Hybrid Parsing with a Maximum Spanning Tree Predictor

by Lidia Khmylko

supervised by:

Prof. Dr.-Ing. Dr. habil. Karl-Heinz Zimmermann
Institute of Computer Technology
Hamburg University of Technology

Prof. Dr.-Ing. Wolfgang Menzel
Natural Language Systems, Department of Informatics
University of Hamburg

Master Thesis

Hamburg University of Technology

Hamburg, June 2007

Declaration

I declare that this work has been prepared by myself, all literal or content-based quotations are clearly pointed out, and no other sources or aids than those declared have been used.

Lidia Khmylko, December 16, 2007 _____

Acknowledgments

My sincere thanks go to all who contributed to the success of this work:

Prof. Dr. Karl-Heinz Zimmermann — for his consent to supervise this master thesis on the side of Hamburg University of Technology;

Prof. Dr. Wolfgang Menzel — for the exciting topic and the outstanding supervision and support of this work;

Dr. Kilian Foth — for providing research materials and motivating hints, help with the WCDG as well as for mentoring my previous work in the "Partial Parsing" that served as a starting point for my interest in syntactic parsing;

Klaus Dahlinghaus — for valuable comments and discussions;

Dr. Ryan McDonald — for sharing the MSTParser and timely information;

Computer Center of the Department of Informatics, University of Hamburg — in person, Gerhard Friesland-Köpke, Reinhard Zierke and Marc Klegin — for providing the necessary hardware resources and fast and reliable computer administration support.

A special thank to my dearest friends, Jasmin Kominek and Endri Deliu, for helping me with mathematical and other relevant and not relevant ambiguities the whole time this work was in progress.

Contents

Revised Version
December 16, 2007

Chapter 1

Introduction

"What makes the desert beautiful," says the little prince, "is that somewhere it hides a well."

Antoine de Saint-Exupéry

There are a lot of application areas that could benefit greatly from automatic syntactic analysis of natural language. Question answering and machine translation, information extraction and grammar checking in word processors are among a few to mention. Because of the fast development of the Web, the need for automatic processing of written text becomes even more urgent. Together with the practical gains, a reliable solution would also contribute to the understanding of the nature and the inner organizational structure of language as well as provide more insights into the process of learning and understanding of language by humans themselves. Unfortunately, the goal of automatic language processing in general and syntactic analysis in particular could not be solved to the full extent yet. This can be ascribed to the complexity of language itself for not even a theoretical model has succeeded to cover all its phenomena. Besides, natural language processing and computational linguistics accumulating the research in this field are still comparatively young.

In the recent years, the most prominent advances in the area of syntactic analysis of natural language have been seen in connection with stochastic methods. Another distinguishable direction of research is combining the results obtained by different methods of analysis, an approach often referred to as *hybrid*, although the understanding of the term may differ greatly, the meaning relevant for this thesis will be given in a moment. Before that, the main conceptual view on modeling the structure of a sentence as the main syntactic unit is to be introduced shortly.

1.1 Sentence as Dependency Structure

Modeling natural language in terms of *dependencies* has gained popularity in natural language processing due to the success of statistical parsers [1]. Its strongest rival, especially in the English language research community, the *constituent model* [2], dominated the scene until recently, and was used for lexicalized phrase structure representations. In that approach, a phrase structure of a sentence is understood as a set of nested constituents, i.e., those words or word groups that function as a single unit within a hierarchical structure. E.g., the German sentence "Mein Bruder ging am Sonntag in den Zoo," may get the analysis shown in Figure 1.1.

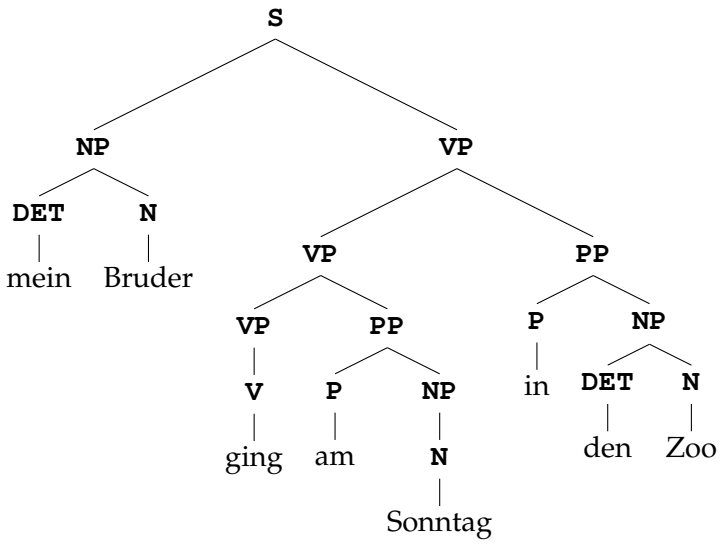


Figure 1.1: An example of the phrase structure analysis.

When validating the correctness of the constituent structure, such usual checks as moving constituents around in the sentence or substituting them by pronouns are applied. The modified sentence will still make sense if the constituent has been distinguished correctly. Thus, the transformations "*Am Sonntag* ging mein Bruder in den Zoo" and "*Er* ging am Sonntag in den Zoo" would help identify the **PP** "am Sonntag" and the **NP** "mein Bruder".

In contrast, a *dependency structure* renounces the phrasal nodes and is built wholly of lexical nodes connected by binary relations. Every lexical node *modifies* some other lexical node that is called its *head* or *regent*. This structure can be represented as a *directed graph* having a unique *root* node which is inserted artificially. The nodes of the graph are built by the lexical elements and the edges are drawn to show the dependency relations. Commonly, each dependency graph is acyclic and connected. Furthermore, it will be assumed that each node except the root has exactly one incoming edge in the graph, and the root

[1] It is more correct to say, it was re-discovered as the lexical dependency notion of grammar was already known in the linguistic tradition of ancient Greece and India [JM00].

[2] This approach was already used by the ancient Stoics [Fot06]

has no incoming edge. A dependency graph satisfying these properties is in fact a tree and thus is often referred to as a *dependency tree*. Figure 1.2 shows an example of a dependency tree of the same German sentence with an artificially inserted root as the left-most element.

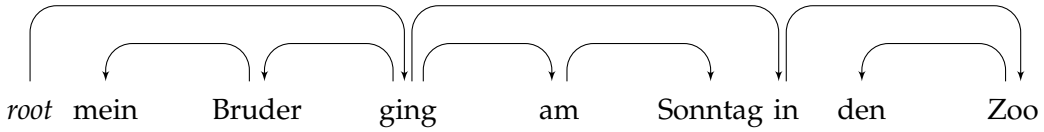


Figure 1.2: An example of a (projective) dependency graph.

The advantages of dependency parsing over phrase structure parsing are connected to the properties of the structures themselves. In the former case, the sentence is decomposed into much smaller units, these occur more often and count for a better learning rate of statistical parsers. Unfortunately, some of the higher-order knowledge of the sentence phrasal structure is lost in the dependency tree in comparison to the constituent model, but per-word relationships can be easier understood and explored as lexical information is accessible at the moment when syntactic decisions are made. Besides, for other applications using the analyzed representation of the syntactic structure, the dependency structure is easier to process. Another important advantage is that although not many manually annotated dependency corpora exist, applying simple converting rules those corpora annotated with the phrasal structures can be transformed into dependency trees — while probably not the optimal ones — so that statistical parsers have plenty of training material they need. Last but not least to mention is the facilitation of measuring the accuracy of a dependency structure as each word-to-word dependency is either correct or wrong while such mistakes in the phrase structure as an incorrect part of a correctly detached phrase or the partition of the constituents make them difficult to score.

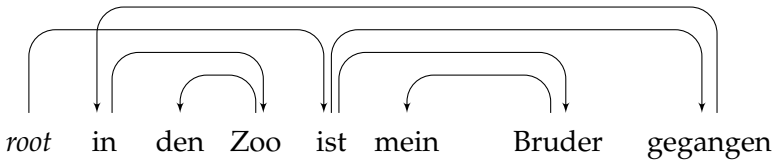


Figure 1.3: An example of a non-projective dependency graph.

Figure 1.2 has shown an example of a *projective* dependency tree, all edges of which are drawn above the sentence with a root to the left of linearly ordered words and no edges are crossing. If some edges are crossing, the tree is said to be *non-projective*. The results of the transformations from the phrase structure trees are always projective. In the English language, the number of non-projective sentences is very low. In German, and even more often in languages with more flexible word-order non-projective structures are not unusual. For example, a slightly modified sentence from the previous example shown in Figure 1.3 is clearly non-projective. In this case, the edge from the root to "ist" is crossing the edge from "gegangen" to "in" and there is no possibility to draw this graph in such a way that these edges do not cross.

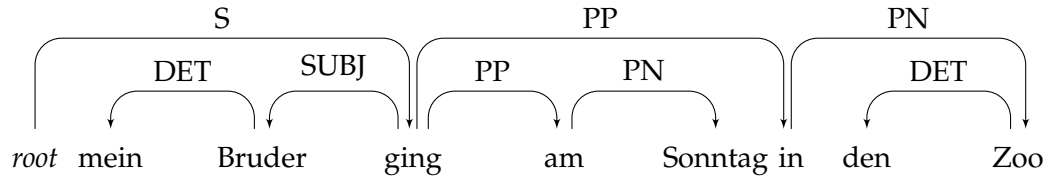


Figure 1.4: An example of a labeled dependency graph.

One may associate labels to the edges of the dependency graph to indicate syntactic relationships between the head and modifier, the result is referred to as a *labeled dependency tree*, such as the one presented in Figure 1.4.

1.2 Parsing Paradigms

Syntactic parsing can be understood as a task of recognizing a sentence and assigning a syntactic structure to it [JM00]. It should be noted that such a definition is not broad enough as it does not reflect the fact that universal 'yes' or 'no' judgment about *grammaticality* apparently does not exist [Cho55]. There are utterances which human speakers fail to understand, although they are made up analogous to other syntactic constructions in the language. Just think of the "Buffalo buffalo buffalo buffalo buffalo" by Covington [Fot06] unclear and ungrammatical to most readers until hinted for "Boston cattle bewilder Cleveland cattle". On the other hand, in many cases small errors in the syntactic construction do not prevent comprehension. For instance, uncouth foreign language learners may be unaware of many mistakes they make as no native speaker reclaims them. Differences in the social, local or historic conditions may account for the fact that quite different constructions are preferred by the speakers and thought of to be grammatical. In fact, the grammaticality is not a binary, but rather a *graded* property [Kel00]. Thus, it is desirable that parsing results include some kind of ranking, i.e., not only assign a possible grammatical structure to the input, but also a score to it.

One of the greatest hurdles for syntactic parsing is the *ambiguity* inherent to language structures. Thus, when encountering the sentence "He ate the bread with his wife," human speakers imagine a nice family meal rather than a spine-chilling sandwich. In this case, they may rely on the general world knowledge of what kinds of bread exist, such as "bread with meat" or "bread with cheese", to correctly choose for attaching the **`PP`** to the verb and not to the **`NP`** preceding it. For an automatic parser this choice between alternatives would be much harder to make.

The methods chosen for syntactic disambiguation, allow for classifications of the existing parsers. The two main parsing paradigms distinguishable from the very beginning of the language processing research are rule-based and data-driven parsing briefly presented next.

Rule-Based Parsing

Within the *rule-based*, or *symbolic*, paradigm, the language is understood as a system of regularities and, thus, an attempt is made to formulate all of them through a grammar of explicit linguistic rules and interactions of rules.

For example, the grammar according to which the sentence presented in Figure 1.1 is analyzed contains at least the following set of rules:

$$\begin{aligned}
\texttt{S} &\rightarrow \texttt{NP VP} \\
\texttt{NP} &\rightarrow \texttt{DET N} \\
\texttt{VP} &\rightarrow \texttt{VP PP} \\
\texttt{PP} &\rightarrow \texttt{P NP} \\
\texttt{VP} &\rightarrow \texttt{V} \\
\texttt{NP} &\rightarrow \texttt{N}
\end{aligned}$$

A common disadvantage of rule-based systems is that much expert knowledge and effort is needed to create, enhance and maintain the rules together with the lexicon mostly used in such systems. The creation of *broad coverage* parsers, i.e., parsers working on unrestricted input is difficult as, on the one hand, the complete generally applicable set of rules has never been set down in theory or practice for any language, and, on the other hand, if it were, the growth in the number of rules would multiply the number of ambiguity of the results and increase the search space substantially. Moreover, as not only the type of the word, but also its identity is important for the restriction of the syntactic roles it can play, another shortcoming of symbolic parsers comes into foreground for it is clearly infeasible to accumulate all lexicalized rules manually.

Data Driven Parsing

Data driven parsers, also referred to as *statistical*, have recently left the rule-based parsers behind. They first gained popularity in the 50s, but after the n-gram models were judged to be unsuitable for natural language modeling in [Cho55] and the limitations of the perceptron algorithm were emphasized [MP69], they were almost forgotten until a recent revival in the 90s.

Data driven parsing does not need any underlying grammar. Instead, some *similarity measure* is defined and the disambiguation is understood as similarity maximization. While parsing the data, an alternative is preferred that has the greatest similarity to those seen during the *training* phase [3] which is done on the data annotated with correct structures.

The dependency on the similarity at the same time accounts for a well-known restriction of this approach: the parsing results will depend on the similarity between the training

[3] The term 'learning' is used synonymously to 'training', but it sounds misleading as it suggests the similarity of automatic learning to human learning which is not generally applicable.

and testing data. So, such properties of the data as its domain, style, or the percentage of lexical units during the test, will make influence on the results. Even different sections of the same corpus can comprise quite non-similar data in this sense. Besides, the volume of the training data is also a property statistical parsers depend on since both the scarcity and the abundance of the training data may contribute to the degradation of the results. The former is just a limiting factor typical for some languages for which a sufficient amount of annotated data has not yet been collected, the latter sometimes becomes a time problem of the parsers due to the exponential complexity of the algorithms. Another disadvantage of data driven parsing is that fine-tuning cannot be done easily, e.g., by improving the performance on some isolated source of errors. The similarity measures are generally defined for the data as a whole and changes will effect all the results after the reparsing.

Still, the advantages of this approach are much more prominent in comparison to the disadvantages of the rule based paradigm. Data driven parsers adapt well to new languages. Lexicalization becomes easy as one only has to enforce the similarity measure to include the lexical units. Besides, the achievable coverage is greater than what is known for the rule-based parsers despite the points mentioned in the previous passage.

Within the data driven approach, probabilistic models can be subdivided into generative and discriminative with respect to the prediction task.

A *generative model* is motivated by the following properties. If objects and their classifications are generated randomly from a joint probability distribution $P(\boldsymbol{x}, \boldsymbol{y})$, then the optimal way to predict the class $\boldsymbol{y}$ for an input $\boldsymbol{x}$ is to maximize $P(\boldsymbol{y}|\boldsymbol{x})$. Applying Bayes' rule, this is equivalent to maximizing $P(\boldsymbol{x}|\boldsymbol{y})P(\boldsymbol{y})$. Thus, in the generative classifier the training data is used to learn the conditional probability $P(\boldsymbol{x}|\boldsymbol{y})$ for all the variables and the marginal probability $P(\boldsymbol{y})$ for the different classes $\boldsymbol{y}$. The results are used to approximate the behavior of the optimal predictor for the source [DHS00, JH99, RSNM04].

In the *discriminative approach*, the learning algorithm simply tries to find a classifier that performs well on the training data. Fitting the model requires estimating the conditional probability $P(\boldsymbol{y}|\boldsymbol{x})$. It does not attempt to model the underlying distribution of the variables and features, only the resulting classification boundary or function approximation accuracy is adjusted, the intermediate goal of forming a generator that can model the variables is left out [JH99, LS06, Vap00].

A generative model must make simplifying independence assumptions about the entire $P(\boldsymbol{x}, \boldsymbol{y})$, while the discriminative model makes many fewer assumptions by focusing on $P(\boldsymbol{y}|\boldsymbol{x})$. In fact, by optimizing the model to fit the joint distribution $P(\boldsymbol{x}, \boldsymbol{y})$, one may tune the approximation away from optimal conditional distribution $P(\boldsymbol{y}|\boldsymbol{x})$, which is used to make the predictions. Given sufficient data, the discriminative model will learn the best approximation to $P(\boldsymbol{y}|\boldsymbol{x})$ possible using its inner parameter $\boldsymbol{w}$, while the generative model will not necessarily do so. Still, generative models actually need fewer samples to converge to a good estimate of the joint distribution than discriminative models need to accurately represent the conditional distribution [Tas04, JH99]. But with enough data in use, the result may look quite differently.

Examples of generative training methods include Hidden Markov Models [JM00], Markov Random Fields [BVZ98], Bayesian Networks [DHS00], and of the discriminative — Gaussian processes [Ras06], Support Vector Machines [Vap00], as well as Neural Networks [McK03]. Eisner used a cubic parsing algorithm [Eis96] together with a generative model. The shift-reduce dependency parser of English of Yamada and Matsumoto [YM03] is trained with Support Vector Machines. Thereby, the classifiers are trained on individual decisions rather than on the overall quality of the parse. A similar approach is applied in the parser of Ratnaparkhi [Rat99] which is trained to maximize the conditional likelihood of each parsing decision and is probably the earliest work on discriminative parsing.

