO	219	CJ	296

(b) German.

	correct head		wrong head	
rank	label	freq	label	freq
1	OBL	3029	ATT	1116
2	OBJ	1505	Exd	574
3	PRED	250	COORD	313
4	ATT	185	OBL	311
5	DAT	152	OBJ	139

(c) Hungarian.

Table 4.6: Top 5 functions with which subjects were confused when parsing with NO-M models. _M marks a coordinated function in Czech.

As expected, the results in Table 4.6 show confusion errors with functions that are marked with different case values: For Czech, when the head was chosen correctly, *Obj4*, *Obj3*,

and *Obj7* (accusative, dative, and instrumental objects respectively) are all marked by a different case value and their confusion rates follow their frequency in the data. *Pnom* (nominal predicates) are marked by nominative case like subjects. If the head was chosen incorrectly, the parser assigns *Obj4* and coordinated subjects and objects (*Sb_M, Obj_M*). Adverbial (*Adv*) and attributive functions (*Atr*) are expected as they mark adjunct functions that can be filled by nominal elements. For German, we see confusions with the object functions (accusative *OA* and dative objects *DA*), predicates (*PD*) and the *EP* function marking expletive pronouns in subject position. Furthermore, the parser confuses subjects with *MO, NK*, and *AG*, which are the three adjunct functions that can be filled by nominal elements (e. g. *AG* marks genitive adjuncts). *CJ* finally marks coordinated elements, which is an expected error if the head was chosen incorrectly, but unlike in the Czech treebank, we cannot tell by the coordination label the particular function the element would have if it were not coordinated. In Hungarian, we also find errors across the board, with argument functions not marked by nominative case (accusative objects *OBJ*, dative objects *DAT*), the predicate function *PRED*, and all types of adjuncts (*ATT* (attributives) and *OBL* (obliques)). Obliques are especially interesting in Hungarian since the language has only a small number of prepositions. Most oblique adjunct functions are realized by a particular case (hence the about 20 different case values) which for a parsing model using no morphological information makes it rather difficult to distinguish them from the core argument functions. In summary, we find the expected picture of confusion errors across the case paradigm.

Turning now to the GOLD-M models, we can see whether the parser is able to learn the mapping between case and its associated functions. If so, we expect confusion errors with functions that are all compatible with the case value of the correct function. Table 4.7 shows the top five confusion errors that the GOLD-M models make on the subject function. Here, we see a completely different picture compared to the NO-M model errors in Table 4.6. In all three languages, we find — regardless if the head is correct or not — confusions only with functions that are compatible with the nominative case. In Czech, subjects are mostly confused with predicates (*Pnom*) and coordinated subjects (*Sb_M*). *ExD* marks suspended nodes in an elliptical construction. The label does not tell whether the node would be a subject with regard to the empty node but it may be, so it is compatible with nominative case. *Atr* between nominal elements may mark close appositions, which would be marked as nominative by default. *ObjX* marks objects with no annotated case value (mostly for foreign words). Of all the functions, only *Obj4* cannot be signaled by nominative case. If one checks those 69 cases, only 22 are annotated with accusative case

	correct head		wrong head	
rank	label	freq	label	freq
1	Pnom	583	Sb_M	1142
2	ObjX	102	Atr	711
3	Adv	102	ExD_M	162
4	Obj4	69	ExD	145
5	ExD	45	Pnom	65

(a) Czech.

	correct head		wrong head	
rank	label	freq	label	freq
1	PD	773	NK	555
2	EP	323	CJ	245
3	MO	117	PNC	139
4	OA	112	PD	129
5	PH	96	APP	127

(b) German.

	correct head		wrong head	
rank	label	freq	label	freq
1	PRED	264	ATT	678
2	Exd	102	Exd	494
3	OBL	94	COORD	249
4	ATT	90	NE	32
5	OBJ	50	DET	22

(c) Hungarian.

Table 4.7: Top 5 functions with which subjects were confused when parsing with the GOLD-M models. _M marks a coordinated function in Czech.

in the gold standard, the rest consists mostly of various, high-frequent numerals in neuter gender and quantifiers, most of which are ambiguous between nominative and accusative. In these cases, the case feature seems to be overruled by other information. We get the same picture for German and Hungarian, both models making errors that are compatible with the nominative case value. Of the 112 errors with accusative objects (*OA*) in German, only 36 have the correct case value in the gold standard. We thus conclude that the parser learns that subjects are marked by nominative case.

Direct Objects (Accusative). We now switch to accusative objects and compare the performance of the GOLD-M models with their respective PRED-M counterparts. We make the switch to show results on another function and not just on subjects. The basic pattern is the same in both cases. Table 4.8 shows the confusion errors for accusative objects. On the left, the GOLD-M errors are shown, the PRED-M errors are displayed on the right. For

the GOLD-M models, the picture is basically the same as with subjects, with the small exception that all three languages show confusion with subjects under the top five.[11] Although the effect is not strong, it shows that the morphological features are sometimes simply overruled by other, non-morphological features.

The interesting difference however can be seen when comparing to predicted morphological information. The overall number of errors increases, but the largest increase occurs for subjects in German (*SB*) and in Czech (*Sb*), while the same is not observable in Hungarian (*SUBJ*). Of the 2,945 confusion errors in Czech, where the PRED-M model incorrectly predicts an accusative object, 891 have been marked as accusative by the morphological prediction despite being nominative in the gold standard while 1,505 have been classified as nominative although being accusative. Checking the gender value of these instances, we find the overwhelming majority to be neuter, feminine, or masculine inanimate, exactly those genders whose inflection paradigms show syncretism between nominative and accusative forms. The same effect can be found in the German errors. The syncretism in the two languages causes the automatic morphological analyzers to confuse these case values more often, which subsequently leads to errors in the parser due to error propagation in the pipeline architecture. If the morphological analyzer cannot reliably predict the case values, the parser has a hard time making decisions that are based on this information.

That the parser so frequently falls for incorrect annotation is another proof that it has learned the mapping between case and its associated grammatical functions. Also as expected, we do not find this effect for Hungarian. Since there is almost no syncretism in the Hungarian case paradigm, it should subsequently not lead to this kind of error propagation.

4.4 Discussion

Since it would not contribute something new to the picture, we do not go into detail for the remaining grammatical functions. It is clear from the analysis that the parsing model

[11] The *AuxT* label in the Czech errors is used to mark certain kinds of reflexive pronouns, which can be in accusative or dative case. The criterion for deciding whether a reflexive pronoun is labeled *AuxT* or *Obj4*, i. e. accusative object, is whether the governing verb denotes a conscious or unconscious action. This is a very tough criterion to learn for a dependency parser. In any case however, *AuxT* is perfectly compatible with accusative case.

	GOLD-M				PRED-M			
	correct head		wrong head		correct head		wrong head	
rank	label	freq	label	freq	label	freq	label	freq
1	Adv	274	Obj_M	750	Sb	2354	Atr	687
2	AuxT	270	Atr	172	Adv	262	Obj_M	660
3	Sb	69	ExD_M	67	AuxT	256	Sb	594
4	ExD	34	Adv	65	Obj3	137	Sb_M	108
5	AuxR	28	Atv	53	Obj2	109	ExD_M	94

(a) Czech.

	GOLD-M				PRED-M			
	correct head		wrong head		correct head		wrong head	
rank	label	freq	label	freq	label	freq	label	freq
1	MO	283	NK	357	SB	2176	SB	1329
2	SB	112	CJ	191	DA	610	NK	606
3	DA	55	SB	121	MO	308	CJ	365
4	CJ	43	APP	97	CJ	46	AG	137
5	EP	25	MO	55	EP	40	APP	136

(b) German.

	GOLD-M				PRED-M			
	correct head		wrong head		correct head		wrong head	
rank	label	freq	label	freq	label	freq	label	freq
1	OBL	90	COORD	119	OBL	119	COORD	140
2	ATT	60	Exd	81	SUBJ	86	Exd	111
3	SUBJ	50	ATT	44	ATT	65	ATT	78
4	Exd	23	OBL	18	Exd	19	ROOT	18
5	MODE	14	ROOT	13	MODE	16	OBL	15

(c) Hungarian.

Table 4.8: Top 5 functions with which accusative objects were confused when parsing with the gold (left) and predicted (right) morphology models. _M marks a coordinated function in Czech.

learns the association of case with the respective grammatical function and it makes errors when the case information is wrong. These errors are in line with the syncretism in the

morphological systems of German and Czech and the fact that we do not find the same effect for Hungarian shows that syncretism is likely to be the source.

The syncretism in Czech and German makes the task of learning the morphosyntactic rules of these language much more difficult for a statistical parser in a pipeline architecture. As we described in Chapter 3, the morphological analyzer makes mistakes because it is lacking the relevant context and these mistakes are propagated to the parser because it relies on this information. The fact that these errors occur despite the parser being trained via jackknifing shows that jackknifing does not solve the problem even though there would be many more errors if the parser had been trained with gold-standard morphology. Essentially, jackknifing is a general method to improve machine learning and it helps to some extent. But the problem here is not that the statistical model cannot learn the correct rules, which is demonstrated by the experiments with the gold-standard morphology. It is rather that the parsing architecture itself does not properly model the interdependency of morphology and syntax. This can only be changed by changing the model itself.

With a high amount of syncretism, it becomes questionable whether it is right to fully disambiguate certain morphological properties of a word (e. g. case) without taking the syntactic context into account. That syntactic information can indeed help predicting morphology for Czech and German is demonstrated in the next chapter. However, the analysis in this chapter also suggests that for Hungarian, a pipeline architecture may be adequate with respect to morphological prediction. Joint models are only needed if the morphological analysis cannot reliably predict the necessary information, and this is not the case for Hungarian albeit for a different reason than for English. But since a joint model would subsume a pipeline, we could just go with joint models for all languages.

There are two additional observations that we think are worth mentioning: First, the experiments show that a parsing model for German can rely to a large extent on lexical and configurational information and while morphological information is useful, it is not as vital as in Czech and Hungarian. This illustrates the role of German as a borderline case of a morphologically rich language, which resides in the middle ground between morphologically poor languages like English and morphologically rich languages like Czech and Hungarian.

Second, we can see from the analysis that gold-standard morphological information improves the parsing results considerably in all three languages with huge improvements

for Czech and German with respect to grammatical functions. While it is clear that this is an oracle experiment, one should be aware that this is a rather unrealistic oracle. Given the linguistic interdependency between morphology and syntax in these languages, it does not make much sense to assume a situation where the morphology is correctly predicted without access to the syntactic context of the word. As a comparison, think of an oracle used in parser reranking that is used to assess the quality of n-best lists. Given a set of n-best lists, the oracle computes what the maximum score would be if the system always picked the best scoring item in the list. If the lists always contain the correct solution, the maximum score is 100%, but since they usually do not it is lower. There is in principle no (obvious) reason why the n-best lists could not always contain the correct solution, just sometimes not in the top position. But for languages like Czech and German, syncretism constitutes a principled reason why we cannot expect a system to predict the correct morphological description of a word without access to its syntactic context. An oracle that uses gold-standard morphology is therefore more unrealistic than the one for ranking.

A typical result in parsing experiments with morphological features, particularly with case, is that case is the best feature when gold-standard morphology is used, but it is much less useful (and sometimes even harmful) when automatically predicted (see for example the results on Arabic in Marton et al. 2013). This is a consequence of the difficulty of predicting morphology and it shows that experiments with gold-standard morphology should always be accompanied by experiments using automatically predicted morphology since otherwise the interaction between morphology and syntax is systematically excluded.

Chapter 5

Morphology Prediction with Structural Context

In the last chapter, we saw that syncretism (and morphological ambiguity in general) cause the morphological analysis step in a pipeline to make mistakes that are then passed on to the parser. Morphological ambiguity very generally describes the situation when a given word form has several potential analyses, which automatic morphological prediction must disambiguate. Since the word form itself does not provide any clues, the system must rely on the syntactic context in which the word form occurs in order to find the correct analysis. As mentioned in Chapter 3, the standard models for such systems are sequence models (e.g. Hajič 2000, 2004, Smith et al. 2005, Chrupała et al. 2008), which model the syntactic context of a word as the immediate surrounding words to the left and the right as well as the predicted analyses for a fixed number of preceding words.

The claim that we put forward in Chapter 3 states that this model of context is insufficient for languages with rich morphology since they also normally allow for free word order. With free word order, a fixed context window around the word form that is to be disambiguated becomes unreliable because free word order means that the word can occur in any context in the sentence. Sequence models are by design not able to capture long distance dependencies and in a free word order language, long distance dependencies, think for example grammatical agreement, are much more frequent.

Consequently, our hypothesis is that a context model that also has access to the syntactic structure of the sentence (e.g., in form of a dependency tree) performs better for a morphologically rich language than a model that models syntactic context as a window around the word form only. This hypothesis is tested empirically in this chapter.[1] Answering it positively will provide additional support for jointly modeling morphology and syntax.

To be clear, we do not argue that sequence models are entirely inappropriate for predicting morphology in morphologically rich languages. They still capture the majority of phenomena because they can capture frequency effects and oftentimes the context window is big enough to capture the short "long distance" dependencies. In Chapter 3, we give an example of a German noun phrase (Figure 3.8) where the context window needs to be at least 13 words in order to capture the agreement relation between determiner and head noun. However, most noun phrases in the German TiGer treebank are short enough to be captured with a standard trigram model. But while the surrounding context can model many cases (and we make use of this in the experiment as well), the hypothesis states that explicit access to the syntactic structure models the underlying relationship more faithfully and therefore leads to even better models.

In the following experiments, we compare a baseline model that uses a context window with itself when it has additional context information from a dependency parser. We compare these models on the same languages that we used in the previous chapter, namely Czech, German, and Hungarian. To these three we add Spanish as a fourth language. We will see in the experiments that explicit syntactic information indeed leads to better models for Czech and German while it causes only tiny improvements on Hungarian and Spanish. In the analysis we show that the explanation for this outcome aligns with the conclusions that we drew in the previous chapter, namely that Czech and German profit because their morphological system relies on mechanisms like agreement for disambiguation while Hungarian does not.

In an additional experiment, we compare the influence of explicit syntactic information with the information that is provided by external lexicons. Lexicons are commonly used by state-of-the-art systems to increase the coverage (Hajič 2004, Müller et al. 2013). The comparison shows that syntactic information is complementary to lexical information and having access to both provides the best systems for Czech and German.

[1] The content of this chapter is published in Seeker and Kuhn (2013b).

In Section 5.1, we present the experimental design that we use in this chapter and describe the data sets. Section 5.2 presents the results of the experiments and in Section 5.3, we analyzes the outcome of the experiments from different angles. The chapter concludes with a discussion in Section 5.4.

5.1 Experimental Design

In order to test our hypothesis, we employ a two-step approach. The input data is preprocessed by assigning lemma and part-of-speech information, morphological features are annotated and all sentences are parsed. Then, the morphological features are predicted again (possibly overwriting the original annotation) this time with access to features extracted from the predicted parse trees. Figure 5.1 shows a graphical representation of the experimental setup.

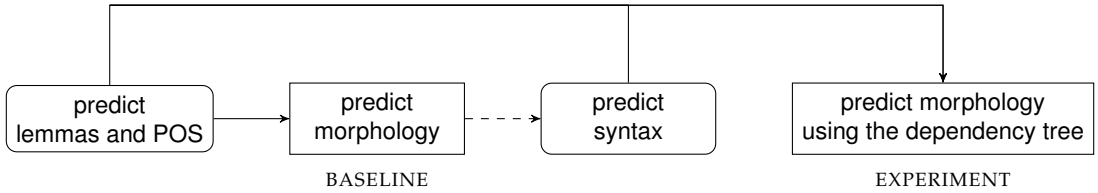


Figure 5.1: Experimental setup to predict morphology using features from syntax.

The setup is inspired in part by Versley et al. (2010) who manually annotate a German treebank with morphological information. To support the annotators, a morphological analyzer is first run to provide a set of possible candidates. This set is further reduced by applying hand-crafted rules leaving only few options to choose from. The crucial point is that these rules also make use of the available syntactic structure in the treebank in order to reduce the morphological ambiguity. It should be mentioned that the experimental setup in this chapter is used to test a particular hypothesis and is not meant to be a proposal for a system that can predict morphology, as it is rather inefficient. Proposals for models that predict syntax and morphology jointly are discussed in the next two chapters.

Furthermore, be aware that the process in Figure 5.1 is not a stacking setup, i.e., the second annotation of morphology does not use the output of the first one. It accesses information about lemmas, part-of-speech, and parse trees only. The reason for using the first round of

morphological annotation is that the quality of the parse trees is much better if the parser is provided with morphological features (see also Chapter 4). But these features are not used by the second morphology prediction.

In Section 5.1.2, we describe a morphological tagging system that we use for all the experiments. In its basic form it constitutes our baseline system because it uses a standard context window. We compare this baseline to a version of the tagger where we extend its feature set with features that are extracted from the dependency tree of the given sentence. This version constitutes the model that has access to the actual syntactic structure. The feature set of the baseline is a proper subset of the feature set of the model that uses the dependency tree. Because of that, the comparison between the two can be used to show the influence and contribution of the structural context. We pad this core comparison with two additional models. First, we compare the models to external baselines for which we run off-the-shelf morphological taggers. The purpose of the external baseline is to show that our basic model and the extended model both give state-of-the-art performance. This fact is important because otherwise we cannot exclude the possibility that the structural context adds information that may also be obtained with a standard context model and a better feature set. Finally, we add an oracle experiment by providing gold-standard dependency trees instead of automatically predicted ones. Comparing the oracle to the model using structural context allows us to see the influence of parsing errors on the quality of morphological prediction.

Syntactic information has been approximated before by incorporating long distance information about verbs into a sequence model. The idea behind this is to capture subject verb agreement in languages where the subject and the verb are unlikely to be captured by a fixed context window. Prins (2004) splits the states of an HMM model depending on information about verbs in the previous part of the sentence. Votrubec (2006) adds features to a discriminative model that record the presence of verbs in a large window (30 tokens) around the current token. In both cases, long distance syntactic information is approximated with the assumption that the nearest verb is likely to be a syntactic head. We use the features by Votrubec (2006) in our baseline system in order to arrive at a competitive baseline.

5.1.1 Data Sets

We run the experiment on treebanks of four different languages: Czech, German, Hungarian, and Spanish.[2] We use the CoNLL 2009 Shared Task data sets (Hajič et al. 2009) for Czech and Spanish with their respective splits. For German, we use the dependency conversion of the TiGer treebank (Brants et al. 2002) by Seeker and Kuhn (2012) without ellipses, splitting it into 40k/5k/5k for training/development/test. We use the Szeged Dependency Treebank (Vincze et al. 2010) for Hungarian with the split of Farkas et al. (2012). Preprocessing consists in assigning each token to a lemma and part-of-speech. For Czech and Hungarian, we keep the annotation provided with the respective treebank. For German and Spanish, we predict the information using the lemmatizer and part-of-speech tagger from mate tools.[3] Training sets are annotated via 10-fold jackknifing.

We add Spanish to the set because it differs from all these languages in interesting ways. Spanish is a Romance language and has a complex verbal morphology. The word order is not as restricted as in English but far from the possibilities in Hungarian. The interesting difference that sets Spanish aside from the other three is that it has only a residual system of case marking which is used for pronouns only (similar to English).[4]

5.1.2 System Description

We now describe the morphological tagging system that we use in the experiments. The system is a token-based tagger that assigns full morphological descriptions to each word in a sentence. It extracts features from word forms, lemmas, and part-of-speech for a token and surrounding context tokens. It furthermore incorporates features from the morphological descriptions it predicted previously by performing two passes over the input sentence. Finally, the tagger uses a tag filter, which deterministically assigns tags to word forms that are unambiguous in the training data. The system is trained with passive-aggressive online training (Crammer et al. 2003, 2006), which we run for ten iterations.

[2] Note that these are not the same data sets as in Chapter 4.

[3] code.google.com/p/mate-tools

[4] Unfortunately, Spanish seems to be a bad choice due to the data set that we tested on. See the discussion at the end of the chapter.

As in the previous chapter, we use mate parser (Bohnet 2009, 2010) to predict syntactic structures (see also Section 4.1.2). In order to keep training as close as possible to testing, all training sets are annotated via 10-fold jackknifing for information that is used by other models. Models that are applied to development and test data are trained on the full training sets.

5.1.3 Feature Sets

The basic feature set for the morphological tagger is shown in Table 5.1. The feature set was obtained by running an automatic forward/backward feature selection process for each of the four languages separately (Czech, German, Hungarian, Spanish) in order to arrive at a baseline system that is as good as possible without using information from the dependency tree. As mentioned before, this is important because otherwise the structural context might provide information that can be obtained by simpler means, which would then overestimate the utility of structural context.

The feature set in Table 5.1 distinguishes between *static* and *dynamic* features. Dynamic features are features extracted from morphological tags that the tagger predicted itself. Recall that the tagger performs two passes over the sentence such that during the second pass, it can use its previous predictions as features. Static features are the features that do not depend on the taggers output.

Given a token t, Table 5.1 uses feature functions to describe the feature set. Feature conjunctions are indicated by $\oplus$. form(t), lemma(t), pos(t), and mtag(t) extract the word form, the lemma, the part-of-speech, and the morphological tag, respectively. number(t) indicates whether the word form of t contains a digit, consists entirely of digits, or has no digit at all. uppercase(t) does the same with uppercase characters. prefix(t, n) and suffix(t, n) extract the first/last n characters from the word form of t. Context features are extracted from up to two tokens to the left and right of t, denoted t_{l2}, t_l, t_r, and t_{r2}.

The last-verb/next-verb and case(t) features are variants of the features in Votrubec (2006). The verb features extract information about the first verb within a window of 10 previous and 30 following tokens around t in the sentence. They approximate long-distance syntactic information by assuming that the closest verb is likely to be a syntactic head.

The feature model can thus approximate e.g., subject-verb agreement. The case(t) feature collects case information from previously tagged tokens.

STATIC FEATURES			
form(t)	pos(t_{l2})	pos$(t) \oplus$ suffix$(t,1)$	suffix$(t,1) \oplus$ suffix$(t_r,1)$
form(t_l)	pos(t_r)	pos$(t) \oplus$ suffix$(t,2)$	suffix$(t,2) \oplus$ suffix$(t_r,2)$
form(t_{l2})	suffix$(t_l,1)$	pos$(t) \oplus$ suffix$(t,3)$	last-verb-lemma(t)
form(t_{l3})	form$(t) \oplus$ pos(t)	pos$(t) \oplus$ suffix$(t,4)$	last-verb-pos(t)
form(t_r)	form$(t) \oplus$ form(t_l)	pos$(t) \oplus$ number(t)	next-verb-lemma(t)
lemma(t_{r2})	lemma$(t) \oplus$ prefix$(t,2)$	pos$(t) \oplus$ pos(t_r)	next-verb-pos(t)
pos(t_l)	lemma$(t) \oplus$ prefix$(t,3)$	pos$(t) \oplus$ pos$(t_l) \oplus$ pos(t_{l2})	
DYNAMIC FEATURES			
mtag(t_r)	pos$(t_l) \oplus$ case(t_l)	mtag$(t_l) \oplus$ mtag(t_{l2})	mtag$(t_{l2}) \oplus$ mtag(t_{l3})
last-verb-mtag	mtag$(t_r) \oplus$ mtag(t_l)	mtag$(t_r) \oplus$ mtag(t_{r2})	mtag$(t_{r2}) \oplus$ mtag(t_{r3})
next-verb-mtag	pos$(t_l) \oplus$ case$(t_l) \oplus$ pos$(t_{l2}) \oplus$ case(t_{l2})		
CZECH ONLY FEATURES			
pos$(t) \oplus$ prefix$(t,2)$			
HUNGARIAN ONLY FEATURES			
pos$(t) \oplus$ uppercase(t)			
SPANISH ONLY FEATURES			
suffix$(t,5)$	suffix$(t_r,2)$	suffix$(t_r,3)$	suffix$(t_r,4)$
prefix$(t,1)$	prefix$(t,4)$	prefix$(t,5)$	

Table 5.1: Baseline feature set.

Table 5.2 shows the structural features that we add to the baseline features when the dependency trees are available to the system. There are two kinds of features that are extracted: features of the syntactic head of token t (denoted head(t)) and features of the left-most dependent of t (denoted lmdep(t)). The head direction of a dependency arc from d to h is extracted with dir(d,h). We experimented with other types, e.g., the right-most dependent, but these features did not lead to better models. But note that this result may be specific to the way these languages encode morphological and syntactic information and other languages might profit from such a feature. Another feature that we have not tried but might be useful are the dependency relations on the arcs.

STATIC FEATURES		
lemma(head(t))	dir(t, head(t))	dir(t, head(t)) $\oplus$ pos(head(t))
suffix(head(t), 2)	suffix(head(t), 3)	pos(t) $\oplus$ pos(head(t))
suffix(lmdep(t), 1)	suffix(lmdep(t), 2)	suffix(t, 1) $\oplus$ suffix(head(t), 1)
prefix(lmdep(t), 1)	prefix(lmdep(t), 4)	
DYNAMIC FEATURES		
mtag(head(t))	mtag(lmdep(t))	

Table 5.2: Structural features.

5.2 Experiments

This section presents the experiments that we ran with the system described in the previous section. The purpose of these experiments is to test our hypothesis about explicit structural context. We first present an experiments that tests the influence of the structural context on a purely statistical morphological tagger. In the second step, we compare the purely statistical systems to a system that additionally has access to features from a lexicon, showing that the structural context and the lexicon features both contribute different information and complement each other. We follow the presentation of results with an analysis of different aspects of the experiments.

5.2.1 The Effect of Syntactic Features on Morphology Prediction

We use the off-the-shelf tagger *morfette* (Chrupała et al. 2008) as an external baseline (ext. baseline). Morfette is a system for morphological tagging and lemmatization. It trains two separate linear regression models and combines their output to produce coherent combinations of lemmas and morphological tags. In the experiments, we use the morphological tags only. The feature set of morfette extracts similar features as our baseline system, e.g., surface form, affixes, and the last two predicted tags. Its context window includes the two tokens immediately to the left and the token to the right.

The tagger described in the previous section using the basic feature set (Table 5.1)) consti-

tutes the internal baseline (int. baseline). The two baselines are compared to the tagger that uses features from the dependency tree (Table 5.2)), which we extract from automatically predicted trees (pred tree) and from the treebank trees (gold tree), which represents the oracle system.

The systems are evaluated for full tag accuracy, i.e., a token is correctly classified if the full predicted morphological description coincides with the one in the gold standard. Unknown words are evaluated separately to measure the influence of the structural information on the classification of words that the statistical model could not see in the training data.

	CZECH		GERMAN		HUNGARIAN		SPANISH	
	all	oov	all	oov	all	oov	all	oov
ext. baseline	90.37	68.66	86.78	66.37	96.19†	85.82†	97.83	89.67
int. baseline	92.51	73.12	90.92	72.52	96.08	84.49	97.83	89.05
pred tree	93.18†	74.04	92.07†	75.06	96.18	84.70	97.84	89.08
gold tree	93.64†	75.20	92.70†	76.29†	96.46†	85.30	98.11	90.34

(a) Development sets.

	CZECH		GERMAN		HUNGARIAN		SPANISH	
	all	oov	all	oov	all	oov	all	oov
ext. baseline	90.01	67.25	84.58	61.05	95.99	85.43†	97.76	91.00
int. baseline	92.29	72.58	89.11	69.67	95.94	83.76	97.59	90.88
pred tree	92.82†	73.11	90.10†	71.18	96.11	83.85	97.67	90.91
gold tree	93.30†	74.96	90.87†	73.20†	96.35†	84.50	97.88	91.61

(b) Test sets.

Table 5.3: The effect of structural context when predicting morphological information. The tables show accuracies for all tokens (all) and out-of-vocabulary tokens only (oov). † marks statistically significantly better models compared to the internal baseline (sentence-based t-test with $\alpha = 0.05$).

Tables 5.3a and 5.3b show the experimental results for development and test sets, respectively. The results show several trends, which are consistent between development and test sets: First of all, the features from the dependency tree are very useful for Czech and German, but do not yield any significant improvements for Hungarian or Spanish. For Czech and German, improvements are between 0.5 (Czech) and 1.0 (German) percentage points absolute. On unknown words, improvements are 0.2 (Czech test set) and 2.5 (German development set) percentage points absolute.

Our system performs on par with, or better than, the external baseline in terms of token accuracy, but morfette outperforms our system on unknown token accuracy for Hungarian. For Spanish, all systems yield similar results. The oracle experiments with gold-standard trees show that all languages can profit from the additional information, which is clearly visible for Czech and German, but also the experiments on Hungarian and Spanish show small but visible increases in accuracy. Finally, the pronounced differences in Czech and German between the system using predicted trees and the system using gold-standard trees demonstrate the propagation of parsing errors.

This experiment thus shows that our hypothesis, namely that explicit structural context leads to better models than a fixed-size context window, is indeed true. However, we also see that a strong effect occurs only for Czech and German, but not for Hungarian or Spanish, even though the oracle experiment indicates that there is some small potential. One thing that can be noted for Hungarian and Spanish is that all models give very high accuracies. In Spanish, the accuracies are so high that it is doubtful that any more improvement can be made on this data set. For Hungarian, we already saw in the previous chapter that predicting morphology seems to be an easier task than for Czech and German due to the low rate of ambiguity.

For Czech and German, however, we find a strong effect. Clearly, the model that uses the structural context has access to information that the standard context model cannot find. In Section 5.3, we show that part of the improvement comes from a better modeling of grammatical agreement in the two languages.

5.2.2 Including Information from Lexicons

But before we analyze the experimental findings in more detail, we present a second experiment where we add information from lexicons to the four systems that we compared in the previous section. Lexicons encode knowledge that is at least to some extent difficult to pick up for a purely statistical system. For example, the gender value of a noun in Czech or German cannot be read off of a word form. While there are systematic patterns in the gender assignment in these languages, especially in Slavic languages, there is always at least a small semantic core in a noun class system, where gender is assigned according to the meaning of a word, and not the form Corbett (1991). For these words, the gender has

to be learned together with the word. In this respect, gender is very different from case because case is a structural feature that is closely related to the syntax of the sentence. This difference shows that it may not be such a good idea to treat all morphological features the same way.

Lexicons can help a statistical system to learn information like gender for words that do not occur in its training data, and thus are a good means to deal with data sparsity. Lexicons can also be used to speed up processing considerably by restricting the search space of the statistical model, but we do not pursue this approach here.

The current state-of-the-art in predicting morphological features makes use of morphological lexicons (e.g. Hajič 2000, Hakkani-Tür et al. 2002, Hajič 2004, Spoustová et al. 2009, Müller et al. 2013). We extend the system described in Section 5.1.2 to include information from a morphological dictionary. Morphological dictionaries return all possible analyses for a given word form, but do not disambiguate the input. For Czech, we use the morphological analyzer distributed with the Prague Dependency Treebank 2 Hajič et al. (2006). For German, we use DMor Schiller (1994). For Hungarian, we use Trón et al. (2006), and for Spanish, we use the morphological analyzer included in Freeling Carreras et al. (2004). The output of the analyzers is given to the system as features that simply record the presence of a particular morphological analysis for the current word. The system can thus use the output of any tool regardless of its annotation scheme which is important if the annotation scheme of the treebank is different from the one of the morphological dictionary.