Hybrid Parsing

The opposition of the two approaches just described, and the possible synergies that may emerge after the combination of both, were the reasons for the appearance of hybrid parsing methods which have been first applied in the area of shallow parsing, such as supertagging or attachment prediction [WH02], and later transferred to full syntactic parsing as well. For example, the rule-based parsers presented in [RM90] or [CC04] use statistical methods to restrict the space of search possibilities.

The greatest gains of hybrid methods lie in the combination of the different sources of knowledge. This may be of advantage even among data-driven approaches. For example, in [SL06] several statistical parsing solutions are applied to the same data and an additional algorithm makes further decisions based on the suggested variants. Or, purely stochastic solutions may be trained on different data collections and their results may be combined for parsing. In the recent time, much research in the area of hybrid methods is being done [Ric94, HB02, Zu05]. In this work, an extension of a rule-based system with a statistical parser is pursued.

The main hurdle that hybrid methods have to overcome is the imperfection of the components combined. Errors in one component may be propagated into the next and lead to a substantial decrease of performance, e.g., a parser that completely relies on the part of speech information provided by the tagger will most probably make a mistake when the tagger fails. A *preference-based solution* is one of the possibilities to deal with this problem. According to it, conflicting information from different sources should be assessed for its reliability and even overridden if enough evidence is present for some other alternative [FM06b].

Evaluation Measures

To evaluate parsing results, such measures as *precision* p, defined as the ratio of the number of correct predictions to the number of all the predictions made by the parser, and *recall* r, understood as the ratio between the number of correct predictions and desired predictions, are mostly used [Fot06]. Considering either of them in isolation does not

make sense as they are antagonistic in their nature, they are also combined as part of the so-called *f-measure* [Rij79] which may by defined as

$$F = \frac{(\beta^2 + 1)pr}{\beta^2 p + r},$$

where β is a parameter that allows to favor either of the two, precision or recall, as desired.

As already mentioned, it is rather straightforward to evaluate an analysis made by a dependency parser, as per edge correctness may be easily computed. For parsers, that never fail to return an analysis such as those that will be considered in the present work, the measures of parsing performance are summed up under the term *accuracy* since precision, recall and f-score coincide. Thereby, *structural accuracy* points out the percentage of words correctly attached to their regent, and *labeled accuracy* is the ratio of the correctly attached words which also have the correct label.

Although well-defined general measures to evaluate parsing performance exist, it is difficult to directly compare the results reported for different parsers as the evaluations are influenced by the data used during the experiment, the domain of the data, or various annotation guidelines as well as the source of the part of speech information that may be obtained from the manual annotations or disambiguated during the parsing process. Even such a condition as inclusion or not of the punctuation into the results has not yet become a standard.

1.3 Scope of This Thesis

This work is a successor of the previous efforts on hybrid methods of syntactic language processing realized with *Weighted Constraint Dependency Grammar*, WCDG, [Fot06, FHS+05, Sch02] at the *Natural Language Systems Division* (NATS) of the Department of Informatics, University of Hamburg. The WCDG system has successfully combined rule-based parsing, the rules in this case are written in form of constraints, with five additional statistical predictor components which improved the performance of the system by over twenty percent.

Integrating predictors into WCDG has shown itself as a promising way of parse improvements. At the moment, a simple shift-reduce parser is acting in WCDG as a full parsing predictor. It is known for its rather modest parsing accuracy and inability to deal with non-projectivity. Even such an unreliable predictor brings a performance increase of over 3 percent. A recent study of statistical parsers during the CoNLL-X shared task 2006 [BMDK06] has reported the best results among 14 languages of the *MSTParser* by R. McDonald [MLP06] who combined discriminative learning with maximum spanning tree parsing. The perspectives of integrating the MSTParser as a predictor for WCDG were outlined in [FM06b]. In addition to high performance (the reported accuracy on the German data is 90.4% structural and 87.3% labeled accuracy, especially attractive is the

stated ability of this parser to deal with non-projective data and the low complexity of suggested algorithms.

The evaluation of the perspectives to use the MSTParser as a predictor for WCDG, restricted to the German language is the main goal of the present work. Because of the reasons outlined in the previous section, the evaluation of the MSTParser on the in-house data is a necessary step on the way to it. This would also allow for a reliable comparison of the performance of the two parsers as the measurements will be performed on the same data according to the same criteria.

This thesis is organized as follows. The next two chapters study the MSTParser in detail: the theoretical background about online learning methods and the MIRA algorithm used in the MSTParser are revised in Chapter 2 and the leveraged parsing algorithms are summarized in Chapter 3. Chapter 4 gives an overview of the WCDG system and the hybrid parsing methods successfully applied in it so far. Chapter 5 presents the results of evaluation of the MSTParser on the same German data WCDG has been mostly tested on and Chapter 6 sums up the results of integrating the MSTParser as a predictor for WCDG. Finally, in Chapter 7, the achieved results are summarized and an outlook on further research is given.

Chapter 2

Online Learning for Natural Language Parsing

An important feature of a learning machine is that its teacher will often be very largely ignorant of quite what is going on inside, although he may still be able to some extent to predict his pupil's behavior.

Alan Turing

The maximum spanning trees parsing framework constructed by R. McDonald [McD06, MLP06], or the *MSTParser*, takes a set of parsed sentences during the *training phase* and outputs a *parsing model*. The responsibility for the success of the training is taken by a *learning algorithm* which is generic and will produce different parsing models under different input data. The *test phase* is controlled by the *parser* which consists of both the parsing model and an *inference algorithm*, also called the *parsing algorithm*. When the parser gets a new sentence $\boldsymbol{x}$ it uses the model parameters to produce a syntactic representation $\boldsymbol{y}$.

This chapter will analyze the online learning algorithm building the center of R. McDonald's framework. The theoretical background, necessary for its understanding, such as machine learning approaches to the classification problem as well as specific solutions like large margin classifiers, will be revised first. After that it will be shown how these methods can be applied to the structural classification problems and natural language parsing as their subtype.

2.1 Classification Problem in Machine Learning

Much of this section is drawn from [CST03, SS02, TGK04, Vap00].

Traditional programming methods explicitly specify how the correct output should be computed from the input data. An alternative to this approach provided by the *learning methodology* suggests ways to learn the input/output mapping directly from existing examples, especially, in the cases where no known method for computing the desired output from a set of inputs is known (such as classification of protein types based on DNA sequence from which they are generated) or the methods are too complex (like handwriting analysis or natural language parsing). There are several types of learning suggested by the learning methodology, among them *supervised learning*, in which case the input data is provided in pairs with the corresponding output (both together referred to as *training data*) and which is of utmost importance for the present work. [1]

The input/output pairs in the general case reflect a functional relationship mapping of inputs to outputs called the *target function*, or *decision function*, exception being made for situations in which the outputs are corrupted by noise. Usually, before the attempt is made to learn the correct function, a particular set or class of candidate functions referred to as *hypotheses* is chosen.

One of the fundamental supervised learning problems is *classification*. A classifier is a function that maps an input to an output. In *binary classification problems*, the goal is to assign each input instance one of two possible labels. For example, classifying sounds of a language strictly into vowels and consonants is a binary classification problem. If the number of labels is finite, the problem is called *multi-class classification*, such as in the case of the part of speech tagging. Binary classification can be viewed as a multi-class classification with only two classes. If the output is not a single value, but a set or a sequence of values, such as when assigning a parse tree to an input sentence, the problem is referred to as *structured classification* and as practice has shown that it is a good abstraction for natural language parsing [TGK04, McD06] and will be thoroughly presented in the next section.

The supervised classification setting can formally be described as follows. Let $\boldsymbol{x}$ denote an input, $\boldsymbol{x} \in \mathcal{X}$, where $\mathcal{X}$ is the space of all possible inputs, or *input space*. Further, let y denote the output, $y \in \mathcal{Y}$, and $\mathcal{Y}$ be the discrete *output space*. The input space is an arbitrary set, often chosen as $\mathcal{X} = \mathbb{R}^n$, while the output space for a multi-class classification is $\mathcal{Y} = \{y_1, \cdots, y_k\}$, where k is the number of classes.

A classifier, or hypothesis, h is a function from $\mathcal{X}$ to $\mathcal{Y}$, $h : \mathcal{X} \to \mathcal{Y}$. The set of all classifiers that the learning program can produce is denoted as $\mathcal{H}$ (*hypothesis class*). The input to a learning algorithm is a set of m i.i.d. (independent and identically distributed) samples $S = \{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{m}$, drawn from a fixed but unknown distribution D over $\mathcal{X} \times \mathcal{Y}$ and the learning program aims at finding a classifier $h \in \mathcal{H}$ that will work well on unseen examples, i.e., such that $h(\boldsymbol{x})$ will approximate y on new samples from the distribution $(\boldsymbol{x}, y) \sim D$; this property being one of the most important of h is called *generalization*.

[1] Other types of learning include, e. g., *unsupervised learning* in which case no output values are provided and some understanding of the process that generates the data should be achieved or *reinforcement learning* whereby the learner can take some of eligible actions to move toward states where they can expect high rewards.

The variation in classification algorithms is achieved by choosing the hypothesis class $\mathcal{H}$ and the criterion for selection of a hypothesis h from $\mathcal{H}$ given the training data. The task of selecting a hypothesis h is reduced to estimating model parameters. As already mentioned in Section 1.2, probabilistic estimation methods associate a joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$ or conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ with h and select a model based on likelihood of the data. Joint distribution models are often called *generative*, while conditional models are called *discriminative*.

A criterion for selection of h from $\mathcal{H}$ can be derived by qualifying what it means for $h(\boldsymbol{x})$ to approximate y. The measure for the expected error of the approximation is *risk functional*, defined as:

$$\mathcal{R}^{\mathbb{L}}_{D}[h] = \mathbf{E}_{(\boldsymbol{x},y)\sim D}[\mathbb{L}(\boldsymbol{x}, y, h(\boldsymbol{x}))], \tag{2.1}$$

where $\mathbb{L} : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^{+}$ is the *loss function* which measures the penalty for predicting $h(\boldsymbol{x})$ on the sample $(\boldsymbol{x}, y)$. In general, it is assumed that $\mathbb{L}(\boldsymbol{x}, y, \hat{y}) = 0$ if $y = \hat{y}$.

A common loss function for classification is *0/1 loss*:

$$\mathbb{L}^{0/1}(\boldsymbol{x}, y, h(\boldsymbol{x})) = \mathbb{I}(y \neq h(\boldsymbol{x})),$$

where $\mathbb{I}(\cdot)$ denotes the indicator function, $\mathbb{I}(true) = 1$ and $\mathbb{I}(false) = 0$.

Since the distribution D is generally not known, the risk of h is estimated using its *empirical risk* $\mathcal{R}^{\mathbb{L}}_{S}$, which on the training sample S is given by:

$$\mathcal{R}^{\mathbb{L}}_{S} = \frac{1}{m}\sum_{i=1}^{m}\mathbb{L}(\boldsymbol{x}^{(i)}, y^{(i)}, h(\boldsymbol{x}^{(i)})) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{L}_{i}(h(\boldsymbol{x}^{(i)})), \tag{2.2}$$

where $\mathbb{L}_{i}(h(\boldsymbol{x}^{(i)}))$ is the abbreviation for $\mathbb{L}(\boldsymbol{x}^{(i)}, y^{(i)}, h(\boldsymbol{x}^{(i)}))$. For $0/1$ loss, $\mathcal{R}^{\mathbb{L}}_{S}$ is simply the proportion of training examples that h misclassifies. $\mathcal{R}^{\mathbb{L}}_{S}$ is also called *training error* or *training loss*.

Empirical risk minimization as a criterion for the selection of the solution hypothesis from the hypothesis class might not provide good generalization. This can be illustrated by the fact that for each h and any training sample $\{x^{(i)}, y^{(i)}\}_{i=1}^{m}$, satisfying $\{x^{(1)}, \cdots, x^{(m)}\} \cap \{\bar{x}^{(1)}, \cdots, \bar{x}^{(m)}\} = \emptyset$, there exists another hypothesis h^{*} such that $h^{*}(x^{(i)}) = h(x^{(i)})$ for all $i = 1, \cdots, m$, yet $h^{*}(\bar{x}^{(i)}) \neq h(\bar{x}^{(i)})$ for all $i = 1, \cdots, m$.

The key to selecting an optimal hypothesis h^{*} is to trade-off complexity of class $\mathcal{H}$ with the error on the training data as measured by the empirical risk (2.2). This fundamental balance is generally achieved by minimizing the weighted combination of the two criteria:

$$h^{*} = \arg\min(\mathcal{D}[h] + C\mathcal{R}^{\mathbb{L}}_{S}[h]), \tag{2.3}$$

where $\mathcal{D}[h]$, often referred to as *regularization*, measures the inherent dimension of complexity of h, and $C \geq 0$ is a trade-off parameter. Derivation of the different complexity

measures is the main task of the statistical learning theory, or VC (Vapnik-Chervonenkis) theory [Vap00] which shows that it is imperative to restrict the set of functions from which h^* is chosen to one that has *capacity* suitable for the amount of available training data. VC theory provides *bounds* on the test error. The minimization of these bounds which depend on both the empirical risk and the capacity of the function class, leads to the principle of *Structural Risk Minimization*.

Most of the recent research has been concentrated on linear models where the optimal hypothesis h^* can be found efficiently using convex optimization in polynomial time. The linear model for binary classification as its simplest kind allows to quickly define and demonstrate the main concepts and abstractions regarding classification problem description, learning algorithms and their analysis and will be introduced next.

2.1.1 Linear Classification

Linear functions are intensively used in different fields of mathematical research, among them in traditional statistics and machine learning. Their properties are thoroughly studied and they are simple to apply and implement. A linear function of the form:

$$h(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \tag{2.4}$$

where $(\boldsymbol{w}, b) \in \mathbb{R}^n \times \mathbb{R}$ are the parameters controlling the function, can be used as a decision function for binary classification if the decision rule is given by $\mathrm{sgn}(f(\boldsymbol{x}))$.[2] According to the learning methodology, the parameters $(\boldsymbol{w}, b)$ have to be learned from the data.

This kind of hypothesis is geometrically a *linear separator*, i. e., the hyperplane $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ that splits the input space into two parts. In Figure 2.1, the hyperplane is the dark line separating the two classes presented by crosses and naughts respectively. The parameter $\boldsymbol{w}$, referred to as a *weight vector*, is a vector orthogonal to the hyperplane and affects the orientation of the hyperplane in the space, while the *bias* parameter b controls the distance of the hyperplane from the origin.

If a hyperplane that correctly classifies the training data exists, the data is called *linearly separable*, in the reverse case, the data is called *non-separable*.

Very important parameter concerning the hyperplane for the further discussion is the quantity called the *margin*. It also plays the central role in the learning algorithm used in the framework of R. McDonald.

Definition 2.1. The *functional margin of an example* $(\boldsymbol{x}_i, y_i)$ with respect to a hyperplane $(\boldsymbol{w}, b)$ is the quantity

$$\gamma_i = y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b). \tag{2.5}$$

[2] It will be assumed here that $\mathrm{sgn}(0) = 1$.

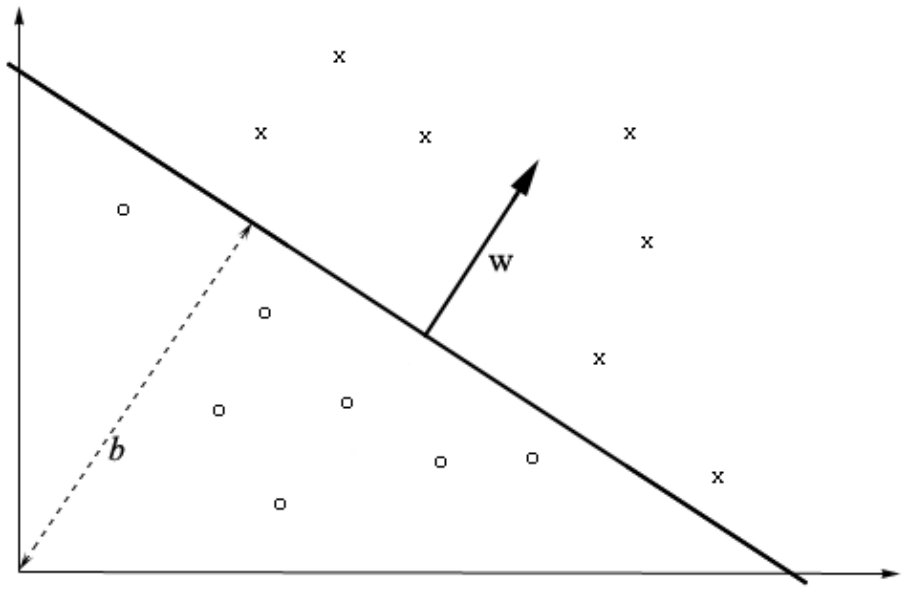


Figure 2.1: A separating hyperplane $(\boldsymbol{w}, b)$ for a two-dimensional training set; cf.: [CST03].