In the following experiment, we change the external baseline for Czech and German because the standard implementation of morfette cannot make use of external lexicons. For Hungarian and Spanish, we did not run external baselines using lexicons. As far as we can tell, the automatically predicted information that comes with the data sets might have used external lexicons, but since we cannot verify this and the quality is rather poor we do not present it here. For Czech, we show results from *featurama*[5] with the feature set developed by Votrubec (2006) as the external baseline. Featurama is a standard sequence model that was specifically developed for Czech. For German, we compare to RFTagger Schmid and Laws (2008) as external baseline.[6] RFTagger decomposes the morphological

[5] http://sourceforge.net/projects/featurama/

[6] RFTagger uses SMOR (Schmid et al. 2004) as morphological lexicon, which was originally based on DMOR but was improved since.

tags into their individual attributes (e.g., gender, number, and case) and models the context with a hidden markov model. Both Featurama and RFTagger use the Viterbi algorithm to find the optimal sequence over the given input sentence.

	CZECH		GERMAN		HUNGARIAN		SPANISH	
	all	oov	all	oov	all	oov	all	oov
ext. baseline	94.75	84.12	90.63	72.11				
int. baseline	93.80	80.47	92.59	80.73	97.27	92.61	98.23	92.46
pred tree	94.40†	81.51	93.70†	82.71	97.38	92.39	98.24	92.30
gold tree	94.80†	82.45	94.28†	84.12†	97.63†	92.79	98.40	92.82

(a) Development sets.

	CZECH		GERMAN		HUNGARIAN		SPANISH	
	all	oov	all	oov	all	oov	all	oov
ext. baseline	94.78	84.23	89.04	70.80				
int. baseline	93.57	80.53	91.48	78.83	97.03	91.28	98.02	93.15
pred tree	94.24†	81.61	92.51†	80.20	97.19	91.50	98.07	93.03
gold tree	94.64†	82.80	93.32†	82.35†	97.45†	91.92	98.22†	93.64

(b) Test sets.

Table 5.4: The effect of explicit structural context when predicting morphological information using lexicons. The tables show accuracies for all tokens (all) and out-of-vocabulary tokens only (oov). † marks statistically significantly better models compared to the internal baseline (sentence-based t-test with $\alpha = 0.05$).

Tables 5.4a and 5.4b present the experimental results when all systems have access to information from a morphological lexicon. The results show the expected increase in overall performance for all systems compared to Table 5.3, especially on out-of-vocabulary tokens. This shows that even with the considerable amounts of training data available nowadays, morphological dictionaries are important resources for morphological description (cf. Hajič 2000). Note also that featurama outperforms our best system, even the one using gold-standard syntax. This, however, does not mean that syntactic information is unnecessary. We show in Section 5.3.2 that featurama also improves when given information extracted from dependency trees. Its superior performance here is due to featurama's better feature set and its Viterbi decoding.

Figure 5.2 compares the contribution of syntax and lexicon features for all four languages. Lexicon features lead to a large improvement in all four languages, syntax features lead

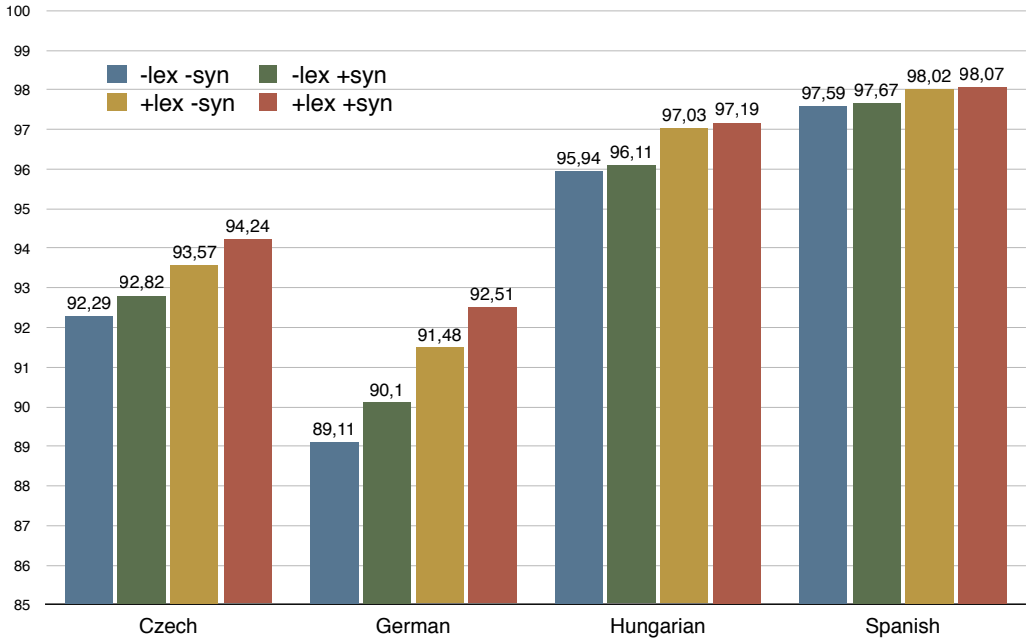


Figure 5.2: The contribution of structural and lexicon features.

to smaller improvements in Czech and German. However, it becomes clear from the comparison that the contribution of lexicons and structural context are almost completely orthogonal. Both sources contribute complementary information with almost no overlap. While lexicons provide information about a word form, e.g., inherent features like gender, structural features provide information about the syntactic context of the word in the sentence that is necessary to correctly disambiguate the word form. For example, we show in Section 5.3.1 that syntactic feature help in Czech and German with getting grammatical agreement right.

5.3 Analysis

This section presents analyses of the experimental results, that investigate the experiments from different angles. First, we look at the improvements with respect to a specific morphosyntactic phenomenon, namely grammatical agreement. Then, we show that featurama, the external baseline that we compared with for Czech and which outperformed

our best system also improves when provided with structural features. We then try to turn the wheel one step further and test whether the parser can profit from the improved morphology in Czech and German (it does not). Finally, we conduct an experiment to find out the minimum amount of parsing information that is necessary to train a parser for predicting morphology with structural features.

5.3.1 Agreement

From the experimental results we concluded that structural information helps for predicting morphology in Czech and German, but not for Hungarian and Spanish. A closer look at the output shows an interesting difference between these two pairs of languages with respect to tokens that are in a grammatical agreement relation with another token. Agreement is a phenomenon where morphology and syntax strongly interact. Morphological features co-vary between two items in the sentence, but the relation between these items can occur at various linguistic levels (Corbett 2006). If structural information helps with predicting morphological information, we expect this to be particularly helpful with getting agreement right, since agreement relations can hold over long distances in a sentence and are therefore difficult to capture for a sequence model.

All four languages in our experiments show agreement to some extent. Specifically, all languages show agreement in number (and person) between the subject and the verb of a clause. Czech, German, and Spanish show agreement in number, gender, and case (not Spanish) within a noun phrase. Hungarian shows case agreement within the noun phrase only rarely, e.g., for attributively used demonstrative pronouns. In order to test the effect on agreement, we measure the accuracy on tokens that are in an agreement relation with their syntactic head. We counted subject-verb agreement as well as agreement with respect to number, gender, and case (where applicable) between a noun and its dependent adjective and determiner. The patterns were defined manually over part-of-speech tags in the respective treebank.

Table 5.5 displays the counts from the development sets of each language. We compare the internal baseline system that does not use any structural information with the output of the morphological tagger that uses the gold-standard tree. We use the gold-standard trees rather than predicted ones in order to eliminate any influence from parsing errors. Note

that we assume here that an agreement relation holds along a direct link in the dependency tree. This depends on the treebank annotation scheme, which in our case holds in all four treebanks.

agreement type	baseline	gold tree
	CZECH	
subject-verb	3199/4044 = 79.10	3264/4044 = 80.71
NP case	8719/9132 = 95.48	8821/9132 = 96.59
NP number	8933/9132 = 97.82	9016/9132 = 98.73
NP gender	8493/9132 = 93.00	8768/9132 = 96.01
	GERMAN	
subject-verb	4412/ 4696 = 93.95	4562/ 4696 = 97.15
NP case	13340/13951 = 95.62	13510/13951 = 96.84
NP number	13631/13951 = 97.71	13788/13951 = 98.83
NP gender	13253/13951 = 95.00	13528/13951 = 96.97
	HUNGARIAN	
subject-verb	8653/10219 = 84.68	8655/10219 = 84.70
NP case	402/ 891 = 45.12	412/ 891 = 46.24
	SPANISH	
subject-verb	1930/2004 = 96.31	1932/2004 = 96.41
NP number	8810/8849 = 99.56	8816/8849 = 99.63
NP gender	8810/8849 = 99.56	8821/8849 = 99.68

Table 5.5: Accuracies on agreement in morphological annotation compared between the baseline system and the oracle system using gold trees.

The results in Table 5.5 show that the quality of morphological tags for tokens that are in an agreement relation with their head token improves in Czech and German when structural context is available, whereas in Spanish and Hungarian, only very tiny changes occur. For Czech and German, these results confirm that structural context helps with agreement.

We believe that the reasons why it does not help for Hungarian and Spanish are the following: for Spanish, we see that also the baseline model achieves very high accuracies (cf. Table 5.3) and also high rates of correct agreement, as is visible from Table 5.5. It seems that for Spanish, structural context is simply not necessary to make the correct prediction. However, the results are so high that we cannot exclude the possibility that this is an

artifact not of the language but of the data set. The same experiment would need to be repeated on additional data sets of Spanish to really confirm that it is indeed the language for which predicting morphology is easy.

For Hungarian, we believe that the reason is in the inflectional paradigms of the language, which show very few if any form syncretism, meaning that word forms in Hungarian are usually not ambiguous within one morphological category (e.g., case). Making a morphological tag prediction, however, is difficult only if the word form itself is ambiguous between several morphological tags. If the word form is unambiguous, structural context is unnecessary for disambiguation. In Czech and German on the other hand, where form syncretism is pervasive in the inflectional paradigms, features from the dependency tree provide informative context to better disambiguate syncretic word forms.

5.3.2 Featurama with Features from the Dependency Tree

In the Czech experiments in Section 5.2.2, the external baseline tool, featurama, performs as good as our best system, which has access to gold-standard trees. It outperforms our baseline system by a percentage point absolute (and even more for unknown tokens). The question then arises whether the structural context actually contributes something new to the task, or whether the same effect could also be achieved with a better feature model alone as in featurama.

In order to test this we ran an additional experiment, where we added some of the structural features to the feature set of featurama. Specifically, we add the static features from Table 5.2 that do not use lemma or part-of-speech information, because featurama predicts this information on its own. Due to the Viterbi decoder in featurama, we cannot use features from the morphological tags (the dynamic features). The results in Table 5.6 show that featurama profits from structural context as well thus corroborating the findings from the experiments in Section 5.2.

	dev set		test set	
	all	oov	all	oov
featurama	94.75	84.12	94.78	84.23
pred syntax	95.18	84.65	95.09	84.52
gold syntax	95.39†	84.62	95.34†	85.03

Table 5.6: Structural features for featurama (Czech). † mark statistically significantly better models compared to featurama (sentence-based t-test with $\alpha = 0.05$).

5.3.3 Can the Improved Morphology Help the Parser?

Now that we have seen that structural context information improves a model for predicting morphology for Czech and German, where syntax and morphology interact considerably. A natural follow-up question is whether the improvement also occurs in the other direction, namely whether the improved morphology also leads to better parsing models.

In the previous experiments, we ran a 10-fold jackknifing process to annotate the training data with morphological information using no structural features and afterwards use jackknifing with the parser to annotate syntax. The predicted dependency trees are then used to extract features for the experiments that use structural context. The same process can be applied once more with the morphology prediction in order to annotate the training data with morphological information that is predicted using information from the trees. A parser trained on this data can then use the improved morphology in its feature set. If the improved morphology has an impact on the parser, the quality of the second parsing model should then be superior to a baseline parsing model, which uses morphology predicted without structural context. Note that for the following experiments, neither morphology model uses the morphological lexicon.

Table 5.7 presents the evaluation of two parsing models (one using morphology without structural information, the other one using the improved morphology). The results show no improvement in parsing performance when using the improved morphology. Looking closer at the output, we find differences between the two parsing models with respect to grammatical functions that are morphologically marked. For example, in German, performance on subjects and accusative objects improves while performance for dative

	CZECH				GERMAN			
	dev set		test set		dev set		test set	
	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS
baseline morph	81.73	88.45	81.02	87.77	91.16	92.97	88.06	90.24
morph w/ syntax	81.63	88.37	80.83	87.61	91.20	92.97	88.15	90.34

Table 5.7: Impact of the improved morphology on the quality of the dependency parser for Czech and German.

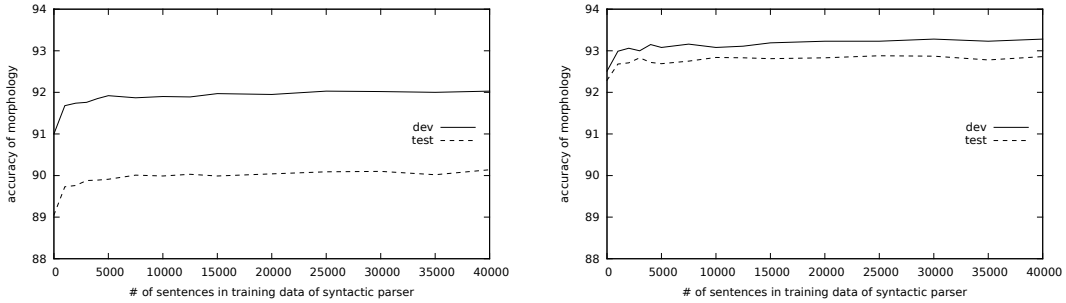
objects and genitives decreases.

5.3.4 Minimum Parsing Quality

As a last set of analyses, we test how good the parser needs to be. We do this in two ways: limiting the amount of training data, and using an extremely simple parser.

In the first step, we train several parsing models on increasing amounts of syntactically annotated data. For example, the first model uses the first 1,000 sentences of the treebank. We perform 5-fold jackknifing with the parser on these sentences to annotate them with dependency trees. Then we train one parsing model on these 1,000 sentences and use it to annotate the rest of the training data as well as the development and the test set. This gives us the full data set annotated with syntax that was learned from the first 1,000 sentences of the treebank. The morphological tagger is then trained on the full training set and applied to development and test set.

Figure 5.3 shows the dependency between the amount of training data given to the parser and the quality of the morphological tagger using structural features provided by this parser. The left-most point corresponds to a model that does not use any information from the dependency tree. For both languages, German and Czech, we find that already 1,000 sentences are enough training data for the parser to provide useful structural information to the morphological tagger. After 5,000 sentences, both curves flatten out and stay on the same level. These results show that the parser already picks up the relevant information



(a) German. **(b)** Czech.

Figure 5.3: Dependency between amount of training data for the syntactic parser and quality of morphological prediction using structural context.

quite early.

In the second step, we test now whether we can get the same effect with a very simple parser. We use the brute-force algorithm described in Covington (2001), which simply selects a head for each token in the sentence. It does not have any tree requirements, so there is no guarantee for a cycle-free tree structure. In Table 5.8, we compare the simple parser with mate parser, both trained on the first 5,000 sentences of the treebank. We use 5,000 sentences to be on the safe side even though the previous experiment suggests that 1,000 sentence might already be enough. Evaluation is done with labeled (LAS) and unlabeled attachment score (UAS).

	CZECH				GERMAN			
	dev set		test set		dev set		test set	
	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS
simple parser (5k)	71.57	78.96	69.09	77.23	83.06	85.23	78.56	81.18
full parser (5k)	76.77	84.38	74.70	83.00	87.56	90.08	83.69	86.58

Table 5.8: Simple parser vs full parser – syntactic quality. Trained on first 5,000 sentences of the training sets.

As expected, the simple parser performs much worse in terms of attachment accuracy. Table 5.9 shows the performance of the morphological tagger when using the output of both parsers as the structural context. For Czech, both parsers seem to supply similar

information to the morphological tagger, while for German, using the full parser is clearly better. In both cases, the morphological tagger outperforms the models that do not have access to structural information. The performance on unknown words is however much worse, probably because the simple parser is very unreliable on unknown words.

	CZECH				GERMAN			
	dev set		test set		dev set		test set	
	all	oov	all	oov	all	oov	all	oov
no syntax	92.51	73.12	92.29	72.58	90.92	72.52	89.11	69.67
simple syntax	92.96	73.45	92.53	72.66	91.52	73.34	89.66	70.52
full syntax	93.08	73.64	92.69	73.39	91.92	83.46	89.91	80.50

Table 5.9: Simple parser vs full parser – morphological quality. The parsing models were trained on the first 5,000 sentences of the training data, the morphological tagger was trained on the full training set.

These two experiments show that the information that the parser needs to provide to the morphology prediction seems to be very easy to learn. Even with rather little training data and a very simple-minded parsing approach, the morphology is able to improve to some extent albeit not as much as with a full-blown parser. This fits well with the observation that syntax helps in Czech and German with getting agreement right, since most of the agreement relations that we tested in Section 5.3.1 holds within noun phrases, which are very frequent and parsers should learn their general structure already from few sentences.

5.4 Discussion

In this chapter, we set out to demonstrate empirically that for morphologically rich languages, modeling the syntactic context structurally leads to better models than a regular window-based context model. The experiments show that this is indeed the case. When our baseline system is extended with features that are extracted from a dependency tree, it outperforms the baseline system without this information significantly for Czech and German. The analysis shows that this is partly due to a better modeling of grammatical agreement, one of the long-distance morphosyntactic relations that occur in languages with rich morphology and free word order.

We also see the same difference that we already observed in the previous chapter between Czech and German on one side and Hungarian on the other. While the serious problem for Czech and German is the ambiguity that needs syntactic information to be resolved, the main problem for the Hungarian data set seems to be the coverage of the lexicon rather than the high amount of ambiguity in the word forms. This can be seen also from the improvements in Hungarian once an external lexicon is added. For the Spanish data set, the case is a bit different. As for the other three languages, it is the case that a morphological dictionary improves the accuracy, but generally, the results are so good that it is doubtful that this can be improved further. 98% accuracy is so high that is reasonable to assume that most of the remaining 2% are likely to be annotation errors. In this case, a system cannot achieve better numbers on this data set (unless it somehow learns the idiosyncrasies of the annotation errors). This leaves to options: either, predicting morphology in Spanish is so easy that there is no need for better models. Or the data set is not representative, in which case the experiments for Spanish should be verified on a different data set before final conclusions can be drawn. It should be said, however, that from a linguistic point of view, it makes sense to assume that predicting morphology for Spanish is probably an easier task than for the other three languages.

For all four languages, external lexicons improve the prediction accuracy. This is due to increased coverage because the statistical model can make better conclusions about unknown words when the additional information from the lexicon is available. The experiments show that both syntactic context as well as lexicon information improve the results, but the overlap between the two is rather small so they seem to provide complementary information.

In the analysis, we find that the improved morphology cannot improve the parser further. We believe that this is because of two reasons. The theoretical reason is in the nature of syncretism: Baerman et al. (2005) describe syncretism informally as *Morphology lets the syntax down*, meaning that a syncretic word form fails to make a syntactically relevant distinction. This description explains, why we can use syntax to disambiguate syncretic word forms, because if we know the correct syntactic structure, we can use it to decide what the correct interpretation of the syncretic word form must have been. The practical reason is that we perform jackknifing to provide morphology to the parser. The main motivation for jackknifing is to provide as realistic input as possible during training compared to testing, so that the parser can learn *when to trust the information from the morphology*. This alleviates error propagation due to incorrectly predicted morphology

since the parser already learns not to trust some annotations in the first place (we have seen in the previous chapter that this may still lead to errors though). For this reason, we get syntactic trees that are good enough to help us disambiguate some of the morphologically ambiguous word forms, but the improved morphology cannot improve the syntax further since the parser already learned to predict the correct structure without the additional information it provides. As an example, consider a noun phrase in German. Assuming that the determiner is ambiguous between two values then the model will only be able to correctly disambiguate the word via agreement if the determiner has been attached to the correct head noun because only from there it can get the necessary information. But if the head can be correctly attached already, the model does not need the correct morphological information from the determiner to find the correct head. However, we can see from the oracle experiments that there is still room to improve, so there must be cases where incorrectly predicted syntax prevents the correct prediction of the morphology.

For this reason (and for efficiency considerations) the experimental setup in this chapter is useful for showing the effect of structural information on morphological prediction, but it is not a joint model in itself. Joint models do not work in cyclic dependencies as here but they model the dependencies jointly. In a joint model, the head selection can be made while simultaneously optimizing for the best morphological tag for the determiner that is consistent with the attachment. In this and the previous chapter, we have provided evidence for the claim that pipeline models are deficient models for morphologically rich languages. In the next two chapters, we now propose two joint models for morphology and syntax and show that these models indeed outperform a pipeline.

5.5 Automatic Prediction of Morphological Features

Automatically predicting morphology is an old task and for most languages, it is a crucial part of the processing pipeline. Hence, systems have been developed for many languages, often combining a morphological lexicon with a statistical disambiguation model (Hakkani-Tür et al. 2002, Hajič 2004, Smith et al. 2005, Schmid and Laws 2008, Spoustová et al. 2009). These systems cast the task as a sequence labeling task that finds the best sequence of morphological tags given a sequence of words.

Chapter 6

Morphosyntax with Symbolic Constraints

In Chapter 4, we have shown that error propagation is a problem for pipeline models because the relevant syntactic context for making correct decisions is not always available. In Chapter 5, we have then shown that automatic prediction of morphology becomes better when the model has access to the full syntactic structure of the sentence. Furthermore, the analysis in both chapters has shown that syncretism is one source for errors and that the structural information helps modeling long distance morphosyntactic dependencies like agreement.

In this chapter, we turn the question of the previous chapter around and test whether we can improve the parser by explicitly modeling such morphosyntactic dependencies. To this end, we combine and extend the work by Riedel and Clarke (2006) and Martins et al. (2009). Riedel and Clarke (2006) propose to view dependency parsing as a constrained optimization problem, where the objective is to find the combination of arcs between the words in a sentence that maximizes the overall score of the tree. A set of constraints is defined to ensure that the solution to this optimization is a proper dependency tree. They formulate the parser as an integer linear program, which they can solve with any general-purpose constraint solver for linear programs. Although solving an integer linear program is NP hard in the general case, the formulation allows them to add additional constraints to the problem easily. They use these constraints to model linguistic knowledge, for example

that a word should have at most one subject.

The original formulation by Riedel and Clarke (2006) had one flaw, namely that they could not find a concise formulation to enforce the tree structure of the output. They resorted to an iterative solution algorithm to find the best scoring well-formed tree but this procedure is very inefficient. A concise formulation of the problem that enforces the tree structure immediately was later presented by Martins et al. (2009).

We adopt the formulation by Martins et al. (2009) and use the parser to test the effect of explicit morphosyntactic constraints.[1] We take the idea of linguistic constraints by Riedel and Clarke (2006) and implement it with constraints for case licensing, agreement, and uniqueness. The idea is to restrict the search space of the parser such that it can only output trees that do not violate these morphosyntactic rules. The underlying statistical model that drives the parser remains untouched be the constraints but we use automatically predicted morphological information in the feature set. This means that the statistical model is still used to find the optimal dependency tree but the parser sets linguistically motivated boundaries within which the model must stay.

The motivation for this comes from the findings in the previous two chapters, where we saw that the statistical model can indeed learn the rules but still makes mistakes due to error propagation and because other features sometimes overrule the evidence from morphology. However, we know that for a given language certain morphosyntactic rules hold and an alternative to letting the statistical model learn them is to simply enforce them in the decoder. This is the purpose of the morphosyntactic constraints in the parser. The idea is to see these rules as structural requirements for well-formed dependency trees rather than as features in the statistical model. The hypothesis that we test in this chapter is thus that the explicit modeling of morphosyntactic rules can improve the parsing quality because the statistical model can rely on the rules to guide it towards the correct structure.

We evaluate the morphosyntactic constraints on Czech, German, and Hungarian. We use the mate parser models from Chapter 4 as a state-of-the-art pipeline baseline and compare it to two versions of the constraint parser, namely one with and one without the additional morphosyntactic constraints. We start by giving the full formulation of the integer linear programming parser in Section 6.1 and define the morphosyntactic

[1] The content of this chapter is published in Seeker and Kuhn (2013a).

constraints in Section 6.1.3. Section 6.2 presents the empirical evaluation of the constraints and we conclude the chapter with a discussion of the results in Section 6.3.

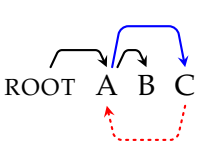
6.1 Parsing with Morphosyntactic Constraints

As we said before, the parser in this chapter defines dependency parsing as a constrained optimization problem implemented as an integer linear program following Martins et al. (2009). For the sake of completeness, we present the full formulation of the parser. To the most part, the formulation very closely follows the one presented by Martins et al. (2009) but we additionally extend it to labeled parsing. This extension increases the number of variables in the model and might be thought to make the model difficult to solve. Our experiments however show that with careful pruning, the introduction of labels into the parsing process does not have a big impact on parsing speed. We describe the set of morphosyntactic constraints in the second part of this section.

6.1.1 Dependency Trees as Indicator Vectors

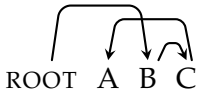
In this and the following chapter, we represent dependency trees formally as indicator vectors. An indicator vector is a vector of binary variables, where each variable indicates the presence of a particular dependency arc in the tree. There is one binary variable for each possible dependency arc between the words of a given input sentence. Figure 6.1 illustrates how a dependency structure for a three word sentence (+ artificial root node) is encoded as an indicator vector. The first three positions encode the arcs between the root node and each of the other words. The following six positions (3×2) encode the arcs from each word to the other two. In Figure 6.1a the color coding exemplifies the encoding of single arcs. The blue arc (arc from A to C) is encoded by the blue 1 in the 5th position. Because the dotted red arc is not present in the tree, the 8th position (arc from C to A) in the vector is 0.

Given the set of tokens T in a sentence and a set of dependency labels L, we define an



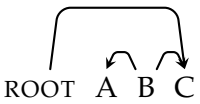
$\langle 1, 0, 0, 1, 1, 0, 0, 0, 0 \rangle$

(a) A dependency tree.



$\langle 0, 1, 0, 0, 0, 0, 1, 1, 0 \rangle$

(b) A non-projective tree.



$\langle 0, 0, 1, 0, 0, 1, 1, 0, 0 \rangle$

(c) Not a well-formed tree.

Figure 6.1: Encoding graph structures as binary indicator vectors.

index set A for dependency arcs as

$$A := \{ \langle h, d, l \rangle \mid h \in T \cup \{\text{ROOT}\}, d \in T, l \in L, d \neq h \} \tag{6.1}$$

A dependency tree can thus be represented by an indicator vector

$$\boldsymbol{y} := \langle y_a \rangle_{a \in A} \tag{6.2}$$

with $y_a \in \{0, 1\}$. We then define $\mathcal{Y}$ as the set of all well-formed dependency trees, where well-formed means that the tree fulfills the three conditions on the structure (root has no head, one head per word, no cycles). This includes non-projective trees (Figure 6.1b), but excludes indicator vectors like the one in Figure 6.1c because this graph is not a well-formed dependency tree (word C has two heads, word B has none).

6.1.2 The Basic Architecture

The general idea of the parser is to represent each arc in a fully connected graph as a binary variable of an integer linear program, each of which associated with a score from a statistical model. A set of constraints imposes the formal properties of a dependency tree such that finding the highest-scoring combination of arcs under this set of constraints outputs the best-scoring dependency tree. Typologically, this parser belongs to the graph-based paradigm as it performs global optimization to find the best spanning tree over a given set of tokens.

Formally, let T be the set of tokens in a given sentence and $T_0 = T \cup \{\text{ROOT}\}$ be the set of tokens including a special root token. Furthermore, let L be the set of dependency relations or dependency labels. A and U then define two index sets, one for labeled and one for unlabeled arcs, respectively.

$$A := \{ \langle h, d, l \rangle \mid h \in T_0, d \in T, l \in L, h \neq d \} \tag{6.3}$$

$$U := \{ \langle h, d \rangle \mid h \in T_0, d \in T, h \neq d \} \tag{6.4}$$

A dependency tree is an indicator vector of binary variables,

$$\boldsymbol{y} := \langle y_a \rangle_{a \in A} \tag{6.5}$$

where $y_a = 1$ means that arc a is in the parse, otherwise $y_a = 0$. We define $\mathcal{Y}$ to be the set of all well-formed dependency trees (projective and non-projective).

The basic parser assumes an arc-factored model (McDonald et al. 2005), in which the score of a dependency tree is defined as the sum of the scores of the individual arcs in that tree. The objective function of the integer linear program is thus to find the combination of arcs that has the highest sum of arc scores

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \sum_{a \in A} y_a \left(\boldsymbol{w} \cdot \phi_{\text{ARC}}(a) \right) \tag{6.6}$$

with $\boldsymbol{w}$ being the weight vector and ϕ_{ARC} being a function that represents arcs as feature vectors. Solving the integer linear program finds the $\hat{\boldsymbol{y}}$ that maximizes the objective function given a set of additional constraints.

In order to formulate the set of constraints that force the variables in $\hat{\boldsymbol{y}}$ to form a dependency tree, we need some auxiliary definitions. First, we define the set of potential incoming and outgoing arcs for each token (one version each for labeled and unlabeled

arcs):

$$A_h^{in} := \{ \langle g, h, l \rangle \mid g \in T_0, l \in L, g \neq h, \langle g, h, l \rangle \in A \} \tag{6.7}$$

$$A_h^{out} := \{ \langle h, d, l \rangle \mid d \in T, l \in L, h \neq d, \langle g, h, l \rangle \in A \} \tag{6.8}$$

$$U_h^{in} := \{ \langle g, h \rangle \mid g \in T_0, g \neq h, \langle g, h \rangle \in U \} \tag{6.9}$$

$$U_h^{out} := \{ \langle h, d \rangle \mid d \in T, h \neq d, \langle g, h \rangle \in U \} \tag{6.10}$$

It holds that $A_h^{in} \subset A$, $A_h^{out} \subset A$, $U_h^{in} \subset U$, and $U_h^{out} \subset U$.

With these subsets, we can define the first property of a dependency tree, namely that each token has exactly one head

$$\sum_{a \in A_t^{in}} y_a = 1 \quad \text{for all } t \in T \tag{6.11}$$

Note that this constraint is defined as in Martins et al. (2009) but it ranges over labeled arcs. There is no need to change anything because also for labeled arcs, only one of them can be active at any given time. Furthermore, Martins et al. (2009) have an additional constraint that states that the root node does not have a head. Here, this constraint is not necessary because the index set of arcs does not contain incoming arcs to the root (Equation (6.3)).