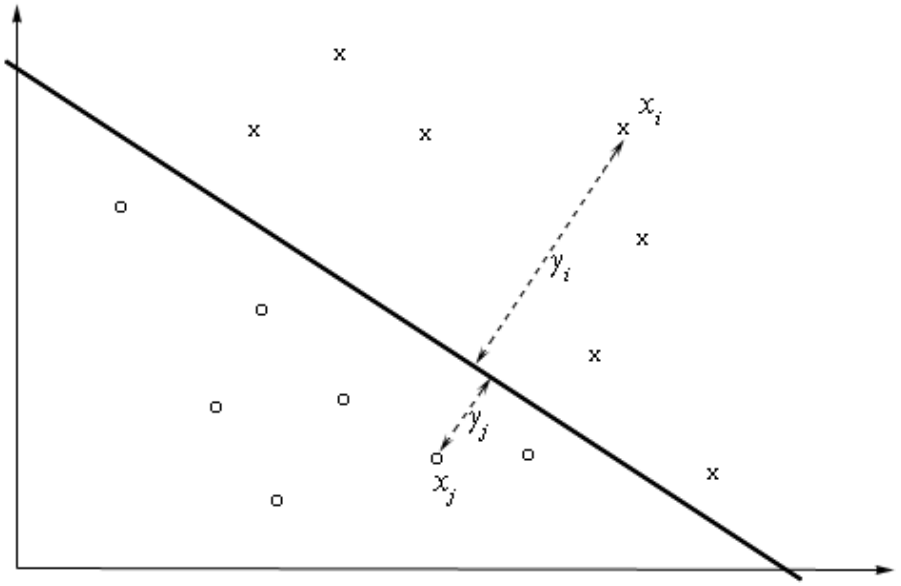


Figure 2.2: The geometric margin of two points; cf.: [CST03].

The margin $\gamma_i > 0$ implies that the example $(\boldsymbol{x}_i, y_i)$ from S has been classified correctly.

The minimum of the margin distribution (i. e. the distribution of the margins of the examples in S) is often referred to as the *functional margin of a hyperplane* $(\boldsymbol{w}, b)$ with respect to a training set S. The *geometric margin* is obtained from the functional margin for the normalized linear function $\left(\frac{1}{\|\boldsymbol{w}\|}\boldsymbol{w}, \frac{1}{\|\boldsymbol{w}\|}b\right)$ and measures the Euclidean distances of the points

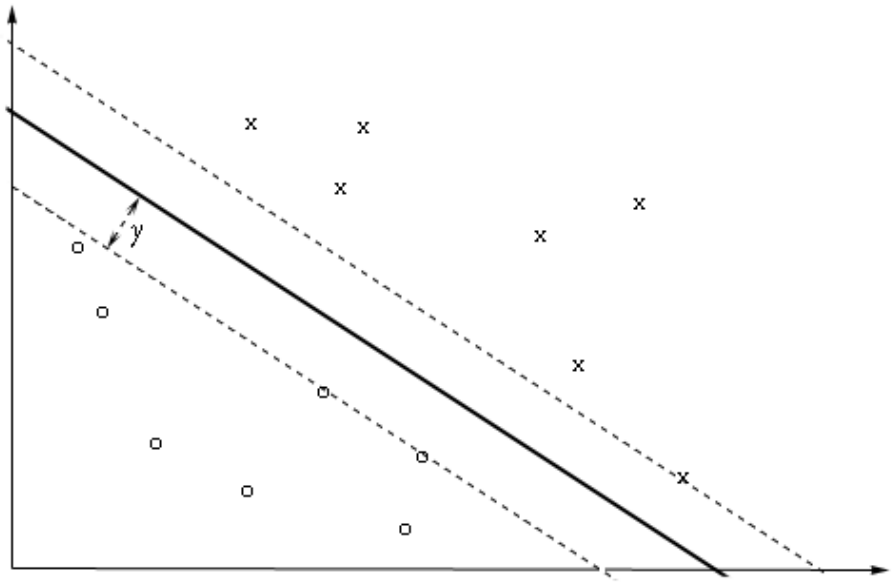


Figure 2.3: The margin of a training set; cf.: [CST03].

from the decision boundary in the input space (see Figure 2.2). The geometric margin will equal the functional margin if the weight vector is a unit vector. Besides, the *margin of a training set* S (Figure 2.3) is the maximum geometric margin over all hyperplanes. A hyperplane realizing this maximum is known as the *maximal margin hyperplane*. For a linearly separable training set, the size of the margin will be positive.

The earliest usage of linear functions in the context of machine learning goes back to Fisher [Fis52] who suggested them for classification more than 70 years ago. One of the first algorithms utilizing linear functions that recently gained its popularity again [Col02, CR04, RSCJ04, Moo05] is the F. Rosenblatt's Perceptron dating back to 1956. Since then, it has proved to be effective for a broad rage of applications and gave birth to a number of enhancements, one of them being the learning algorithm used for R. McDonald's learning framework.

The Perceptron algorithm is used in its original form for binary classification of a linearly separable training set S of length m, it maintains a separating hyperplane $(\boldsymbol{w}, b)$ that is used for prediction. Given an input instance $\boldsymbol{x}$ the Perceptron algorithm predicts its label by first computing $\hat{y} = \text{sgn}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle)$. The algorithm modifies $\boldsymbol{w}$ only on rounds with prediction errors changing $\boldsymbol{w}$ to $\boldsymbol{w} + \boldsymbol{x}$ if the correct label is $y = 1$ and to $\boldsymbol{w} - \boldsymbol{x}$ if $y = -1$. This update rule can be summarized by $\boldsymbol{w} = \boldsymbol{w} + y\boldsymbol{x}$. The pseudocode of the algorithm is shown in the left column of Figure 2.4. It is referred to as a *primal form* to reflect the fact that it updates the weight vector and the bias directly in contrast to a *dual form* presented on the left of Figure 2.4 which will be discussed in a moment.

Initialize:

- Set $b = 0$.
- Set $R = \max_{1 \leq i \leq m} \|\boldsymbol{x}_i\|$.

Primal

- Set $\boldsymbol{w}^{(0)} = 0$.

Dual

- Set $\alpha = 0$.

Loop: $i = 1, \cdots, m$

- Get a new instance $\boldsymbol{x}_i \in \mathbb{R}^n$.

Primal

- Predict $\hat{y}_i = \langle \boldsymbol{w}^{(i)}, \boldsymbol{x}_i \rangle$.
- Get a new label y_i.
- Compute $L^{0/1} = L^{0/1}\big(\boldsymbol{w}^{(i)}, (\boldsymbol{x}_i, y_i)\big)$.
- Update :

Primal

- If $L^{0/1} = 0$ set $\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)}$.
- If $L^{0/1} = 1$ set $\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)} + y_i \boldsymbol{x}_i$
- and set $b^{(i+1)} = y_i R^2$.

Dual

- Predict $\hat{y}_i = \sum_{j=0}^{i-1} \alpha_j \langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle$.

Dual

- If $L^{0/1} = 0$ preserve α_i.
- If $L^{0/1} = 1$ set $\alpha_i = \alpha_i + 1$
- and set $b^{(i+1)} = y_i R^2$.

Output:

Primal

- $h(\boldsymbol{x}) = \langle \boldsymbol{w}^{(m)}, \boldsymbol{x} \rangle$.

Dual

- $h(\boldsymbol{x}) = \sum_{j=0}^{m} \alpha^j y_j \langle \boldsymbol{x}_j, \boldsymbol{x} \rangle$.

Figure 2.4: The Perceptron algorithm: primal and dual form; cf. [Cra04, CST03]

It does not influence the generality if the initial weight vector is the zero vector as it is assumed by the Perceptron algorithm. In combination with the additive update rule, this accounts for the fact that the final hypothesis is a linear combination of the training set:

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i,$$

where, since the classification y_i provides the sign of the coefficient of $\boldsymbol{x}_i$, the α_i are positive values proportional to the number of times misclassification of $\boldsymbol{x}_i$ has caused the weight vector to be updated. Once a sample S has been fixed, one can think of the vector α as alternative representation of the hypothesis in different or *dual coordinates*. This expansion is however not unique: different α can correspond to the same hypothesis $h(\boldsymbol{x})$. The α_i can also be regarded as an indication of the information content of the example $\boldsymbol{x}_i$.

The analysis of the Perceptron algorithm made by Novikoff [Nov62] and Block [Blo62] has shown that for linearly separable data with non-zero margin, the algorithm terminates and the number of mistakes the Perceptron algorithm will perform is bounded.

Besides, what makes it especially attractive for present research, thirty years later Freund and Schapire have extended the analysis for the non-separable case [FS98].

2.1.2 Quadratic Optimization

For what follows, some words should be said about the algorithmic techniques for optimizing quadratic convex functions with linear constraints. The general form of the problem that is particular important for the next discussion is:

Definition 2.2. (*Primal optimization problem*) Given functions $f, g_i, i = 1, \ldots, k$, and $h_i, i = 1, \ldots, m$, defined on a domain $\boldsymbol{\Omega} \subseteq \mathbb{R}^n$,

$$\begin{array}{lll} \text{minimize} & f(\boldsymbol{w}), & \boldsymbol{w} \in \Omega, \\ \text{subject to} & g_i(\boldsymbol{w}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\boldsymbol{w}) = 0, & i = 1, \ldots, m, \end{array}$$

where $f(\boldsymbol{w})$ is called the *objective function*, and the remaining relations are called, respectively, the *inequality* and *equality constraints*. The optimal value of the objective function is called the *value of the optimization problem*.

Only a restricted class of *convex* quadratic optimization problems is important for the present discussion. That is, the problems in which the objective function is convex and quadratic and $\Omega = \mathbb{R}$ and the constraints are linear.

If a function f is convex, any local minimum $\boldsymbol{w}^*$ of the unconstrained optimization problem with objective function f is also a global minimum, since for any $\boldsymbol{u} \neq \boldsymbol{w}^*$, by the definition of a local minimum there exists θ sufficiently close to 1 that

$$\begin{aligned} f(\boldsymbol{w}^*) &\leq f(\theta \boldsymbol{w}^* + (1-\theta)\boldsymbol{u}) \\ &\leq \theta f(\boldsymbol{w}^*) + (1-\theta) f(\boldsymbol{u}). \end{aligned}$$

And it follows that $f(\boldsymbol{w}^*) \leq f(\boldsymbol{u})$. It is this property of convex functions that makes optimization problems tractable when such functions are involved.

Lagrangian treatment of convex optimization problems (see Appendix B) leads to an alternative dual description, which often turns out to be easier to solve than the primal problem as handling inequality constraints directly is difficult. The dual problem is obtained by introducing Lagrange multipliers, also called *dual variables*, that become the fundamental unknown of the problem.

Primal can be transformed into dual by setting the derivatives of the Lagrangian with respect to the primal variables to zero and substituting the relations obtained back into the Lagrangian. This way, the dependence on the primal variables is removed and corresponds to explicitly computing the function

$$\theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_{w \in \Omega} L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}).$$

This transformation is applied to the maximal margin classifier that will be considered in the next section (2.1.3).

There are also ready-made programmatic applications to the described technique, such as Hildreth's algorithm [CZ97] that was specially designed for the case when functions of the form $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|^2$ should be optimized (see Figure 2.5).

It is a primal-dual algorithm that iteratively updates primal $\{\boldsymbol{x}^\nu\}$ and dual $\{\alpha^\nu\}$ and iterates in such a manner that only one component ν of the dual vector is actually changed in a single iterative step. The size of the change c_ν is such that it guarantees dual ascent. The convergence of Hildreth's algorithm has been thoroughly analyzed in literature, e. g. [Hil57, D'E59, LC91].

Problem:

$$\text{Minimize} \qquad \frac{1}{2}\|\boldsymbol{x}\|^2 \tag{2.6}$$

$$\text{such that} \qquad \langle \boldsymbol{a}^i, \boldsymbol{x}\rangle \leq b_i, \qquad i \in [n], \tag{2.7}$$

where $f(x) = \frac{1}{2}\|\boldsymbol{x}\|^2, \quad f : \mathbb{R}^m \to \mathbb{R}$ and $\boldsymbol{a}^i : \mathbb{R}^m \to \mathbb{R}$ are convex functions and $[n] \in \mathbb{N}$.

Initialization: $\alpha^0 \in \mathbb{R}^m_+$ is arbitrary and $\boldsymbol{x}^0 = -\boldsymbol{A}'\alpha^0$.

Iterative step:

$$\boldsymbol{x}^{\nu+1} = \boldsymbol{x}^\nu + c_\nu \boldsymbol{a}^{i(\nu)}, \tag{2.8}$$

$$\alpha^{\nu+1} = \alpha^\nu - c_\nu \boldsymbol{e}^{i(\nu)}, \tag{2.9}$$

with

$$c_\nu = \min\left(\alpha^\nu_{i(\nu)}, \lambda_\nu \frac{b_{i(\nu)} - \langle \boldsymbol{a}^{i(\nu)}, \boldsymbol{x}^\nu\rangle}{\|\boldsymbol{a}^{i(\nu)}\|^2}\right). \tag{2.10}$$

Relaxation parameters: For all $\nu \geq 0, \quad \epsilon_1 \leq \lambda_\nu \leq 2 - \epsilon_2$, for some arbitrary small but fixed $\epsilon_1, \epsilon_2 > 0$.

Control: The sequence $\{i(\nu)\}$ is almost cyclic on $[n]$; $\boldsymbol{e}$ is the basis vector.

Figure 2.5: Hildreth's Algorithm; cf. [CZ97].

A geometric interpretation of the algorithm is as follows. At one iteration, $\boldsymbol{x}^\nu$, α^ν and the closed half-space $L_{i(\nu)} = \{\boldsymbol{x} \in \mathcal{R}^m | \langle \boldsymbol{a}^{i(\nu)}, \boldsymbol{x}\rangle \leq b_{i(\nu)}\}$, determined by the $i(\nu)$th inequality of (2.7) are given. If $\boldsymbol{x}^\nu \notin L_{i(\nu)}$, then $\boldsymbol{x}^{\nu+1}$ is the orthogonal projection of x^ν onto $L_{i(\nu)}$. If $\boldsymbol{x}^\nu$ belongs to the bounding hyperplane $h_{i(\nu)}$ then $\boldsymbol{x}^{\nu+1} = \boldsymbol{x}^\nu$. Finally, if $\boldsymbol{x}^\nu \in \operatorname{int} L_{i(\nu)}$, i.e., if $\langle \boldsymbol{a}^{i(\nu)}, \boldsymbol{x}^\nu\rangle < b_{i(\nu)}$, then a move perpendicular to the bounding hyperplane $h_{i(\nu)}$ is made. In this case, either $c_\nu = \alpha^\nu_{i(\nu)}$ or $\boldsymbol{x}^{\nu+1}$ is the orthogonal projection of $\boldsymbol{x}^\nu$ onto $h_{i(\nu)}$. These possibilities are shown in Figure 2.6.

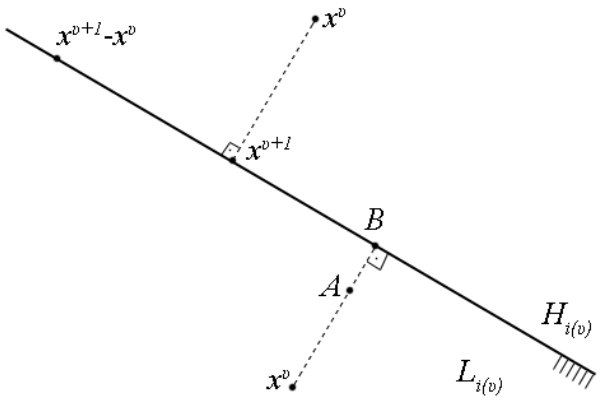


Figure 2.6: Possible cases in the iterative step of Hildreth's algorithm; cf.: [CZ97].

2.1.3 Maximal Margin Classifier

One of the main propositions made by the already mentioned statistical learning theory is that the generalization error of linear classifiers is bounded in terms of the margin $m_S(f)$ of the hypothesis f with respect to the training set S. The dimensionality of the space in which the data are separated does not play any role. Besides, the statistical learning theory proves that this bound will be optimal if the hyperplane separating the data is the maximal margin hyperplane. It is the geometric margin that is referred to but the inherent degrees of freedom in the definition of linear classifiers allow to use the functional margin instead. The function associated with the hyperplane $(\boldsymbol{w}, b)$ does not change if the hyperplane is rescaled to $(\lambda\boldsymbol{w}, \lambda b)$ for $\lambda \in \mathcal{R}^+$. There will, however, be a change in the margin as measured by the function output as opposed to the geometric margin. Hence, the geometric margin can be equally maximized by fixing the functional margin to be equal to 1 (the hyperplane in this case is referred to as *canonical hyperplane*) and minimizing the norm of the weight vector.