For the second property, acyclicity, Martins et al. (2009) employ a single-commodity flow formulation (Magnanti and Wolsey 1995). The idea is to enforce acyclicity by enforcing connectedness with the root for each token. The root sends units of flow along the arcs of the dependency tree and each token consumes one unit of this flow. A token can only consume its unit of flow if there is a path from the root to this token in the tree. Since each token can only have a single head (Equation (6.11)), only acyclic trees can fulfill this condition for each token simultaneously. This is because any cycle necessarily disconnects the nodes in the cycle from the rest of the tree and then there will be no path from the root node to the nodes in the cycle.

To model the flow, we need an additional set of variables. The flow variables are represented as a vector of integer variables, one for each unlabeled arc in the graph. The flow variables are not restricted to binary values because their value represents the flow on the respective arc:

$$\boldsymbol{f} := \langle f_u \rangle_{u \in U} \tag{6.12}$$

The flow variables are connected to the labeled arcs in the graph via an inequality. Only if one of the labeled arcs between a pair of dependent and head is active, the flow variable can carry flow:

$$|T| \sum_{l \in L} y_{\langle h,d,l \rangle} \geq f_{\langle h,d \rangle} \quad \text{for all } h \in T_0, d \in T \tag{6.13}$$

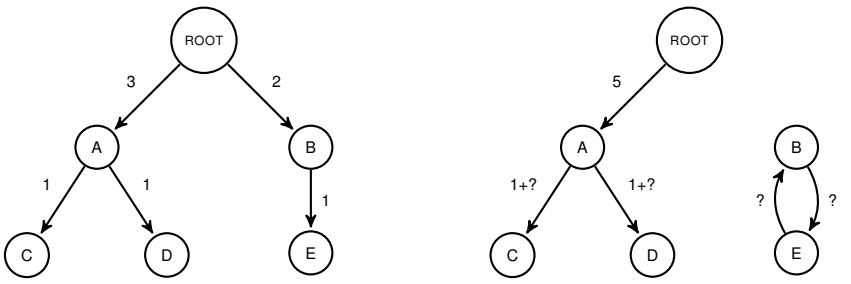
Note that because of the single-head constraint, the value of the sum on the left-hand side can be either 1 or 0.

The root node is treated differently than the other nodes. Acylicity is modeled by ensuring that there is a path from the root node to every other node in the tree. Since every node is supposed to consume one unit of flow, the flow on the outgoing arcs of the root node is set to the number of other nodes (i.e., the number of tokens in the sentence), which means it sends $|T|$ units:

$$\sum_{u \in U^{out}_{\text{ROOT}}} f_u = |T| \tag{6.14}$$

Every other token consumes one unit of flow, i.e., the difference between the incoming flow and the outgoing flow is one.

$$\sum_{u \in U^{in}_t} f_u - \sum_{u \in U^{out}_t} f_u = 1 \quad \text{for all } t \in T \tag{6.15}$$



(a) In well-formed dependency trees, flow flows from root to each of the nodes.

(b) Cycles disconnect the structure such that flow cannot reach all nodes.

Figure 6.2: Schema of how flow constraints prevent cycles.

Figure 6.2 illustrates the idea of Martins et al. (2009) of using single-commodity flow to enforce acyclicity. Two structures are shown, one that fulfills the constraints (Figure 6.2a) and one that does not (Figure 6.2b). In Figure 6.2a, the root node sends 5 units of flow (distributed over the outgoing arcs as 3+2), one for each other node in the tree. This is

the content of Equation (6.14). Every other node consumes one unit of flow such that the difference between the flow on the incoming arcs and the flow on the outgoing arcs is 1. This is the content of Equation (6.15). Figure 6.2b now shows a structure that violates Equation (6.15) in several places. The cycle that is formed by nodes B and E makes it impossible to fulfill the constraint simultaneously for both nodes because the incoming flow of B should have a value that is one larger than the outgoing flow, but the same is supposed to hold for E. Furthermore, the root node sends 5 units of flow to node A which distributes them to its dependents. However, since nodes C and D have no outgoing arcs, the difference between the incoming and the outgoing flow is going to be bigger than one. We see then that cyclic structures cannot fulfill the flow constraints and are therefore excluded as valid output structures of the parser.[2]

Equations (6.6), (6.11) and (6.13) to (6.15) plus the domain restrictions

$$\boldsymbol{y} \in \mathbb{B}^d, \boldsymbol{f} \in \mathbb{Z}^d \tag{6.16}$$

form an integer linear program and represent a first-order graph-based dependency parser. Finding the best solution of this integer linear program solves the same task as running the Chu-Liu-Edmonds algorithm (Chu and Liu 1965, Edmonds 1967) as proposed in McDonald et al. (2005). It can be considerably slower than Chu-Liu-Edmonds since solving an integer linear program is of exponential complexity in the worst case. However, unlike the Chu-Liu-Edmonds algorithm, it allows us to add additional conditions to the constraint set, for example to model second-order features but also constraints to model morphosyntax (see Section 6.1.3).

Second-order Parsing

Martins et al. (2009) add several additional constraints to the basic formulation to facilitate second-order features. In our parser, we adopt two of them, namely what they call *all siblings* and *all grandchildren*. The idea in both cases is to introduce an auxiliary variable for each pair of arcs, e.g., two arcs with the same head (siblings). This auxiliary variable is coupled with the respective arc variables and is only active if both of the arcs of the pair

[2] It should be noted that the flow is used as a metaphor in this formulation. The constraint solver searches for a structure that fulfills all constraints simultaneously and there is no distribution of flow values in cyclic trees that would do so. But there is not actually anything that flows.

are active. If it is active, it contributes its weight to the total score of the tree.

We define index sets for sibling and grandchildren pairs of arcs (S and G, respectively):

$$S := \{ \langle h, d, s \rangle \mid h \in T_0, d \in T, s \in T, h \neq d, h \neq s, d \neq s \} \tag{6.17}$$

$$G := \{ \langle g, h, d \rangle \mid g \in T_0, h \in T, s \in T, g \neq h, h \neq d, g \neq d \} \tag{6.18}$$

Note that we do not include labels in second-order features.

In the following, we demonstrate the all sibling formulation. The grandchildren work completely analogously. First, we define the binary variables, one for each sibling factor:

$$\boldsymbol{s} := \langle s_a \rangle_{a \in S} \tag{6.19}$$

A set of three constraints for each sibling factor couples the variable with the two arcs in the dependency tree:

$$s_{\langle h,d,s \rangle} \leq \sum_{l \in L} y_{\langle h,d,l \rangle} \tag{6.20}$$

$$s_{\langle h,d,s \rangle} \leq \sum_{l \in L} y_{\langle h,s,l \rangle} \tag{6.21}$$

$$s_{\langle h,d,s \rangle} \geq \sum_{l \in L} y_{\langle h,d,l \rangle} + \sum_{l \in L} y_{\langle h,s,l \rangle} - 1 \tag{6.22}$$

Equations (6.20) and (6.21) ensure that the sibling factor cannot be active if one of the arc variables is not active. Equation (6.22) states that the sibling factor must be active if both of the arc variables are active. The equations loop over the labels for each arc because the actual arc label is unknown at this point. The single head constraint (Equation (6.11)) guarantees that at most one of the arcs will be active, i.e., the sums over labels in these equations will always evaluate to either 0 or 1.

The objective function changes accordingly to also factor over siblings

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \sum_{a \in A} y_a \left(\boldsymbol{w} \cdot \phi_{\text{ARC}}(a) \right) + \sum_{a \in S} s_a \left(\boldsymbol{w_s} \cdot \phi_{\text{SIB}}(a) \right) \tag{6.23}$$

with $\boldsymbol{w_s}$ representing the weights for siblings and ϕ_{SIB} being the feature function for sibling factors. As before, the constraint solver searches for the solution with the highest overall

score, which now also includes scores from sibling factors.

Inference, Learning, and Feature Model

Inference is performed using a general purpose constraint solver for linear programs, in our case the GUROBI constraint solver.[3] Since solving integer linear programs is exponentially hard in the general case, we follow Martins et al. (2009) and solve the relaxation of the problem, which means dropping the integer constraint on the variables. If the solver outputs an integer solution, as it often does, then this solution is guaranteed to be optimal. If the solution is not integer, we use the first-order formulation as defined above to project the fractional solution to integer space using the fractional values of the variables as arc weights. To further reduce the complexity of the problem, the graphs are pruned before parsing by choosing for each token the ten most probable heads using a linear classifier that is not restricted by structural requirements. We also use an arc filter that blocks arcs that do not occur in the training data based on the label of the arc and the part-of-speech tag of the head and the dependent.

The weight vector is trained with loss-augmented passive-aggressive online learning (Crammer et al. 2003) and averaged afterwards (Freund and Schapire 1999). The feature model was modeled after mate parser's but is not identical. The full set is described in Appendix A.

6.1.3 Morphosyntactic Constraints

We now present a set of constraints for each language that is added to the integer linear program that the parser solves. The idea follows and extends the proposal in Riedel and Clarke (2006). They augment their decoder with several linguistically motivated constraints and show improvements over the unaugmented decoder. Unlike Riedel and Clarke (2006), we have the concise formulation of Martins et al. (2009) in the background, which makes the parser much more efficient.

[3] `www.gurobi.com`, version 4.0.

The additional constraints implement linguistic restrictions on the morphosyntactic structures of Czech, German, and Hungarian. The purpose of these constraints is to restrict the search space of the parser to those dependency trees that comply with the linguistic constraints. Note that the linguistic constraints are independent of the statistical model that drives the parser towards the optimal solution. They thus act as a filter that filters out the linguistically implausible solutions (according to the constraint set).

Linguistically, the constraints implement knowledge about the relation between morphology and syntax by imposing restrictions on the arcs based on the morphological information on the words. In order to do this, we require an underspecified symbolic representation of morphological information over which the constraints can be defined. For example, the case feature of a word is represented as a set of binary variables M for which 1 signals the presence of a particular value and 0 signals its absence. For Hungarian, we only model the different case values, which leads to one binary variable for each of the values. For Czech and German, we include the gender and the number features which then gives, for each case marked word, a binary variable for every combination of the case, number, and gender values.

The values of the morphological indicator variables are specified before parsing by annotating the data sets with morphological dictionaries.[4] If a certain feature value is excluded by the analyzers, the value of the indicator variable for this feature is fixed at 0, which then means that the decoder cannot set it to 1. This way, all morphological values that cannot be marked by the form of the token (according to the morphological analyzer) are blocked and thereby also all parser solutions that depend on them. Words unknown to the analyzers are left completely underspecified so that each of the possible values is allowed. The symbolic, grammar-based preannotations thus set some of the morphological indicator variables to 0 where the word form gives enough information while leaving other variables open to be set by the parser, which can use syntactic context to make a more informed decision. Form syncretism is thus explicitly modeled by leaving more than one variable unspecified.

As an example, take the word form *Hund* (*dog* in German). *Hund* can be nominative singular masculine, dative singular masculine, and accusative singular masculine. The

[4] Czech: http://ufal.mff.cuni.cz/pdt/Morphology_and_Tagging/Morphology/index.html
German: Schiller (1994)
Hungarian: Trón et al. (2006)

three variables representing these three morphological descriptions are thus left unspecified, whereas all other values are fixed to 0, for example nominative plural neuter. An non-syncretic word form like *Hundes*, which can only be genitive singular masculine, has only one variable that is not fixed to 0.

Case Licensing

We now present several constraints that model the relation between morphology and syntax. The first constraint models the mapping between a function label and the case value that it requires. Equation (6.24) shows an example of a case licensing constraint for the *DAT* label in Hungarian. A dependent d cannot be attached to a head with label *DAT* if its morphological indicator variable for dative case (m_d^{dat}) is zero.

$$\sum_{a_{\langle h,d,DAT\rangle} \in A_d^{in}} a_{\langle h,d,DAT\rangle} \leq m_d^{dat} \quad \text{for all } d \in T \tag{6.24}$$

Agreement

The second constraint models the morphological agreement between dependents and their heads in noun phrases (Equations (6.25) and (6.26)), for instance determiners and adjectives with their head noun in the noun phrases in Czech and German. In the treebanks, the relation is marked by *NK* for German and *Atr* for Czech.[5] The constraints relate the morphological indicators for an adjective and a noun in the following way: As long as there is no arc ($a_{\langle h,d,NK\rangle}$ is zero) between the adjective (d) and the noun (h) the two constraints allow for any value in the morphological indicator variables of both words. If the arc is established ($a_{\langle h,d,NK\rangle}$ is set to 1), the two constraints form an equivalence forcing

[5] In German and mostly also in Czech, if an adjective is attached to a noun by *NK* (or *Atr*), they stand in an agreement relation. This allows us to bind the agreement constraint to these function labels (and to the involved part-of-speech tags). However, in a (very) small number of cases in the Czech treebank, an adjective is attached to a noun by *Atr* but there is no agreement. This happens for example if the adjective is actually the head of another noun phrase that stands in attributive relation (*Atr*) to the noun. The *Atr* label was not meant to mark agreement relations, it just happens to coincide for most of the cases. But it might be worth considering whether morphosyntactic relations like agreement should be represented explicitly in syntactic treebanks.

all the morphological indicators to agree on their value, i.e., to be both 1 or both 0. We additionally require every word to have at least one morphological indicator variable set to 1. Thus, if there is no solution to the equivalence, the arc between the adjective and the noun cannot be established with this function label.

$$m_h^{dat-pl-fem} \leq m_d^{dat-pl-fem} + 1 - a_{\langle h,d,NK\rangle} \tag{6.25}$$
$$m_h^{dat-pl-fem} \geq m_d^{dat-pl-fem} - 1 + a_{\langle h,d,NK\rangle} \tag{6.26}$$

Uniqueness

For the third constraint, Equation (6.27) shows a constraint that was already proposed by Riedel and Clarke (2006). It models label uniqueness by forcing label l to appear at most once on all the dependents of a head h. The fact that verbs have at most one subject can be modeled directly by this constraint. It thus defines a very conservative version of subcategorization. We define a set of unique labels L_u for each of the treebanks by counting the number of times a label occurs more than once on the dependents of a given token. Labels that occurred more than once only for a small number of times were still included into L_u if it made sense linguistically. The constraint is applied to every word in a sentence.

$$\sum_{a \in \{\, a_{\langle d,h,l\rangle} \mid l \in L_u,\ a_{\langle d,h,l\rangle} \in A_h^{out} \,\}} a \leq 1 \quad \text{for all } h \in T_0 \tag{6.27}$$

Parsers like mate parser, which use the decoder by Carreras (2007), have by design no means to enforce such a constraint for two reasons: First, the decoder collects left and right dependents independently of each other. Second, the feature model never sees two arc labels together and thus cannot learn a relation between two labels.

To illustrate this, Table 6.1 shows the number of times a grammatical function occurs more than once per head in the treebank (TRBK) and how often it was annotated by the mate parser models used in Chapter 5. While doubly annotated argument functions almost

never appear in the treebank, the parser frequently annotates them because it has no way of checking whether the function has already been annotated (see also Khmylko et al. 2009).

	Czech			German			Hungarian		
	TRBK	GOLD	PRED	TRBK	GOLD	PRED	TRBK	GOLD	PRED
subjects	0	772	1,723	44	1,170	2,403	0	586	670
predicates	7	174	190	6	92	108	1	17	19
obj (dat.)	0	28	46	0	33	46	0	9	5
obj (acc.)	22	284	602	2	364	912	0	182	189

Table 6.1: Number of times a core grammatical function was annotated more than once in the treebank (TRBK), by a mate parser model using gold morphology (GOLD), and by one using predicted morphology (PRED).

Other Constraints

In addition to the constraints shown here, the parser uses several other constraints that deal with more specific phenomena, e.g., a constraint that forbids accusative objects in passive constructions of German. Other constraint apply uniqueness over analytic verb forms so that the parser cannot annotate two subjects even though one would be attached to the auxiliary and one to the main verb.

Constraint Interaction

Each individual constraint already reduces the choices that are available to the parser. However, they exclude additional incorrect analyses by interaction. Figure 6.3 illustrates the interaction between the three constraints for the German sentence *den Mädchen helfen Frauen* meaning *women help the girls*.

Each individual word displays a high degree of syncretism. But when the syntactic structure is decided, many options mutually exclude each other. The agreement constraints in Equations (6.25) and (6.26) disambiguate *den Mädchen* for dative plural feminine. Case licensing (Equation (6.24)) then blocks the subject and direct object labels for *Mädchen* because they cannot be marked by dative case, and Equation (6.27) ensures uniqueness

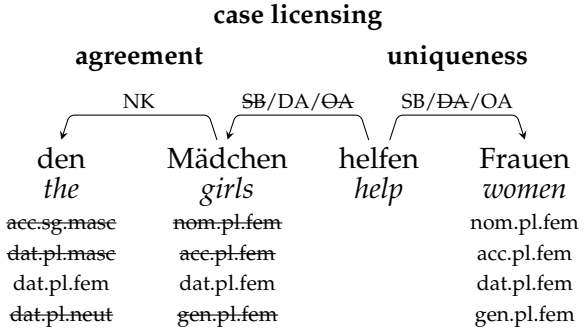


Figure 6.3: Constraint interaction for the German sentence *den Mädchen helfen Frauen* meaning *women help the girls*.

by blocking the dative label for *Frauen*. The parser now has to decide whether *Frauen* is subject, accusative object, or something else completely. The constraints are applied online during the decoding process. If the statistical model would strongly prefer *Mädchen* to be accusative object, the parser could label it with *OA*. However, in that case, it would not be able to establish the *NK* label between *den* and *Mädchen*, since the agreement constraint would be violated. Thus, the constraints filter out incorrect solutions but the decoder is still driven by the statistical model.

6.2 Experiment

To test the parser and the morphosyntactic constraints, we apply it to the same data sets as were described in Section 4.1.1, Chapter 4. The data sets are annotated with syntax via 5-fold jackknifing using the same preprocessing as in the previous experiment. We train two models for the constraint parser, one without constraints and one with constraints (denoted C and NO-C, respectively). For the C model, constraints are applied during training and testing. We compare these two models to the mate parser models from Chapter 4. The preprocessing is identical between the three models except that we only train models on automatically predicted morphology.

6.2.1 Results

Table 6.2 shows the overall performance of the two models compared to the performance of mate parser repeated from Table 4.4 in Chapter 4. Recall that GOLD-M, PRED-M, and NO-M are the models that use gold-standard, automatically predicted, and no morphological information. The model that compares directly to the constraint parser is the PRED-M model, as the features are most similar. The results in Table 6.2 show that the basic constraint parser performs slightly below mate parser (PRED-M), but the model that uses the linguistic constraints performs slightly better or equal for Czech and German.

	Czech		German		Hungarian	
	LAS	UAS	LAS	UAS	LAS	UAS
GOLD-M	82.49	88.61	91.26	93.20	86.70	89.70
PRED-M	81.41	88.13	89.61	92.18	84.33	88.02
NO-M	79.00	86.89	89.18	91.97	78.04	86.02
ILP NO-C	81.69	88.09	89.30	91.98	84.01	87.12
ILP C	81.91*	88.18	89.93*	92.25	84.35	87.39

Table 6.2: Overall performance of mate parser and the ILP parser. Results for German and Hungarian are without punctuation. $*$ marks statistical significance with respect to the respective PRED-M model, measured with a two-tailed t-test, $\alpha = 0.05$.

Performance on Grammatical Functions

Table 6.3 shows the performance on the argument functions for the unconstrained (*no-c*), constrained (*c*) ILP models, and the PRED-M models of mate parser. In this evaluation, we only evaluate tokens that actually carry case morphology (cf. Chapter 4). For each language, the best results are bold-faced. In addition to the results for the different argument functions, a total score is computed over all argument functions (all arg funcs) and another is computed over all tokens that are not included in the first set (all other). The latter illustrates the performance of the parsing models on the functions that are not marked by case morphology.

For each language, we get the same basic picture: while the unconstrained ILP model performs slightly worse than (German, Hungarian) or equally well as (Czech) the PRED

	Czech			German			Hungarian		
	NO-C	C	PRED	NO-C	C	PRED	NO-C	C	PRED
subject	85.41	**87.23***	85.46	90.02	**92.91***	90.59	85.05	**87.67***	86.53
predicate	87.13	**90.09***	87.11	72.86	**80.70***	74.33	74.16	**78.88***	74.79
obj (nom)	47.48	**53.19***	38.74	–	–	–	–	–	–
obj (gen)	70.15	**72.54**	70.27	31.41	**42.98**	34.26	–	–	–
obj (dat)	79.99	**80.42**	79.54	65.21	**77.78***	71.05	75.33	**77.92***	73.49
obj (acc)	84.27	**86.79***	84.12	83.74	**87.96***	84.86	91.96	**93.21***	92.53
obj (instr)	67.36	**68.76**	65.02	–	–	–	–	–	–
all arg funcs	84.33	**86.37***	84.21	86.27	**90.11***	87.24	86.87	**89.04***	87.78
all other	81.37	**81.37**	81.05	89.79	89.88	**89.98**	82.73	82.86	**83.43**

Table 6.3: Parsing results for the unconstrained (NO-C) and the constrained (C) ILP models, and mate parser (PRED) in terms of f-score for core grammatical functions marked by case. We omit locative objects in Czech, and second accusative objects in German because of their low frequency. $*$ marks statistically significant differences when comparing the performance on a grammatical function for the C model to the PRED-M model (α = 0.05, two-tailed t-test).

model of the mate parser, the constrained ILP model clearly outperforms both on the argument functions. On each of them, the constrained ILP model improves over the other two models, raising the score by between one percentage point for e. g. subjects in Hungarian up to 7 percentage points on dative objects in German (compared to the PRED model). We can see that in general, the improvements seem to be higher on the more infrequent arguments like dative objects and predicates than on the frequent arguments like subject or accusative object. However, it is not the case that the performance of one of the infrequent functions suddenly surpasses the performance of a more frequent function. Those two effects are to be expected since the ILP parser is still a data-driven parser. The constraints support it by excluding morphosyntactically incorrect analyses but they do not resolve ambiguous cases. These are still decided by the statistical model.

6.3 Discussion

The experiments in this chapter are meant to test whether the explicit modeling of morphosyntactic constraints can improve a statistical parser. While the overall parse quality is

only marginally better compared to a state-of-the-art baseline without such constraints, a focused analysis that evaluates the argument functions targeted by the constraints reveals that the model indeed becomes better for those functions. We also again see that the improvements are smallest for Hungarian. Evidently, there is less need for explicit constraints when parsing Hungarian.

The linguistic constraints that we used implement rules that are part of the grammar of Czech, German, and Hungarian. We implement them as hard constraints on the output structures of the parser and not as features in the statistical feature model even though the feature model of the parser already includes features that relate the morphology of dependent and head with each other. Ideally, the statistical model would learn the morphosyntactic rules of a language on its own and predict structures that adhere to them. The fact that the constrained model outperforms the one without constraints shows that this is not necessarily the case.

Goldberg and Elhadad (2013) take a similar approach as we do and model agreement for Hebrew as a filtering process that is applied on an n-best list output by their lattice parser. In their parser like in ours, agreement is modeled as hard constraints and is decoupled from the actual statistical model. As they put it: *[...] agreement is a part of the parser and not of the grammar* (Goldberg and Elhadad 2013: 146). However, while they find improvements due to the filter they are very small. Their analysis shows that this is mostly because the statistical model already makes a correct decision in most cases. Other reasons include the restricted number of analyses in the n-best list and the fact that agreement alone does not always create the correct parse. The difference in our parser is that the constraints are already in place during parsing thus obviating n-best lists. But more crucially, the constraints are also applied during training so that the statistical model never actually makes agreement mistakes during training.

Hard constraints seem to be an attractive solution because the parser cannot ignore them and they cannot be overruled by stronger features. The uniqueness constraint for example models something that statistical parsing models usually do not model on its own. For example, Carreras' decoder (Carreras 2007) which is used in mate parser derives left and right dependents of a word independently of each other. It also never relates two arc labels with each other. The same holds for the statistical model underlying the parser used in this chapter. Such design decisions are made for efficiency, but they prevent the statistical model to learn that some labels occur at most once per head. The uniqueness

constraint does not add this knowledge to the statistical model but it still forces the parser to comply with it. Similarly, the agreement constraints enforce morphological agreement along a single arc. We have seen in Chapter 4 that a statistical model can learn these rules but that errors in the preprocessing cause the parser to make mistakes nonetheless. Also here, hard constraints can avoid the mistakes that are caused by faulty information.

The relation between the statistical model and the constraints can be seen as two components of a grammar where the constraints represent the hard rules in a language that always need to hold whereas the statistical model models preferences and frequency effects. Besides a well-formed tree structure, this parser additionally guarantees some form of grammaticality with respect to case marking and agreement. Rules and preferences are clearly both part of language and it is an open question how they should be modeled in a parser. Krivanek and Meurers (2011) compare a data-driven Nivre et al. (2007) and a rule-based dependency parser Foth and Menzel (2006) on learner and newspaper corpora and find that while the former is better on modifier functions (e. g. PP-attachment), the latter performs better on argument functions. Their explanation is that while the data-driven parser has access to lots of data and can pick up statistical effects in the data like semantic or selectional preferences, the rule-based parser has access to deep lexical and grammatical information and is thus able to model argument structure in a better way. The results of our experiment seem to support the claim that rules can help with argument structure, which also means that argument structure is still a problem for standard parsing models.

The big disadvantage of hard rules, however, is their brittleness. The constraints that we develop here are purposefully formulated in a very defensive way to account for this. For example, the case licensing constraint does not say that a word in dative case is either indirect object or an argument of a preposition etc., but it merely states that if a word form cannot be dative, the word cannot fill the role of an indirect object. It is very difficult to find rules that always hold in a language without exception in the first place, and these rules can always be violated, may it be on purpose or by mistake. It would be much more elegant if the statistical model could learn the relation between morphology and syntax on its own. The parser in this chapter does not actually model the morphological features of the words in the sentence but only uses it to restrict the search space. In the next chapter, we propose a parser that models morphology and syntax jointly such that the statistical model itself must decide on the best combination of syntactic structure and morphological features.

Chapter 7

Graph-based Lattice Dependency Parsing

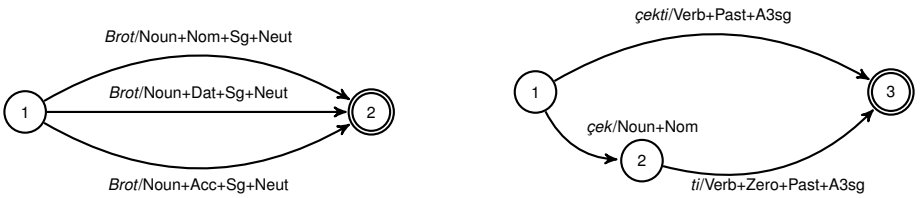
The last chapters have presented motivation and justification for jointly modeling morphological and syntactic analysis and we showed how morphosyntactic rules can be integrated into a parser by constraining the output structures. While the constraint parser in the previous chapter models the interaction between morphology and syntax, it does so without involving the statistical model of the parser. Because of this, the constraint parser cannot actually predict the morphological features of words. It can restrict morphological ambiguity by percolating the morphosyntactic rules down to the word level and in the best case, only one option remains. However, it cannot disambiguate between options that are both compatible with the morphosyntactic constraints.

In this chapter, we develop a parser that jointly predicts the morphological features of words as well as their syntactic structure in the sentence.[1] The statistical model predicts the best combination of syntactic structure and morphological features and thus truly learns a joint distribution. This model cannot suffer from error propagation because it does not rely on information from another processing step. We use this parser to empirically demonstrate that joint modeling is superior to a pipeline parser that separates

[1]The content of this chapter is published in Seeker and Çetinoğlu (2015). My contribution to this joint work with Özlem Çetinoğlu consists of the formulation, development, and testing of the parser as well as participating in the design of the evaluation.

morphological and syntactic analysis.

To facilitate the joint prediction, the parser operates on morphological lattices, which represent the potential morphological analyses for a given word or sentence. Lattices are often used in NLP to compactly represent ambiguous analyses for a given input, for example multiple interpretations of a sound signal in speech recognition (Ortmanns et al. 1997, Jurafsky and Martin 2009) or candidate output sentences in machine translation (Ueffing et al. 2002). Figure 7.1a shows a representation of the German word *Brot* (bread). Each path through the lattice encodes one morphological interpretation of the word form. Chaining several such word lattices together creates a lattice for a sentence. Choosing a path through such a sentence lattice means to select a morphological interpretation for each word.



(a) The German word *Brot*. **(b)** The Turkish word *çekti*.

Figure 7.1: A lattice representation of morphological ambiguity.

However, in this chapter we do not work on German but instead extend the joint model beyond the prediction of morphological features and apply it to Turkish and Hebrew. Recall from Chapter 3, Section 3.3, that in these languages words can be segmented into meaningful units. For Turkish, these units correspond to inflected stems or derivation suffixes and are called *Inflectional Groups* (IG). The segmentation thus represents the derivational genesis of the word. For Hebrew, the segments represent single morphemes that may belong to different syntactic contexts. For illustration, Figure 7.1b shows a word lattice for the Turkish word *çekti* that encodes two possible segmentations of the word. The upper path encodes an interpretation as an inflected verb stem that would be translated as *it pulled*. The lower path represents an interpretation as a noun that was derived into a verb with the copula suffix. This interpretation means *it was a cheque*.

Selecting a path in a lattice such as the one shown for Turkish thus means to not only predict the morphological features of the words but also to decide the segmentation of words into smaller units. As discussed in Chapter 3, it is these smaller units over which we

want to predict a syntactic structure because syntactic relations hold between the smaller units rather than the words themselves. The task of the joint parser is therefore to select a path through the lattice while at the same time predicting a syntactic structure over the segments of the selected path. Solving these three tasks, i.e., segmentation, morphological analysis, and parsing, jointly allows the model to use information from all three sources. Syntactic information can support choosing the correct segmentation (cf. Figure 3.6) and morphological analysis. At the same time, segmentation defines the tokens over which the syntactic structure is built. In a pipeline model, segmentation mistakes can never be repaired. The joint model instead can learn how segments, morphological features, and dependency arcs relate to each other without being restricted by decisions that it cannot change. Such a model is then trivially applicable also to less complex lattices as the one shown for German in Figure 7.1a.

Building a parser for the joint task is challenging because the search space for this parser is huge. Not only does it have to find the best dependency tree among an exponentially large number of trees, the number of trees is additionally increased because now they can differ with respect to the segmentation and morphological features on the words as well. We solve this problem by using a dual decomposition approach (Rush et al. 2010, Koo et al. 2010, Martins et al. 2010b). The joint problem is decomposed into smaller subproblems, each of which can be solved efficiently with known algorithms. An iterative optimization algorithm then solves the subproblems repeatedly, adjusting weights after each round to enforce agreement between them. Decomposing the joint problem in this way keeps the model tractable as the complexities of the subproblems are combined additively instead of being multiplied.

Almost as challenging as building the parser is its evaluation. Since the parser predicts the segments over which the parser predicts the dependency tree, they may be different from the segments in the gold standard tree. Simple attachment scores therefore cannot be applied anymore. Drawing on previous work for Hebrew, Turkish, and Chinese, we discuss and apply different evaluation metrics for evaluating the joint task.