Thus, if $\boldsymbol{w}$ is the weight vector realizing a functional margin of 1 on the positive point $\boldsymbol{x}^+$ and the negative point $\boldsymbol{x}^-$, the geometric margin can be computed as follows. The functional margin of 1 implies

$$\begin{aligned}\langle \boldsymbol{w}, \boldsymbol{x}^+ \rangle + b &= +1, \\ \langle \boldsymbol{w}, \boldsymbol{x}^- \rangle + b &= -1,\end{aligned}$$

while to compute the geometric margin one has to normalize $\boldsymbol{w}$ first. The geometric margin γ is then the margin of the resulting classifier

$$\begin{aligned}\gamma &= \frac{1}{2}\left(\langle \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}, \boldsymbol{x}^+ \rangle - \langle \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}, \boldsymbol{x}^- \rangle \right) \\ &= \frac{1}{2\,\|\boldsymbol{w}\|}\left(\langle \boldsymbol{w}, \boldsymbol{x}^+ \rangle - \langle \boldsymbol{w}, \boldsymbol{x}^- \rangle \right) \\ &= \frac{1}{\|\boldsymbol{w}\|}.\end{aligned}$$

Hence, the resulting geometric margin will be equal to $\frac{1}{\|\boldsymbol{w}\|}$ and the whole result can be summed up as follows.

Given a linearly separable training sample $S = \{\boldsymbol{x}_i, y_i\}_{i=1}^m$, the hyperplane $(\boldsymbol{w}, b)$ that solves the optimization problem

$$\begin{aligned} &\text{minimize}_{\boldsymbol{w},b} && \langle \boldsymbol{w}, \boldsymbol{w} \rangle, \\ &\text{subject to} && y_i\big(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b\big) \geq 1, \\ & && i = 1, \ldots, m, \end{aligned}$$

realizes the maximal margin hyperplane with geometric margin $\gamma = \frac{1}{\|\boldsymbol{w}\|}$.

According to the technique borrowed from the Lagrangian theory (Appendix B) and presented shortly in the previous section, the corresponding dual problem of this quadratic optimization problem can be derived. The primal Lagrangian is in this case

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w} \rangle - \sum_{i=1}^{m} \alpha_i [y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) - 1]$$

where $\alpha_i \geq 0$ are Lagrangian multipliers.

To find the corresponding dual one has to differentiate with respect to $\boldsymbol{w}$ and b

$$\begin{aligned} \frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} &= \boldsymbol{w} - \sum_{i=1}^{m} y_i \alpha_i \boldsymbol{x}_i = \boldsymbol{0}, \\ \frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} &= \sum_{i=1}^{m} y_i \alpha_i = 0, \end{aligned}$$

whereby the following relations are obtained

$$\begin{aligned} \boldsymbol{w} &= \sum_{i=1}^{m} y_i \alpha_i \boldsymbol{x}_i, \\ 0 &= \sum_{i=1}^{m} y_i \alpha_i, \end{aligned}$$

the first of them is the optimization theory evidence to the dual representation already introduced in Section 2.1.1. Substituting both relations into the primal yields

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w} \rangle - \sum_{i=1}^{m} \alpha_i [y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) - 1] \\ &= \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \sum_{i=1}^{m} \alpha_i \\ &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \end{aligned}$$

Thus, the dual optimization problem that has to be solved is

$$\begin{aligned} &\text{maximize}_{\boldsymbol{\alpha}^*} && W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\ &\text{subject to} && \sum_{i=1}^{m} y_i \alpha_i = 0, \\ & && \alpha_i \geq 0, i = 1, \ldots, m, \end{aligned}$$

and the weight vector $\boldsymbol{w}^* = \sum_{i=1}^{m} y_i \alpha_i^* \boldsymbol{x}_i$ realizes in this case the maximal margin hyperplane with geometric margin $\gamma = \frac{1}{\|\boldsymbol{w}^*\|}$.

2.1.4 Online Learning

The Perceptron algorithm (Figure 2.4) presented in Subsection 2.1.1 belongs to the so-called group of online learning algorithms. A general form of an online learning algorithm is shown in Figure 2.7. An online algorithm works in rounds. On round i, it receives an instance $\boldsymbol{x}_i$. Given $\boldsymbol{x}_i$, it outputs a prediction $\hat{y}_i = h(\boldsymbol{x}_i)$. It then receives the correct label $y_i \in \mathcal{Y}$ which is evaluated according to the loss function $\mathbb{L}(y_i, \hat{y}_i)$ used. The algorithm then updates its prediction rule h which for the linear functions means updating the weight vector $\boldsymbol{w}$ according to the rule specific to the algorithm. The auxiliary vector $\boldsymbol{v}$ accumulates the successive values of $\boldsymbol{w}$ so that the final weight vector is the *average* of the weight vectors after each iteration which helps to reduce the so-called *overfitting* as suggested in [Col02].

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{T}$

1. $\boldsymbol{w}^{(0)} = 0; \boldsymbol{v} = 0; i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Given $\boldsymbol{x}_i$, predict $\hat{y}_i$
5. Get y_i and compute loss $\mathbb{L}(y_i, \hat{y}_i)$
6. $\boldsymbol{w}^{(i+1)}$ = update $\boldsymbol{w}^{(i)}$ to minimize loss $\mathbb{L}$
7. $\boldsymbol{v} = \boldsymbol{v} + \boldsymbol{w}^{(i+1)}$
8. $i = i + 1$
9. $\boldsymbol{w} = \boldsymbol{v}/(N * T)$

Figure 2.7: Generic online learning algorithm; cf. [McD06].

Hypotheses are said to *overfit* if they become too complex to find an accurate fit to the training data. This happens, for example, if the hypothesis is allowed to grow unboundedly in size.

While the online model focuses on the *learning process* itself as the goal of an online algorithm is to minimize the cumulative loss during the training process, its opposite, the *batch model*, divides the learning process into two phases — a training and a test — and does not care about the performance in the training phase in which the learning algorithm has access to a finite set of examples and constructs a prediction function taking all available instances into consideration simultaneously. It optimizes the values so that the algorithm performs best in the test phase in which the prediction function is not modified.

On one iteration, online algorithms consider only one instance, whereas batch algorithms optimize their parameters according to several instances at once. Practical results proved that this potential weakness is balanced over by the simplicity of online learning which scales better to large problems than batch algorithms as shown by [McD06].

One essential drawback of the Perceptron algorithm is that is does not optimize any notion of the classification margin which, as shown in the previous subsection, would be a good way to reduce the generalization error. Y. Crammer [Cra04] has proposed a large-margin modification of the Perceptron algorithm that does not suffer from this weakness.

The *Margin Infused Relaxed Algorithm* (MIRA) of Y. Crammer for binary classification derives its update rule (line 6 of the generic online algorithm in Figure 2.7) as the solution to the following problem

$$\begin{aligned} \boldsymbol{w}^{i+1} = \arg\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}^i\|^2 \\ \text{such that } \mathbb{L}_\gamma(\boldsymbol{w}, (\boldsymbol{x}, y)) = 0, \end{aligned} \tag{2.11}$$

$\boldsymbol{w}^{i+1}$ is set to be the projection of $\boldsymbol{w}^i$ onto the set of all weight vectors that guarantee a zero loss. For the problem of binary classification it is a half space, $\{\boldsymbol{w} : y_i\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle \geq \gamma\}$. On each update,MIRA attempts to keep $\boldsymbol{w}^{i+1}$ as close to $\boldsymbol{w}^i$ as possible, while forcing $\boldsymbol{w}^{i+1}$ to provide a zero loss on the most recent example. Thus, the algorithm maximizes the margin on the current example.

As shown in [Cra04], the solution to the optimization problem in (2.12) yields the following update rule:

$$\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)} + \alpha^i y_i \boldsymbol{x}_i, \tag{2.12}$$

where α^i is the corresponding dual with value

$$\alpha^i = \frac{\mathbb{L}_\gamma(\boldsymbol{w}, (\boldsymbol{x}_i, y_i))}{\|\boldsymbol{x}_i\|^2}. \tag{2.13}$$

MIRA algorithm has been successfully adapted to the setting of the structural classification by B. Taskar [TGK04]. The next section will demonstrate how these adaptations were used for syntactic parsing by R. McDonald.

2.2 Natural Language Parsing as Structured Classification Problem

In syntactic parsing, the provided input in form of a sentence, should be mapped to the output in form of, e.g., a dependency tree which can be abstractly seen as an ordered set of labels. That is why, syntactic parsing fits into the setting of the structured classification problems in machine learning.

Formally, for parsing we want to know a function $s : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ is a set of sentences and $\mathcal{Y}$ is a set of valid parse trees according to a fixed grammar $\mathbf{G}$. $\mathbf{G}$ maps an input $x \in \mathcal{X}$ to a set of candidate parses $\mathbf{G}(x) \subseteq \mathcal{Y}$. The function s is further referred to as the *score function* and takes the following linear form

$$s(x, y) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(x, y) \rangle, \tag{2.14}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ is a weight vector and $\boldsymbol{\Phi}$ is a feature-vector representation of a parse tree $\boldsymbol{\Phi} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$. The goal of learning is to obtain $\boldsymbol{w}$ so that correct outputs are given a high score according to s and incorrect outputs a low score.

As shown by B. Taskar [TGK04], maximal margin methods can be successfully used for this setting, too. He defines the margin of the parameters $\boldsymbol{w}$ on the example i and parse $\hat{y}$ as the difference value between the true parse y_i and $\hat{y}$:

$$s(x_i, y_i) - s(x_i, \hat{y}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(x_i, y_i) \rangle - \langle \boldsymbol{w}, \boldsymbol{\Phi}(x_i, \hat{y}) \rangle = \langle \boldsymbol{w}, \boldsymbol{\Delta\Phi}_i \rangle, \tag{2.15}$$

whereby the size of the margin quantifies the confidence in rejecting the mistaken parse y using the score function s, modulo the scale of the parameters $\|\boldsymbol{w}\|$. This rejection confidence has to be larger when the mistake is more severe, i. e. $\mathbb{L}(y, \hat{y})$ is large.

B. Taskar [TGK04] shows through the standard transformations (similar to those shown in Section 2.1.3) that maximizing the margin in this case corresponds to minimizing the norm of the weight vector $\boldsymbol{w}$ and the following optimization problem should be solved as a result:

$$\begin{aligned} &\min \|\boldsymbol{w}\| \\ &\text{such that } \langle \boldsymbol{w}, \boldsymbol{\Delta\Phi}_i \rangle \geq \mathbb{L}(y, \hat{y}) \\ &\forall \hat{y} \in \mathbf{G}(x_i). \end{aligned}$$

R. McDonald uses the MIRA algorithm introduced in the previous section to solve this problem as maximizing the margin is already the internal property of MIRA and thus the MIRA update rule (2.12) will look like:

$$\begin{aligned} &\mathbf{w}^{(i+1)} = \arg\min_{\mathbf{w}^*} \|\mathbf{w}^* - \mathbf{w}^{(i)}\| \\ &\text{such that } \langle \boldsymbol{w}, \boldsymbol{\Delta\Phi}_i \rangle \geq \mathbb{L}(y, \hat{y}), \text{with respect to } \boldsymbol{w}^* \\ &\forall \hat{y} \in \mathbf{G}_{best_k}(x_i), \end{aligned} \tag{2.16}$$

where $\mathbf{G}_{best_k}(x_i)$ denotes the k parses of x with the best score as constraints derived from all the possible parses of x make the quadratic program intractable. Still even small values of k yield near optimal performance as shown in [McD06].

The loss function chosen is closely related to the Hamming loss often used in sequences and is defined for a graph as the number of words with incorrect incoming edges relative to the correct parse.

The problem in 2.16 is a quadratic problem that is solved in the McDonald's framework with the help of Hildreth's algorithm shown in Section 2.1.2.

Hildreth's algorithm gets the values of $\boldsymbol{\Delta\Phi}_i$ and d_i denoting

$$d_i = \mathbb{L}(y, \hat{y}) - \langle \boldsymbol{w}, \boldsymbol{\Delta\Phi}_i \rangle \tag{2.17}$$

for each of the parses k of a sentence x. The constraint according to which 2.16 should be optimized takes the form $d_i \leq 0 \, \forall i \in [k]$. The sequence $i(\nu)$ is chosen to be dependent on the largest value d_{max} of d_i so that in one iteration the most broken presently constraint is chosen.

Not to depend on the weight vector $\boldsymbol{w}$ during iterations, the update rules from the general Hildreth's algorithm are slightly modified. First, the parameter c_ν (cf. Eq. 2.10) can be computed faster by

$$c_\nu = \min \left(\alpha^\nu_{i(\nu)}, \frac{d_{i(\nu)}}{\|\boldsymbol{\Delta\Phi}^{i(\nu)}\|^2} \right),$$

λ_ν is assumed to be equal 1. Second, because of the inequality sign direction change in the constraint of 2.16, the dual will be updated by

$$\alpha^{\nu+1} = \alpha^\nu - (-c_\nu \boldsymbol{e}^{i(\nu)}) = \alpha^\nu + c_\nu \boldsymbol{e}^{i(\nu)}$$

Besides, the update rule for the weight vector (Eq. 2.8) substituted into the expression for d_i (2.17) yields

$$\begin{aligned}
&\mathbb{L}(y, \hat{y}) - \langle \boldsymbol{w}^{\nu+1}, \boldsymbol{\Delta\Phi}_i \rangle = \\
&\mathbb{L}(y, \hat{y}) - \langle (w^\nu + c_\nu \, \boldsymbol{\Delta\Phi}_{max}), \boldsymbol{\Delta\Phi}_i \rangle = \\
&\mathbb{L}(y, \hat{y}) - \langle \boldsymbol{w}^\nu, \boldsymbol{\Delta\Phi}_i \rangle - c_\nu \, \langle \boldsymbol{\Delta\Phi}_{max}, \boldsymbol{\Delta\Phi}_i \rangle = \\
&d_i^\nu - c_\nu \, \langle \boldsymbol{\Delta\Phi}_{max}, \boldsymbol{\Delta\Phi}_i \rangle, \forall i \in [k],
\end{aligned}$$

so that one has to actually update the expression for d_i during each iteration to

$$d_i^{\nu+1} = d_i^\nu - c_\nu \, \langle \boldsymbol{\Delta\Phi}_{max}, \boldsymbol{\Delta\Phi}_i \rangle, \forall i \in [k] \tag{2.18}$$

Thus, the update rule becomes independent of the value of the weight vector $\boldsymbol{w}$.

The algorithm terminates if one of the d_i gets smaller than, e.g., $\epsilon = 10^{-8}$ or the maximum allowed iteration number (set to 10^4) has been reached.

2.3 Summary

The background knowledge necessary for understanding of discriminative learning in general and one of its online algorithm, MIRA, in particular has been presented in this chapter. This algorithm was chosen for the dependency parsing framework of R. McDonald, because of the three following reasons. It is accurate — it outperforms the standard perceptron algorithm and performs not worse than batch alternatives; it is efficient — with the exception of the perceptron algorithm, MIRA is the fastest discriminative learning algorithm [MCP05]. Besides, its implementation is simple — it only relies on inference and Hildreth's algorithm to solve the quadratic programming problem.

Chapter 3

Dependency Parsing as Maximum Spanning Tree Search

This chapter revises the parsing algorithms used by the MSTParser. [1] The main idea behind all the algorithms is to reduce the parsing problem to finding maximum spanning trees in dependency graphs which enables to find the correct parse through a search for a tree with a maximum score based on the introduced scoring functions. If only scores over single edges are taken into consideration when calculating the score of a dependency tree, the corresponding algorithms are referred to as *first-order parsing* algorithms in contrast to *second-order* algorithms which also incorporate information about adjacent edges in addition to single edges. This chapter also covers the features used in the MSTParser and the approaches taken to add labels to the parsing decision.

3.1 First-Order Parsing Algorithms

According to the definition of R. McDonald [McD06], the *score of an edge* is the dot product $\langle . , . \rangle$ between a high dimensional representation of the edge and a weight vector,

$$s(i,j) = \langle \boldsymbol{w}, \mathbf{f}(i,j) \rangle, \tag{3.1}$$

where $(i,j) \in \boldsymbol{y}$ if there is a dependency in the generic dependency tree $\boldsymbol{y}$ from word x_i to word x_j.

Then the *score of a dependency tree*, if only single edge scores are taken into consideration, is given by,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j) \in \boldsymbol{y}} s(i,j) = \sum_{(i,j) \in \boldsymbol{y}} \langle \boldsymbol{w}, \mathbf{f}(i,j) \rangle.$$

If an appropriate feature representation is chosen and the weight vector $\boldsymbol{w}$ is known then the dependency tree with the highest score will present the dependency parse for a sentence $\boldsymbol{x}$.

[1] For details s. [McD06, MLP06].

For a directed graph $G = (V, E)$ in which each edge $(i, j), v_i, v_j \in V$, has a score $s(i, j)$, the *maximum spanning tree* (MST) is defined as a tree $\boldsymbol{y}$ that maximizes $\sum_{(i,j)\in\boldsymbol{y}} s(i, j)$, such that $(i, j) \in E$ and every vertex in V is used in the construction of $\boldsymbol{y}$.

The properties of the maximum spanning trees can be applied for dependency trees since for each sentence $\boldsymbol{x}$ a directed graph $G_{\boldsymbol{x}} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$ can be defined such that

$$V_{\boldsymbol{x}} = \{x_0 = \text{root}, x_1, \ldots, x_n\},$$
$$E_{\boldsymbol{x}} = \{(i, j) : x_i \neq x_j, x_i \in V_{\boldsymbol{x}}, x_j \in V_{\boldsymbol{x}} - \text{root}\}.$$

Thus, if the score of a tree is factored into the sum of single edge scores as defined previously then finding the (projective) dependency tree of highest score is equivalent to finding the maximum (projective) spanning tree in G_x with an artificial root. This problem is referred to as the *first-order* maximum spanning tree problem, or *first-order* dependency parsing problem. The MSTParser uses separate algorithms for first-order projective and non-projective parsing which will be presented next.

3.1.1 Eisner Projective Parsing Algorithm

Projective parsing algorithms are actually suitable for languages the syntactic structures of which are mostly projective (e.g., English — the largest source of the English dependency trees, the Penn Treebank [MSM93], only includes projective sentences) which is unfortunately not true for the German language having around 14 percent of non-projective sentences. [2] Still, the first-order projective parsing algorithm used by the MSTParser is an important element on the way to understanding the approximate second-order non-projective algorithm most promising for parsing German and so it also deserves its attention here.

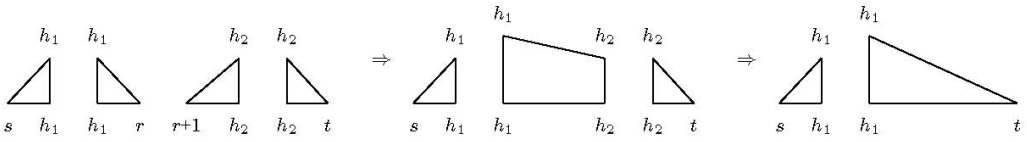


Figure 3.1: Eisner algorithm (first-order); cf [McD06].

Projective parsing algorithms in the MSTParser are based on the algorithm introduced by J. Eisner [Eis96] which was a successful attempt to reduce the complexity of the well-known chart parsing CKY (Cocke, Younger, Kasami) algorithm [You67]. Whereas the chart parsers derived directly from the CKY algorithm, such as [Als96, Col96], need $O(n^5)$ time to create the whole dependency tree forest in practice, Eisner observed that if the left and right dependencies of a word are parsed independently and combined only later then it is possible to parse in $O(n^3)$ whereby no additional head indices of the original algorithm are needed any more and only two additional variables to gather left and right dependencies respectively are required.

[2] See 5.2 for information about different corpora for German

In Figure 3.1 illustrating the Eisner algorithm, r, s, and t represent the start and end indices of chart items, h_1 and h_2 stay for the indices of the heads of chart items. Initially, the items are complete (shown by right angle triangles). The first step creates an incomplete item with h_1 as the head of h_2 and the second checks whether it can be completed. Larger items are created bottom-up from pairs of smaller items as in the normal CKY parsing. Each chart item also stores the score of the best possible subtree it has been created from. [3]

The Eisner dependency parsing algorithm is a bottom-up dynamic programming chart parsing algorithm and, in fact, it provides the solution of the maximum spanning tree problem if a linear ordering of the vertices in the graph, such as the order of words in the sentence, is given [McD06]. k-best extensions of this algorithm that increase complexity by a multiplicative factor of $O(k \log k)$ also exist.

3.1.2 Chu-Liu-Edmonds Non-Projective Parsing Algorithm

For the non-projective case, the entire space of spanning trees has to be searched with no restrictions and the MSTParser applies the algorithm of Chu-Liu-Edmonds [CL65, Edm67] [4] for directed graphs for that purpose. During successive steps, each vertex of the graph greedily selects the incoming edge with highest score. If the formed structure is not a tree, there must be a cycle. The cycle is identified and contracted into a single vertex whereby the weights of the edge going into and out of the cycle are recalculated. As soon as a tree is formed after a succession of such steps, it should be the maximum spanning tree. The algorithm can recursively call itself on the contracted graph as Georgiadis showed [Geo03] that the MST on the contracted graph is equivalent to the MST of the original graph. Besides, Tarjan's efficient implementation of the algorithm [Tar77] counts for only $O(n^2)$ time complexity and the k-best case extensions are possible with $O(kn^2)$.

3.2 Second-Order Parsing Algorithms

Second-order spanning tree parsing methods factor the score of the tree into the sum of adjacent edge pairs and, thus, allow to benefit from the previous parse decisions.

For second-order spanning tree parsing, the score function (Eq. 3.1) has been modified to $s(i, k, j)$, which is the score of creating a pair of adjacent edges, from word x_i to words x_k and x_j, and thus, for a word x_{i_0} having the m modifies $x_{i_1}, \ldots, x_{i_j}, x_{i_{j+1}}, x_{i_m}$, the first

[3] See Appendix C for a complete listing of the Eisner algorithm (Figure C.3).

[4] See Appendix C for a complete listing (Figure C.4) and a detailed example (Figure C.5) of the Chu-Liu-Edmonds algorithm.

j modifiers of which are on its left and the rest $m - j$ on its right, it is formally defined by

$$s(i,k,j) = \sum_{k=1}^{j-1} s(i_0, i_{k+1}, i_k) + s(i_0, -, i_j) + \\ + s(i_0, -, i_{j+1}) + \sum_{k=j+1}^{m-1} s(i_0, i_k, i_{k+1}),$$

where $s(i,-,j)$ is the score when x_j is the first left/right dependent of word x_i. The left/right independence assumption made in this definition is common and allows to define polynomial second-order projective parsing algorithms. This function is reduced to the standard first-order function if this score function ignores the middle modifier, or sibling.

The score of a tree for second-order parsing becomes equal to the sum of adjacent edge scores,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,k,j) \in \boldsymbol{y}} s(i,k,j).$$

The notion of adjacent edges can be applied only to graphs having a fixed order over their vertexes, and it is fortunately the case for the words of a sentence comprising a dependency tree.

The gain introduced by the second-order features is the ability to use most recent parsing decisions and condition on the last dependent found for a particular word. As in the first-order case, algorithms for projective and non-projective case have been presented.

3.2.1 Extension to Eisner Algorithm

The algorithm of Eisner in its original form [Eis96] has the extension for the second-order case which is running in $O(n^3)$. The principle of the algorithm is sketched in Figure 3.2, which shows that h_1 creates a dependency to h_3 with the second-order information that the last dependent of h_1 was h_2. Part (B) of the figure shows how this is done through the creation of a sibling item. [5] The main difference to the first-order case is the delay of the completion of the items until all the dependencies of the head have been found. This allows for the collection of pairs of sibling dependents in a single stage without increasing the cubic time. With this algorithm, the k-best parses can be found in $O(k \log(k) n^3)$.

3.2.2 Approximate Non-Projective Algorithm

R. McDonald has proved that the second-order non-projective MST parsing is $\mathcal{NP}$-hard [MP06], but provided an efficient approximate algorithm for the non-projective case. It is similar

[5] See Appendix C for the listing of the second-order Eisner algorithm (Figure C.6).

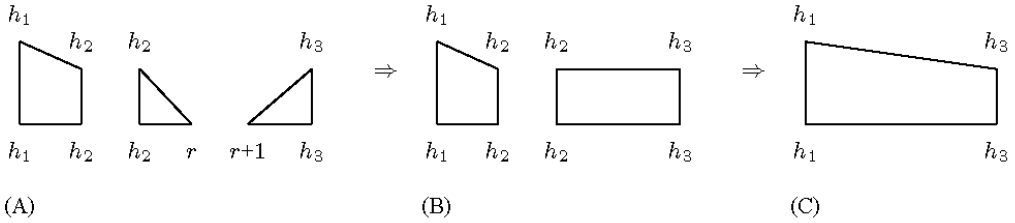


Figure 3.2: Eisner algorithm (second-order); cf [McD06].

to the approach applied in WCDG [FMS00] which to overcome the difficulties in a standard constraint dependency grammar starts with a suboptimal solution and does subsequent attempts to reach the global optimum through local constraint optimizations.

The approximate algorithm is based on the projective parsing algorithm just presented. First, the highest scoring projective parse is found with the second-order projective algorithm, and then the edges in the tree are rearranged, one at a time, as long as such rearrangements increase the overall score and do not violate the tree constraint. [6] The approach is based on the fact that even in non-projective languages like German the majority of the sentences are projective and even in non-projective sentences most edges but a few are projective. [McD06] has analyzed the properties of this algorithm and proved its termination.

3.3 Second-Stage Labeling

As already mentioned in Section 1.2, an important quality of a good parser nowadays is its ability not only to find the correct dependency structure, but also to assign labels to result edges. The algorithms presented in this chapter identify only the modifiers of a word and although it is easy to enrich the edge score functions with labeling information and then incorporate labeling into the parsing algorithms directly, this approach leads to the growth of the complexity of the algorithms. The complexity of the first-order projective algorithm becomes $O(n^3 - |T|n^2)$ and for the non-projective algorithms $O(|T|n^2)$, where T is the set of labels used. For the respective second-order algorithms time requirements are even higher, $O(|T|n^3)$ and $O(|T|^2)n^3$, respectively, the latter with a potential to become a problem for large label sets.

That is why, in correspondence with the idea of Dan Klein who showed that it is much harder to find the correct structure for dependency trees than to assign labels to ready-made trees [Kle04], a second stage that takes the result parse $\boldsymbol{y}$ and classifies each edge $(i, j) \in \boldsymbol{y}$ with a particular label $t \in T$, was added into the MSTParser [MLP06]. In addition to the complexity reduction of the overall system, one gets the possibility to consider all the available information about the fixed output and not restrict oneself to single

[6] See Appendix C for the listing of the second-order non-projective approximate algorithm (Figure C.7).

edges or pairs of edges as suggested by the scoring functions used by the corresponding algorithms.

The idea is to consider an edge $(i, j) \in \boldsymbol{y}$ as input for sentence $\boldsymbol{x}$ and check for the label with the highest score,

$$t = \arg\max_t s(t, (i, j), \boldsymbol{y}, \boldsymbol{x})$$

Even more benefit can be assured if the labels of nearby edges are taken into consideration, especially for a word and its dependents. Specifically, if a word x_i has M dependents $x_{j_1}, \ldots, x_{j_M}$ then labeling of the edges $(i, j_1), \ldots, (i, j_M)$ is treated in the MSTParser as a sequence labeling problem,

$$(t_{(i,j_1)}, \ldots, t_{(i,j_M)}) = \boldsymbol{t} = \arg\max_t s(\bar{t}, i, \boldsymbol{y}, \boldsymbol{x}).$$

The scores of the adjacent edges (i, j_m) and (i, j_{m-1}) in the tree $\boldsymbol{y}$ are thus seen as factors of a first-order Markov factorization

$$\boldsymbol{t} = \arg\max_t \sum_{m=2}^{M} s(t_{(i,j_m)}, t_{(i,j_{m-1})}, i, \boldsymbol{y}, \boldsymbol{x})$$

The score functions are, as in the other algorithms, standard dot products between high dimensional feature representations and a weight vector. The highest scoring label sequence can be found with Viterbi's algorithm. The MIRA online learning algorithm presented in Section 2.1.4 is used to set the needed weights in this case as well.

3.4 Feature Space

For all the algorithms, the choice of the proper feature space seems to be one of the crucial reasons for their actual success. It should be noted that the features in this approach should be chosen in such a way that when a single edge is considered the whole information needed to that point is present and one is independent of the parsing decisions taken outside this edge.

First-order features used in the MSTParser are summarized in Table C.1 (a), (b), (c) and an illustration to building features for one edge of an English sentence is given in Figure C.1 (both in Appendix C). The first two most general feature groups, referred to as uni-gram and bi-gram according to the number of the elements that are considered, take the word identities and their parts of speech (POS) categories into account combining the information about the head or the modifier of the edge or both in all possible variations. The third group of features includes the in-between features, the trigrams built of the POS of the head and the modifier and the POS of the word between them for all the words in between, as well as 4-gram features including the information about the POS of the words before and after the head and modifier pair and thus reflecting the local context information. In the first version of the MSTParser, the example in the Appendix is based

on, for words longer than 5 characters, the same features are added for the 5-gram prefix as for the entire words which is just an ad-hoc measure to extract the lemma from an inflected form, beginning from the CoNLL task the MSTParser uses word lemmas.

A number of specific features for second-order parsing (Table C.1 (d) in Appendix C) add the information about the word as well as the POS of the middle modifier, or sibling, according to the meaning used in Section 3.2. Whereas all the first-order features include the direction of attachment and the distance between the head and the modifier (that are not reflected in the tables of the Appendix), second-order features exist in two variations, the first with and the second without the distance between the two siblings and the direction of attachment.

The feature set of the MSTParser was optimized for the English language. The impact of each of the feature groups was tested with common (in the discriminative learning setting) leave out feature tests, a simple approach to compare the test results obtained after the parsing models have been successively trained without a certain feature group. For the first-order model, the performance is shown to be highly dependent on knowing the attachment direction as well as the distance from the head to the modifier. When removed, the performance drops from 90.7 % to 88.1 %. Even stronger is the dependence on the POS context and in-between features which, when removed together, are responsible for a modest performance value of only 86.0 % which is in fact smaller than when all the edge relevant features about the head and the modifier are removed (87.3). In fact, the POS context and the in-between features can be seen as simulating higher order feature representations.

The second-order model shows other dependencies, the highest performance degradation occurs contrary to the first-order case if one removes all the edge relevant information about the head and the modifier from 91.5 % to 89.5 %. The POS context together with in-between features or even attachment direction and distance seem to be overlapping with other second-order features as their removal has much smaller impact on the performance than in the first-order case (91.3 % accuracy in both cases instead of 91.5 % for the full feature set).

To allow for the labeling stage, appropriate features are added to those already mentioned. Choosing labeling features, there is no need to restrict oneself to the scope of only one edge as labeling is not based on the local factorization (see Section 3.3), the gain which should be paid for with the renouncement of the knowledge of the combination about the label and the structure that could be done if labeling were made together with the structural analysis. Thus, the edge, sibling and context features that are similar to those defined for the dependency algorithms, such as head and modifier POS, attachment direction or words in between and their POS, or add other, e.g., sharing the same suffix, sharing the same POS), are extended by non-local features, such as the number of modifiers for the modifier in question or whether the modifier is the left- or right-most of the head (see Figure C.2 for the complete list of labeling features).

Furthermore, the features have been extended for the case that the training and testing data are enriched with the language specific morphological information to benefit from

it as well. For this purpose, the structural feature representation for each edge between the head x_i and the modifier x_j having morphological features M_i and M_j respectively is extended with M_i as head features and M_j as modifier features and a conjunction of features from both sets. It is a way to easily find such common properties between a head and a modifier like gender, case, or number. Morphological additions are also made for the second-stage labeler, they are labeled with (M) in Figure C.2, in Appendix C.

3.5 Summary

To consider the dependency trees as directed graphs and to view syntactic parsing as a problem of finding maximum spanning trees in directed graphs, is a novel abstraction on the parsing problem successfully applied by R. McDonald. Similar approaches have been described in [Hir01] and [Rib04], but they are known for their complexity and lack of description abstractions. The formalism of maximum spanning trees allows to find efficient solutions for both the projective and non-projective cases. The existing projective algorithm of Eisner is the starting point for both first-order (incorporating features over only one edge) and second-order (adding features about adjacent edges) cubic parsing algorithms. The non-projective algorithm of Chu-Liu-Edmonds using first-order features has an even lower complexity of $O(n^2)$ and although the use of the second-order features makes it intractable, an approximate algorithm for this case is also suggested. Besides, these algorithms are extended with a second stage to add labeling.

Thereby, the role of the discriminative learning framework itself should not be underestimated for the success of the MSTParser. R. McDonald stresses that "the weaknesses of edge-based factorization can be overcome with discriminative learning that exploits rich feature sets describing properties of each dependency as well as their local context" [McD06]. So the weights for statistically dependent features, such as the edge features and part of speech context together with the in-between features can be efficiently learned. The success of the methods is directly connected with online learning.

In the CoNLL-X 2006 shared task [BMDK06], the MSTParser achieved the best results compared to other present-day parsers. Its structural accuracy ranged from 79.3% for Arabic to 91.5% for English. The reported results for German in the CoNLL-X task are 90.4% and 87.3% structural and labeled accuracy respectively. The German model was trained on 39,216 sentences from the the TIGER treebank [BDH$^+$02] previously automatically translated into the dependency structures, and 357 sentences of it were used for testing.

Chapter 4

WCDG System

There are many paths to the top of the mountain, but only one view.

Harry Millner

This chapter shortly presents the WCDG system. [1] The details about it can be obtained from [Fot06, FHS+05, Sch02] reflecting the milestones of its development as well from more recent publications [FM06b, FM06c, FBM06] this chapter is mostly based on. The most important practical aspects are summed up in [FHS+05].

4.1 Weighted Constraint Dependency Grammar

The formalism of a *Constraint Dependency Grammar*, CDG, was first introduced by Maruyama [Mar90] and suggests modeling natural language with the help of constraints. Later [Sch02], it was extended to a *Weighted* CDG, on the one hand, following the awareness of the *graded* nature of grammaticality [Abn96, Kel00], and on the other hand, with the motivation to use the weights of the constraints during parsing to disambiguate between alternatives. A WCDG models natural language as labeled dependency trees and is entirely declarative, i.e., it has no derivation rules. How well-formed trees may be built is expressed with the help of constraints.