The parser presented in this chapter is the first graph-based dependency parser that solves the full joint task of segmentation, morphological analysis, and parsing. Furthermore, the results presented for Turkish are the first results on full lattice parsing for this language.

The chapter is organized into four main parts: Section 7.1 introduces the concept of lattice

parsing and presents the formal description of the parser. Section 7.2 discusses the problem of evaluating a lattice parser and the approaches that have been put forward to do so. Section 7.3 presents an empirical evaluation of the parser on Turkish and Hebrew and Section 7.4 finally compares the parser to related work for joint models in parsing.

Terminology. In the rest of the chapter, we will use the term *word* to refer to input strings separated by whitespace, and the term *token* to refer to the smallest unit of processing as in the output of a tokenizer. For Turkish, a token corresponds to an *Inflectional Group*, whereas for Hebrew, it refers to a morpheme. Thus, a Turkish or Hebrew input sentence can have seven words, but the number of tokens in that sentence is unknown and must be predicted by the parser.

7.1 Lattice Parsing

Lattices are directed acyclic graphs with one defined initial state and one defined final state. Figure 7.2 shows the lattice representations of a Turkish and a Hebrew word. The initial state is the state that has no incoming transition (shown as the left-most state) and the final state is the state with no outgoing transitions (shown as the right-most state). Each path from the initial state to the final state represents one possible analysis of the input word and there is exactly one path per analysis.

(a) The Turkish word *ekmek*. **(b)** The Hebrew word *hneim*.

Figure 7.2: Lattice representations of morphologically ambiguous words in Turkish and Hebrew. The Turkish example is from Seeker and Çetinoğlu (2015) and the Hebrew example is from Goldberg and Tsarfaty (2008).

Given a vocabulary $\mathcal{X}$, a morpheme inventory $\mathcal{T}$, and a word $x \in \mathcal{X}$, a morphological lexicon $M(x) \subseteq \mathcal{T}^+$ maps words to a set of sequences over $\mathcal{T}$. These sequences of morphemes are the morphological analyses of word x. The elements of $\mathcal{T}$ can simply

be the surface forms but may also include information about part-of-speech tags and morphological features as shown in Figure 7.2. The set of morphological analyses for a word x can be represented by a lattice. A path $p \in M(x)$ through the lattice from the start state to the final state then represents one possible morphological analysis of the word.

The lattice representation for words can be extended to sentences by concatenating word lattices into what Cohen and Smith (2007) call a "sausage lattice". An example for such a lattice is shown in Figure 7.3 for the Turkish sentence *ekmek aldım* (I bought bread).[2] A sentence lattice represents all possible morphological analyses of the given sentence. Since it is a concatenation of the word lattices, the paths converge after each word and there cannot be transitions that skip a word boundary. A sentence lattice therefore represents the Cartesian product over the analyses of the individual words.

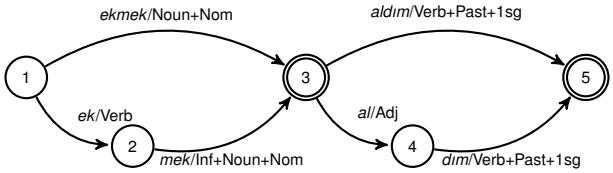


Figure 7.3: A sentence lattice for the Turkish sentence *ekmek aldım* (I bought bread). Double circles mark word boundaries (i.e., whitespace). Example taken from Seeker and Çetinoğlu (2015).

It is straightforward to extend M to sentences: Given a sentence $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_n \rangle$, the morphological lexicon maps sentences to sets of sequences of sequences of morphemes $M(\boldsymbol{x}) \subseteq (\mathcal{T}^+)^+$, which is the same as $M(\boldsymbol{x}) \subseteq \mathcal{T}^+$. $M(\boldsymbol{x})$ can be represented as a sentence lattice as in Figure 7.3 and the elements of $M(\boldsymbol{x})$ are paths through the sentence lattice.

Lattice parsing is the task of predicting a parse tree given a sentence lattice (Chappelier et al. 1999, Hall 2005). During parsing, a lattice parser must select one of the paths in the lattice, thereby deciding on the number of tokens in the sentence and their morphosyntactic properties as part of the parsing process. A lattice parser run on a morphological lattice as shown in Figure 7.3 thus predicts segmentation, morphology, and syntax of this sentence simultaneously.

Lattice parsing subsumes two interesting special cases: (1) The lattice encodes multiple paths but all paths are of equal length. In this case, the lattice does not encode any

[2] `https://www.youtube.com/watch?v=tAboni_Mqv4`

ambiguity about segmentation but the morphological properties of the tokens remain underspecified. Languages without segmentation ambiguity can be represented by these lattices. (2) There is only a single path in the lattice. In this case, there is no uncertainty about segmentation or morphological properties of the tokens at all. This is the standard case used in pipeline architectures.

7.1.1 A Graph-based Lattice Dependency Parser

In this section, we present the construction of the lattice parser. We cast the problem of lattice parsing as a constrained optimization problem, in which we seek the highest-scoring dependency tree under the constraint that the tokens that it spans over form a consecutive path through the lattice. Since solving this constrained problem is difficult, we use a dual decomposition approach (see Chapter 2, Section 2.2 for a description of the underlying idea of dual decomposition), in which we decompose the task into two subtasks. The first subtask finds a consecutive path through the sentence lattice. The second subtask computes a spanning tree over the lattice. To these two subtasks, we add two constraints that ensure that the path that is found by the first task coincides with the tokens that the second task predicts to be part of the final solution. A dual decomposition algorithm then finds the optimal solution to this problem by solving the subproblems repeatedly until all constraints are fulfilled.

As said before, we will refer to the minimal unit of parsing as a *token*. In the input lattices for the parser, a token corresponds to a single transition between two states. For convenience, we define the set of tokens T to hold all tokens represented in a (sentence) lattice. Tokens represent IGs in the Turkish treebank and morphemes in the Hebrew treebank.

We assume two different structures, lattices and dependency trees. Dependency trees are represented as directed acyclic trees with a special root node (ROOT); lattices are directed acyclic graphs as defined above. For dependency trees, we will use the terms *node* and *arc* to refer to the vertices and the edges between the vertices, respectively. Tokens are represented as nodes in the dependency tree. For lattices, we use the terms *state* and *transition* to refer to the vertices and their edges in the lattice. Contrary to dependency trees, tokens are represented as transitions in the lattice.

Finding The Path. A token bigram in a lattice $M(\boldsymbol{x})$ is a pair of two transitions $\langle t, t' \rangle$, such that the target state of t in $M(\boldsymbol{x})$ coincides with the source state of t' in $M(\boldsymbol{x})$. A chain of overlapping bigrams that starts from the initial state and ends in the final state forms a path through the lattice. We represent the ROOT token as the first transition, i.e., a single transition that leaves the initial state of the lattice.

Given a lattice $M(\boldsymbol{x})$, we define the index set of token bigrams in the lattice to be

$$S := \{ \langle t, t' \rangle \mid t, t' \in T, \text{target}(t) = \text{source}(t') \}. \tag{7.1}$$

For later, we furthermore define the set of bigrams that have t at the second position:

$$S_{|t} := \{ \langle k, t \rangle \mid \langle k, t \rangle \in S,\ k \in T \} \tag{7.2}$$

A consecutive path through the lattice is defined as an indicator vector

$$\boldsymbol{p} := \langle p_s \rangle_{s \in S} \tag{7.3}$$

where $p_s = 1$ means that bigram s is part of the path, otherwise $p_s = 0$. We define $\mathcal{P}$ as the set of all well-formed paths, i.e., all paths that lead from the initial to the final state.

We use a linear model that factors over token bigrams. Given a scoring function f_{P} that assigns scores to paths, the path with the highest score can be found by

$$\begin{aligned} \hat{\boldsymbol{p}} &= \arg\max_{\boldsymbol{p} \in \mathcal{P}} f_{\text{P}}(\boldsymbol{p}) \\ \text{with} \quad f_{\text{P}}(\boldsymbol{p}) &= \sum_{s \in S} p_s\ \boldsymbol{w} \cdot \phi_{\text{SEG}}(s) \end{aligned} \tag{7.4}$$

where ϕ_{SEG} is the feature extraction function for token bigrams. Given a weight vector, the highest-scoring path through the lattice can be found with the Viterbi algorithm. In Section 7.3, we also use the bigram model as a standalone disambiguator for morphological lattices to find the highest-scoring path in a lattice.

Finding The Tree. We define the index set of arcs in a dependency tree as

$$A := \{ \langle h, d, l \rangle \mid h \in T, d \in T - \{\text{ROOT}\}, l \in L, h \neq d \} \tag{7.5}$$

with L being a set of dependency relations. A dependency tree is defined as an indicator vector

$$\boldsymbol{y} := \langle y_a \rangle_{a \in A} \tag{7.6}$$

where $y_a = 1$ means that arc a is in the parse, otherwise $y_a = 0$. We define $\mathcal{Y}$ to be the set of all well-formed dependency trees (projective and non-projective).

We assume an arc-factored model as commonly done in dependency parsing (McDonald et al. 2005, Koo et al. 2010). Given a scoring function f_{T} that assigns scores to trees, the problem of finding the highest scoring tree is defined as

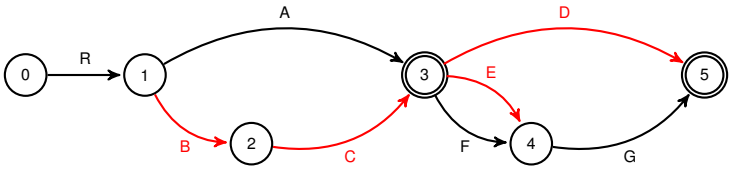
$$\begin{aligned} \hat{\boldsymbol{y}} &= \arg\max_{\boldsymbol{y} \in \mathcal{Y}} f_{\mathrm{T}}(\boldsymbol{y}) \\ \text{with} \quad f_{\mathrm{T}}(\boldsymbol{y}) &= \sum_{a \in A} y_a \, \boldsymbol{w} \cdot \phi_{\mathrm{ARC}}(a) \end{aligned} \tag{7.7}$$

where ϕ_{ARC} is the feature extraction function for single arcs and $\boldsymbol{w}$ is the weight vector. We follow Koo et al. (2010) and use the Chu-Liu-Edmonds algorithm (CLE) to find the highest-scoring parse (Chu and Liu 1965, Edmonds 1967). CLE enforces the tree properties of the output, i.e., acyclicity and exactly one head per token. Note that the algorithm includes all tokens of the lattice into the spanning tree, not just some tokens on some path.

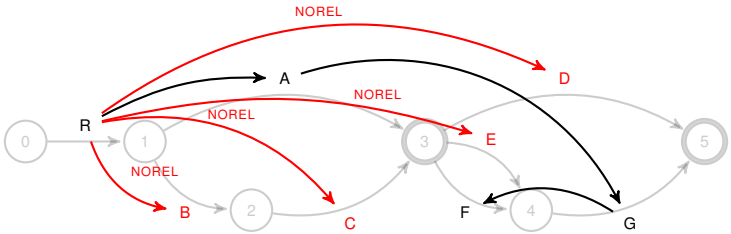
Agreement Constraints. To make the path and the parse tree agree with each other, we introduce an additional dependency relation NOREL into L, the set of dependency relations. We define a token that is attached to ROOT with relation NOREL to be *not on the path through the lattice*. These arcs are not scored by the statistical model, they simply serve as a means for CLE to mark tokens as not being part of the solution by attaching them to ROOT with this relation.[3]

When the bigram model finds a path through the input lattice, it effectively partitions the set of tokens T into two sets, namely the tokens that are on the path and the tokens that are not (see Figure 7.4a). By means of the NOREL label, the arc-factored model is able to mark a token as being part of the solution tree or not. The tokens that are part of the solution should form a dependency tree, whereas the other ones should simply be dependents of ROOT. Currently, however, a tree token can be a dependent of a non-tree

[3]The parser can predict the NOREL label only on arcs attached to ROOT.



(a) The bigram model partitions the tokens into two sets. The tokens on the path, $\{R, A, F, G\}$, and the tokens not on the path, $\{B, C, D, E\}$.



(b) The tree model also partitions the tokens into two sets. The tokens in the output tree, $\{R, A, F, G\}$, and the tokens that are not part of the output tree, $\{B, C, D, E\}$. Due to the first agreement constraint, there can be internal structure among the tokens in the first set only.

Figure 7.4: The bigram model and the tree model both partition the set of tokens into two sets. The second agreement constraint ensures that the two partitionings coincide.

token. To prevent this, we introduce a constraint that disallows tokens that are attached to ROOT with NOREL to have dependents on their own. The constraint is implemented as an *implication factor* ($\Longrightarrow$, Martins et al. 2015). It states that an active NOREL arc for a token h implies an inactive arc for all arcs having h as head. There is one such constraint for each possible arc in the parse.

$$y_{\langle \text{ROOT}, h, \text{NOREL} \rangle} \implies \neg y_{\langle h, d, l \rangle} \quad \text{for all} \quad \langle h, d, l \rangle \in A, h \neq \text{ROOT}, l \neq \text{NOREL} \tag{7.8}$$

By introducing the constraint in Equation (7.8), the CLE cannot mix tree tokens and non-tree tokens and thus cleanly partitions T into two sets, namely tree tokens and non-tree tokens (see Figure 7.4b). Simultaneously, it predicts a dependency tree over the tree tokens. The final step is now to ensure that the partitioning of T by selecting a path through the lattice is identical to the partitioning of T by computing the spanning tree. This can be achieved with a second constraint, which is defined over token bigrams and arcs. It

states that for a token t, either one of its bigrams[4] or its NOREL-arc must be active. It is implemented as an *XOR factor* ($\oplus$, Martins et al. 2011b) and there is one such constraint for each token in the lattice.

$$\bigoplus_{s \in S_{|t}} p_s \ \oplus y_{\langle \text{ROOT},t,\text{NOREL} \rangle} \quad \text{for all } t \in T \tag{7.9}$$

By means of the constraint in Equation (7.9), both subtasks have to agree on the path through the lattice. The Viterbi algorithm ensures that the solution will be a coherent path while the CLE predicts a dependency tree over this path. All tokens that are not on the path are discarded before the parser returns the parse tree.

7.1.2 Inference

The objective function of the lattice parser is

$$\underset{\boldsymbol{y} \in \mathcal{Y}, \boldsymbol{p} \in \mathcal{P}}{\arg\max} f_{\text{T}}(\boldsymbol{y}) + f_{\text{P}}(\boldsymbol{p}) \tag{7.10}$$

subject to the two agreement constraints in Equations (7.8) and (7.9).

We use *Alternating Directions Dual Decomposition* or *AD*^{3} (Martins et al. 2011a)[5] to find the optimal solution to this constrained optimization problem. AD^3 is an approximate algorithm that finds the optimal solution in an iterative fashion. If the algorithm converges, the solution is guaranteed to be optimal. AD^3 can efficiently handle first-order logic constraints, which we use to implement the agreement constraints. Similar approaches, where such constraints are used to ensure certain properties in the output structures, have been used, e.g., in semantic parsing (Das et al. 2012), compressive summarization (Almeida and Martins 2013), and joint quotation attribution and coreference resolution (Almeida et al. 2014).

CLE can be implemented such that its worst case complexity is $\mathcal{O}(T^2)$, while the Viterbi

[4] The lattice structure ensures that always only one of the bigrams with the same token in second position can be part of a path.

[5] http://www.ark.cs.cmu.edu/AD3/

algorithm needed to find the path is of worst case complexity $\mathcal{O}(QT^2)$, where Q is the number of states in the lattice. Instead of combining these two problems directly, which would multiply their complexity, AD^3 combines them additively, such that the complexity of the parser is $\mathcal{O}(k(T^2 + QT^2))$ with k being the number of iterations that AD^3 is run.

7.1.3 Second-Order Parsing

The formulation of the parser given so far describes a first-order dependency parser, where features are extracted on single arcs only. However, higher-order models that consider sibling or grandparent information are known to be superior to first-order models (McDonald and Pereira 2006, Carreras 2007, Koo and Collins 2010).

Koo et al. (2010) present a formulation for a dependency parser that uses head automata (Alshawi 1996, Eisner 2000) to model second-order factors, i.e., sibling and grandparent features (see also Martins et al. 2013). In this model, a dependency structure is decomposed into smaller graphs, two for each token in the sentence. The smaller graphs model the left and right direct dependents for each token, independent of each other and, crucially, independent of the other tokens in the sentence.

Finding the optimal left and right dependents for each token independently of the other tokens does however not guarantee that the partial solutions agree with each other and form a coherent dependency tree. Therefore, Koo et al. (2010) use CLE to ensure proper spanning trees in the output of the parser. They use dual decomposition to find the optimal dependency tree defined by the head automata and CLE under the constraint that both agree on the structure.

We adopt the formalization given in Martins et al. (2013) extending it slightly to include arc labels. Let

$$A_h^{in} := \{\, \langle g, h, l\rangle \mid g \in T, l \in L, g \neq h \,\} \tag{7.11}$$

$$A_h^{out} := \{\, \langle h, d, l\rangle \mid d \in T - \{\text{ROOT}\}, l \in L, h \neq d \,\} \tag{7.12}$$

be the sets of incoming and outgoing arcs of token h, respectively. The set of outgoing arcs

is further split into the set of outgoing arcs to the right and to the left of h.

$$A^{out}_{h,\rightarrow} := \{\, \langle h, d, l\rangle \,|\, d \in T - \{\text{ROOT}\}, l \in L, h < d \,\} \tag{7.13}$$

$$A^{out}_{h,\leftarrow} := \{\, \langle h, d, l\rangle \,|\, d \in T - \{\text{ROOT}\}, l \in L, h > d \,\} \tag{7.14}$$

The set of candidate arcs for a right (left) grandparent-sibling head automaton (Koo et al. 2010) is then

$$A^{\text{GSIB}}_{h,\rightarrow} = A^{out}_{h,\rightarrow} \cup A^{in}_{h} \tag{7.15}$$

$$A^{\text{GSIB}}_{h,\leftarrow} = A^{out}_{h,\leftarrow} \cup A^{in}_{h} \tag{7.16}$$

Note that the terms *right* and *left* are a bit fuzzy when it comes to lattices, and the definition in Equations (7.13) and (7.14) depends on the numbering of tokens in the lattice. In our lattices, tokens (i.e., transitions) are numbered such that a higher index means that the token is either on a competing path or it is further to the right on the same path. This way, the indices of tokens on the same path increase from left to right.

For each head token h in the sentence, let $\langle d_0, d_1, \ldots, d_{m+1}\rangle$ be the sequence of right dependents of h (the left case works analogously). d_0 and d_{m+1} are special symbols for the begin and end of the sequence. Furthermore, let $g := head(h)$ be the head of h (the grandparent). The scoring function for a given grandparent-sibling head automaton is then defined as

$$f^{\text{GSIB}}_{h,\rightarrow}(\boldsymbol{y}_{|A^{\text{GSIB}}_{h,\rightarrow}}) = \sum_{k=1}^{m+1} \boldsymbol{w} \cdot (\phi_{\text{SIB}}(h, d_{k-1}, d_k, l_k) + \phi_{\text{GP}}(g, h, d_k, l_k)), \tag{7.17}$$

where $\boldsymbol{y}_{|A^{\text{GSIB}}_{h,\rightarrow}}$ is the subvector of $\boldsymbol{y}$ that contains all indicator variables indexed by $A^{\text{GSIB}}_{h,\rightarrow}$. ϕ_{SIB} and ϕ_{GP} are feature functions that extract features from the head, the dependent, its immediate next sibling, and the grandparent.

Sibling head automata (without the grandparent part) can be solved in quadratic time using dynamic programming (Koo et al. 2010, Martins et al. 2013) to find the set of right (left) dependents of h with the highest score. To include information about grandparents, this algorithm is run once for each possible grandparent yielding a cubic complexity. As there are two grandparent-sibling head automata per token in a sentence (one for right and one for left dependents), the complexity of solving the head automata for all tokens in the sentence is $\mathcal{O}(T^4 L)$.

Head automata can simply be added to the set of subproblems defined in Section 7.1.1. The objective function for the parser changes accordingly to

$$\arg\max_{\boldsymbol{y}\in\mathcal{Y},\boldsymbol{p}\in\mathcal{P}} f_{\mathrm{T}}(\boldsymbol{y}) + f_{\mathrm{P}}(\boldsymbol{p}) + \sum_{h=0}^{T} f^{\mathrm{GSIB}}_{h,\rightarrow}(\boldsymbol{y}_{|A^{\mathrm{GSIB}}_{h,\rightarrow}}) + \sum_{h=1}^{T} f^{\mathrm{GSIB}}_{h,\leftarrow}(\boldsymbol{y}_{|A^{\mathrm{GSIB}}_{h,\leftarrow}})$$

subject to the two agreement constraints in Equations (7.8) and (7.9). The head automata subproblem gives the parser access to second-order features, but it also increases the complexity of the parser significantly.

7.1.4 Learning

We use passive-aggressive online learning (Crammer et al. 2003) with parameter averaging (Freund and Schapire 1999). We train only one parameter vector that includes features from the tree and from the path. The parser thus always performs full decoding during training. We map features into vector space using a hash kernel as described in Bohnet (2010). It gives a boost in speed and allows the model to learn weights for structures that do not occur in the training data but may be predicted during online training.

The loss function computes Hamming loss between the gold standard tree y and the predicted tree $\hat{y}$.

$$loss(y, \hat{y}) = \sum_{a_{\langle h,d,l\rangle}\in A} y(h,d,l)(1 - \hat{y}(h,d,l)) \tag{7.18}$$

Loss is computed for the full spanning tree over the lattice including the NOREL arcs. One could compute loss only on the actual tokens being selected by the parser, but this could not be done with Hamming loss since the number of tokens in gold and prediction would not necessarily coincide.

However, while the NOREL arcs are considered for computing the loss, they are not scored by the statistical model and thus do not contribute to the score of the actual solution. This is done because the only interpretation of a NOREL arc is that this token is not correct. Preliminary experiments showed that scoring them indeed yields worse models.

Feature Model

The feature model for the parser was developed semi-automatically on the Turkish development set (see Section 7.3.1) by performing a few rounds of leave-one-out feature selection. The main purpose of the feature selection was to make the feature set smaller, in particular for the second-order features. No feature selection or adaptation was done with respect to the Hebrew data. We did not systematically evaluate the importance of individual features. The feature set is an obvious place to improve the parser and to explore the possibilities and restrictions that a lattice imposes on the type of features that are available.

The feature set comprises mostly standard features like surface form, lemmas, part-of-speech tag, morphological features and combinations thereof. The full feature set of the lattice parser is described in Appendix B. Here, we will only discuss a few interesting cases that arise due to the lattice structure.

Distance Features. In standard dependency parsing, the distance between the head and the dependent is computed based on the total ordering of the tokens in the input sentence. When parsing lattices, there is no total order over the tokens anymore, as some tokens are alternatives for each other. To determine the distance between two tokens, we therefore compute the length of the shortest path between the two tokens in the lattice. There is no shortest path for tokens that can never be on the same path, but we do not need to cover this case as the parser does not need to consider arcs between such tokens anyway (see also Section 7.1.5). Note, however, that the shortest distance between two tokens in the lattice may not coincide with the distance of the two tokens in the path that is chosen by the parser. While we do not do this, in-between features (McDonald et al. 2005) could be computed based on the shortest path as normal.

Space-delimited Words. Since for Turkish and Hebrew, tokens in the lattice are segments of larger units, i.e., space-delimited words, features can also be extracted on these words. We extract the surface form of the space-delimited word that the current token is part of and whether the token is a suffix of a space-delimited word. Furthermore, if two tokens are to be related by a dependency arc, features are added that state whether the two tokens

are part of the same space-delimited word and whether they are adjacent to each other. These features are helpful for Turkish because the relations inside a space-delimited word always follow the same pattern.

Latent Context Features. One problem observed for transition-based parsing by Hatori et al. (2011) and Bohnet and Nivre (2012) is that joint models that predict, e.g., syntax and part-of-speech tags together cannot use part-of-speech tags as features on tokens that have not yet received a part-of-speech tag.[6] Hatori et al. (2011) solve this problem by what they call *delayed features*, i.e., they delay the evaluation of features involving part-of-speech tags until the token is assigned a part-of-speech tag. Bohnet and Nivre (2012) stack their joint model on a part-of-speech tagger and features extracted from buffer tokens thus use the top-scoring part-of-speech tag from the tagger even though it might not be the one assigned by the joint model later on.

A similar problem as for the transition-based parsers arises for linear context features in the lattice parser. Due to the lattice structure, it is generally not possible to find a unique left and right neighbor of a token. We define the immediate left neighbor of a token as a token whose target state coincides with the source state of current token. Analogously, the right neighbor token's source state coincides with the target state of the current token. However, as there are multiple paths in a lattice, there may be more than one immediate left/right neighbor for each token. We first tried to compute context features on all of them, but this made the model learn the specific lattice structure and led to strong overfitting effects. We therefore use the path features of the parsing model (see Table B.1 in Appendix B) to determine the context token with the highest score according to the current weight vector and extract context features only on these latent neighbors. The latent neighbors are computed only once and their scores are not influenced by the penalties from the dual decomposition algorithm.

[6] In transition-based parsing, the tokens that are still on the buffer have usually not yet received a part-of-speech tag.

Fractional Solutions

AD^3 solves a relaxation by treating the binary indicator variables (Equations (7.3) and (7.6)) as continuous variables ranging from 0 to 1. This means that in general, the solution produced by the parser is not a binary indicator vector (y) but a vector of real numbers. In most of the cases, the actual solution of the algorithm will however converge to an integer solution once the model is trained. During training, when the model is not finished, fractional solutions happen more frequently. We address this issue differently during testing and during training.

During testing, we follow Martins et al. (2009) and project fractional solutions to the nearest integer solution. They run CLE on the sentence using the fractional values of the solution as arc weights rather than the actual scores from the model. In the lattice parser, we first run the Viterbi algorithm with the fractional bigram values (p) to find the best path through the lattice. Afterwards, we run CLE on the tokens selected by the first step weighted with the fractional arc values from y.[7] In the experiments described in Section 7.3, fractional solutions occur in about 9% of the sentences in the Turkish development set during testing.

During training, we skip the projection step and instead perform updates with the fractional solution. If AD^3 finds an integer solution, the learning algorithm performs the normal update. If it finds a fractional solution, features are weighted by the fractions that were assigned to the factor they were computed on. Fractional solutions also lead to fractional loss.

7.1.5 Pruning

The search space of the parser is defined as the set of well-formed non-projective dependency trees ($\mathcal{Y}$). It is already restricted to those trees that form a coherent path through

[7] The second step can potentially fail if the pruning step (see Section 7.1.5) has pruned off all arcs that would link a token to the other tokens on the path selected by the first step. In this case, all arcs from such a token to any other token on this path are allowed. As there is no weight for these arcs, the parser will make a very poor choice attaching this token. This solution is motivated solely from the need for a proper spanning tree, it does not give the parser means to make a sensible attachment. Fortunately, this case occurs only very rarely.

the lattice by the formulation of the parser. Reducing this search space even further can lower parsing time considerably. It is commonly done and often necessary in higher-order graph-based dependency parsers (Martins et al. 2009, Koo and Collins 2010, Koo et al. 2010, Rush and Petrov 2012, Zhang and McDonald 2012, Martins et al. 2013).

Excluding Arcs between Competing Paths

First of all, we can exploit the formal properties of the lattice itself to reduce the number of arcs that the parser needs to consider. Some tokens in the lattice can never end up on the same path because they are parts of competing analyses of a word. The parser therefore does not need to consider arcs between these tokens, because they cannot both be part of a dependency tree simultaneously. Arcs between them are thus never allowed and can therefore safely be discarded from the beginning.

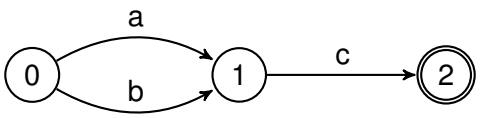


Figure 7.5: Pruning arcs between edges in the lattice that can never be on the same path. Token a can never be on the same path as token b and thus dependency arcs between a and b can be excluded beforehand. Both a and b can be on the same path as c.

To find such arcs we define C_t as the set of tokens that can be connected with t by a dependency arc. C_t contains all tokens in the lattice that can be part of a path through the lattice that also contains t. Let source(t) be the source state of a token (recall that a token is represented as an edge in the lattice) and let target(t) be the target state of that token. The set C_t is defined recursively as

1. if target(t') = source(t) then $t' \in C_t$
2. if $t' \in C_t$ then $C_t = C_t \cup C_{t'}$
3. if $t \in C_{t'}$ then $t' \in C_t$

The first condition states that two adjacent tokens can always be part of the same path. The second condition defines transitivity, i.e., that all tokens that can be on the same path

as token t' can also be part of the same path as token t if t' and t can be on the same path. The third condition defines symmetry. If t' can be on the same path as t, then t can be on the same path as t'.

All tokens $t' \notin C_t$ can neither be dependents nor heads of t. The corresponding arcs can therefore be removed from the set of candidate arcs.

Excluding Arcs Based on the Annotation Scheme

The second formal property we can use is the annotation scheme of the treebank. For example, the Turkish treebank only allows the last segments of a word to have their head outside of this word. Inner segments of a word have by definition their head immediately to the right. Since word boundaries (not token/segment boundaries) are known before parsing, tokens can be classified beforehand for whether they can be word-final segments or not. For tokens that cannot be word-final segments, we can reduce the set of candidate heads to the ones immediately following this token.

Figure 7.6 shows a schematic lattice. Word boundaries are marked with double circles and edges leading into double circles therefore represent word-final segments. Word internal segments must have their head immediately to the right, which reduces the head options for these tokens considerably. The head options for the word internal segments in Figure 7.6 would be $a : \{b, c\}$, $b : \{d\}$, $c : \{d\}$, $e : \{g\}$, $f : \{g\}$.

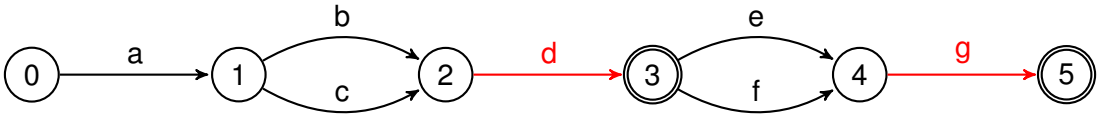


Figure 7.6: Using the annotation scheme of the Turkish treebank to prune dependency arcs. Word boundaries are marked by double circles. Edges leading into double circles represent word-final segments (marked red). All other segments must have their head immediately to the right.

Arc Pruning

A heuristic pruning step is applied after the two pruning steps described above. During heuristic pruning, the number of candidate heads for each token is reduced to a maximum

of ten.

The heuristic pruner is a simple classifier that is trained on the treebank and uses the first-order features of the parser's feature model (ϕ_{ARC}). It ranks all potential heads for each token and keeps only the top ten heads on the list. It does not enforce a tree structure but simply considers all pairs of tokens for scoring.