Every constraint of WCDG carries a *weight*, also referred to as a *penalty*, in the interval from zero to one, a lower value of the weight reflects its greater importance. If a parse analysis found by the system, or, more precisely a set of dependency edges, violates any of the constraints, it is registered as a *constraint violation* or a *conflict* between the structure and the rules of the language. The *score* of an analysis is the product of all the constraints that are violated by it. In the described setting, it becomes possible to differentiate between the quality of different parse results: the analysis with a higher

[1] The WCDG system is freely available for download at `http://nats-www.informatik.uni-hamburg.de/view/Main/NatsDownloads/`.

score is considered preferable. Surely, under these conditions, an analysis having only a few grave conflicts may unfortunately be preferred by the system against another one with a great number of smaller constraint violations. But an analysis violating any of the constraints with a weight of zero would get the lowest value. These constraints are referred to as *hard*, reflecting their prohibitive nature. The constraints having a weight greater zero, also called *defeasible*, may express universal principles or vague preferences for language phenomena.

In the following examples that are expressed in the WCDG constraint language, the first prohibits adverbs to follow attributive expressions directly and the second helps to recognize incorrectly assigned **SUBJC** labels if an adverb modifies the conjunction.

```
{X/SYN/Y\SYN} : 'adverb after attribute' : order : 0.0 :
  X.label = ATTR -> Y.label != ADV;

{X/SYN\Y/SYN} : 'SUBJC should be NEB' : category : 0.0 :
  X@word = so & Y.label = KONJ
  ->
  ~is(Y^id,SUBJC);
```

In this more or less intuitive notation, **X** and **Y** are the edges, **SYN** in the signature stays for the syntactic level of analysis as extrasyntactical analysis is also conceptually possible within the WCDG (at the moment, there is also the **REF** level that is used to connect relative pronouns with their antecedents) and **X**@ and **Y**^ are used to express the modifier and the regent respectively. [2] The sign in the signature between the edge and the analysis level is a direction indicator for this edge. Thus, `{X/SYN}` and `{X\SYN}` refer to an edge **X** pointing to the right or left, `{X|SYN}` means that **X** points to the root which is opposed to `{X!SYN}` meaning that **X** does not point to the root while `{X:SYN}` does not impose any restriction on the edge under analysis. The constraints in the examples have referred to two edges so the information about their relative position is also included into the signature. Thus, `{X:SYN/Y:SYN}` means that **X** is above **Y** and `{X:SYN\Y:SYN}` is the opposite.

The following defeasible constraint refers to adjacent and connected genitive constructions and would in most situations disprefer a second genitive that is on the left of its regent.

```
{X/SYN\Y!SYN} : 'nested genitive' : category : 0.1 :
  X.label = GMOD
  ->
  Y.label != GMOD;
```

[2] Section 1.4 of [FHS+05] describes the constraint syntax in detail.

The concrete weights for a constraint are chosen by the grammar writer. Attempts have been made to compute the weights of a WCDG automatically by observing which weight vectors perform best on a given corpus, but the computations did not bring improvements to the manually assigned scores [SPMF01]. Much more important are the relative values of the weights, so that more often constructions are preferred, but seldom variations are also allowed.

At the moment, there are around 1000 constraints defined to cover modern German language completely [FDM05] that are robust against many kinds of language errors. The last fact accounts for a very great number of defeasible constraints to allow an error principally to happen, but, as a result, the search space of possible trees gets even larger than for a prescriptive grammar.

Parsing problem is being treated in the WCDG system as a Constraint Satisfaction Problem which is "the problem of assigning values to a finite set of variables while obeying a set of conditions on combined assignments" [Fot06]. The expressiveness of the constraint language comes at the expense of complexity, the parsing problem becomes $\mathcal{NP}$-complete. Still, a reliable heuristic alternative to a complete search has been suggested: the *transformation-based solution methods* that make a guess to get the initial variant of the optimal tree and the constraint violations are used as a control mechanism leading to the change of labels, subordinations, or choice of the lexical variants. The basic algorithm for heuristic transformational search is shown in Figure D.1 of Appendix D. It has been described in great detail in [FMS00] and [Fot06].

The transformation-based search cannot guarantee to find the best solution to the constraint satisfaction problem, but in comparison to a complete search that requires much time and space, it only needs limited resources and can be interrupted at any time. Besides, a complete search will not always return a solution as it will probably not terminate at all. The transformation-based algorithm, even interrupted, will always return an analysis, together with a list of the violated constraints that it probably has not fixed yet. The algorithm terminates on its own before the timeout when no violated constraints with a weight over a predefined threshold remain.

4.2 Statistical Enhancements

Realistic broad-coverage parsing should operate on raw data. In this respect, several limitations of WCDG can be observed. If the input is annotated with part of speech (POS) information, the parser reaches 90.4% structural and 88.8% labeled accuracy. But if this information is not available, the accuracy drops down to $72.6\%/68.3\%$ respectively.

There exist weaknesses in the constraint formalism itself, that cannot be overcome only by writing more constraints, e.g., having lexicalized constraints is possible, but practically infeasible. Besides, the success of the parse entirely depends on a human grammar writer and there are language phenomena that can be easily overlooked.

Moreover, the solutions should always be searched by WCDG from scratch, the parser does not learn from previous errors or successful decisions.

These are the main problems statistical methods should help to overcome. At the present moment, WCDG successfully integrates five additional statistical components: POS tagger, chunker, supertagger, PP attacher and a shift-reduce oracle parser.

POS Tagging

As already mentioned, possessing reliable POS information about the input is one of the essential conditions for the parsing success. Purely stochastic methods have achieved very good results as contextual information can be very effectively used to disambiguate the categories of the words in the utterance. The TnT tagger [Bra00] was added to WCDG [HF02] and after some improvements were applied to the tagger [Fot06] it has brought the parser onto almost the same accuracy level it had with the knowledge of the tags from the gold standard ($89.7\%/87.9\%$). One of the important aspects of the integration of the tagger is that it submits its decisions to the parser in form of graded predictions of the POS categories of which the parser may choose basing its choice on the value of the prediction which is incorporated into the weight of the tagger constraint for this purpose. Thus, the parser is not forced to accept the decision of the tagger and may overwrite it if enough evidence is present against it.

Chunk Parsing

The chunker is responsible for predicting the boundaries of the main constituents in a sentence and reduces the number of search alternatives during parsing. According to the WCDG constraint, all chunk heads should be attached outside the chunk itself. For chunking, the tree model TreeTagger [Sch94] was used after it has been trained on articles from *Stuttgarter Zeitung* and achieved a precision of 88.0% and a recall of 89.5%.

Supertagging

Supertagging is another level of category disambiguation that takes not only the syntactic category into consideration, but also the local syntactic environment of each word. In an approach, most similar to that of [WH02], the predictions used in WCDG were the label of each word, whether it follows or precedes its regent, and what other types of relations are found below it and are obtained after a re-train of the TnT. As in other cases, a constraint with a preference to a predicted value of the supertag was added to the constraint grammar. If one measures the average accuracy of the supertag predictions according to the individual predictions for distinct properties, it reaches around 84.5% [FBM06].

PP Attachment

The attachment of prepositional phrases depends to a very high extent on the semantics of the words in their vicinity and presents a big problem for syntactic parsing as it should be based on the lexicalized information that cannot be incorporated into the constraints in the needed amount. To disambiguate PP-attachment, a probabilistic model that counts only occurrences of prepositions and potential attachment words and ignores the identity of the words that belong to the preposition, has been trained on both the treebanks and raw text and achieved an accuracy of 79.4% and 78.3% respectively [FM06a]. All the solutions that do not correspond to the prediction of the PP-attacher are penalized by another WCDG constraint.

Shift-Reduce Oracle Parser

Most relevant for the present work are the previous experiments to integrate a full parser as a predictor of the attachment for all words. The probabilistic shift-reduce parser constructed for this purpose according to the model described in [Niv03] sees parsing as a series of simple actions — shift, reduce and attach — that are used to construct dependency trees. A maximum likelihood model that takes such properties of the current state into account as the categories of the current and following words, the environment of the top stack constructed so far, and the distance between the top word and the next word, predicts parse actions with an accuracy of $84.8\%/80.5\%$ [FM06b]. But it can only predict projective trees, the limitation that was tolerated during the first experiments with a full oracle predictor for simplicity and that should be resolved by this work.

4.3 Summary

Although Weighted Constraint Dependency Grammar has limitations typical for a rule-based system, such as the development and parsing effort, the formalism turned out to be flexible enough to incorporate other sources of knowledge with the help of the same constraints that describe grammar rules serving as parsing guidelines in general. What is even more important, it allows the parser not only to avoid error propagation successfully although the predictor components show an accuracy that is mostly — with the exception of the tagger — below that of the parser itself, but also improve its own results in the range of very slight improvements to tens of percent (Table 4.1). Moreover, in this successful realization of hybrid parsing methods, multiple components may be used according to the same principles. Table 4.1 shows that the accuracy improves also in parsing experiments in which multiple components interact.

The WCDG system benefits from additional sources of knowledge both in those cases where it does not possess the information that comes from the predictors itself, such as it is almost entirely unreliable in determining PP-attachments or POS-tags, and in situations where it is able to find a similar solution, but only after probably inefficient heuristic

	Accuracy, %	
Experiment	structural	labeled
none	72.6	68.3
POS only	89.7	87.9
POS+CP	90.2	88.4
POS+PP	90.9	89.1
POS+ST	92.1	90.7
POS+SR	91.4	90.0
POS+PP+SR	91.6	90.2
POS+ST+SR	92.3	90.9
POS+ST+PP	92.1	90.7
all five	92.5	91.1

Table 4.1: WCDG parsing accuracy with various predictor components (from [FM06b]). POS — part of speech tagger, CP — chunker, ST — supertagger, PP — prepositional attacher, SR — shift-reduce oracle parser.

search, so that a global optimum can in the end be found much faster when the predictors are used than without them.

This work goes on investigating the benefits that combinations of different sources of knowledge may bring for syntactic parsing. In the following chapters, the prospects of adding another oracle parser to WCDG will be evaluated. In comparison to the presently integrated shift-reduce parser, the MSTParser has shown a much higher accuracy in external experiments and uses a parsing algorithm that is able to produce non-projective trees. Before integrating the MSTParser into the WCDG system as an oracle predictor, it is sensible to evaluate it on the WCDG compliant data which will be done next, in Chapter 5.

Chapter 5

Parsing German with the MSTParser

When all else fails, read the manual.
Anonymous

In this chapter, the results of the evaluation of the MSTParser on the German data will be presented. The experiments were aimed at comparing the performance of the MSTParser and WCDG and showing the perspectives of integrating the former as an oracle parser for WCDG.

Two goals were pursued during the performance measurements. On the one hand, backward comparison with the previous WCDG results should be achieved. On the other hand, the comparison with the largest available study of the performance of the present parsers, the CoNLL-X shared task, should also be made possible. With respect to these two goals, measurements were done both according to the CoNLL standard where the data is provided with the correct POS tags and morphological features as well as under more realistic conditions with a real and not a gold POS tagger, a challenge broad coverage parsers should be able to cope with. When not said otherwise, results will be given including the punctuation into the overall score as done in WCDG as well as MSTParser internal tradition. Still, the more important results will be doubled excluding the punctuation to allow for a comparison with the CoNLL-X task results which also ignore it. The performance of the parser will be evaluated according to the accuracy measures presented in Section 1.2 [1].

5.1 Parsing Experiments

The experiments used the MSTParser version 0.4.3b from April 4, 2007, freely downloadable from `sourceforge.net/projects/mstparser/`, and were performed on a 32-bit 4-processor Athlon with GNU/Linux which allowed to use 2000 MB for the Java

[1] The MSTParser did fail on extremely long sentences due to out of memory errors so that the three measures would differ slightly sometimes, but in the absolute majority of cases a parse was returned so these measures can be simplified and the exceptions will be pointed out specially.

heap (JDK1.5.0_10). As input, the MSTParser needs data that has already been tagged with POS information. Optionally, the data may also include base forms of the words and morphological information. Certainly, the training data should also be annotated with the correct regent and label.

Experiment Settings

The evaluations were performed on a thousand sentences ($18,602$ – $19,601$) from the NEGRA treebank. It is the same data set that was previously used in the performance evaluations of WCDG (e.g., [FM06b] and [Fot06]) and, thus, it provides a good opportunity for the comparison of the results. The NEGRA treebank is a collection of newspaper articles; in the original, it stores phrase structure annotations. These were available in the form after being translated into dependency trees with the DEPSY tool [DFM04] and manually improved afterward. The manual corrections were necessary to bring the dependency trees in accord with the internal annotation guidelines of WCDG. Most inconsistencies were removed in the cases where the annotation guidelines of the WCDG differ from those of NEGRA, such as in their treatment of non-projectivity (WCDG only allows non-projectivity in the attachment of verbal arguments, relative clauses and coordinations, i.e., the cases where it would decrease ambiguity) and in situations in which the annotations of NEGRA itself turned out to be inconsistent (e.g., in connection with the co-ordinated and elliptical structures, adverbs and subclauses). In addition, the existing version of DEPSY still uses 33 labels, while WCDG already uses 36, although two of them are, in fact, rather rare (**`ETH`** and **`NP2`**). The third, **`OBJP`**, occurs more often, it was introduced to set apart the prepositional phrases that are required by the valency of the verb.

The choice of the data set on which to train the parser was not so straightforward. The performance of a data-driven parser depends strongly on the similarity of the training set to the data used for the test. As parts of the same corpus usually have a greater similarity to each other than parts from different corpora, it would be a good decision to train the MSTParser on the NEGRA corpus as well. The problem is that although the whole treebank has been translated into the dependency trees using DEPSY, only another three thousand of them have been manually corrected, a quantity that seems too small for effective training. Moreover, previous evaluations of the MSTParser have used much larger training sets. E.g., during the CoNLL-X shared task 39,216 sentences from the TIGER Treebank [BDH$^+$02] were used.

Another candidate to become a training set is the `heiseticker` corpus. It contains the texts from the online archive of `www.heise.de`, $70,000$ of which have been manually annotated at NATS according to the WCDG guidelines. The texts in this corpus are all from roughly the same domain, and although very many technical terms and proper nouns are used, the sentences have only a slightly longer mean length which is an advantage for the training.

A cross-corpus experiment between NEGRA and `heiseticker` was performed first. As the NEGRA training set all the available NEGRA sentences excluding the test set were taken (sentences 1 – $18,601$ and $19,602$ – $20,602$) independent of the fact whether they were manually improved or not. Twenty thousand sentences from the `heiseticker` corpus, $10,000$ – $29,000$, were

used as the other training set. For the `heiseticker` test set, sentences $6,001 - 7,000$ were arbitrary chosen.

	Training data		Test data	
	NEGRA	`heiseticker`	NEGRA	`heiseticker`
Sents.	19,601	20,000	1,000	1,000
Tokens	337,425	369,688	16,687	18,050
Mean length	17.2	18.5	16.7	18.1
Longest s.	117	79	66	68
% PUNC	14.5	11.7	14	13
% LEMMA	0.05	47.1	36.3	49.7
% H. = 0	0.9	1.0	1.08	1.0
% H. left	45.2	45.5	43.6	46.7
% H. right	47.0	48.1	48.9	46.7
% N.P. sents.	18.2	16.8	14.5	16.2
% N.P. edges	0.03	0.02	0.02	0.02
Lex. units	49,703	43,833	5,154	5,191
% new lex. (n)	-	-	29.8	38.2
% new lex. (h)	-	-	57.6	30.0

Table 5.1: Characteristics of the training and test data sets.

The properties of the data sets, similar to those provided during the CoNLL-X task [BMDK06] are summarized in Table 5.1 which presents the following characteristics,

- **Sents.:** The number of sentences in the data set.
- **Tokens:** The number of tokens in the data set.
- **Mean length:** Mean length of a sentence.
- **Longest s.:** The longest sentence in the data set.
- % **PUNC:** Percentage of punctuation tokens.
- % **LEMMA:** Percentage of non-punctuation data annotated with lemmas (base forms).
- % **H. = 0:** Percentage of words per sentence whose head is the root of the graph (excluding the punctuation).
- % **H. left:** Percentage of words whose head is on its left (excluding tokens having the root as their head).
- % **H. right:** Percentage of words whose head is on its right (excluding tokens having the root as their head).
- % **N.P. sents.:** Percentage of sentences with at least one non-projective edge.
- % **N.P. edges:** Percent of all non-projective edges.
- **Lex. units:** Number of different lexical units in the data set.
- % **new lex. (n)** Percentage of new lexical units in the test set (not present in the NEGRA training set).
- % **new lex. (h)** Percentage of new lexical units in the test set (not present in the `heiseticker` training set).

Then four experiments were made, all with the second-order non-projective algorithm: one parsing model was trained on NEGRA and the other on `heiseticker` and each was tested on both test sets. Although the data is annotated with morphological features, they could not be included as they caused the MSTParser to stop because of out of memory errors, [2] still the lemmas were preserved. The results summarized in Table 5.2 show that both models achieve the same structural accuracy of 91.9% on the NEGRA test set, the labeled accuracy of the `heiseticker` model is even slightly better (89.3% vs. 89.1%) whereas the accuracy of the `heiseticker` model on the own test set is over two percent higher than of the NEGRA parsing model on it (93.8 % vs. 91.4 % structural and 89.3 % vs. 88.4 % labeled accuracy). This asymmetric behavior can be explained, on the one hand, by the already mentioned annotation inconsistencies inside the part of the NEGRA treebank available for training, on the other hand, by the lack of lemmas in the annotations of NEGRA and the smaller number of words in general in it. Besides, the NEGRA model encounters almost twice so many new lexical units on the `heiseticker` test set. These factors also explain the difference in the number of features between the two models: the training on `heiseticker` creates over 1.5 milion features more, another quality making the model better for generalizations.