It also does not take into account the dependency relations, i.e., it decides only on unlabeled arcs. The admissible dependency relations for each head-dependent pair are decided in a rule-based fashion by the part-of-speech tags of the head and the dependent. If the specific combination of part-of-speech tags and dependency relation does not occur in the training data of the parser, then the particular arc will be pruned away.

The ten heads that the pruner outputs for each token are combined with all dependency relations with which they can occur in the training data. This set of candidate arcs constitutes the graph in which the parser searches for the maximum spanning tree.

As heuristic pruning can in principle remove correct arcs from the input graph, the correct arcs are always added back to the graph, when the parser is trained. This way, the parser is always able to predict the correct tree during training, but it may not learn to handle a situation well in which the correct arc has been pruned away.

Path Pruning

A second way of speeding up the parser is to reduce the ambiguity in the lattice. To achieve this, we use the bigram model (Equation (7.4) in Section 7.1.1) together with an n-best Viterbi decoder to predict the n-best paths through the lattice. We then prune away all edges in the lattice that are not part of any of the n-best paths. Since the n-best lists only contain full paths, pruning does not create dead ends in the lattice.

Note that this is not the same as keeping n-best paths in the lattice. Since all of these paths pass through the convergence points after each word, the combination of two distinct paths creates more than two paths in the pruned lattice.

The intention of this procedure is to preserve ambiguity where the scores are close to each other. The idea is inspired by the observation by Bohnet et al. (2013) that keeping few alternatives that are close in score to the top-scoring item is better than keeping more alternatives. Zhang et al. (2015) use a similar idea to create input lattice for Chinese.

7.2 Evaluation

In standard parsing, the parser predicts a syntactic structure over a fixed number of tokens. In lattice parsing, the number of tokens is not fixed because deciding on the number of tokens is part of the prediction. Simple accuracies that measure how many tokens were correctly attached are therefore not applicable in lattice parsing, as the number of tokens in the prediction can differ from the number of tokens in the gold standard. In this section, we review proposals on how to evaluate lattice parsers. The actual metrics that we use for our experiments are described in Section 7.3.4.

7.2.1 Precision and Recall

Tsarfaty (2006) proposes a variant of PARSEVAL (Black et al. 1991) to evaluate constituent parsers on Hebrew. Instead of anchoring constituents to the tokens of a parse tree, they are anchored to the (non-whitespace) characters over which they span. This way, an incorrect segmentation of a given substring will not affect the parsing score as long as the correct constituent was predicted for it. For an entirely correct segmentation, this metric collapses to standard PARSEVAL.

Let G be the set of all constituents in the gold standard parse and let P be the set of all constituents in the predicted parse. Precision is computed by dividing the number of correctly predicted constituents by the number of all constituents in the prediction. Recall is the ratio between the number of correctly predicted constituents and the number of constituents in the gold standard.

$$\text{precision} = \frac{|G \cap P|}{|P|} \qquad \text{recall} = \frac{|G \cap P|}{|G|}$$

This metric is currently the standard for evaluating constituent lattice parsers on Hebrew (Goldberg and Tsarfaty 2008, Goldberg et al. 2009, Goldberg and Elhadad 2011, 2013) and has also been applied for Arabic (Green and Manning 2010). Cohen and Smith (2007) propose a variation of this metric to deal with cases where the concatenation of the segments of a word does not coincide with the original surface form of the word (see for example Tsarfaty and Goldberg (2008) for Hebrew). They first compute an alignment between the morphemes in the gold standard and in the prediction. The metric then also punishes violations of one-to-one mappings and character edits necessary to transform the predicted sequence into the gold sequence.

PARSEVAL and the described variant are intended for evaluating constituent parsers and are not directly applicable to dependency parsing. Goldberg (2011: 53) proposes a variation on (un-)labeled attachment score that uses the same idea as the PARSEVAL variant. Morphemes in the gold standard and the prediction are indexed by the ID of the (whitespace-delimited) word to which they belong. To give Goldberg's example, the word **מספר** can be segmented as one (**מספר**) or two tokens (**מ ספר**). All segments of the word are indexed with the word ID: ⟨5,**מספר**⟩ or ⟨5,**ספר**⟩,⟨5,**מ**⟩. Dependency arcs in a dependency tree thus connect pairs of such indexed segments. Precision and recall are then computed as for PARSEVAL, with G being the set of all arcs in the gold standard and P being the set of all arcs in the predicted dependency tree. If the segmentations in the gold standard and the prediction are identical, this metric reduces to standard (un-)labeled attachment score.

A particular segmentation error that can occur in Hebrew is the incorrect prediction or omission of a covert determiner. Under certain circumstances, a given word is ambiguous between a definite and an indefinite reading which manifests in the presence or absence of an additional segment (Bar-Haim et al. 2005: 44). This error is sometimes excluded from evaluation by arguing that this should be treated as an error in the morphological analysis rather than a segmentation error (Goldberg and Tsarfaty 2008, Goldberg 2011).

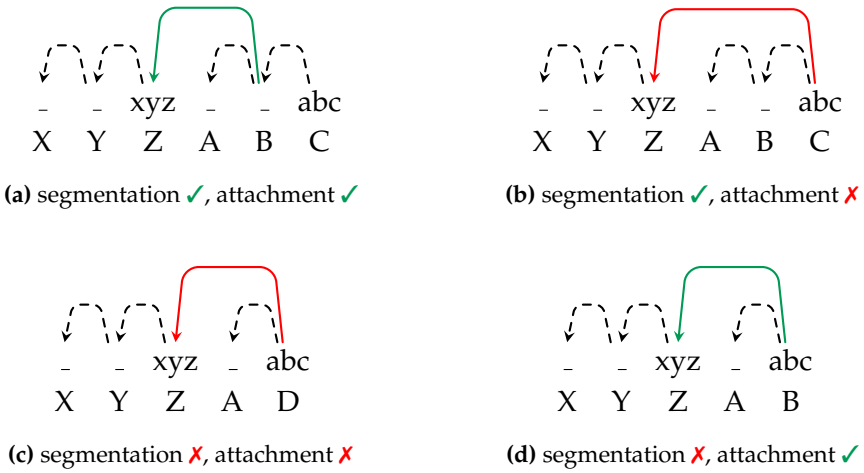
Li and Zhou (2012) and Hatori et al. (2012) use a similar definition of precision and recall to evaluate a parser for Chinese dependency parsing. The parser jointly decides on word boundaries, part-of-speech tags, and syntactic structure. Their definition of precision and recall is stricter than the one proposed in Goldberg (2011: 53), because in addition to the correct surface segmentation, the segments are required to also have the correct part-of-speech tag.

7.2.2 Evaluating Turkish

The metrics proposed for Hebrew are all applicable to Turkish dependency parsing as well, but because of the different nature of the segmentation problem in Turkish, a Turkish-specific metric can be defined meaningfully. Recall that contrary to Hebrew, Arabic, or Chinese, the Turkish treebank annotates dependency relations over inflectional groups (IGs, see Section 3.3). Within a word, the dependency structure between the IGs of this word is fixed and deterministically predictable: each IG except for the last has its head immediately to the right. A sequence of IGs in Turkish encodes different stages of morphological derivation, whereas in Hebrew, different segments of a whitespace-delimited word may constitute different words on their own which may belong to completely different syntactic contexts.

Because the internal structure of a word in the Turkish treebank is deterministic, it is traditionally not considered in evaluation. Eryiğit et al. (2008) (also in Eryiğit 2012) define a metric that evaluates the attachment of the last IG of each word in the sentence. For an arc to be correct, the IG has to be attached to the correct IG in the correct word (Figure 7.7a). This requires the segmentation of the head word to be correct, because otherwise, the correct IG cannot be determined. If the segmentation of the head word is not correct, the metric accepts an attachment as correct if the head IG is part of the correct head word and the head IG has the same part-of-speech tag as the gold-standard head IG (Figure 7.7d).

The segmentation of a word is considered correct if all IGs in the predicted word segmentation have the same morphological analysis as their counterparts in the gold standard. Unlike for Hebrew, the segmentation cannot be checked on the word forms alone, as the dependency representation of the Turkish treebank annotates word forms for non-final IGs as underscores. The metric does not consider the segmentation of dependent words during evaluation, and it loosens the requirements if the segmentation of the head word is incorrect. The metric thus abstracts away from the segmentation problem as much as possible in order to evaluate syntactic structure in isolation.



(a) segmentation ✓, attachment ✓

(b) segmentation ✓, attachment ✗

(c) segmentation ✗, attachment ✗

(d) segmentation ✗, attachment ✓

Figure 7.7: Evaluation for Turkish in Eryiğit et al. (2008). The four schemas show two words (xyz and abc) with their segmentations indicated by underscores. (a) and (d) show cases that are counted as correct, (b) and (c) show cases that are considered incorrect. Uppercase letters symbolize part-of-speech tags. Dashed arcs are word internal and are not considered for evaluation.

7.2.3 TedEval

TedEval is a metric that was developed to facilitate cross-framework evaluation, e.g., comparing different dependency annotations (Tsarfaty et al. 2011) or comparing constituent structures to dependency structures (Tsarfaty et al. 2012a). It has also been proposed for evaluating morphological segmentation and parsing (constituents and dependencies) jointly (Tsarfaty et al. 2012b).

TedEval is based on tree edit distance, which is the number of operations like insertion and deletion of nodes that is needed to transform one tree into another. The two trees that are compared are first transformed into *functional trees*, formal tree structures that represent the functional information of a parse tree but abstract away from the directionality of the syntactic relations. Each nonterminal in these trees is labeled with the grammatical function of the substring that it spans over. This process is performed for the gold standard tree and the predicted parse tree.[8] The minimal tree edit distance is then computed

[8]When comparing to multiple gold standards, the gold standard tree is actually a generalization over multiple trees from different gold standards.

between the functional trees of both gold standard and prediction.

For the joint evaluation, Tsarfaty et al. (2012b) propose the following operations: adding a nonterminal node, deleting a nonterminal node, adding a terminal node, and deleting a terminal node. Adding and deleting terminal nodes allows the metric to measure segmentation errors, as each terminal that needs to be replaced with another constitutes an error in the predicted segmentation. The terminals that can be added are restricted by the input lattice and segments can only be replaced by segments that are available for the same position in the sentence. Tsarfaty et al. (2012b) then define the actual TedEval metric as

$$\textsc{TedEval}(G, P) = 1 - \frac{\textsc{ted}(G, P)}{|P| + |G| - 2} \tag{7.19}$$

where $\textsc{ted}(G, P)$ is the minimum tree edit distance between the gold tree G and the predicted tree P. The edit distance is normalized by the number of nodes in both trees minus the two root nodes.

When using this metric for evaluation of joint morphological segmentation and dependency parsing, the dependency trees have to be transformed into functional trees first. Non-projective trees are projectivized to allow for the transformation. A dependency tree is then evaluated based on the structure of its functional tree. It has been used for example in Seddah et al. (2013) and Zhang et al. (2015) to evaluate the joint task.

However, the transformation seems to influence the outcome of the evaluation: As an experiment, we took a sentence from the Turkish treebank and created several versions of it where the 10th token is attached to each of the other 13 in the sentence unless this would lead to a cyclic structure. All other tokens were left untouched. This procedure thus created 12 different versions of the dependency structure, 11 with exactly one attachment error and 1 tree that was completely correct. Running TedEval on the 11 trees with the attachment error returned 5 different scores, because the functional trees that are created from the dependency trees have different edit distances to the gold even though all of the underlying dependency trees have exactly one attachment error.

This experiment shows that TedEval and attachment score are not equivalent, which is to be expected. But since different errors in the dependency trees can lead to different TedEval scores, it shows that there is a gradation of incorrectness in the metric. Some errors are punished more by TedEval than others, and they are punished in an unpredictable manner.

A difference in scores between two parsers can be caused by one parser getting more of the structure correct than the other. But in TedEval, it can also be caused by one parser making better errors than the other, where better means more favorable functional tree structures. Using TedEval in this manner will thus bias models towards higher TedEval scores, but it is not clear whether this bias relates to anything meaningful and if so, what it would be. We see the metric proposed by Goldberg (2011: 53) as a good alternative for TedEval in dependency parsing, as it performs a very similar evaluation by taking the surface form of the tokens into account, but it does not need to transform the dependency structures.

7.2.4 Evaluating the Path

The lattice parser performs all subtasks of a typical pipeline simultaneously, namely segmentation, part-of-speech tagging, morphological analysis, and syntactic analysis. The first three are solved by finding a single path through the lattice, the last one is solved by finding the best spanning tree over such a path. While it is difficult to evaluate the parsing task without the other ones, we can evaluate the first three tasks without the last one.

The definitions of precision and recall described above are easily adapted by simply disregarding the syntactic structure. A segment is considered correct if it matches its counterpart in the gold standard. Precision and recall are defined as before over the number of segments in the prediction and the gold standard, respectively.

Evaluating the segmentation based solely on the surface form of the segments makes more sense for the Hebrew treebank than for the Turkish treebank, since the latter annotates underscores for the surface forms of word-internal segments, which makes the matching of surface forms mostly uninformative.

Another possibility to evaluate the path selection of the lattice parser is to use Word Error Rate (WER) (Jurafsky and Martin 2009: 362). This metric computes minimum string edit distance between the predicted segmentation and the gold standard. Given the three edit operations *insert*, *delete*, *substitute*, WER is defined as

$$\text{WER} = 100\,\frac{\text{insertions} + \text{deletions} + \text{substitutions}}{\text{number of words in gold standard}}$$

Tsarfaty et al. (2012b) define SEGEVAL, which is based on string edit distance as well.

In addition to the segment-based metrics, we can also give a word-based evaluation. Such a metric would return the ratio of correctly segmented words to the number of words in the sentence. For this metric, only entirely correctly segmented words count as true positives.

7.3 Experiments

In this section, we present the experimental evaluation of the lattice parser by testing it on the Turkish treebank and the treebank of Modern Hebrew. We compare the joint model to two state-of-the-art pipeline systems and to a pipeline system that uses the exact same feature set and decoding algorithms as the joint parser.

7.3.1 Data Sets

Turkish. The training set for Turkish consists of the 5,635 sentences of the METU-Sabancı Turkish Treebank (Oflazer et al. 2003b). We use the *detached* version of the Turkish Treebank (Eryiğit et al. 2011), where multiword expressions are represented as separate tokens. We use the 300 sentences of the ITU validation set (Eryiğit 2012) as test set. As there is no separate development set, we split the training set into 10 parts and use 2 of them as development data. All models run on this development set are trained on the remaining 8 parts. Table 7.1 summarizes the split sizes.

This version of the treebank contains 49 sentences with circular annotations. We manually corrected these sentences and use the updated version in our experiments.[9]

Hebrew. The Hebrew data comes from the SPMRL Shared Task 2014 (Seddah et al. 2014), which provides lattices and pre-disambiguated input files. This data is based on the

[9]The corrected data is available under `http://www.ims.uni-stuttgart.de/institut/mitarbeiter/ozlem/seekerTACL2015.html`.

	Turkish	Hebrew
training	5,635 (4,508)	5,216
development	0 (1,127)	500
test	300	500
total	5,935	6,216

Table 7.1: Data split sizes. The numbers in brackets for Turkish refer to the set sizes when the development set is evaluated.

Modern Hebrew Treebank (Sima'an et al. 2001), which was extended according to the Universal Stanford Dependency scheme (Tsarfaty 2013) and converted to dependencies using the method described in Goldberg (2011). The 2014 version also uses the original Hebrew characters instead of the ASCII transliteration that was used in the SPMRL Shared Task 2013.

The treebank consists of 6,216 annotated sentences, of which the first 500 are used as development set and the last 500 are used as test set. The remaining sentences are used as training set. The sizes of the different sets are summarized in Table 7.1. The training and development lattices contained a number of circular structures from which we removed the edges causing the cycles.

7.3.2 Experimental Setup

The lattice parser is trained on morphological lattices that represent different segmentations and morphological analyses of a given sentence. For Turkish, these lattices are produced by passing the sentences through a morphological analyzer for Turkish (Oflazer 1994). For Hebrew, we use the lattices provided by the SPRML Shared Task 2014. For both Turkish and Hebrew, the training lattices may not contain the correct paths due to shortcomings of the lexicons and analyzers involved in their creation. In these cases, we add the correct path to the lattices so that it is always possible for the parser to find the correct solution during training. This procedure is done during training only. We do not add any paths to the lattices during testing.

In order to provide disambiguated lattices to the baseline systems, we train the bigram model (Equation (7.4) in Section 7.1.1) separately to predict the best path through a lattice. The feature set is the same as in the lattice parser. In the following, we will refer to this model as the *bigram segmenter*. The segmenter is trained on the respective training data for each language to predict paths for the development and test sets.

Compared to the Turkish data, the Hebrew lattices are so large that training times for the lattice parser became unacceptable. We therefore use the bigram segmenter in combination with an n-best Viterbi decoder to predict the 10 best paths for each lattice (the training set is annotated via 10-fold jackknifing). All edges in the lattice that are not part of one of these 10 paths are discarded. All experiments with the joint model for Hebrew are conducted on such pruned lattices (see also Section 7.1.5).

The bigram segmenter and the lattice parser are trained for 10 iterations. After each iteration, the training samples are deterministically shuffled. AD^3 is run for maximally 1,000 iterations both during training and testing.

7.3.3 Baselines

We use three baseline parsing systems in the experiments: MATE, TURBO, and PIPELINE. The first two baselines are off-the-shelf dependency parsers that currently represent the state-of-the-art, the purpose of the third baseline is to measure the effect of the joint decoding compared to a pipeline setting.

1. MATE. Mate parser[10] (Bohnet 2009, 2010) is a graph-based dependency parser that uses Carreras' decoder (Carreras 2007) and approximate search (McDonald and Pereira 2006) to output non-projective dependency structures.

2. TURBO. TurboParser[11] (Martins et al. 2013) is a graph-based parser that decomposes the parsing problem into smaller subproblems and solves the overall task using dual decomposition. The main principles behind the construction of the lattice dependency parser presented in this chapter are in fact modeled after TurboParser.

[10] http://code.google.com/p/mate-tools
[11] http://www.ark.cs.cmu.edu/TurboParser/

3. PIPELINE. The third baseline system runs the joint parser on a pre-disambiguated lattice, i.e., in a pipeline setup. The lattices are disambiguated using the same decoder and feature set that the lattice parser uses in the bigram model. The only difference between the PIPELINE baseline and the lattice parser is therefore the fact that the lattice parser performs segmentation and parsing jointly.

All three baselines are pipeline setups and operate on disambiguated lattices. Since the bigram segmenter uses the same feature model (for segmentation) as the lattice parser, there is no difference between the lattice parser and any of the baseline parsers concerning the features that are available during segmentation.

The first two baselines allow us to compare the joint parser to the current state-of-the-art. Note, however, that the feature sets are different between the joint parser and the off-the-shelf baseline systems. A difference in performance between the lattice parser and the first two baseline systems might therefore be caused by a difference in the feature sets rather than the fact that the former performs joint decoding. The third baseline eliminates this difference in the feature sets and allows us to test directly the influence of joint decoding.

All three baseline systems are trained on the gold standard segmentation (and thus gold morphological analyses) in the training data, since predicted paths are not guaranteed to be compatible with the gold dependency structures. During testing, the lattices are first disambiguated using the bigram segmenter, and the selected paths are then given to the parser.

7.3.4 Evaluation Metrics

We follow Hatori et al. (2012) and use a strict definition of precision and recall (PREC, REC, F1) over tokens to evaluate the full task. We first align the tokens of each word in the parser output with the tokens of the corresponding word in the gold standard using the Needleman-Wunsch algorithm (Needleman and Wunsch 1970), which we modified so it does not allow for mismatches. A token in the parser output that is not in the gold standard is thus paired with a gap and vice versa. Two tokens must have the same morphological

analysis in order to match.[12] A true positive is a token that has the same segmentation, morphological analysis, and head as its matching token in the gold standard.

This metric is very strict and requires all levels of analysis to be correct. In order to evaluate the syntax as independently as possible, we furthermore report IGeval for Turkish, with and without the aforementioned backoff strategy (IGeval and IGeval STRICT). For Hebrew, we report on a version of precision and recall as defined above that only requires the surface forms of the tokens to match.[13] This metric is almost the one proposed in Goldberg (2011), but we do not exclude errors with respect to covert determiners. All reported evaluation metrics ignore punctuation.

7.3.5 Results

We present the evaluation of the system in four steps: We start by measuring how well the joint system is able to select a correct path through the input lattice. We then show the competitiveness of the feature model of our system by applying it to correctly disambiguated lattices. After that, we evaluate the full task, in which the system has to predict segmentation, part-of-speech tags, morphological features, and syntactic structure. Finally, we have a closer look at where the improvements occur in the joint system.

The purpose of the experiments is two-fold: Firstly, we want to see how the lattice parser compares to state-of-the-art pipeline systems that perform the joint task as a series of independent steps. Secondly, we want to compare the lattice parser to a pipeline system that uses the exact same feature model and decoding algorithms in order to see whether the very fact that it uses joint decoding makes the lattice parser superior to a pipeline setup.

[12] The method does not create cross, many-to-one, or one-to-many alignments, which can be important because in very rare cases the same token occurs twice in one word.

[13] The metric would not work for Turkish, as the surface forms of non-final IGs are all represented as underscores.

Path Selection

We first evaluate the effect of syntactic information on the path selection quality and leave aside the syntactic structure for now. We compare three systems for each treebank, a baseline system (BASELINE), the bigram segmenter (BIGRAM SEGMENTER), and the full lattice parser (JOINT).

The baseline system for Turkish is the morphological disambiguator described in Sak et al. (2008). We do not use pre-trained models but train the disambiguator on the training data of the Turkish treebank. We use the pre-disambiguated lattices provided by the SPMRL 2014 Shared Task as the baseline for Hebrew. Seddah et al. (2013: 159) give a description on how these lattices are produced.

We use segment-based precision, recall, and f-score (PREC, REC, F1) and word-based accuracy (ACC_w) to evaluate the systems. The metrics are described in Section 7.2.4. Recall that these metrics measure the quality of the full path selected by the system. Only a segment/word that has the correct surface form, part-of-speech tag, and morphological analysis is counted as a true positive.

data	system	PREC	REC	F1	ACC_w
dev	BASELINE	89.59	88.14	88.86	87.97
	BIGRAM SEGMENTER	90.69	89.52	90.10	89.45
	JOINT	**90.80**	**90.22**	**90.51**	**89.94**
test	BASELINE	89.46	88.51	88.99	87.95
	BIGRAM SEGMENTER	89.96	89.23	89.59	88.71
	JOINT	**90.19**	**89.74**	**89.97**	**89.25**

Table 7.2: Path selection quality for Turkish.

Table 7.2 shows the experimental results for the segmentation evaluation on the Turkish data. The bigram segmenter considerably outperforms the baseline system. This shows that the feature model of the bigram segmenter is better suited to the Turkish treebank, which is of course partly due to the fact that this particular feature model was developed on the Turkish development set. The bigram model is in turn outperformed by the full lattice parser. While there is a modest improvement for precision, recall and word-based accuracy show a considerable improvement over the bigram model. The improvements over the bigram model must be due to the syntactic information that the lattice parser can

access, since the bigram model and the lattice parser both use the same feature model for path selection.

data	system	PREC	REC	F1	ACC_w
dev	BASELINE	85.99	84.07	85.02	80.30
	BIGRAM SEGMENTER	**86.84**	86.30	86.57	83.46
	JOINT	86.68	**87.49**	**87.08**	**84.67**
test	BASELINE	81.79	79.83	80.80	74.85
	BIGRAM SEGMENTER	**84.44**	83.22	83.83	79.60
	JOINT	83.88	**83.99**	**83.94**	**80.28**

Table 7.3: Path selection quality for Hebrew.

Table 7.3 shows the results of the same experiment when conducted on the Hebrew data. Also for Hebrew, the bigram segmenter and the lattice parser outperform the baseline. As the feature model of these two systems was not specifically adapted to the Hebrew treebank in any way, this result suggests that the feature model performs quite well in the general case. Comparing the bigram segmenter with the lattice parser, we see that for Hebrew, the bigram segmenter gives better precision than the lattice parser, especially on the test set. But like in Turkish, the parser is considerably ahead in recall and word-based accuracy, and the higher f-score indicates that the improvement in recall outweighs the disadvantage in precision to some extent.

So far, the experiments indicate that syntactic information is useful for the segmentation task. This result corroborates previous results for Hebrew (Cohen and Smith 2007, Goldberg and Tsarfaty 2008) and also demonstrates it for Turkish, even though the segmentation ambiguities in both treebanks have different origins.

Performance on Gold-standard Segmentation

Next, we present experimental results on the development sets of both languages, when the different systems are trained and tested on correctly disambiguated lattices. We compare the lattice parser (JOINT) to the three baselines described in Section 7.3.3, namely MATE, TURBO, and PIPELINE. In this setting, the PIPELINE and the JOINT systems give identical results, because they use the same feature set and decoders. We can use standard labeled and unlabeled attachment scores (LAS, UAS) for evaluation because segmentation

in the parser output is always correct.

		Turkish		Hebrew	
data	system	LAS	UAS	LAS	UAS
dev	MATE	73.05	81.61	82.80	89.68
	TURBO	73.85	82.87	82.51	89.63
	PIPELINE/JOINT	74.40	82.81	83.06	89.63

Table 7.4: Parsing results for Turkish and Hebrew using gold standard segmentation.

Table 7.4 mainly serves to show that the feature model of the lattice parser is competitive with the external baseline parsers. For unlabeled scores, all three systems perform on par with each other, whereas for labeled scores, the lattice parser outperforms the baselines. The numbers also show that MATE performs worse than TURBO for Turkish, but is slightly ahead for Hebrew.

Evaluating the Full Task

We now evaluate the performance of the lattice parser on the full task, i.e., segmentation, morphological analysis, and dependency parsing. We evaluate with segment-based precision, recall, and f-score (PREC, REC, F1) both for labeled and unlabeled dependencies as described in detail in Section 7.2. The input to all baseline systems has been produced using the bigram segmenter trained on the respective training data.

Table 7.5 gives the results on the Turkish treebank, evaluating the systems on the development set, the test set, and on the training set via 10-fold cross-validation (10cv). The TURBO baseline performs consistently better on the Turkish data than the MATE baseline. It is in turn outperformed by the PIPELINE baseline in all experiments except for unlabeled scores on the development set, where TURBO and PIPELINE give results that are close to each other. That the PIPELINE baseline is better than the two external baselines is partly due to the fact that the feature model of the PIPELINE baseline was specifically developed on the Turkish development set, which is not the case for the external baselines. The JOINT system is the best performing system, consistently outperforming even the PIPELINE baseline.

		LABELED			UNLABELED		
data	system	PREC	REC	F1	PREC	REC	F1
dev	MATE	62.54	61.73	62.14	69.44	68.54	68.98
	TURBO	63.54	62.71	63.12	70.68	69.76	70.22
	PIPELINE	63.86	63.03	63.44	70.65	69.73	70.19
	JOINT	**64.21**	**63.79***	**64.00**	**70.96**	**70.50***	**70.73**
10cv	MATE	63.28	62.49	62.88	70.37	69.50	69.94
	TURBO	63.82	63.03	63.42	71.12	70.24	70.68
	PIPELINE	64.97	64.17	64.57	71.71	70.83	71.27
	JOINT	**65.27**	**64.84**†	**65.06**	**72.05***	**71.58**†	**71.82**
test	MATE	64.64	64.12	64.38	70.62	70.04	70.33
	TURBO	65.36	64.83	65.09	71.66	71.08	71.37
	PIPELINE	66.40	65.86	66.13	72.30	71.72	72.01
	JOINT	**67.33**	**66.99***	**67.16**	**72.94***	**72.58***	**72.76**

Table 7.5: Parsing results for Turkish. Statistically significant differences between the joint system and the pipeline system are marked with † ($p < 0.1$) or $*$ ($p < 0.5$). Significance testing was performed for precision and recall using the Wilcoxon Signed Rank Test.

Table 7.6 shows the same experiments conducted on the Hebrew treebank. We compare the same systems as for Turkish, but note that the joint system is now called JOINT10 to indicate that the lattices for this system are pruned as described in Section 7.3.2. Unlike for Turkish, we cannot find big differences between the three baseline systems. They perform mostly on par with each other, this time with the MATE baseline being slightly ahead of the TURBO baseline. However, as for Turkish, the lattice parser is better than the three baselines.

For both languages, the differences in recall between the pipeline baseline and the joint system are statistically significant, but this is most of the time not the case for precision (with the exception of unlabeled precision on 10cv and test for Turkish). It appears that the syntactic information that is available to the joint system mostly helps with recovering correct segments but cannot prevent overprediction of segments better than the pipeline system. However, the improvements in recall do not seem to occur at the expense of precision, as the precision of the joint system is in general at least as high or higher than the precision of the pipeline baseline.

		LABELED			UNLABELED		
data	system	PREC	REC	F1	PREC	REC	F1
dev	MATE	65.41	65.00	65.20	70.65	70.21	70.43
	TURBO	65.12	64.72	64.92	70.44	70.00	70.22
	PIPELINE	65.64	65.23	65.44	70.65	70.21	70.43
	JOINT10	**66.82**	**67.44**†	**67.13**	**71.47**	**72.13***	**71.80**
test	MATE	63.16	62.25	62.70	67.52	66.55	67.03
	TURBO	63.06	62.16	62.61	67.27	66.31	66.79
	PIPELINE	63.63	62.72	63.17	67.62	66.65	67.14
	JOINT10	**63.81**	**63.89**†	**63.85**	**67.79**	**67.88**†	**67.84**

Table 7.6: Parsing results for Hebrew. Statistically significant differences between the joint system and the pipeline system are marked with † ($p < 0.1$) or $*$ ($p < 0.5$). Significance testing was performed for precision and recall using the Wilcoxon Signed Rank Test.

Evaluating the Syntax

As became clear in Section 7.2, it is very difficult to evaluate the parsing system only for the syntactic performance independently from the segmentation quality. Nonetheless, while we are generally interested in improving the full parsing task, knowing at which level the improvements occur is helpful to understand the system better.

Because of its particular type of annotation, we can evaluate the lattice parser on the Turkish treebank with two additional metrics. We use the word-based accuracy proposed by Eryiğit and Oflazer (2006), which is described in detail in Section 7.2.2. We also use a more strict version of this metric. In the strict version, the head word is always required to have the correct segmentation. We denote these metrics IGeval and IGeval STRICT and report on labeled (LAS_{IG}) and unlabeled dependencies (UAS_{IG}).

Table 7.7 shows the results of the experiments on the Turkish treebank when evaluated with the IGeval metrics. For the strict version, the picture is similar to the one we saw in Table 7.5, where the lattice parser outperforms all baselines. For the non-strict version, however, the difference between the lattice parser and the PIPELINE baseline disappears.

Recall that the IGeval metric is designed to abstract as much as possible from the segmen-

		IGeval STRICT		IGeval	
data	system	UAS_{IG}	LAS_{IG}	UAS_{IG}	LAS_{IG}
dev	MATE	70.60	60.10	74.88	63.46
	TURBO	72.22	61.24	76.58	64.73
	PIPELINE	72.26	61.82	**76.64**	65.49
	JOINT	**72.66***	**62.40**	76.61	**65.59**
10cv	MATE	71.75	61.26	75.84	64.42
	TURBO	72.77	61.89	76.93	65.09
	PIPELINE	73.66	63.52	77.68	66.82
	JOINT	**73.93**	**63.85**	**77.74**	**66.83**
test	MATE	71.99	61.84	77.08	65.98
	TURBO	73.16	62.76	78.37	67.02
	PIPELINE	74.33	64.40	**79.61**	**69.02**
	JOINT	**75.02**	**65.32**	79.45	68.99

Table 7.7: Parsing results for Turkish using IGeval. Statistically significant differences between the joint system and the pipeline system are marked with †. Significance testing was performed using the Wilcoxon Signed Rank Test with $p < 0.01$.

tation of the words that are connected by the dependency structure. Thus, it is able to approximate a syntactic evaluation that is independent of potential segmentation mistakes made by the system. The reason for this is the annotation in the Turkish treebank, in which the word-internal dependency arcs are deterministic and thus allow for a word-based evaluation.

The disappearance of the difference between the PIPELINE baseline and the lattice parser suggests that most of the improvements in the joint system occur in the segmentation and morphological analysis. The syntactic quality produced by the lattice parser stays on the same level as the pipeline baseline.

We can perform a similar experiment for Hebrew. Since the Hebrew data encodes actual word forms for the segments (compared to the underscores in the Turkish treebank), we can evaluate the syntax jointly with the surface segmentation disregarding any errors in the predicted part-of-speech tags or morphological features. This metric is very similar to the metrics used by Goldberg and Tsarfaty (2008) and is also comparable in spirit to the

version of TedEval in Tsarfaty et al. (2012b).

		LABELED			UNLABELED		
data	system	PREC	REC	F1	PREC	REC	F1
dev	MATE	68.05	67.62	67.83	74.70	74.24	74.47
	TURBO	67.97	67.54	67.75	74.58	74.12	74.35
	PIPELINE	68.56	68.14	68.35	74.84	74.37	74.60
	JOINT10	**69.23**	**69.87**†	**69.55**	**74.88**	**75.58**†	**75.23**
test	MATE	66.17	65.22	65.69	71.62	70.60	71.11
	TURBO	66.14	65.19	65.66	71.38	70.35	70.86
	PIPELINE	**66.81**	65.85	66.33	**71.82**	70.79	71.30
	JOINT10	66.63	**66.72**†	**66.68**	71.48	**71.57**†	**71.52**

Table 7.8: Parsing results for Hebrew, only requiring surface forms of segments to be correct. Statistically significant differences between the joint system and the pipeline system are marked with †. Significance testing was performed for precision and recall using the Wilcoxon Signed Rank Test with $p < 0.01$.

Table 7.8 shows the results of this kind of evaluation. Unsurprisingly, the absolute figures are higher than the ones in Table 7.6, since we now disregard two classes of errors. We still get the same picture as before with the lattice parser being ahead of the baselines in recall, but like in Table 7.3, the pipeline baseline gives higher numbers for precision. The differences in recall between these two systems are statistically significant.

7.3.6 Conclusion

The experiments presented in this section show that the lattice parser performs better than a pipeline model when evaluated for the full task of segmentation, morphological analysis, and parsing. The improvements mostly occur in recall and the effect is the same for both languages, Turkish and Hebrew. For Turkish, the improvements occur in the lower levels of analysis, i.e., segmentation and morphology. The IGeval metric shows that the pipeline and the joint model perform equally well on the syntactic level. This effect might be due to the special nature of segmentation ambiguity in the Turkish treebank. For Hebrew, an evaluation that disregards morphological analysis still shows improvements in recall for the joint model compared to the pipeline. That a joint model is better suited to model Hebrew has been shown by Cohen and Smith (2007), Goldberg and Tsarfaty (2008), and Goldberg and Elhadad (2013) for constituency parsing and our experiments demonstrate

the same result for dependency parsing.

7.4 Discussion

We start the discussion by giving some background on the joint models proposed for parsing. We then compare our parser to this work and discuss alternative modeling choices.

7.4.1 Joint Constituency Parsers

There exists a long line of work on joint models for parsing. It was originally motivated by the word segmentation problem in Chinese, Hebrew, and Arabic and was conducted in the context of constituency parsing. The segmentation problem in Chinese is different to the one in Turkish or Hebrew as it is not a problem about splitting words into smaller parts but rather a problem of grouping sequences of characters into words. Luo (2003) already proposes a constituency parser that integrates segmentation and part-of-speech tagging into the parsing process by modeling word structure as part of the constituency structure. Words are thus treated as the lowest tier of constituents. They find that the structural information from the parse tree is less important but part-of-speech information is highly beneficial to the task of word segmentation. This finding is confirmed later by Ng and Low (2004) with a serious of comparisons between pipeline and joint models for joint segmentation and part-of-speech tagging. Several other models for segmentation that exploit part-of-speech information were developed since then, e.g., Zhang and Clark (2008b), Jiang et al. (2008), and Zhang and Clark (2010).

Joint models for Hebrew were first proposed by Tsarfaty (2006), who argues that a joint model for segmentation, morphological analysis, and syntactic analysis is better suited for Hebrew than a pipeline setup. She defines a probabilistic model for the full joint task but presents experiments with a model that performs the joint task only for the first two steps (segmentation and morphological analysis). Nonetheless, the experimental results already demonstrate the superiority of the (semi-)joint model compared to a setup where none of the tasks are solved jointly.

The first full joint constituent parser for Hebrew was proposed one year later by Cohen and Smith (2007). It is a parser that operates on morphological lattices and searches for the best combination of parse tree and lattice path. Their model factors the joint probability into two different sub-models: a morphological disambiguation model and a PCFG, which they train separately. The morphological disambiguation model assigns weights to the transitions in the lattice, which are integrated into the score for the tree when the transition is selected by the lattice parser. They show that a good morphological model is an important prerequisite for the success of the joint parser. In particular, they need the morphological model to be good at ranking the alternatives to the best solution in order to be beneficial to the parser.

Goldberg and Tsarfaty (2008) present further experiments with a lattice constituent parser for Hebrew. Contrary to Cohen and Smith (2007), their model does not have a separate sub-model for morphological disambiguation but relies entirely on the parsing model to disambiguate the morphology. They test a series of extensions to a plain PCFG creating increasingly rich grammars and demonstrate that the richer grammars are more accurate. Although they do not directly compare to a pipeline model, their best parser outperforms the results reported by Cohen and Smith (2007) for their pipeline and their joint setting. This work is continued in Goldberg and Elhadad (2011) where they extend the Berkeley parser (Petrov et al. 2006) to accept lattices as input. The latent annotations learned by the LA-PCFG yield an error reduction of 20% over their previous model. In Goldberg and Elhadad (2013), they extend the parser once more by adding a post-filter on top of the LA-PCFG lattice parser to model morphological agreement, showing that such a filter can lead to grammatically more sound analyses.

For Arabic, Green and Manning (2010) adapt the Stanford parser to lattices and conduct a comparison of a joint model to a standard pipeline. They find that weighting the lattice as suggested by Cohen and Smith (2007) gives better performance than using the unweighted lattices as proposed by Goldberg and Tsarfaty (2008). However, their joint model does not outperform the pipeline setup. They suggest that this result might be due to the fact that lattices are basically longer sentences and that parse quality is generally worse on longer sentences.

7.4.2 Joint Models for Transition-based Dependency Parsers

Joint models for dependency parsing were pioneered by Hatori et al. (2011) for Chinese. They design a transition-based parser that performs part-of-speech tagging and parsing jointly. In this parser, the *shift* operation of the arc-standard decoder (Nivre 2008) is defined such that every time a token is pushed onto the stack the parser also selects a part-of-speech tag for it. Instead of one shift transition that the model has to predict, the parser now chooses between shift transitions for each tag in the part-of-speech tag set. This increase in number of transitions adds a constant factor to the overall complexity of the parser, which means that the parser is not significantly slower, especially since part-of-speech tag sets for languages with no morphology are usually rather small. A problem that they encounter is that due to the joint modeling, the parser has no information about the part-of-speech of tokens that it has not shifted yet. To deal with this, they introduce *delayed features* in the feature model, which postpones the evaluation of some features until all necessary information is available.

This parser was soon extended to integrate word segmentation into the model thus performing the full joint task of segmentation, part-of-speech tagging, and dependency parsing (Hatori et al. 2012, Li and Zhou 2012). The full joint parsers operate on character level and form words by means of an *append* action, an additional operation in the transition system that concatenates characters. The first models worked with an ad-hoc representation of the inner structure of the words. Linguistically motivated word-internal structures were shown to yield even better results by Zhang et al. (2014a).

The joint transition-based parsers that were developed for Chinese were quickly adapted to other languages as well. As we have argued in this dissertation, joint models are interesting for languages with rich morphology because they can model interaction between morphology and syntax. Bohnet and Nivre (2012) define a similar parser as the one in Hatori et al. (2011) but extend it to non-projective parsing in order to handle the free word order in morphologically rich languages.

Bohnet et al. (2013) extend the parser further to include prediction of morphological features into the model. However, including morphological features is not as straightforward as including the part-of-speech tags was for the Chinese parsers because morphological tag sets are much larger than part-of-speech tag sets. Simply having the parser choose for

each shifted word a morphological tag out of a set of potentially more than 1,000 tags has a big impact on the run-time of the parser. Bohnet et al. (2013) therefore provide the parser with an n-best list of possible tags for each word, which they predict with a standard sequence model. They find that keeping at most two tags for each word already gives them the best results. It may be surprising that two is already enough, but today's sequence models for predicting part-of-speech tags or morphology are quite good. Keeping a list of the two best tags essentially means that one trusts the sequence model to rank the correct candidates high, but wants to keep some options open for the parser to make the final decision when it sees more context. That the preprocessing has to be good in ranking the correct alternatives high was already found by Cohen and Smith (2007) for their Hebrew constituency parser (see above).

Bohnet et al. (2013) encounter another problem related to rich morphology in their beam-search decoder: the beam quickly loses variants that differ with respect to the part-of-speech tags or morphological features and only keeps structural variants around. They avoid this effect by reserving a portion of the beam to be filled by morphological variants exclusively. The problem seems related to one encountered by Zhang et al. (2014a), who weight the features for segmentation and part-of-speech tagging four times as high as the parsing features because otherwise the parsing features dominate the feature model due to their larger number.

The parsers discussed in this section were developed for languages with rich morphology but that do not have a word segmentation problem as in Hebrew, Arabic, Turkish, or Chinese. However, it is straight-forward to adapt the parser. In essence, the append action that is used in Chinese to form words from characters can be used to split words into smaller parts. A parser that uses this idea is described in Tratz (2013) for parsing Arabic. The parser is based on the easy-first decoding algorithm (Shen and Joshi 2008, Goldberg and Elhadad 2010a) and differs from standard transition-based algorithms in that it can operate on any pair of words in the sentence. The parser defines additional operations like part-of-speech tagging, morphological tagging, and affix splitting. The operations are ordered such that the parser can only link two tokens with a dependency relation if both have already received a part-of-speech tag. The same idea was implemented in a parser for Chinese for joint part-of-speech tagging and parsing (Ma et al. 2012).

There is some work on standard dependency parsing for Hebrew (Goldberg and Elhadad 2009, 2010a, Goldberg 2011), but only little work has been done on lattices. De La Clergerie

(2013) reports on experiments with a transition-based dependency parser for lattices. The results for the joint model are however slightly behind the ones for their pipeline baseline. Köhn et al. (2014) conduct experiments with TurboParser on n-best paths that they predict from Hebrew lattices. Although this is technically not lattice parsing, they show that the parser is able to select better paths from the n-best list than a ranking model without syntactic features.

7.4.3 Joint Models for Graph-based Dependency Parsers

Parallel to the development of transition-based parsers for joint modeling work on graph-based parsers was conducted too. There is however not as much work on joint models for graph-based parsing due to the high complexity of graph-based decoders. Transition-based parsers are generally more attractive for joint modeling due to their incremental processing that looks at one word at a time. Having a transition-based parser predict part-of-speech tags adds the size of the part-of-speech tag set as a constant to its complexity. This constant is usually small for part-of-speech tags but as we discussed above, it can become very large for morphological tags and already then it is better to prune the set of possible tags for each word. However, the tag set constant is considerably larger in graph-based parsers because graph-based parsers consider factors over multiple words.

Li et al. (2011) extend Eisner's decoder (Eisner 1997, 2000) and several higher-order versions of it to jointly predict part-of-speech tags and syntactic structure. In each cell of the chart, the parser predicts the highest-scoring combination of arc and part-of-speech tags on the head, the dependent, and the two words at which the combined spans meet. Depending on the order of the factors, this adds polynomials of the tag set size to the complexity of the parser. The complexity of their first-order model includes the factor q^4 where q is the size of the tag set. These parsers can still be efficient when combined with heavy pruning and they produced state-of-the-art results for Chinese at the time. However, the approach does not scale well with respect to large tag sets as are common for morphologically rich languages.

Lee et al. (2011) define a graphical model for joint morphological prediction and parsing, modeling morphosyntactic interaction via dedicated factors in the model. They use loopy belief propagation (Smith and Eisner 2008) to do inference in the model and demonstrate

improvements over their own baselines. However, their results fell far behind the state-of-the-art of that time.

7.4.4 Comparison

We now compare our parser to the work that we described in the beginning of this section. The comparison is grouped into different aspects that we think are relevant for this work.

Architecture

The parser that we develop in this chapter is a graph-based dependency parser. The complexity issue that arises due to the global optimization is addressed by using inexact search (i.e., dual decomposition) that allows us to add the complexity of the sub-problems rather than multiplying them. In this respect, it is more efficient than an extension of the dynamic programming decoders in Li et al. (2011) to segmentation and morphological prediction. However, like their parsers ours requires pruning to keep the runtime tractable. The decoder is therefore more similar to the work by Smith and Eisner (2008) as they also use inexact search for inference and model different aspects of the problem with different dedicated factors.

The architecture of the lattice parser resembles the construction by Rush et al. (2010) for combining a part-of-speech tagger with a constituency parser in the sense that the lattice parser also combines a sequence model with a parsing model. However, due to the lattice structure, the lattice parser needs additional constraints to ensure an agreement between the outputs of the two models. These constraints are implemented as first-order logic factors, which can be handled efficiently by AD^3 (Martins et al. 2011b, 2015). Such constraints have previously been used to enforce structural requirements in systems for semantic parsing (Das et al. 2012), compressive summarization (Almeida and Martins 2013), and joint quotation attribution and coreference resolution (Almeida et al. 2014). We adopted the idea of using the Chu-Liu-Edmonds algorithm to ensure the tree structure from Koo et al. (2010) as well as using head automata to model second-order dependencies. Non-lattice dependency parsers that decode with dual decomposition algorithms are proposed in Koo et al. (2010) and Martins et al. (2013).

One important motivation for the graph-based parser in this chapter is to deal with segmentation ambiguity. This is done by searching for the combination of best parse tree and best path in a morphological lattice. The transition-based parsers can easily be adapted to solve this task as well. For example, the parser by Hatori et al. (2012) uses an append operation to form new words from single characters. This operation can be changed to split incoming words into smaller parts. A similar idea is implemented in Tratz (2013) for easy-first parsing.

Lattice parsing in a graph-based setting can also be done without dual decomposition. The chart-based lattice parsers that are used in the constituency parsers work similarly with Eisner's decoder. This technique has been applied for example in the context of machine translation (Carreras et al. 2008). The difference to our parser is then that our parser natively outputs non-projective structures whereas Eisner's decoder alone is restricted to projective structures.

Finally, recent work suggests that global optimization is unnecessary in dependency parsing in the first place. This assumption underlies the transition-based approach as well, namely that it may be enough to make local decisions since this will lead to an optimal tree anyway. The work by Zhang et al. (2014b) pushes this assumption to the limit by using a very simple decoder based on sampling that in turn allows them to use any kind of scoring function they want. In Zhang et al. (2015), they present an application of this approach to joint modeling of segmentation, morphological analysis, and dependency parsing for Arabic and Chinese, demonstrating state-of-the-art results. The scoring function in this parser permits any kind of global feature that one can think of. In principle, this can be modeled in our lattice parser with additional soft constraints for which the parser learns a weight. However, this would mean many more constraints that would likely lead to a slower parser. But more importantly, their work shows that a good feature model, i.e., a model that captures the relevant information well, is more important than a decoder that considers all possible combinations.

Lattices

One question that is answered differently by the different parsers is the question of how to represent morphological (and segmentation) ambiguity. Almost all of the constituency

parsers described above work with explicit morphological lattices that are created beforehand and serve as input to the parser. However, the transition-based parsers mostly do not adopt this setting but instead represent ambiguity by simply optimizing over all possible part-of-speech tags or morphological tags. Conceptually, this is not so different to a lattice since all possible part-of-speech tags can also be represented in a lattice structure. However, it makes a difference if the lattice is not complete, i.e., if the lattice only encodes the options that are known to the morphological dictionary from which it was created. In this case, the lattice restricts the possible options defined by a tag set to those that are represented in the dictionary. This feature is helpful because it the restriction lowers the amount of ambiguity which in turn restricts the search space of the parser. However, it can also lead to problems if the dictionary does not cover all phenomena or is missing particular analyses.

One reason why the transition-based parsers do not use explicit lattices seems to be the fact that they were developed either for Chinese or for morphologically rich languages that have no segmentation problem (with the exception of the parser by Tratz (2013)). The opposite is true for the constituency parsers for Hebrew and Arabic, which were from the start meant to deal with segmentation ambiguity. When the segmentation problem enters into the picture, explicit lattices (or at least a restriction of options via an external dictionary) are almost unavoidable because otherwise every possible segmentation at character level would need to be considered and the model would also need to learn what real words are.

However, transition-based parsers for Chinese still do not use explicit lattices even though they solve the word segmentation problem. This is due to two differences between the Chinese word segmentation problem and the one in Hebrew, Turkish, etc. One is that a word in Chinese is always going to be the concatenation of the characters that it is composed of. This is for example not the case for Hebrew. Goldberg and Tsarfaty (2008) describe *super-segmental morphology* as one problem of Hebrew parsing. Super-segmental morphology describes cases where the original surface form looks different than the concatenation of all its constituting morphemes. This phenomenon can be triggered for example by phonological processes that delete or change sounds. For this reason, a model for word segmentation must know more about the morphology of words than just the points where the surface string needs to be split. The other reason is that Chinese words consist of only few characters on average which results in less split points compared to a Latin-based script. Nonetheless, it is common that Chinese models for word segmentation

(including the joint models discussed above) use large lexicons of Chinese words in their feature set.

A question that is related to the use of lattices is whether to use scores of other models to weight the lattices or even prune them. Cohen and Smith (2007), Green and Manning (2010), Bohnet et al. (2013) use additional models to prescore the options that the parser can choose from when predicting the morphological information and find that this prescoring is important for the success of the overall model. On the other hand, Goldberg and Tsarfaty (2008) explicitly state that they want the parsing model to learn the decision on its own. Our lattice parser does not use scores from external models and even though it uses two models for scoring path and tree respectively, they are learned jointly. However, we believe that it makes a lot of sense to rely on the knowledge that is represented in specialized tools like part-of-speech taggers.

Features

One problem that was already observed by Hatori et al. (2011) is that if the parser is supposed to predict part-of-speech tags, it cannot use it as features for doing so. In particular, the transition-based parser that they design cannot access part-of-speech information on tokens that have not been shifted because it is the shift operation that assigns a part-of-speech tag to the token. However, information about the part-of-speech of the buffer tokens provides valuable context. Hatori et al. (2011) address this problem by introducing delayed features, which are evaluated once the information is available. Bohnet and Nivre (2012) and Bohnet et al. (2013) use the best prediction of a part-of-speech tagger as context features but the parser can overwrite this information later on. The same procedure is used by Li et al. (2011) who fix the part-of-speech tags for context tokens to the single best analysis from a part-of-speech tagger.

In our parser, we use the features of the path model to determine the best preceding and following token for each token in the lattice. These are then used as context features in the feature model. It is a slightly different approach because the path model is trained jointly with the parser and the best preceding and following token can therefore change during training.

Another technique that was reported to be useful is to give higher weight to the path features. Zhang et al. (2014a) describe how they weight the features for segmentation and part-of-speech tagging four times higher than the parsing features since otherwise they are outweighed by the large number of parsing features. We do not perform such weighting in our model. Exploring different weighting schemes may give some insights into the model, especially which part of the model is most important. Instead of weighting the path features higher, one could also make them very small. Koo et al. (2010) for example put almost the entire weight on the head automata, leaving only a small fraction of it in the Chu-Liu-Edmonds algorithm to facilitate tie breaking. In such a weighting scheme, the Chu-Liu-Edmonds (and in out case the path model) simply constitutes another constraint that enforces tree structure (or a consecutive path in the case of the path model). The actual optimization would take place in the head automata, which have access to path as well as structural features.

7.4.5 Evaluation

Even though the segmentation problem can be solved with similar means as part-of-speech tagging, the problems it poses to evaluation (see Section 7.2) show that it is not quite the same. One insight from evaluation is that segmentation is structurally connected to syntax and thus an incorrect segmentation makes the syntactic annotation meaningless. This is not the case for incorrect part-of-speech tags or incorrect morphological features. Although one can argue that an incorrect part-of-speech tag makes the interpretation of a sentence meaningless as well, it does not affect the overall structure of the sentence. It therefore seems to us that segmentation and syntax should be seen as a unit that makes sense only in combination.

Segmentation ambiguity also has repercussions on the training of pipeline architectures: When training pipelines for languages with an uncertain segmentation, jackknifing cannot be done with respect to the segmentation. The problem is that for predicted segmentation, a gold-standard dependency tree will not be available and a supervised parser cannot be trained. In this regard, segmentation is different to part-of-speech information. However, training via jackknifing usually gives better models since the later steps in the pipeline can learn which annotations from the lower levels can be trusted and which ones cannot, thus leading to less severe error propagation in pipelines. Joint models that also predict

segmentation do not suffer from error propagation, since they predict the segmentation themselves. While error propagation in pipelines could so far be remedied (to some extent) by applying jackknifing, including segmentation into the parsing task introduces a processing step to which jackknifing cannot be applied. For parsing languages with segmentation ambiguity, joint models are thus the best option that is currently at our disposal.

Chapter 8

Conclusion

In this dissertation, we presented experiments to demonstrate the shortcomings of a pipeline architecture for parsing languages with rich morphology. We developed parsers to test an alternative model that solves the task of morphological and syntactic analysis jointly and showed experimentally that they are superior to the pipeline. The purpose of this dissertation was to demonstrate that the reason why pipeline architectures work well for English is not because pipelines are good models for parsing *per se*, but because English is a special case among the languages in the world with respect to morphology. In linguistic theory, syntax cannot and is not considered independent of morphology, and parsing should not be thought independent of morphological analysis either. In the following, we summarize the main contributions.

Syncretism. Syncretism introduces ambiguity because relevant grammatical information is not fully specified morphologically. The syntactic context of a syncretic word form can give the relevant information to properly resolve the ambiguity but in languages with free word order, this context can be anywhere in the sentence. Morphosyntactic mechanisms like agreement can hold over long distances in the sentence even though syntactically they may be local. A pipeline model disconnects the prediction of syntactic structure from the prediction of morphological information and thus limits access to syntactic structure for predicting morphology. We demonstrated this with an analysis that shows how parsing errors with respect to argument functions can directly be related

to mistakes of the morphological prediction due to syncretism. We then showed how access to the full syntactic structure of a sentence improves automatic morphological analysis. In particular, these improvements are orthogonal to improvements from large-scale lexicons showing that the syntactic context of a word contributes information that cannot be obtained otherwise.

Languages like Czech and German show an intricate interaction between morphology and syntax. These languages rely (among other things) on syntactic information and morphosyntactic rules like agreement to resolve syncretism. However, our experiments have also shown that parsing models for Hungarian do not suffer as much from error propagation as Czech and German. This is because there is almost no syncretism in Hungarian. It seems that jointly modeling morphology and syntax for Hungarian may not need to be as necessary as it is for German and Czech. Even though joint models subsume pipelines, pipelines are more efficient and may be suitable for Hungarian. At the end of the day, this shows that using morphology to encode grammatical information is not the problem in itself, but it is the specific ways these mechanisms are implemented in the individual languages that causes problems for our standard models.

Restricting the Search Space. The constraint parser that we presented in Chapter 6 models morphosyntax by imposing constraints on the search space of the parser. These constraints import a linguistic aspect into the parser's decoder by defining a well-formed output not just by imposing a tree structure but also the requirement that they do not violate the morphosyntactic rules of the language. The underlying statistical model, on the other hand, remains untouched by the constraints. Although the parser performs better on argument functions compared to an unconstrained version, the statistical model itself does not model morphology and syntax jointly.

The fact that we can use these constraints to improve the performance of the parser for argument functions also shows that the parsing models still have a deficit with respect to modeling argument frames. Restricting the search space is a step in the right direction because it excludes for example structures where the same verb has two different subjects, but what we would like to see is that also the statistical model has a representation of the argument frame of a verb. We would like the model to select an argument as the subject because it considered the other potential candidates and decided that among these this one is the best choice. To do this, it must however see all candidates together rather

than making a decision for each one individually and independent of the other options. Sibling models, which model two dependents of the same head together, are the closest to model the interaction between different arguments of the same verb. But they only consider two dependents, often ones that are next to each other, and they almost never consider the dependency relations of both siblings. Such design decisions are often made out of efficiency considerations, and for good reasons. However, free word order and non-configurationality, which are found aplenty in morphologically rich languages, make it necessary to reconsider these decisions.

Joint Modeling. The second parser that we presented is a graph-based parser that operates on lattices and jointly predicts a segmentation of the words into smaller units, their morphological features, and the syntactic structure connecting them. It is an implementation of the joint model that we have argued for in this dissertation and our experiments with it show that joint decoding indeed outperforms an identical pipeline model that only differs in the fact that it separates morphological and syntactic analysis. One challenge for joint models is their increased complexity. We addressed this issue in the lattice parser by decomposing the problem into efficiently solvable sub-problems and an agreement constraint. Inexact optimization via dual decomposition then finds the best dependency structure that fulfills the agreement constraint.

The experiments on Turkish and the subsequent analysis show that the improvements due to the joint modeling almost exclusively occur in the segmentation and morphological analysis while the parsing quality stays at the same level as for the pipeline model. We observed a similar effect in a side experiment in Chapter 5, where we automatically predicted morphology while having access to the full syntactic structure of the sentence. When using the improved morphological prediction as input to another parsing step, the parsing quality stayed the same. It seems like the morphological analysis can profit from the parsing but not necessarily the other way around. This should be further investigated by testing the parser on more languages, especially also languages without the segmentation problem. This would also facilitate a meaningful comparison to other joint parsers.

There are also several aspects of the lattice parser itself that are worth further investigation. Like previously published joint parsers there is the question of how to represent the context of words in the feature model. In our case, it is the question of what the next/previous

word in a lattice should be and how it should be represented. Another aspect is to adapt the morphosyntactic constraints from the constraint parser and integrate them into the lattice parser.