	Accuracy, %			
Experiment	structural	labeled	Features	Time
Training on NEGRA:			$9,121,567$	8 h 11 min 6 s
Test on NEGRA	91.9	89.1		7 min 37 s
Test on `heiseticker`	91.4	88.4		7 min 36 s
Training on `heiseticker`:			$10,785,884$	8 h 16 min 19 s
Test on NEGRA	91.9	89.3		9 min 4 s
Test on `heiseticker`	93.8	91.7		9 min 19 s

Table 5.2: The influence of the training set (`heiseticker` vs. NEGRA). With the approximate algorithm.

Leaving out the lemmas from the annotations dropped the performance of the `heiseticker` model on the NEGRA testset by ca. 3 percent down to 88.3%/86.2% and to 90.1% /88.3% on the `heiseticker` test set. Although this shows a strong dependency of the MSTParser to the presence of lemmas in the data, they were left in the annotations for the other experiments for if the MSTParser should be integrated into WCDG as a predictor it would be quite easy to automatically add the lemmas from the lexicon to the data sent to the MSTParser for pre-processing. Surely, this automatism would introduce some errors into the lemma information which should be noticeable in the performance, but, on the other hand, all the data, and not some percent of it, as presently, would be annotated, and this would also bring some improvements.

Another test was done to see how strong the dependency on the annotation correction really is. For this purpose, the MSTParser was trained on the first 3 thousand sentences of NEGRA that have been manually annotated as well as on 4 disjunct sets of sentences, of 3 thousand items each, that have been returned by DEPSY. All five parsing models produced this way were compared on the standard NEGRA test set (manually reannotated) and alternatively on another thousand of the sentences transformed by DEPSY. Table 5.3 shows a comparison of the results on the manually annotated and automatically extracted dependency sets. Although the training on the manually

[2] In fact, the choice of only twenty thousand sentences to train on is also motivated by the memory issues as this was the limit manageable by the JVM on the used architecture.

annotated set (m) has the best results on the manually annotated test set, it cannot outperform any of the other models built as a result of training with sentences automatically annotated by DEPSY ($d1$ to $d4$). Still, the dependency on what kind of data is used for training is really not strong and the difference is never greater than 1.5 percent for structural accuracy and 2 percent for labeled accuracy what allows to conclude that the manual corrections do not change the main principles according to which the dependency trees have been built automatically.

Train set	Features	Labels	Accuracy on reannotated			Accuracy on transformed		
			structural	labeled	(%)	structural	labeled	(%)
m	$2,744,966$	36	89.7	87.0		89.1	86.0	
$d1$	$1,979,592$	33	88.7	85.5		90.0	87.3	
$d2$	$1,932,997$	33	88.1	85.0		89.3	86.6	
$d3$	$2,085,645$	33	88.7	85.6		89.4	86.5	
$d4$	$1,904,947$	33	88.2	85.1		89.3	86.3	

Table 5.3: Comparison of the results on the automatically transformed and reannotated data. With the approximate algorithm.

All said has disqualified NEGRA as a training set, but it also proved that `heiseticker` is a decent alternative. It should also be pointed out that 91.9% structural accuracy is a really good state-of-the-art result, but this is still below the performance that WCDG achieves. On the other hand, it is 1.5% higher than what WCDG could reach with the POS annotations from the gold standard and no statistical enhancements ([Fot06] reported 90.4% structural and 88.8% labeled accuracy on the same test set). More details about the performance of the `heiseticker` model can be extracted from Table 5.4 that together with presenting a typical dependency of the parsing accuracy on the sentence length, provides an alternative measurement excluding the punctuation. It turns out that the accuracy reached on the NEGRA corpus ($90.5\%/87.5\%$) is comparable to that reported by the authors of the MSTParser for the TIGER data during the CoNLL task (90.4%/87.3%).

		With punctuation, %		Without punctuation, %	
Sentence length	Instances	structural	labeled	structural	labeled
1 – 10	340	94.3	91.1	93.2	89.4
11 – 20	323	92.8	90.2	91.7	88.7
21 – 30	229	92.0	89.5	90.7	87.8
31 – 40	76	89.8	87.3	88.1	85.2
≥ 40	32	89.1	86.5	87.2	84.3
overall	1000	91.9	89.3	90.5	87.5

Table 5.4: Dependency on the sentence length (training on `heiseticker` and testing on NEGRA with POS annotations from the gold standard with the approximate algorithm).

Tests on Other Corpora

An important quality of a broad-coverage parser is its ability to generalize well to other domains, it is also an essential property of a parser-predictor. The MSTParser was used to parse the following collections of the dependency trees:

- **Grundgesetz**: The German federal constitution (Revision 2003).
- **Genesis**: The German translation of Genesis from 1960.
- **wyvern**: A part of a contemporary fantasy novel.
- **EU**: Constitutional Treaty proposal for Europe in German (2005).
- **azure**: A German translation of a fantasy roleplay.

All corpora were parsed completely, the only reduction was necessary for the EU corpus, because of out of memory errors on extremely long sentences. Sentences EU-s2 (433 words), EU-s767 (292 words), EU-s1644 (238 words) and EU-s2525 (255 words) were removed from the test set. Thus, the longest sentence that could be parsed with the MSTParser with the available hardware was 217 words long (EU-s1057).

	Grundgesetz	Genesis	wyvern	EU	azure
Sents.	1,154	2,709	9,547	2,564	10,706
Tokens	21,256	43,127	131,886	61,370	151,123
Mean length	18.4	15.9	13.8	23.9	14.1
Longest s.	150	115	103	217	74
% PUNC	10.0	14.6	19.5	8.8	18.4
% LEMMA	47.8	51.7	31.0	58.8	30.0
% H. = 0	0.9	0.9	1.0	0.8	1.0
% H. left	46.2	52.2	44.3	48.1	43.1
% H. right	47.1	39.6	46.3	46.5	47.8
% N.P. sents.	12.0	15.4	10.0	19.1	11.3
% N.P. edges	0.02	0.02	0.02	0.03	0.02
Lex. units	3,332	3,793	10,727	5,461	13,546
% new lex. (n)	35.1	48.6	50.6	43.9	54.9
% new lex. (h)	42.6	59.1	63.2	46.5	65.9

Table 5.5: Characteristics of the corpora used in the tests.

The details about the properties of these corpora can be seen in Table 5.5. The following Table (5.6) summarizes the results (including the punctuation) achieved with `heiseticker` as the training set and using the gold POS tags during the test. [3]

The results show that the same parsing model can be successfully applied to parsing data of various complexity from quite different domains. While the model has been trained on online article

[3] Alternative results were achieved with NEGRA as the training set. In addition, alternative measurements excluding the punctuation as well as measurements of accuracy dependent on sentence length are presented in Tables E.1, E.2, E.3 and E.4 in Appendix E.

			Accuracy, %	
Corpus	Length	Instances	structural	labeled
Grundgesetz	18.4	1,154	91.1	88.2
Genesis	15.9	2,709	91.7	86.7
wyvern	13.8	9,547	93.0	88.7
EU	23.9	2,564	92.0	89.7
azure	14.1	10,706	93.3	89.0

Table 5.6: Testing the `heiseticker` model on other corpora (including punctuation).

material it achieves good results on both simple language of the fantasy novel (93.3%/89.0%) and on the complex structured constitution (91.1%/8.2%). It turns out that the mean sentence length is not the main factor influencing the performance. For example, the parser performs almost three percent better on the sentences from the **EU** collection having over 40 words per sentences on average than on the sentences of the comparable length from the **Grundgesetz** collection. Known lexical information seems to play a more important role for the success of the parser. Thus, in the previous example, 58.8% of the **EU** corpus are annotated with lemmas vs. only 47.8% of **Grundgesetz** that have lemmas while other data properties are comparable. Another reason may also be the greater part of flat structures in **Grundgesetz** in comparison to **EU**.

Parameter Variation

Although the available hardware did not allow to test the full version of the MSTParser that includes the morphological features, this configuration could be evaluated on a training set of three thousand sentences long which was used as another chance to compare the performance of training on NEGRA (first three thousand manually corrected sentences) and on `heiseticker` (sentences $10,000-12,999$). Table 5.7 shows that the accuracy increased by an approximately half a percent for the NEGRA model and quite surprisingly got slightly worse for the `heiseticker` model in comparison to the case when the morphological features are not included. Unfortunately, the reasons for this decline are not obvious. In each of the chosen training sets, around 100 morphological feature values come across, they differ in 30 positions, but around the same number of differences exist with each of the test sets. Thus, to the question which of the morphological features were really responsible for the performance degradation no fast answer can be given. One can be said for sure, the morphological features are not repeated often in all necessary combinations so that the parsing model would become flexible enough to cope with different input. In any case, the usage of morphological features is not realistic for broad coverage parsing at the moment for it has to be investigated first as to how the parser has to adjust its behavior to deal with errors introduced by the morphological tagger after the POS tags from the gold standard are substituted by a real automatic mechanism — and it should be noted than no reliable means exist so far to predict the morphological information.

Another important conclusion that can be made of the previous experiment with morphological features is that the `heiseticker` test set is easier to parse since the results of both parsing models are much better on the `heiseticker` test set than on the NEGRA test set. Besides, this comparison shows that the result of the parse depends on what corpus has been used for training. Neither of the two can outperform the model that is tested on the same corpus it was trained on. Thus, one can claim that the MSTParser would achieve a better result (that would influence all the

	Without morphology			With morphology		
Experiment	structural	labeled	(%)	structural	labeled	(%)
(a) Training on 3 thousand sentences of NEGRA:						
Test on NEGRA	89.7	87.0		90.1	87.3	
Test on `heiseticker`	90.5	87.5		91.0	88.0	
(a) Training on 3 thousand sentences of `heiseticker`:						
Test on NEGRA	89.5	86.6		89.4	86.5	
Test on `heiseticker`	91.4	88.6		91.2	88.3	

Table 5.7: Influence of the inclusion of the morphological features. With the approximate algorithm.

results presented in this work) if, by a happy chance, the manual annotations were consequently provided for the complete NEGRA treebank and one could use it to train the MSTParser.

The benefit of the k-best extensions of the MIRA learning algorithm can only be evaluated for the non-projective algorithm of the first order (because of the approximation of the second-order non-projective algorithm). Here, the accuracy has slightly improved for $k = 10$ in comparison to the results obtained for $k = 1$ used otherwise by default, the structural by 0.1% from 91.1% to 91.2%, and the labeled by 0.2% from 88.5% to 88.7% at the expense of additional six and a half hours of training. This result corresponds to the previously published data [McD06] and is in any case worse than that of the approximate non-projective algorithm of the second-order without k-best extensions.

Until now all the experiments had access to the gold standard part of speech annotation. But, unfortunately no real POS tagger produces ideal output, thus, an important feature for a good parser is to be robust to tagging errors.

	With gold tagging			With real tagging		
Experiment	structural	labeled	(%)	structural	labeled	(%)
Training on `heiseticker`	91.9	89.3		91.0	88.0	
Training on NEGRA	91.9	89.1		91.0	87.8	

Table 5.8: Influence of an automatic tagger (training on `heiseticker`, test on NEGRA with the approximate algorithm).

Table 5.8 draws the comparison between the results of parsing with an ideal and a real POS tagger, in the latter role the POS tagger of WCDG was used (see Section 4.2). As the used version of the MSTParser does not allow to submit POS tag variants to the parser, only the tag variant with the highest score returned by the POS tagger was provided. The results suggest that the MSTParser is much more sensitive to the absence of lemmas in the test data than to the errors in the POS tags. The structural accuracy has dropped by less than one percent to 91.0%, for the labeled accuracy the use of the POS tagger had a slightly greater effect: it is reduced by 1.3%. It is interesting that the accuracy changes by exactly the same values for both the `heiseticker` and the NEGRA model.

5.2 Error Analysis

This section primarily concentrates on the parse results on the same NEGRA test set obtained after the parser has been trained on `heiseticker` with the approximate non-projective algorithm. To eliminate the dependency of the following error discussion on the failures of the POS tagger, the results of the MSTParser are analyzed based on the results obtained using POS tags from the gold standard.

	Accuracy, %			
Algorithm	structural	labeled	Features	Time (to train and test)
Projective:				
first-order	83.0	80.7	10,381,139	4 h 16 min 59 s
second-order	82.9	80.7	10,785,884	7 h 56 min 5 s
Non-Projective:				
first-order	91.1	88.5	10,381,139	4 h 24 min 35 s
second-order	91.9	89.3	10,785,884	8 h 20 min 12 s

Table 5.9: Performance of the MSTParser algorithms (training: `heiseticker`, test: NEGRA).

Non-Projectivity

The reduction of the parsing accuracy in connection with the sentence length has already been pointed out in the previous section. Differences in the performance of the different MSTParser algorithms are presented in Table 5.9, thereby projective algorithms reach a 10 percent lower accuracy than non-projective on the German data. When the accuracy on non-projective sentences is compared to that on projective sentences, for projective algorithms the difference between the two reaches around 5 percent (the accuracy of the projective first-order algorithm on the non-projective sentences equals 78.8%/77.0%, second-order — 78.5%/76.7%; and on the projective sentences they reach both 84.1%/81.8%.

For the non-projective algorithms, this difference is reduced to three and a half percent. The accuracy of the non-projective approximate algorithm on the non-projective vs. projective sentences including the dependency on the sentence length is summed up in Table 5.10. It should be mentioned that the overall values in this table actually cannot be directly compared as the non-projective sentences tend to be longer, in the test set their mean length is 25.0 vs. 15.3 for projective sentences. It is notable that the accuracy for the non-projective sentences (Table 5.10) quite untypically increases with the length of the sentence (excluding extremely long sentences with more than 40 tokens). This may be ascribed to the fact that in the training set the mean length of the non-projective sentences was 24.5 (vs. 17.3 for the projective) and the scoring function does not generalize well for shorter sentences, e.g., as they can have a smaller number of dependents than longer sentences.

Table 5.11 suggests that the non-projective algorithms generally tend to find much more non-projective edges than the data has, while the precision remains restricted. The number of non-projective edges is in this case incremented by one if some edge crosses some other edge once. Thus, Figure 5.1 shows three non-projective edges and Figure 1.3 (Chapter 1) has one. It is surely a rather strict measure. Under the non-projective edge precision in Table 5.11, the absolute number

		Non-projective, %			Projective, %	
Sentence length	Instances	structural	labeled	Instances	structural	labeled
1 – 10	8	85.2	83.6	332	94.6	91.4
11 – 20	44	88.6	85.6	279	93.5	90.9
21 – 30	57	89.4	86.8	172	92.8	90.4
31 – 40	22	89.8	87.5	54	89.7	87.2
≥ 40	14	88.3	86.4	18	89.6	87.0
overall	145	89.1	86.6	855	92.6	90.0

Table 5.10: Accuracy on non-projective vs. projective sentences of the test set.

	Non-projective edges				Non-projective sentences			
Algorithm	recall		precision		recall		precision	
non-projective second-order	346	(29%)	102	(39%)	90	(84%)	76	(52%)
non-projective first-order	393	(23%)	90	(34%)	78	(85%)	66	(46%)
projective second-order	0		0		0		0	
projective first-order	0		0		0		0	
Gold	261 (100%)				145 (100%)			

Table 5.11: Precision and recall of the non-projective edges and sentences for different MSTParser parsing algorithms.

of correct edge-crossings is given as well as the ratio of these to the corresponding value in the gold standard. Under non-projective edge recall, all the non-projective edges that were found by the MSTParser algorithms are included as well as the ratio of the correctly found non-projective edges to all non-projective edges found.

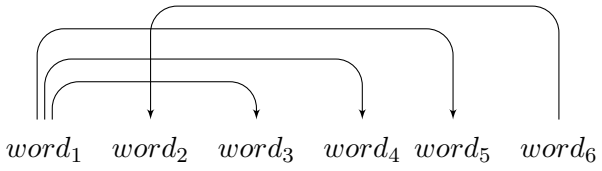


Figure 5.1: An example of three non-projective edges.

A more relaxed measure presented in the same table is the precision and recall of non-projective sentences. If at least one edge-crossing is correctly identified in a non-projective sentence, it has been added to the correctly identified non-projective sentences shown under non-projective sentences precision. When the identified edge-crossing is actually not present in the sentence, but the sentence still has some other non-projective edges, the non-projective sentence recall is incremented by one. Under these conditions, the non-projective second-order algorithm correctly identifies slightly more than a half of the non-projective sentences and over a third of non-projective edges. In fact, WCDG under harder conditions — with a real POS tagger instead of the POS

tags from the gold standard — finds 91 non-projective sentences and 138 edges correctly which corresponds the non-projective sentence precision of 63% and the non-projective edge precision of 53%. Still, WCDG finds even more incorrect non-projective sentences and edges than present. More importantly, as the present shift-reduce predictor is not reliable for non-projective parsing, its participation reduces the non-projective sentence and edge precision — to 80 (55%) and 123 (47%). This question whether it is beneficial to have MSTParser as a predictor for hybrid non-projective parsing will be addressed again in the next chapter.