Appendix A

Full Feature Set of the Constraint Parser

We use functions form(t), lemma(t), and pos(t) to extract the surface form, lemma, and part-of-speech tag of a token t, respectively. mfeats(t) returns the set of individual morphological features, e.g. {`case=nom, number=pl, gender=fem`}. dist(d, h) returns the distance between the dependent and the head. Context tokens are indicated by subscripts, e.g. d_{+1} denotes the token immediately to the right of the dependent and h_{-2} denotes the token that is two tokens to the left of the head. Conjunction of basic features is marked with $\oplus$.

form(h)	lemma(h)
pos(h)	form$(h) \oplus$ pos(h)
lemma$(h) \oplus$ pos(h)	form(d)
lemma(d)	pos(d)
form$(d) \oplus$ pos(d)	lemma$(d) \oplus$ pos(d)
form$(h) \oplus$ form(d)	lemma$(h) \oplus$ lemma(d)
pos$(h) \oplus$ pos(d)	form$(h) \oplus$ form$(d) \oplus$ pos$(h) \oplus$ pos(d)
form$(h) \oplus$ pos$(h) \oplus$ pos(d)	form$(d) \oplus$ pos$(h) \oplus$ pos(d)
form$(h) \oplus$ form$(d) \oplus$ pos(h)	form$(h) \oplus$ form$(d) \oplus$ pos(d)
lemma$(h) \oplus$ pos$(h) \oplus$ pos(d)	lemma$(d) \oplus$ pos$(h) \oplus$ pos(d)
lemma$(h) \oplus$ lemma$(d) \oplus$ pos$(h) \oplus$ pos(d)	lemma$(h) \oplus$ lemma$(d) \oplus$ pos(h)
lemma$(h) \oplus$ lemma$(d) \oplus$ pos(d)	dist$(d,h) \oplus$ form$(h) \oplus$ form(d)
dist$(d,h) \oplus$ pos$(h) \oplus$ pos(d)	dist$(d,h) \oplus$ lemma$(h) \oplus$ lemma(d)
dist$(d,h) \oplus$ pos$(h) \oplus$ form(d)	dist$(d,h) \oplus$ form$(h) \oplus$ pos(d)
dist$(d,h) \oplus$ pos$(h) \oplus$ pos$(d) \oplus$ pos(h_{+1})	dist$(d,h) \oplus$ pos$(h) \oplus$ pos$(d) \oplus$ pos(h_{-1})
dist$(d,h) \oplus$ pos$(h) \oplus$ pos$(d) \oplus$ pos(d_{+1})	dist$(d,h) \oplus$ pos$(h) \oplus$ pos$(d) \oplus$ pos(d_{-1})
pos$(h) \oplus$ pos$(d) \oplus$ pos(h_{+1})	pos$(h) \oplus$ pos$(d) \oplus$ pos(h_{-1})
pos$(h) \oplus$ pos$(d) \oplus$ pos(d_{+1})	pos$(h) \oplus$ pos$(d) \oplus$ pos(d_{-1})
pos$(h) \oplus$ pos$(d) \oplus$ pos$(h_{+1}) \oplus$ pos(d_{+1})	pos$(h) \oplus$ pos$(d) \oplus$ pos$(h_{+1}) \oplus$ pos(d_{-1})
pos$(h) \oplus$ pos$(d) \oplus$ pos$(h_{-1}) \oplus$ pos(d_{+1})	pos$(h) \oplus$ pos$(d) \oplus$ pos$(h_{-1}) \oplus$ pos(d_{-1})
pos$(d) \oplus$ pos(d_{+1})	pos$(d) \oplus$ pos$(d_{+1}) \oplus$ pos(d_{+2})
form$(d) \oplus$ pos$(d) \oplus$ pos$(d_{+1}) \oplus$ pos(d_{+2})	pos$(d) \oplus$ pos(d_{-1})
pos$(d) \oplus$ pos$(d_{-1}) \oplus$ pos(d_{-2})	form$(d) \oplus$ pos$(d) \oplus$ pos$(d_{-1}) \oplus$ pos(d_{-2})
pos$(d) \oplus$ pos$(d_{+1}) \oplus$ pos(d_{-1})	form$(d) \oplus$ pos$(d) \oplus$ pos$(d_{+1}) \oplus$ pos(d_{-1})
pos$(h) \oplus$ pos(h_{+1})	pos$(h) \oplus$ pos$(h_{+1}) \oplus$ pos(h_{+2})
form$(h) \oplus$ pos$(h) \oplus$ pos$(h_{+1}) \oplus$ pos(h_{+2})	pos$(h) \oplus$ pos(h_{-1})
pos$(h) \oplus$ pos$(h_{-1}) \oplus$ pos(h_{-2})	form$(h) \oplus$ pos$(h) \oplus$ pos$(h_{-1}) \oplus$ pos(h_{-2})
pos$(h) \oplus$ pos$(h_{+1}) \oplus$ pos(h_{-1})	form$(h) \oplus$ pos$(h) \oplus$ pos$(h_{+1}) \oplus$ pos(h_{-1})
form$(d) \oplus$ form(d_{-1})	form$(d) \oplus$ form$(d_{-1}) \oplus$ form(d_{-2})
form$(h) \oplus$ form(h_{-1})	form$(h) \oplus$ form$(h_{-1}) \oplus$ form(h_{-2})
form$(d) \oplus$ form(d_{+1})	form$(d) \oplus$ form$(d_{+1}) \oplus$ form(d_{+2})
form$(h) \oplus$ form(h_{+1})	form$(h) \oplus$ form$(h_{+1}) \oplus$ form(h_{+2})
form$(d) \oplus$ form$(d_{-1}) \oplus$ form(d_{+1})	form$(h) \oplus$ form$(h_{-1}) \oplus$ form(h_{+1})
pos$(h) \oplus$ pos$(d) \oplus$ pos$(h_{+1}) \oplus$ pos(h_{+2})	pos$(h) \oplus$ pos$(d) \oplus$ pos$(h_{-1}) \oplus$ pos(h_{-2})
pos$(h) \oplus$ pos$(d) \oplus$ pos$(d_{+1}) \oplus$ pos(d_{+2})	pos$(h) \oplus$ pos$(d) \oplus$ pos$(d_{-1}) \oplus$ pos(d_{-2})
pos$(d) \oplus$ pos(h_{+2})	pos$(h) \oplus$ form(d_{-1})
pos$(h) \oplus$ pos(d_{-1})	pos$(h) \oplus$ form$(d_{-1}) \oplus$ form(d_{-2})
pos$(h) \oplus$ pos$(d_{-1}) \oplus$ pos(d_{-2})	pos(i) $\forall i, d < i < h$
form(i) $\forall i, d < i < h$	pos$(h) \oplus$ pos$(d) \oplus$ pos(i) $\forall i, d < i < h$
pos$(h) \oplus$ pos$(d) \oplus$ form(i) $\forall i, d < i < h$	lemma$(h) \oplus$ lemma$(d) \oplus$ form(i) $\forall i, d < i < h$
$m \oplus m'$ $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats$(h) \times$ mfeats(d)	
pos$(h) \oplus m \oplus m'$ $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats$(h) \times$ mfeats(d)	
pos$(d) \oplus m \oplus m'$ $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats$(h) \times$ mfeats(d)	
lemma$(h) \oplus m \oplus m'$ $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats$(h) \times$ mfeats(d)	
lemma$(d) \oplus m \oplus m'$ $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats$(h) \times$ mfeats(d)	

Table A.1: Arc features.

form$(d) \oplus$ form(s)	pos$(d) \oplus$ pos(s)
lemma$(d) \oplus$ lemma(s)	form$(d) \oplus$ pos(s)
pos$(d) \oplus$ form(s)	form$(h) \oplus$ form$(d) \oplus$ form(s)
pos$(h) \oplus$ pos$(d) \oplus$ pos(s)	lemma$(h) \oplus$ lemma$(d) \oplus$ lemma(s)
pos$(h) \oplus$ pos$(d) \oplus$ form(s)	form$(h) \oplus$ form$(d) \oplus$ pos(s)
pos$(h) \oplus$ form$(d) \oplus$ form(s)	form$(h) \oplus$ pos$(d) \oplus$ pos(s)
form$(d) \oplus$ pos$(d) \oplus$ form$(s) \oplus$ pos(s)	lemma$(d) \oplus$ pos$(d) \oplus$ lemma$(s) \oplus$ pos(s)
form$(d) \oplus$ form$(s) \oplus$ form(s_{+1})	form$(d) \oplus$ form$(s) \oplus$ form(s_{-1})
form$(d) \oplus$ form$(s) \oplus$ form(d_{+1})	form$(d) \oplus$ form$(s) \oplus$ form(d_{-1})
pos$(d) \oplus$ pos$(s) \oplus$ pos(s_{+1})	pos$(d) \oplus$ pos$(s) \oplus$ pos(s_{-1})
pos$(d) \oplus$ pos$(s) \oplus$ pos(d_{+1})	pos$(d) \oplus$ pos$(s) \oplus$ pos(d_{-1})
form$(d) \oplus$ form$(s) \oplus$ form(s_{+2})	form$(d) \oplus$ form$(s) \oplus$ form(s_{-2})
form$(d) \oplus$ form$(s) \oplus$ form(d_{+2})	form$(d) \oplus$ form$(s) \oplus$ form(d_{-2})
pos$(d) \oplus$ pos$(s) \oplus$ pos(s_{+2})	pos$(d) \oplus$ pos$(s) \oplus$ pos(s_{-2})
pos$(d) \oplus$ pos$(s) \oplus$ pos(d_{+2})	pos$(d) \oplus$ pos$(s) \oplus$ pos(d_{-2})
dist$(d, s) \oplus$ form$(d) \oplus$ form(s)	dist$(d, s) \oplus$ pos$(d) \oplus$ pos(s)
dist$(d, s) \oplus$ lemma$(d) \oplus$ lemma(s)	dist$(d, s) \oplus$ form$(d) \oplus$ pos(s)
dist$(d, s) \oplus$ pos$(d) \oplus$ form(s)	dist$(d, s) \oplus$ form$(h) \oplus$ form$(d) \oplus$ form(s)
dist$(d, s) \oplus$ pos$(h) \oplus$ pos$(d) \oplus$ pos(s)	

Table A.2: Second-order features. For sibling features, s stands for the sibling, while for grandchild features it stands for the grandparent node.

Appendix B

Full Feature Set of the Lattice Parser

We use functions form(t), lemma(t), and pos(t) to extract the surface form, lemma, and part-of-speech tag of a token t, respectively. mtag(t) extracts the full morphological description of a token as one string, e.g. `case=nom|number=pl|gender=fem`. mfeats(t) instead returns the set of individual morphological features, e.g. {`case=nom`, `number=pl`, `gender=fem`}. $\oplus$ denotes conjunction of basic features.

B.1 Path Features

The path model factors over token bigrams, features are therefore extracted for each token bigram $\langle t, t' \rangle$ in the lattice (see Section 7.1.1). Table B.1 shows the feature set used to score token bigrams. All templates in Table B.1 are combined with a binary feature that indicates whether tokens t and t' are part of the same (space-delimited) word or not. Token bigrams involving the first/last token in the lattice are padded with special symbols that represent the boundaries.

B.2 First-order Features

The first-order features are extracted on single arcs between two tokens, the dependent d and the head h. Table B.4 shows the list of features for arcs. The function wsform(d) returns the surface form of the space-delimited word that d is a part of. dist(d, h) returns the length of the shortest path between tokens d and h in the lattice (see Section 7.1.4). is_ws_suffix(d) is a binary function that indicates whether a token is a suffix of a space-delimited word, and adjacent(d, h) returns whether tokens d and h are adjacent in the lattice, i.e. whether the target state of d is the source state of h. All first-order features (including the Turkish-specific features) are combined with the direction of the arc and the dependency label.

For context features, the parser extracts features from each token immediately to the left and right of the dependent and the head. *left* means that the context token's target state is the same as the source state of the current token in the lattice, *right* means that the source state of the context token is the target state of the current token. When using latent context, there is only one context token to each side of the current token (see Section 7.1.4). Table B.2 shows the features exemplarily for the right context token d_r of the dependent.

B.3 Turkish-specific Features

Table B.3 shows features that capture morphological phenomena in Turkish. The basic functions extract specific features from the morphological description of a token. These features are specific to the annotation scheme of the Turkish treebank. Functions subtype(t), case(t), agr(t), tam(t), voice(t), pcase(t), pagr(t), and cop(t) extract the part-of-speech subtype, the case value, the agreement features (number and person), tense-aspect-mood values, voice, the case value governed by a preposition, agreement values on the possessum, and the copula status of a word, respectively. Two binary functions, case_agrees(d, h) and agr_agrees(d, h), return whether the case values and the agreement values of two tokens are the same. Features are extracted on a arc for dependent d and head h.

form(t)	form(t') $\oplus$ pos(t')	pos(t) $\oplus$ mtag(t')
form(t')	form(t) $\oplus$ pos(t')	pos(t') $\oplus$ mtag(t)
lemma(t)	form(t') $\oplus$ pos(t)	form(t) $\oplus$ pos(t) $\oplus$ mtag(t)
lemma(t')	form(t) $\oplus$ mtag(t)	form(t') $\oplus$ pos(t') $\oplus$ mtag(t')
pos(t)	form(t') $\oplus$ mtag(t')	form(t') $\oplus$ pos(t) $\oplus$ mtag(t)
pos(t')	form(t) $\oplus$ mtag(t')	form(t) $\oplus$ pos(t') $\oplus$ mtag(t')
mtag(t)	form(t') $\oplus$ mtag(t)	lemma(t) $\oplus$ pos(t) $\oplus$ mtag(t)
mtag(t')	lemma(t) $\oplus$ pos(t)	lemma(t') $\oplus$ pos(t') $\oplus$ mtag(t')
form(t) $\oplus$ form(t')	lemma(t') $\oplus$ pos(t')	lemma(t') $\oplus$ pos(t) $\oplus$ mtag(t)
lemma(t) $\oplus$ lemma(t')	lemma(t) $\oplus$ pos(t')	lemma(t) $\oplus$ pos(t') $\oplus$ mtag(t')
pos(t) $\oplus$ pos(t')	lemma(t') $\oplus$ pos(t)	pos(t) $\oplus$ pos(t') $\oplus$ mtag(t)
mtag(t) $\oplus$ mtag(t')	lemma(t) $\oplus$ mtag(t)	pos(t) $\oplus$ pos(t') $\oplus$ mtag(t')
form(t) $\oplus$ lemma(t)	lemma(t') $\oplus$ mtag(t')	m $\forall m, m \in$ mfeats(t)
form(t') $\oplus$ lemma(t')	lemma(t) $\oplus$ mtag(t')	m $\forall m, m \in$ mfeats(t')
form(t) $\oplus$ lemma(t')	lemma(t') $\oplus$ mtag(t)	$m \oplus$ pos(t) $\forall m, m \in$ mfeats(t)
form(t') $\oplus$ lemma(t)	pos(t) $\oplus$ mtag(t)	$m \oplus$ pos(t') $\forall m, m \in$ mfeats(t')
form(t) $\oplus$ pos(t)	pos(t') $\oplus$ mtag(t')	
$m \oplus m'$ $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(t) $\times$ mfeats(t')		

Table B.1: Path features.

B.4 Second-order Features

Second-order features are extracted for sibling and grandparent relations. Tables B.5 and B.6 show the features involving sibling tokens (s) and grandparent tokens (g), respectively. Like the first-order features, all second-order features are combined with the direction of the arc and the dependency label.

form(d_r)	lemma(d_r) ⊕ lemma(d) ⊕ lemma(h)
lemma(d_r)	pos(d_r) ⊕ pos(d) ⊕ pos(h)
pos(d_r)	mtag(d_r) ⊕ mtag(d) ⊕ mtag(h)
mtag(d_r)	form(d_r) ⊕ form(d) ⊕ mtag(h)
form(d_r) ⊕ form(d)	lemma(d_r) ⊕ lemma(d) ⊕ mtag(h)
lemma(d_r) ⊕ lemma(d)	form(d_r) ⊕ form(h) ⊕ mtag(d)
pos(d_r) ⊕ pos(d)	lemma(d_r) ⊕ lemma(h) ⊕ mtag(d)
mtag(d_r) ⊕ mtag(d)	form(d_r) ⊕ mtag(d) ⊕ mtag(h)
form(d_r) ⊕ form(h)	lemma(d_r) ⊕ mtag(d) ⊕ mtag(h)
lemma(d_r) ⊕ lemma(h)	mtag(d_r) ⊕ form(d) ⊕ mtag(h)
pos(d_r) ⊕ pos(h)	mtag(d_r) ⊕ lemma(d) ⊕ mtag(h)
mtag(d_r) ⊕ mtag(h)	mtag(d_r) ⊕ form(h) ⊕ mtag(d)
form(d_r) ⊕ form(d) ⊕ form(h)	mtag(d_r) ⊕ lemma(h) ⊕ mtag(d)

Table B.2: First-order context features, shown for the right context d_r of the dependent d.

pos(d) ⊕ subtype(d)	pos(d) ⊕ pos(h) ⊕ case_agrees(d, h)
pos(h) ⊕ subtype(h)	pos(d) ⊕ pos(h) ⊕ agr_agrees(d, h)
pos(d) ⊕ pos(h) ⊕ subtype(d)	pos(d) ⊕ pos(h) ⊕ case(d) ⊕ case(h)
pos(d) ⊕ pos(h) ⊕ subtype(h)	pos(d) ⊕ pos(h) ⊕ agr(d) ⊕ agr(h)
pos(d) ⊕ pos(h) ⊕ case(d)	pos(d) ⊕ pos(h) ⊕ agr(d) ⊕ pagr(h)
pos(d) ⊕ pos(h) ⊕ tam(h)	pos(d) ⊕ pos(h) ⊕ case(d) ⊕ pcase(h)
pos(d) ⊕ pos(h) ⊕ cop(h)	pos(d) ⊕ pos(h) ⊕ case(d) ⊕ agr(d) ⊕ pagr(h)
pos(d) ⊕ pos(h) ⊕ voice(h)	

Table B.3: First-order features to capture Turkish morphology.

form(d)	form(d) $\oplus$ form(h)	wsform(d) $\oplus$ pos(d) $\oplus$ mtag(d)
lemma(d)	form(d) $\oplus$ lemma(h)	form(h) $\oplus$ pos(h) $\oplus$ mtag(h)
pos(d)	form(d) $\oplus$ pos(h)	lemma(h) $\oplus$ pos(h) $\oplus$ mtag(h)
mtag(d)	form(d) $\oplus$ mtag(h)	wsform(h) $\oplus$ pos(h) $\oplus$ mtag(h)
wsform(d)	form(d) $\oplus$ wsform(h)	is_ws_suffix(d) $\oplus$ adjacent(d, h)
form(h)	lemma(d) $\oplus$ form(h)	dist(d, h)
lemma(h)	lemma(d) $\oplus$ lemma(h)	dist(d, h) $\oplus$ form(d) $\oplus$ form(h)
pos(h)	lemma(d) $\oplus$ pos(h)	dist(d, h) $\oplus$ lemma(d) $\oplus$ lemma(h)
mtag(h)	lemma(d) $\oplus$ mtag(h)	dist(d, h) $\oplus$ pos(d) $\oplus$ pos(h)
wsform(h)	lemma(d) $\oplus$ wsform(h)	dist(d, h) $\oplus$ mtag(d) $\oplus$ mtag(h)
form(d) $\oplus$ lemma(d)	pos(d) $\oplus$ form(h)	dist(d, h) $\oplus$ wsform(d) $\oplus$ wsform(h)
form(d) $\oplus$ pos(d)	pos(d) $\oplus$ lemma(h)	$m \oplus$ form(d) $\forall m, m \in$ mfeats(h)
form(d) $\oplus$ mtag(d)	pos(d) $\oplus$ pos(h)	$m \oplus$ lemma(d) $\forall m, m \in$ mfeats(h)
form(d) $\oplus$ wsform(d)	pos(d) $\oplus$ mtag(h)	$m \oplus$ pos(d) $\forall m, m \in$ mfeats(h)
form(h) $\oplus$ lemma(h)	pos(d) $\oplus$ wsform(h)	$m \oplus$ wsform(d) $\forall m, m \in$ mfeats(h)
form(h) $\oplus$ pos(h)	mtag(d) $\oplus$ form(h)	$m \oplus$ form(h) $\forall m, m \in$ mfeats(d)
form(h) $\oplus$ mtag(h)	mtag(d) $\oplus$ lemma(h)	$m \oplus$ lemma(h) $\forall m, m \in$ mfeats(d)
form(h) $\oplus$ wsform(h)	mtag(d) $\oplus$ pos(h)	$m \oplus$ pos(h) $\forall m, m \in$ mfeats(d)
lemma(d) $\oplus$ pos(d)	mtag(d) $\oplus$ mtag(h)	$m \oplus$ wsform(h) $\forall m, m \in$ mfeats(d)
lemma(d) $\oplus$ mtag(d)	mtag(d) $\oplus$ wsform(h)	$m \; \forall m, m \in$ mfeats(d)
lemma(d) $\oplus$ wsform(d)	wsform(d) $\oplus$ form(h)	$m \oplus$ form(d) $\forall m, m \in$ mfeats(d)
lemma(h) $\oplus$ pos(h)	wsform(d) $\oplus$ lemma(h)	$m \oplus$ lemma(d) $\forall m, m \in$ mfeats(d)
lemma(h) $\oplus$ mtag(h)	wsform(d) $\oplus$ pos(h)	$m \oplus$ pos(d) $\forall m, m \in$ mfeats(d)
lemma(h) $\oplus$ wsform(h)	wsform(d) $\oplus$ mtag(h)	$m \oplus$ wsform(d) $\forall m, m \in$ mfeats(d)
pos(d) $\oplus$ mtag(d)	wsform(d) $\oplus$ wsform(h)	$m \; \forall m, m \in$ mfeats(h)
pos(d) $\oplus$ wsform(d)	form(d) $\oplus$ pos(h) $\oplus$ mtag(h)	$m \oplus$ form(h) $\forall m, m \in$ mfeats(h)
pos(h) $\oplus$ mtag(h)	lemma(d) $\oplus$ pos(h) $\oplus$ mtag(h)	$m \oplus$ lemma(h) $\forall m, m \in$ mfeats(h)
pos(h) $\oplus$ wsform(h)	wsform(d) $\oplus$ pos(h) $\oplus$ mtag(h)	$m \oplus$ pos(h) $\forall m, m \in$ mfeats(h)
wsform(d) $\oplus$ mtag(d)	form(d) $\oplus$ pos(d) $\oplus$ mtag(d)	$m \oplus$ wsform(h) $\forall m, m \in$ mfeats(h)
wsform(h) $\oplus$ mtag(h)	lemma(d) $\oplus$ pos(d) $\oplus$ mtag(d)	
$m \oplus m'$ $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) $\times$ mfeats(h)		
$m \oplus m' \oplus$ form(d) $\oplus$ form(h) $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) $\times$ mfeats(h)		
$m \oplus m' \oplus$ lemma(d) $\oplus$ lemma(h) $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) $\times$ mfeats(h)		
$m \oplus m' \oplus$ pos(d) $\oplus$ pos(h) $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) $\times$ mfeats(h)		
$m \oplus m' \oplus$ wsform(d) $\oplus$ wsform(h) $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) $\times$ mfeats(h)		

Table B.4: First-order arc features.

form(s)	lemma(d) $\oplus$ lemma(h) $\oplus$ lemma(s)
lemma(s) $\oplus$ pos(s) $\oplus$ mtag(s)	wsform(d) $\oplus$ wsform(h) $\oplus$ wsform(s)
wsform(s) $\oplus$ pos(s) $\oplus$ mtag(s)	form(d) $\oplus$ form(h) $\oplus$ pos(s)
form(d) $\oplus$ form(s)	pos(d) $\oplus$ form(h) $\oplus$ form(s)
lemma(d) $\oplus$ form(s)	lemma(d) $\oplus$ lemma(h) $\oplus$ pos(s)
lemma(d) $\oplus$ pos(s)	lemma(d) $\oplus$ form(h) $\oplus$ pos(s)
lemma(d) $\oplus$ mtag(s)	form(d) $\oplus$ form(h) $\oplus$ mtag(s)
pos(d) $\oplus$ form(s)	pos(d) $\oplus$ pos(h) $\oplus$ mtag(s)
mtag(d) $\oplus$ pos(s)	$m \oplus$ form(s) $\forall m, m \in$ mfeats(s)
mtag(d) $\oplus$ mtag(s)	$m \oplus$ form(d) $\forall m, m \in$ mfeats(s)
dist(d, s)	$m \oplus$ pos(d) $\forall m, m \in$ mfeats(s)
dist(d, s) $\oplus$ lemma(d) $\oplus$ lemma(s)	$m \oplus$ wsform(d) $\forall m, m \in$ mfeats(s)
$m \oplus m' \oplus$ form(d) $\oplus$ form(s) $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) $\times$ mfeats(s)	
$m \oplus m' \oplus$ wsform(d) $\oplus$ wsform(s) $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) $\times$ mfeats(s)	
pos(s_l)	mtag(d) $\oplus$ mtag(s) $\oplus$ mtag(s_l)
mtag(s_l)	mtag(d) $\oplus$ mtag(s) $\oplus$ form(s_l)
pos(s) $\oplus$ pos(s_l)	mtag(d) $\oplus$ mtag(s) $\oplus$ lemma(s_l)
lemma(d) $\oplus$ lemma(s_l)	mtag(d) $\oplus$ form(s) $\oplus$ mtag(s_l)
form(s_r)	lemma(d) $\oplus$ lemma(s) $\oplus$ lemma(s_l)
form(s) $\oplus$ form(s_r)	form(d) $\oplus$ mtag(s) $\oplus$ form(s_r)
lemma(d) $\oplus$ lemma(s_r)	mtag(d) $\oplus$ lemma(s) $\oplus$ lemma(s_r)
mtag(s) $\oplus$ mtag(s_r)	mtag(d) $\oplus$ mtag(s) $\oplus$ form(s_r)
pos(d) $\oplus$ pos(s) $\oplus$ pos(s_r)	lemma(d) $\oplus$ mtag(s) $\oplus$ mtag(s_r)
	lemma(d) $\oplus$ mtag(s) $\oplus$ lemma(s_r)

Table B.5: Sibling features.

form(g)	form(d) ⊕ pos(g) ⊕ mtag(g)
form(d) ⊕ form(g)	lemma(d) ⊕ lemma(h) ⊕ lemma(g)
form(d) ⊕ mtag(g)	pos(d) ⊕ pos(h) ⊕ pos(g)
form(d) ⊕ wsform(g)	mtag(d) ⊕ mtag(h) ⊕ mtag(g)
lemma(d) ⊕ lemma(g)	pos(d) ⊕ form(h) ⊕ form(g)
lemma(d) ⊕ mtag(g)	pos(d) ⊕ lemma(h) ⊕ lemma(g)
lemma(d) ⊕ wsform(g)	lemma(d) ⊕ lemma(h) ⊕ pos(g)
pos(d) ⊕ lemma(g)	lemma(d) ⊕ form(h) ⊕ pos(g)
pos(d) ⊕ pos(g)	lemma(d) ⊕ form(h) ⊕ lemma(g)
pos(d) ⊕ mtag(g)	lemma(d) ⊕ lemma(h) ⊕ form(g)
pos(d) ⊕ wsform(g)	form(d) ⊕ form(h) ⊕ mtag(g)
mtag(d) ⊕ wsform(g)	lemma(d) ⊕ lemma(h) ⊕ mtag(g)
wsform(d) ⊕ lemma(g)	m $\forall m, m \in$ mfeats(g)
wsform(d) ⊕ pos(g)	$m \oplus$ pos(d) $\forall m, m \in$ mfeats(g)
wsform(d) ⊕ mtag(g)	$m \oplus$ form(d) $\forall m, m \in$ mfeats(g)
wsform(d) ⊕ wsform(g)	$m \oplus$ lemma(d) $\forall m, m \in$ mfeats(g)
wsform(d) ⊕ wsform(g)	$m \oplus$ pos(d) $\forall m, m \in$ mfeats(g)
dist(d, g)	$m \oplus$ form(g) $\forall m, m \in$ mfeats(d)
dist(d, g) ⊕ form(d) ⊕ form(g)	$m \oplus$ lemma(g) $\forall m, m \in$ mfeats(d)
dist(d, g) ⊕ lemma(d) ⊕ lemma(g)	$m \oplus$ pos(g) $\forall m, m \in$ mfeats(d)
form(g) ⊕ pos(g) ⊕ mtag(g)	$m \oplus$ wsform(g) $\forall m, m \in$ mfeats(d)
lemma(g) ⊕ pos(g) ⊕ mtag(g)	$m \oplus$ wsform(d) $\forall m, m \in$ mfeats(g)
$m \oplus m'$ $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) × mfeats(g)	
$m \oplus m' \oplus$ form(d) ⊕ form(g) $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) × mfeats(g)	
$m \oplus m' \oplus$ lemma(d) ⊕ lemma(g) $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) × mfeats(g)	
$m \oplus m' \oplus$ pos(d) ⊕ pos(g) $\forall \langle m, m' \rangle, \langle m, m' \rangle \in$ mfeats(d) × mfeats(g)	
lemma(g_l)	form(d) ⊕ mtag(g) ⊕ form(g_l)
pos(g_l)	lemma(d) ⊕ mtag(g) ⊕ lemma(g_l)
form(g_r)	mtag(d) ⊕ lemma(g) ⊕ lemma(g_l)
pos(g_r)	mtag(d) ⊕ mtag(g) ⊕ form(g_l)
form(g) ⊕ form(g_l)	mtag(d) ⊕ mtag(g) ⊕ lemma(g_l)
form(d) ⊕ form(g_l)	mtag(d) ⊕ form(g) ⊕ mtag(g_l)
lemma(d) ⊕ lemma(g_l)	form(d) ⊕ form(g) ⊕ form(g_r)
mtag(g) ⊕ mtag(g_l)	lemma(d) ⊕ lemma(g) ⊕ lemma(g_r)
form(d) ⊕ form(g_r)	pos(d) ⊕ pos(g) ⊕ pos(g_r)
form(g) ⊕ form(g_r)	mtag(d) ⊕ mtag(g) ⊕ mtag(g_r)
lemma(d) ⊕ lemma(g_r)	lemma(d) ⊕ mtag(g) ⊕ lemma(g_r)
lemma(g) ⊕ lemma(g_r)	mtag(d) ⊕ form(g) ⊕ form(g_r)
pos(g) ⊕ pos(g_r)	mtag(d) ⊕ mtag(g) ⊕ form(g_r)
mtag(d) ⊕ mtag(g_r)	mtag(d) ⊕ mtag(g) ⊕ lemma(g_r)
mtag(g) ⊕ mtag(g_r)	form(d) ⊕ mtag(g) ⊕ mtag(g_r)
pos(d) ⊕ pos(g) ⊕ pos(g_l)	mtag(d) ⊕ form(g) ⊕ mtag(g_r)
mtag(d) ⊕ mtag(g) ⊕ mtag(g_l)	

Table B.6: Grandparent features.

Bibliography

Almeida, M. and Martins, A. (2013). Fast and Robust Compressive Summarization with Dual Decomposition and Multi-Task Learning. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 196–206, Sofia, Bulgaria. Association for Computational Linguistics. 120, 153

Almeida, M. S. C., Almeida, M. B., and Martins, A. F. T. (2014). A Joint Model for Quotation Attribution and Coreference Resolution. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 39–48, Gothenburg, Sweden. Association for Computational Linguistics. 120, 153

Alshawi, H. (1996). Head Automata and Bilingual Tiling: Translation with Minimal Representations (Invited Talk). In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 167–176, Santa Cruz, California, USA. Association for Computational Linguistics. 121

Attardi, G. (2006). Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 166–170, New York City. Association for Computational Linguistics. 17

Baerman, M. and Brown, D. (2013a). Case syncretism. In Dryer, M. S. and Haspelmath, M., editors, *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig. Available online at http://wals.info/chapter/28, accessed on 2014-05-06. 36

Baerman, M. and Brown, D. (2013b). Syncretism in verbal person/number marking. In Dryer, M. S. and Haspelmath, M., editors, *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig. Available online at http://wals.info/chapter/29, accessed on 2014-05-06. 36

Baerman, M., Brown, D., and Corbett, G. G. (2005). *The Syntax-Morphology Interface: A Study of Syncretism*. Cambridge Studies in Linguistics. Cambridge University Press. 35, 36, 89

Bar-Haim, R., Sima'an, K., and Winter, Y. (2005). Choosing an Optimal Architecture for Segmentation and POS-Tagging of Modern Hebrew. In *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pages 39–46, Ann Arbor, Michigan. Association for Computational Linguistics. 43, 131

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). Procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the Workshop on Speech and Natural Language*, pages 306–311. Association for Computational Linguistics. 130

Blake, B. J. (2001). *Case*. Cambridge University Press, Cambridge, New York, 2nd edition. 61

Böhmová, A., Hajič, J., Hajičová, E., and Hladká, B. (2000). The Prague Dependency Treebank: A Three-level annotation scenario. In Abeillé, A., editor, *Treebanks: Building and using syntactically annotated corpora.*, chapter 1, pages 103–127. Kluwer Academic Publishers, Amsterdam. 51

Bohnet, B. (2009). Efficient Parsing of Syntactic and Semantic Dependency Structures. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 67–72, Boulder, Colorado. Association for Computational Linguistics. 53, 74, 138

Bohnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 89–97, Beijing, China. International Committee on Computational Linguistics. 53, 74, 123, 138

Bohnet, B. and Nivre, J. (2012). A Transition-Based System for Joint Part-of-Speech Tagging and Labeled Non-Projective Dependency Parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465, Jeju, South Korea. Association for Computational Linguistics. 125, 150, 156

Bohnet, B., Nivre, J., Boguslavsky, I., Farkas, R., Ginter, F., and Hajič, J. (2013). Joint

Morphological and Syntactic Analysis for Richly Inflected Languages. *Transactions of the Association for Computational Linguistics*, 1:415–428. 130, 150, 151, 156

Boyd, A., Dickinson, M., and Meurers, W. D. (2008). On detecting errors in dependency treebanks. *Research on Language and Computation*, 6(2):113–137. 58

Brants, S., Dipper, S., Hansen-Shirra, S., Lezius, W., and Smith, G. (2002). The TIGER treebank. In *Proceedings of the 1st Workshop on Treebanks and Linguistic Theories*, pages 24–41, Sozopol, Bulgaria. 10, 13, 51, 73

Bresnan, J. (2001). *Lexical-Functional Syntax*. Blackwell Publishers. 33, 34

Buchholz, S. and Marsi, E. (2006). CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City. Association for Computational Linguistics. 58

Carreras, X. (2007). Experiments with a Higher-Order Projective Dependency Parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961, Prague, Czech Republic. Association for Computational Linguistics. 21, 53, 103, 108, 121, 138

Carreras, X., Chao, I., Padró, L., and Padró, M. (2004). FreeLing: An Open-Source Suite of Language Analyzers. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*, pages 239–242. European Language Resources Association (ELRA). 79

Carreras, X., Collins, M., and Koo, T. (2008). TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-Rich Parsing. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 9–16, Manchester, England. Coling 2008 Organizing Committee. 154

Chappelier, J.-C., Rajman, M., Aragüés, R., and Rozenknop, A. (1999). Lattice Parsing for Speech Recognition. In *6ème Conférence sur le Traitement Automatique du Langage Naturel*, pages 95–104. ATALA. 115

Choi, J. D. and Palmer, M. (2010). Robust Constituent-to-Dependency Conversion for English. In *Proceedings of the Ninth International Workshop on Treebanks and Linguistic Theories (TLT-9)*, pages 55–66, Tartu, Estonia. 13

Chrupała, G., Dinu, G., and van Genabith, J. (2008). Learning Morphology with Morfette. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, pages 2362–2367, Marrakech, Morocco. European Language Resources Association (ELRA). 69, 76

Chu, Y.-J. and Liu, T.-H. (1965). On the Shortest Arborescence of a Directed Graph. *Scientia Sinica*, 14(10):1396–1400. 18, 98, 118

Cohen, S. B. and Smith, N. A. (2007). Joint morphological and syntactic disambiguation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 208–217, Prague, Czech Republic. Association for Computational Linguistics. 41, 42, 115, 131, 142, 147, 149, 151, 156

Collins, M. (2002). Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics. 29

Collins, M. (2003). Head-driven Statistical Models for Natural Language Parsing. *Computational linguistics*, 29(4):589–637. 13

Corbett, G. G. (1991). *Gender*. Cambridge Textbooks in Linguistics. Cambridge University Press. 78

Corbett, G. G. (2006). *Agreement*. Cambridge Textbooks in Linguistics. Cambridge University Press. 37, 82

Covington, M. A. (2001). A fundamental algorithm for dependency parsing (with corrections). In *Proceedings of the 39th Annual ACM Southeast Conference*, Athens, Gorgia. ACM. 16, 87

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585. 30, 53, 73

Crammer, K., Dekel, O., Shalev-Shwartz, S., and Singer, Y. (2003). Online passive-aggressive algorithms. In *Proceedings of the 16th Annual Conference on Neural Information Processing Systems*, volume 7, pages 1217–1224, Cambridge, Massachusetts, USA. MIT Press. 30, 53, 73, 100, 123

Csendes, D., Csirik, J., and Gyimóthy, T. (2004). The Szeged Corpus: A POS tagged and syntactically annotated Hungarian natural language corpus. In *Proceedings of the 5th International Workshop on Linguistically Interpreted Corpora*, pages 19–23, Geneva, Switzerland. 51

Das, D., Martins, A. F. T., and Smith, N. A. (2012). An Exact Dual Decomposition Algorithm for Shallow Semantic Parsing with Constraints. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 209–217, Montréal, Canada. Association for Computational Linguistics. 120, 153

De La Clergerie, E. (2013). Exploring beam-based shift-reduce dependency parsing with DyALog: Results from the SPMRL 2013 shared task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 53–62, Seattle, Washington, USA. Association for Computational Linguistics. 151

de Marneffe, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC-2006)*, volume 6, pages 449–454. European Language Resources Association (ELRA). 13

Edmonds, J. (1967). Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B(4):233–240. 18, 98, 118

Eisenberg, P. (2006). *Grundriss der deutschen Grammatik: Der Satz*. Grundriss der deutschen Grammatik. 38

Eisner, J. (1997). Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 5th International Workshop on Parsing Technologies (IWPT)*, pages 54–65, MIT, Cambridge, MA. 18, 152

Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies*, pages 29–61. Springer. 18, 121, 152

Eryiğit, G. and Oflazer, K. (2006). Statistical dependency parsing of Turkish. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 89–96, Trento, Italy. Association for Computational Linguistics. 145

Eryiğit, G. (2012). The Impact of Automatic Morphological Analysis & Disambiguation on Dependency Parsing of Turkish. In Calzolari, N., Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 1960–1965, Istanbul, Turkey. European Language Resources Association (ELRA). ACL Anthology Identifier: L12-1056. 132, 136

Eryiğit, G., Ilbay, T., and Can, O. A. (2011). Multiword Expressions in Statistical Dependency Parsing. In *Proc. of the SPMRL Workshop of IWPT*, pages 45–55, Dublin, Ireland. Association for Computational Linguistics. 136

Eryiğit, G., Nivre, J., and Oflazer, K. (2008). Dependency Parsing of Turkish. *Computational Linguistics*, 34(3):357–389. ix, 40, 132, 133

Farkas, R., Vincze, V., and Schmid, H. (2012). Dependency Parsing of Hungarian: Baseline Results and Challenges. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 55–65, Avignon, France. Association for Computational Linguistics. 73

Foth, K. A. and Menzel, W. (2006). Hybrid Parsing: Using Probabilistic Models as Predictors for a Symbolic Parser. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 321–328, Sydney, Australia. Association for Computational Linguistics. 109

Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296. 29, 100, 123

Goldberg, Y. (2011). *Automatic Syntactic Processing of Modern Hebrew*. PhD thesis, Ben Gurion University, Beer Sheva, Israel. 131, 135, 137, 140, 151

Goldberg, Y. and Elhadad, M. (2009). Hebrew Dependency Parsing: Initial Results. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 129–133, Paris, France. Association for Computational Linguistics. 151

Goldberg, Y. and Elhadad, M. (2010a). Easy-First Dependency Parsing of Modern Hebrew. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 103–107, Los Angeles, CA, USA. Association for Computational Linguistics. 151

Goldberg, Y. and Elhadad, M. (2010b). An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California. Association for Computational Linguistics. 16, 17

Goldberg, Y. and Elhadad, M. (2011). Joint Hebrew Segmentation and Parsing using a PCFGLA Lattice Parser. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 704–709, Portland, Oregon, USA. Association for Computational Linguistics. 131, 149

Goldberg, Y. and Elhadad, M. (2013). Word segmentation, unknown-word resolution, and morphological agreement in a hebrew parsing system. *Computational Linguistics*, 39(1):121–160. 41, 108, 131, 147, 149

Goldberg, Y. and Nivre, J. (2012). A Dynamic Oracle for Arc-Eager Dependency Parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India. The COLING 2012 Organizing Committee. 17

Goldberg, Y. and Nivre, J. (2013). Training Deterministic Parsers with Non-Deterministic Oracles. *Transactions of the Association for Computational Linguistics*, 1(Oct):403–414. 17

Goldberg, Y. and Tsarfaty, R. (2008). A single generative model for joint morphological segmentation and syntactic parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, pages 371–379, Columbus, Ohio. Association for Computational Linguistics. 114, 131, 142, 146, 147, 149, 155, 156

Goldberg, Y., Tsarfaty, R., Adler, M., and Elhadad, M. (2009). Enhancing Unlexicalized Parsing Performance Using a Wide Coverage Lexicon, Fuzzy Tag-Set Mapping, and EM-HMM-Based Lexical Probabilities. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 327–335, Athens, Greece. Association for Computational Linguistics. 131

Gómez-Rodríguez, C., Carroll, J. A., and Weir, D. J. (2011). Dependency Parsing Schemata and Mildly Non-Projective Dependency Parsing. *Computational Linguistics*, 37(3):541–586. 12

Gómez-Rodríguez, C. and Nivre, J. (2010). A Transition-Based Parser for 2-Planar Dependency Structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501, Uppsala, Sweden. Association for Computational Linguistics. 17

Green, S. and Manning, C. D. (2010). Better Arabic Parsing: Baselines, Evaluations, and Analysis. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 394–402, Beijing, China. Coling 2010 Organizing Committee. 131, 149, 156

Hajič, J. (2000). Morphological Tagging: Data vs. Dictionaries. In *Proceedings of the 6th ANLP Conference / 1st NAACL Meeting*, pages 94—101, Seattle, Washington. Association for Computational Linguistics. 69, 79, 80

Hajič, J. (2004). *Disambiguation of Rich Inflection (Computational Morphology of Czech)*. Nakladatelství Karolinum, Prague, Czech Republic. 69, 70, 79, 90

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Stepánek, J., Stranák, P., Surdeanu, M., Xue, N., and Zhang, Y. (2009). The CoNLL-2009 shared task: Syntactic and Semantic dependencies in multiple languages. In *Proceedings of the 13th Conference on Computational Natural Language Learning: Shared Task*, pages 1–18, Boulder, Colorado, USA. Association for Computational Linguistics. 51, 58, 73

Hajič, J., Panevová, J., Hajičová, E., Sgall, P., Pajas, P., Štěpánek, J., Havelka, J., and Mikulová, M. (2006). Prague Dependency Treebank 2.0. 12, 51, 79

Hakkani-Tür, D. Z., Oflazer, K., and Tür, G. (2002). Statistical morphological disambiguation for agglutinative languages. *Computers and the Humanities*, 36(4):381–410. 79, 90

Hall, K. B. (2005). *Best-first Word-lattice Parsing: Techniques for Integrated Syntactic Language Modeling*. PhD thesis, Providence, RI, USA. 115

Hatori, J., Matsuzaki, T., Miyao, Y., and Tsujii, J. (2011). Incremental Joint POS Tagging and Dependency Parsing in Chinese. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1216–1224, Chiang Mai, Thailand. Asian Federation of Natural Language Processing. 125, 150, 156

Hatori, J., Matsuzaki, T., Miyao, Y., and Tsujii, J. (2012). Incremental Joint Approach to Word Segmentation, POS Tagging, and Dependency Parsing in Chinese. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1045–1053, Jeju Island, Korea. Association for Computational Linguistics. 131, 139, 150, 154

Havelka, J. (2007). Beyond Projectivity: Multilingual Evaluation of Constraints and Measures on Non-Projective Structures. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 608–615, Prague, Czech Republic. Association for Computational Linguistics. 12

Huang, L. and Sagae, K. (2010). Dynamic Programming for Linear-Time Incremental Parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala, Sweden. Association for Computational Linguistics. 17

Hudson, R. A. (1984). *Word Grammar*. Basil Blackwell, Oxford. 10

Jiang, W., Huang, L., Liu, Q., and Lü, Y. (2008). A Cascaded Linear Model for Joint Chinese Word Segmentation and Part-of-Speech Tagging. In *Proceedings of ACL-08: HLT*, pages 897–904, Columbus, Ohio. Association for Computational Linguistics. 148

Johansson, R. and Nugues, P. (2007a). Extended Constituent-to-dependency Conversion for English. In *Proceedings of NODALIDA 2007*, pages 105–112, Tartu, Estonia. 13

Johansson, R. and Nugues, P. (2007b). Incremental Dependency Parsing Using Online Learning. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 1134–1138, Prague, Czech Republic. Association for Computational Linguistics. 17

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2nd edition. 112, 135

Khmylko, L., Foth, K. A., and Menzel, W. (2009). Co-Parsing with Competitive Models. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 99–107, Paris, France. Association for Computational Linguistics. 104

Köhn, A., Lao, U. C., Zadeh, A. B., and Sagae, K. (2014). Parsing Morphologically Rich Languages with (Mostly) Off-The-Shelf Software and Word Vectors. In *Notes of the SPMRL 2014 Shared Task on Parsing Morphologically-Rich Languages*, Dublin, Ireland. 152

Koo, T. and Collins, M. (2010). Efficient Third-Order Dependency Parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden. Association for Computational Linguistics. 21, 121, 127

Koo, T., Globerson, A., Carreras, X., and Collins, M. (2007). Structured Prediction Models via the Matrix-Tree Theorem. In *Proceedings of the 2007 Joint Conference on Empirical*

Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 141–150, Prague, Czech Republic. Association for Computational Linguistics. 23

Koo, T., Rush, A. M., Collins, M., Jaakkola, T., and Sontag, D. (2010). Dual Decomposition for Parsing with Non-Projective Head Automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA. Association for Computational Linguistics. 27, 28, 113, 118, 121, 122, 127, 153, 157

Krivanek, J. and Meurers, W. D. (2011). Comparing rule-based and datadriven dependency parsing of learner language. In *Proceedings of the International Conference on Dependency Linguistics*, pages 310–318, Barcelona, Spain. 109

Kübler, S., McDonald, R., and Nivre, J. (2009). *Dependency Parsing*. Synthesis lectures on human language technologies. Morgan & Claypool. 15, 18, 20

Kuhlmann, M. (2013). Mildly Non-Projective Dependency Grammar. *Computational Linguistics*, 39(2):355–387. 12

Lee, J., Naradowsky, J., and Smith, D. A. (2011). A discriminative model for joint morphological disambiguation and dependency parsing. In *Proceedings of the 49th annual meeting of the Association for Computational Linguistics*, pages 885–894, Portland, USA. Association for Computational Linguistics. 152

Li, Z., Zhang, M., Che, W., Liu, T., Chen, W., and Li, H. (2011). Joint Models for Chinese POS Tagging and Dependency Parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1180–1191, Edinburgh, Scotland, UK. Association for Computational Linguistics. 152, 153, 156

Li, Z. and Zhou, G. (2012). Unified Dependency Parsing of Chinese Morphological and Syntactic Structures. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1445–1454, Jeju Island, Korea. Association for Computational Linguistics. 131, 150

Luo, X. (2003). A Maximum Entropy Chinese Character-Based Parser. In Collins, M. and Steedman, M., editors, *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 192–199. 148

Ma, J., Xiao, T., Zhu, J., and Ren, F. (2012). Easy-First Chinese POS Tagging and Dependency Parsing. In *Proceedings of COLING 2012*, pages 1731–1746, Mumbai, India. The COLING 2012 Organizing Committee. 151

Magerman, D. M. (1995). Statistical Decision-Tree Models for Parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Cambridge, Massachusetts, USA. Association for Computational Linguistics. 13

Magnanti, T. and Wolsey, L. (1995). Optimal trees. *Handbooks in Operations Research and Management Science*, 7(April):503–615. 24, 96

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330. 12

Martins, A., Almeida, M., and Smith, N. A. (2013). Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria. Association for Computational Linguistics. 121, 122, 127, 138, 153

Martins, A., Figueiredo, M., Aguiar, P., Smith, N., and Xing, E. (2011a). An Augmented Lagrangian Approach to Constrained MAP Inference. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 169–176, New York, NY, USA. ACM. 27, 120

Martins, A., Smith, N., Figueiredo, M., and Aguiar, P. (2011b). Dual Decomposition with Many Overlapping Components. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 238–249, Edinburgh, Scotland, UK. Association for Computational Linguistics. 27, 120, 153

Martins, A., Smith, N., and Xing, E. (2009). Concise Integer Linear Programming Formulations for Dependency Parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 342–350, Suntec, Singapore. Association for Computational Linguistics. 24, 91, 92, 93, 96, 97, 98, 100, 126, 127

Martins, A., Smith, N., Xing, E., Aguiar, P., and Figueiredo, M. (2010a). Augmented dual decomposition for MAP inference. In *Neural Information Processing Systems: Workshop in Optimization for Machine Learning*, volume 105. 27

Martins, A., Smith, N., Xing, E., Aguiar, P., and Figueiredo, M. (2010b). Turbo Parsers: Dependency Parsing by Approximate Variational Inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 34–44, Cambridge, MA. Association for Computational Linguistics. 24, 113

Martins, A. F. T., Figueiredo, M. A. T., Aguiar, P. M. Q., Smith, N. A., and Xing, E. P. (2015). Ad3: Alternating directions dual decomposition for map inference in graphical models. *Journal of Machine Learning Research*, 16:495–545. 27, 119, 153

Marton, Y., Habash, N., and Rambow, O. (2013). Dependency parsing of modern standard arabic with lexical and inflectional features. *Computational Linguistics*, 39(1):161–194. 68

McDonald, R. and Pereira, F. (2006). Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 81–88, Trento, Italy. Association for Computational Linguistics. 21, 22, 53, 121, 138

McDonald, R., Pereira, F., Ribarov, K., and Hajic, J. (2005). Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics. 18, 95, 98, 118, 124

McDonald, R. and Satta, G. (2007). On the Complexity of Non-Projective Data-Driven Dependency Parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 121–132, Prague, Czech Republic. Association for Computational Linguistics. 23

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University Press of New York, suny serie edition. 10

Müller, T., Schmid, H., and Schütze, H. (2013). Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA. Association for Computational Linguistics. 70, 79

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453. 139

Ng, H. T. and Low, J. K. (2004). Chinese Part-of-Speech Tagging: One-at-a-Time or All-at-Once? Word-Based or Character-Based? In Lin, D. and Wu, D., editors, *Proceedings of EMNLP 2004*, pages 277–284, Barcelona, Spain. Association for Computational Linguistics. 148

Nivre, J. (2003). An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160. Association of Computational Linguistics. 16

Nivre, J. (2004). Incrementality in Deterministic Dependency Parsing. In Keller, F., Clark, S., Crocker, M., and Steedman, M., editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain. Association for Computational Linguistics. 15, 16

Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553. 16, 17, 150

Nivre, J. (2009). Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore. Association for Computational Linguistics. 16

Nivre, J. (2015). Towards a Universal Grammar for Natural Language Processing. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing*, pages 3–16. Springer. 10, 14

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryiğit, G., Kübler, S., Marinov, S., and Marsi, E. (2007). MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135. 109

Nivre, J., Kuhlmann, M., and Hall, J. (2009). An Improved Oracle for Dependency Parsing with Online Reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76, Paris, France. Association for Computational Linguistics. 16

Nivre, J. and McDonald, R. (2008). Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio. Association for Computational Linguistics. 28

Nivre, J. and Nilsson, J. (2005). Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan. Association for Computational Linguistics. 16

Oflazer, K. (1994). Two-level Description of Turkish Morphology. *Literary and Linguistic Computing*, 9(2):137–148. 137

Oflazer, K., Say, B., Hakkani-Tür, D. Z., and Tür, G. (2003a). Building a Turkish Treebank. In *Treebanks*, pages 261–277. Springer Netherlands. 40

Oflazer, K., Say, B., Hakkani-Tür, D. Z., and Tür, G. (2003b). Building a Turkish Treebank. In Abeille, A., editor, *Building and Exploiting Syntactically-annotated Corpora*. Kluwer Academic Publishers, Dordrecht. 136

Ortmanns, S., Ney, H., and Aubert, X. (1997). A word graph algorithm for large vocabulary continuous speech recognition. *Computer Speech & Language*, 11(1):43–72. 112

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (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*, pages 433–440, Sydney, Australia. Association for Computational Linguistics. 149

Pitler, E. (2014). A Crossing-Sensitive Third-Order Factorization for Dependency Parsing. *Transactions of the Association for Computational Linguistics*, 2:41–54. 22

Pitler, E., Kannan, S., and Marcus, M. (2013). Finding Optimal 1-Endpoint-Crossing Trees. *Transactions of the Association for Computational Linguistics*, 1:13–24. 22

Pitler, E. and McDonald, R. (2015). A Linear-Time Transition System for Crossing Interval Trees. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 662–671, Denver, Colorado. Association for Computational Linguistics. 17

Prins, R. (2004). Beyond N in N-gram tagging. In Van Der Beek, L., Genzel, D., and Midgley, D., editors, *Proceedings of the ACL 2004 Student Research Workshop*, pages 61–66, Barcelona, Spain. Association for Computational Linguistics. 72

Riedel, S. and Clarke, J. (2006). Incremental Integer Linear Programming for Non-projective Dependency Parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 129–137, Sydney, Australia. Association for Computational Linguistics. 23, 91, 92, 100, 103

Rush, A. and Petrov, S. (2012). Vine Pruning for Efficient Multi-Pass Dependency Parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 498–507, Montréal, Canada. Association for Computational Linguistics. 23, 127

Rush, A. M. and Collins, M. (2012). A Tutorial on Dual Decomposition and Lagrangian Relaxation for Inference in Natural Language Processing. *Journal of Artificial Intelligence Research*, 45(1):305–362. 25, 26

Rush, A. M., Sontag, D., Collins, M., and Jaakkola, T. (2010). On Dual Decomposition and Linear Programming Relaxations for Natural Language Processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Cambridge, MA. Association for Computational Linguistics. 25, 26, 27, 113, 153

Sak, H., Güngör, T., and Saraçlar, M. (2008). Turkish Language Resources: Morphological Parser, Morphological Disambiguator and Web Corpus. In *Proceedings of GoTAL 2008*, volume 5221 of *LNCS*, pages 417–427. Springer. 43, 141

Schiller, A. (1994). Dmor - user's guide. Technical report, University of Stuttgart. 79, 101

Schiller, A., Teufel, S., Stöckert, C., and Thielen, C. (1999). Guidelines für das Tagging deutscher Textcorpora mit STTS. Technical report, Institut für maschinelle Sprachverarbeitung, Stuttgart, Germany. 52

Schmid, H., Fitschen, A., and Heid, U. (2004). SMOR: A German Computational Morphology Covering Derivation, Composition and Inflection. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*. 79

Schmid, H. and Laws, F. (2008). Estimation of conditional probabilities with decision trees and an application to fine-grained POS tagging. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 777–784, Morristown, NJ, USA. Association for Computational Linguistics. 79, 90

Seddah, D., Kübler, S., and Tsarfaty, R. (2014). Introducing the SPMRL 2014 Shared Task on Parsing Morphologically-rich Languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin, Ireland. Dublin City University. 136

Seddah, D., Tsarfaty, R., Kübler, S., Candito, M., Choi, J. D., Farkas, R., Foster, J., Goenaga, I., Gojenola Galletebeitia, K., Goldberg, Y., Green, S., Habash, N., Kuhlmann, M., Maier, W., Nivre, J., Przepiórkowski, A., Roth, R., Seeker, W., Versley, Y., Vincze, V., Woliński, M., Wróblewska, A., and de la Clergerie, E. V. (2013). Overview of the SPMRL 2013 Shared Task: A Cross-Framework Evaluation of Parsing Morphologically Rich Languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*,

pages 146–182, Seattle, Washington, USA. Association for Computational Linguistics. 134, 141

Seeker, W. and Çetinoğlu, O. (2015). A Graph-based Lattice Dependency Parser for Joint Morphological Segmentation and Syntactic Analysis. *Transactions of the Association for Computational Linguistics*, 3:359–373. 111, 114, 115

Seeker, W. and Kuhn, J. (2011). On the Role of Explicit Morphological Feature Representation in Syntactic Dependency Parsing for German. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 58–62, Dublin, Ireland. Association for Computational Linguistics. 59

Seeker, W. and Kuhn, J. (2012). Making Ellipses Explicit in Dependency Conversion for a German Treebank. In Calzolari, N., Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 3132–3139, Istanbul, Turkey. European Language Resources Association (ELRA). 13, 51, 53, 73

Seeker, W. and Kuhn, J. (2013a). Morphological and Syntactic Case in Statistical Dependency Parsing. *Computational Linguistics*, 39(1):23–55. 49, 92

Seeker, W. and Kuhn, J. (2013b). The Effects of Syntactic Features in Automatic Prediction of Morphology. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 333–344, Seattle, Washington, USA. Association for Computational Linguistics. 70

Sgall, P., Hajičová, E., and Panevová, J. (1986). *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Dordrecht:Reidel Publishing Company and Prague:Academia. 10

Shen, L. and Joshi, A. (2008). LTAG Dependency Parsing with Bidirectional Incremental Construction. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 495–504, Honolulu, Hawaii. Association for Computational Linguistics. 16, 151

Sima'an, K., Itai, A., Winter, Y., Altman, A., and Nativ, N. (2001). Building a tree-bank of modern Hebrew text. *Traitement Automatique des Langues*, 42(2):247–380. 137

Smith, D. and Eisner, J. (2008). Dependency Parsing by Belief Propagation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 145–156, Honolulu, Hawaii. Association for Computational Linguistics. 152, 153

Smith, D. A. and Smith, N. A. (2007). Probabilistic Models of Nonprojective Dependency Trees. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 132–140, Prague, Czech Republic. Association for Computational Linguistics. 23

Smith, N. A., Smith, D. A., and Tromble, R. W. (2005). Context-Based Morphological Disambiguation with Random Fields. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 475–482, Vancouver, British Columbia, Canada. Association for Computational Linguistics. 69, 90

Spoustová, D. J., Hajič, J., Raab, J., and Spousta, M. (2009). Semi-supervised training for the averaged perceptron POS tagger. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 763–771, Athens, Greece. Association for Computational Linguistics. 52, 79, 90

Tesnière, L. (1959). *Éléments de syntaxe structurale*. Editions Klincksieck. 10

Tesnière, L., Osborne, T., and Kahane, S. (2015). *Elements of Structural Syntax*. John Benjamins Publishing Company. 10

Tratz, S. (2013). A Cross-Task Flexible Transition Model for Arabic Tokenization, Affix Detection, Affix Labeling, POS Tagging, and Dependency Parsing. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 34–45, Seattle, Washington, USA. Association for Computational Linguistics. 151, 154, 155

Tratz, S. and Hovy, E. (2011). A Fast, Accurate, Non-Projective, Semantically-Enriched Parser. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1257–1268, Edinburgh, Scotland, UK. Association for Computational Linguistics. 16

Trón, V., Halácsy, P., Rebrus, P., Rung, A., Vajda, P., and Simon, E. (2006). Morphdb.hu: Hungarian lexical database and morphological grammar. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, pages 1670–1673, Genoa, Italy. 79, 101

Tsarfaty, R. (2006). Integrated Morphological and Syntactic Disambiguation for Modern Hebrew. In *Proceedings of the COLING/ACL 2006 Student Research Workshop*, pages 49–54, Sydney, Australia. Association for Computational Linguistics. 130, 148

Tsarfaty, R. (2013). A Unified Morpho-Syntactic Scheme of Stanford Dependencies. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 578–584, Sofia, Bulgaria. Association for Computational Linguistics. 137

Tsarfaty, R. and Goldberg, Y. (2008). Word-Based or Morpheme-Based? Annotation Strategies for Modern Hebrew Clitics. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, B. M. J. M. J. O. S. P. D. T., editor, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/. 131

Tsarfaty, R., Nivre, J., and Andersson, E. (2011). Evaluating Dependency Parsing: Robust and Heuristics-Free Cross-Annotation Evaluation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 385–396, Edinburgh, Scotland, UK. Association for Computational Linguistics. 133

Tsarfaty, R., Nivre, J., and Andersson, E. (2012a). Cross-Framework Evaluation for Statistical Parsing. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 44–54, Avignon, France. Association for Computational Linguistics. 133

Tsarfaty, R., Nivre, J., and Andersson, E. (2012b). Joint Evaluation of Morphological Segmentation and Syntactic Parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 6–10, Jeju Island, Korea. Association for Computational Linguistics. 133, 134, 136, 147

Tsarfaty, R., Seddah, D., Goldberg, Y., Kübler, S., Candito, M., Foster, J., Versley, Y., Rehbein, I., and Tounsi, L. (2010). Statistical parsing of morphologically rich languages (SPMRL): what, how and whither. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 1–12, Los Angeles, California, USA. Association for Computational Linguistics. 5

Ueffing, N., Och, F. J., and Ney, H. (2002). Generation of Word Graphs in Statistical Machine Translation. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 156–163. Association for Computational Linguistics. 112

Versley, Y., Beck, K., Hinrichs, E., and Telljohann, H. (2010). A Syntax-first Approach to High-quality Morphological Analysis and Lemma Disambiguation for the TüBa-D/Z

Treebank. In *9th Conference on Treebanks and Linguistic Theories (TLT9)*, pages 233–244, Tartu, Estland. 71

Vincze, V., Szauter, D., Almási, A., Móra, G., Alexin, Z., and Csirik, J. (2010). Hungarian Dependency Treebank. In *Proceedings of the 7th Conference on International Language Resources and Evaluation*, pages 1855–1862, Valletta, Malta. European Language Resources Association (ELRA). 13, 51, 73

Votrubec, J. (2006). Morphological Tagging Based on Averaged Perceptron. In *WDS'06 Proceedings of Contributed Papers*, pages 191–195, Praha, Czechia. Matfyzpress, Charles University. 72, 74, 79

Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*. Association for Computational Linguistics. 13, 15, 16

Zhang, H. and McDonald, R. (2012). Generalized Higher-Order Dependency Parsing with Cube Pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331, Jeju Island, Korea. Association for Computational Linguistics. 21, 23, 127

Zhang, M., Zhang, Y., Che, W., and Liu, T. (2014a). Character-Level Chinese Dependency Parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1326–1336, Baltimore, Maryland. Association for Computational Linguistics. 150, 151, 157

Zhang, Y. and Clark, S. (2008a). A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii. Association for Computational Linguistics. 17

Zhang, Y. and Clark, S. (2008b). Joint Word Segmentation and POS Tagging Using a Single Perceptron. In *Proceedings of ACL-08: HLT*, pages 888–896, Columbus, Ohio. Association for Computational Linguistics. 148

Zhang, Y. and Clark, S. (2010). A Fast Decoder for Joint Word Segmentation and POS-Tagging Using a Single Discriminative Model. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 843–852, Cambridge, MA. Association for Computational Linguistics. 148

Zhang, Y., Lei, T., Barzilay, R., and Jaakkola, T. (2014b). Greed is Good if Randomized: New Inference for Dependency Parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1013–1024, Doha, Qatar. Association for Computational Linguistics. 28, 154

Zhang, Y., Li, C., Barzilay, R., and Darwish, K. (2015). Randomized Greedy Inference for Joint Segmentation, POS Tagging and Dependency Parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 42–52, Denver, Colorado. Association for Computational Linguistics. 130, 134, 154

Zsibrita, J., Vincze, V., and Farkas, R. (2010). Ismeretlen kifejezések és a szófaji egyértelműsítés. In Tanács, A. and Vincze, V., editors, *VII. Magyar Számítógépes Nyelvészeti Konferencia*, pages 275–283, Szeged, Hungary. 52

Wolfgang Burkhard Werner Seeker

Personal Data

date of birth: 1983/12/21
place of birth: Jena, Germany
nationality: German

Education

since 2010	PhD student Supervisor: Prof. Dr. Jonas Kuhn
2009	Diplom in Computational Linguistics, with honors University of Potsdam, Germany
2003	Abitur Ev. Gymnasium "Zum Grauen Kloster", Berlin

Employment

since 2011	Research assistant at the Sonderforschungsbereich 732 (special research center) Project D8: *Data-driven Dependency Parsing - Context Factors in Dependency Classification* University of Stuttgart, Germany
2010	Research assistant at the Chair for Foundations of Computational Linguistics Institute for Natural Language Processing University of Stuttgart, Germany
2008 – 2009	Internship National Centre for Language Technology Dublin City University, Dublin, Ireland Project: LORG
2007 – 2008	Student assistant Berlin-Brandenburgische Akademie der Wissenschaften, Berlin Project: *Digitales Wörterbuch der deutschen Sprache* Project: *Deutsches Textarchiv*
2005 – 2007	Student assistant at the Chair for Theoretical Computational Linguistics, Institute for Linguistics, University of Potsdam, Germany

Typeset November 15, 2016