Per Label Accuracy

The information about per label accuracy is summarized in Table 5.12. Structural precision stays in this case for the amount of those edges that have a given label in the gold annotation and are correctly attached in the MSTParser result independent of the label assigned. Label recall is understood as the ratio of those edges that are assigned a correct label to the number of all edges with the corresponding label in the gold annotation. Label precision represents the ratio of those edges that were assigned a specific label correctly to all that are having that label in the parse result. The last column of Table 5.12 lists the labels with which the correct labels are mixed by the MSTParser. If some alternative is chosen often, the percentage of its occurrences in the wrong solution is given next to it. Labels in the last column for which the percentage is not given are mixed with the correct solution only occasionally.

Most strong and weak sides of the MSTParser read off Table 5.12 are connected to the locality of the decisions made by its algorithm. Best performance is observed for the labels which can be easily assigned based on the local context, such as **AUX**, **DET**, **ATTR**, **PN**, **PART**, and seem not very difficult for syntactic parsing in general. As for **KONJ** and **AVZ**, it seems that lexicalization has added up to the locality accounting for the absolute label recall.

For those labels that are more or less often in general, three main groups of errors can be identified. The first concerns faults at finding the correct verb compliments, such as **SUBJ**, **OBJA**, **OBJD**, **PRED**, **EXPL** that are mistaken for each other, but also for **GMOD** or **APP**. The MSTParser does not know at a concrete point of analysis whether any other verb compliment has already been assigned, and, thus, sentences with multiple verb compliments of the same kind are as often as those having no subject. An example can be found in the very first sentence of the test set presented in Figure 5.2 [4] in which two objects have been assigned to "stellen" in the first clause and two subjects are assigned to "erhält" in the second clause. Another example where a second **GMOD** instead of an **OBJA** is indentified is shown in Figure 5.3. Parallel to the label confusion, most of these labels, excluding **SUBJ**, also have attachment accuracy below 80% which seems to be connected to the erroneous labels. Thus, if a word is attached to a full verb, it also has a **SUBJ** as its label more often than an **OBJA** or **EXPL** (the latter has not been identified as such in the example sentence presented in Figure 5.4).

As the labels are assigned by the MSTParser algorithm only in the second stage, this seems to be an error coming from the parsing algorithm of the first stage that deals only with the structure. The present features used by the MSTParser and tuned with the help of the English data very probably do not allow it to differentiate enough about the different cases of German.

The second source of errors are all subordinate clauses in the broad sense, such as **NEB**, **REL**, **OBJC**, **OBJI**, or **SUBJC** that are often mixed with each other, where the label **NEB** is assigned more often than the other labels of the group, such as in the structure of the same sentence that was used

[4] Example sentences have been shortened for better presentation.

Label	Gold, (*items*)	Structural precision	Label precision	recall	Label error
EXPL	14	100.0	50.0	42.9	**SUBJ**: 57.1%
DET	2015	99.4	100.0	100.0	
PN	1723	99.4	99.1	99.4	**OBJA**, **ATTR**
PART	87	98.9	98.9	100.0	
AVZ	85	98.8	100.0	100.0	
AUX	635	98.7	98.7	98.6	**NEB**, **CJ**, **SUBJC**
ATTR	1156	98.6	98.8	99.0	**PN**, **OBJA**, **CJ**
KONJ	185	95.1	100.0	100.0	
SUBJ	1210	93.0	84.0	88.8	**OBJA**: 7.8%, **GMOD**, **PRED**
SUBJC	36	91.7	48.3	38.9	**NEB**: 27.7%, **OBJC**, **OBJI**
CJ	430	91.2	93.4	91.6	**PP**, **S**, **KON**, **SUBJ**
APP	470	90.2	84.2	91.9	**OBJA**, **SUBJ**, **GMOD**, **ZEIT**
S	1126	89.6	94.7	90.9	**APP**: 2%, **KON**, **ADV**, **OBJA**
PRED	144	88.9	75.2	63.2	**SUBJ**: 13.2%, **PP**: 15%, **ADV**
OBJP	53	88.7	48.6	32.0	**PP**: 68.0%
GMOD	354	88.4	81.6	85.3	**SUBJ**: 5.1%, **APP**: 4.2%, **OBJA**
KOM	86	86.0	98.9	100.0	
OBJA	604	84.8	71.5	74.3	**SUBJ**: 16.1%, **GMOD**: 5.6%, **APP**
OBJI	57	82.5	74.2	86.0	**NEB**: 12.2%, **SUBJC**
ADV	1234	81.0	98.1	98.5	**PRED**, **CJ**
ZEIT	50	80.0	70.8	68.0	**APP**, **OBJA**, **SUBJ**
KON	436	79.6	93.2	87.8	**APP**, **S**, **OBJC**, **AUX**
PP	1697	79.3	95.8	98.4	**OBJP**: 1.1%, **PRED**
OBJD	88	77.3	61.3	21.6	**OBJA**: 39.8%, **SUBJ**: 27.3%, **GMOD**
NEB	122	75.4	52.9	66.4	**OBJI**: 8.2%, **OBJC**, **S**, **SUBJC**
GRAD	16	75.0	84.6	68.8	**OBJA**, **ZEIT**
ETH	15	73.3	66.7	13.3	**SUBJ**: 40%, **GMOD**, **OBJA**, **OBJD**
REL	122	65.6	85.5	77.0	**NEB**: 11.5%, **OBJC**, **S**
OBJC	53	54.7	45.7	39.6	**NEB**: 47.2%, **REL**
PAR	23	13.0	0.0	0.0	**S**: 73.9%, **KON**

Table 5.12: Per label accuracy of the MSTParser on the NEGRA testset — sorted according to the structural precision (trained on `heiseticker`; tested with POS annotations from the gold standard).

for illustration last (Figure 5.4) the label **NEB** is assigned instead of **SUBJC**. All of these labels excluding **SUBJC** also experience attachment problems which is actually not untypical for syntactic parsers. Thus, in the sentence presented in Figure 5.5 (A1) an edge assigned a **NEB** instead of a **REL** is incorrectly attached under the full verb of the main clause instead of its reference. The attachment problem seems to be the reason for the incorrect labeling again.

The MSTParser is having a preference for attaching the subordinate clauses (in the broad sense) to the full verb of the main clause and while doing so it suggests its labeler assigning a **NEB** to

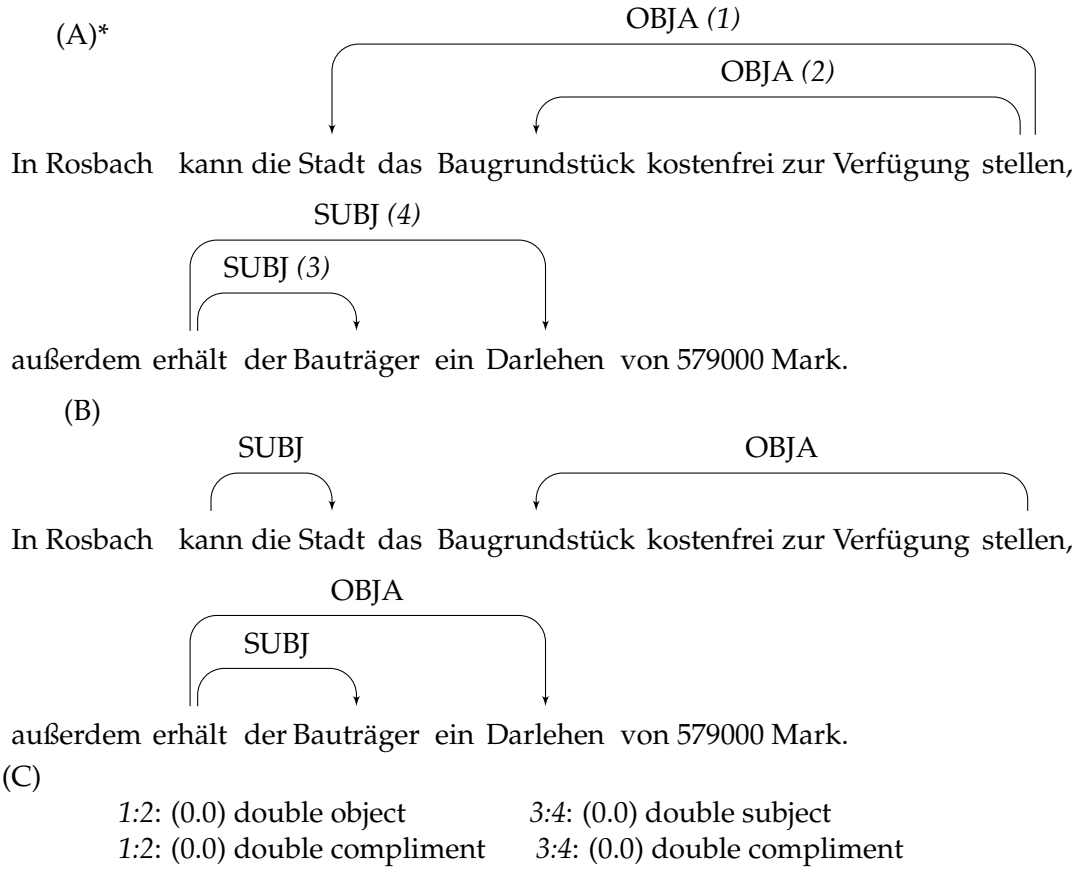


Figure 5.2: Example of verb compliment errors in NEGRA sentence 18602: (A) the analysis by the MSTParser; (B) the correct analysis; (C) WCDG constraints violated by the incorrect edges.

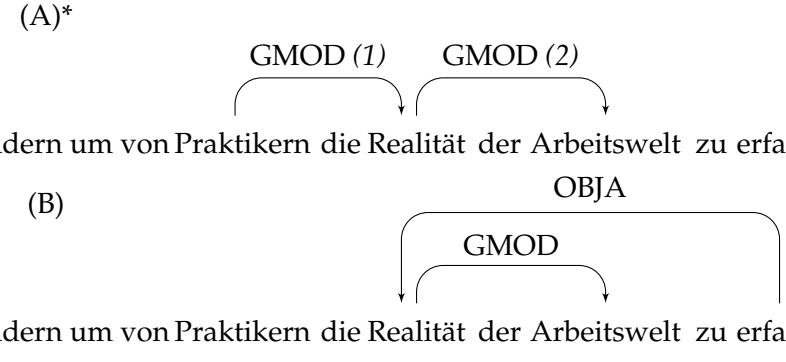


Figure 5.3: Example of a verb compliment error in NEGRA sentence 18718: (A) the analysis by the MSTParser; (B) the correct analysis; (C) a WCDG constraint violated by an incorrect edge.

them incorrectly as the structures where **NEB** is attached to the full verb are more often than if an **OBJC** is attached to the full verb. **NEB** itself is most often mixed with **OBJI**, this happens in

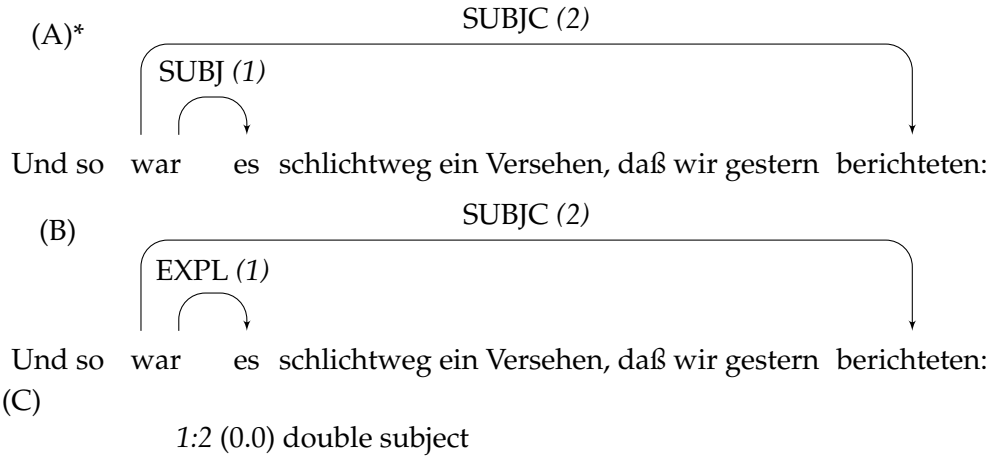


Figure 5.4: Example of a verb compliment error in the NEGRA sentence 18700: (A) the analysis by the MSTParser; (B) the correct analysis; (C) a WCDG constraint violated by an incorrect edge.

the structures similar to the one of sentence in Figure 5.5 (B1). According to the WCDG model of German edge *2* going to the "zu"-infinitive in this sentence should be assigned a **NEB** label as the whole clause is introduced with the conjunction "um". A probable cause of errors of the MSTParser in such situations is that the distance to the conjunction is much larger than to the particle "zu" that has an influence on how the features about these edges are scored.

The group of context-dependent labels (**ADV**, **PP**, **KOM**, **KON**) that will be the last one mentioned at this place has shown typical structural attachment problems. But it should be noted that 79.3% of correctly attached **PP**, 81.0% of **ADV**, 79.6% of **KON** and 86.0% of **KOM** correctly attached are quite state-of-the-art for these difficult cases of disambiguation.

Using WCDG as Evaluator

A very useful property of WCDG is that it cannot only be used as a parser, but also as an analyzer of the output of the other parsers as any dependency trees can be translated into the WCDG constraints easily. The constraint violations show what is grammatically wrong with the parse according to the WCDG constraint grammar. For the MSTParser output the top of the list of the most often violated hard constraints (those with zero weights) is the list presented in Table 5.13. It should be noted that the figures in this table should not be understood as the number of errors in the input. Several different constraints may become violated by one single edge, the examples have already been given in this section. Besides, each figure of this section presenting a parsing analysis of the MSTParser also points out which constraint violations of WCDG have happened to localize the error. Third on the constraint violation list comes the projectivity [5]. How an incorrect analysis may cause non-projectivity is shown in Figure 5.5 (C1). In this case, crossing of edges *1* and *2* is prohibited by WCDG, because it does not fall into any category of non-projectivity described in the constraint grammar.

[5]Under "projectivity", all kinds of projectivity violations have been summed up. The WCDG, in fact, differentiates between finer kinds of projectivity violations.

WCDG constraint name	Occurrences
double complement	230
double subject	156
projectivity	117
double object	78
literary clause without comma	28
NEB without conjunction	28
AUX and object	24
OBJA in passive	22
REL without relative pronoun	18
KOUS at the first position	17
OBJA not allowed	15
conjunction without a clause	14
REL should be **NEB**	14
adverb under adverb	13
root token	13

Table 5.13: WCDG constraints active on the MSTParser output.

5.3 Summary

This chapter reported about the evaluation of the MSTParser on the same part of the NEGRA corpus WCDG was previously evaluated on. Although the former does not outperform WCDG, having it as a predictor would be beneficial. It is a rather reliable parser itself having a very good accuracy of $91.9\%/89.3\%$. Or, without the punctuation, the accuracy equals $90.5\%/87.5\%$ which is not far from the previous evaluations. However, the result of $90.4\%/87.3\%$ during the CoNLL-X shared task has been obtained with morphological features of the gold standard used. Although the parsing accuracy of the MSTParser depends on the presence of lemmas in the data strongly, what is more important, the use of a real tagger only makes the performance decrease by around one percent.

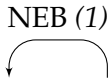
The approximation of the second-order projective algorithm allows the MSTParser to perform well on non-projective input in general. Although the evaluation has shown that the precision in identifying the non-projective edges correctly on the edge or even on the sentence level is higher in the WCDG system, the advantages to have the MSTParser as a predictor for parsing non-projective sentences in particular are still to be evaluated in the next chapter.

This chapter also analyzed typical errors that the MSTParser does on German data. Most helpful in identifying the errors was the WCDG system. Although the comparison to the gold annotation shows which regent or which edge label have been identified incorrectly, it does not "explain" the syntactical reasons. WCDG helps identify the source of failure and even provides the grammatical explanation for the errors in form of violated constraints. The success of WCDG on the output of the MSTParser in this respect may be referred to the fact that WCDG itself has another

source of knowledge and, thus, combining both approaches seems to be a very promising task the realization of which will be described in the next chapter.

For all the evaluations, twenty thousand sentences from the `heiseticker` corpus were used as a training set. The choice was motivated by the annotation guidelines of WCDG to which most of NEGRA dependency trees extracted automatically are not compliant. Still, it was shown that similar results can be obtained when the parsing model is trained on NEGRA itself. At last, it should be noted that the training set of the `heiseticker` corpus was chosen arbitrary and the limitation to only twenty thousand sentences for training was caused by the technical properties of the computational resources used. Alternative training on another arbitrarily chosen twenty thousand sentences set of `heiseticker` ($30,000 - 49,999$) had a slightly worse accuracy (90.8% and 87.8%).

(A1)*



Die Zeltstädte, mit denen Hessen sich auf einen Flüchtlingsansturm aus dem Jugoslawien vorbereiten wollte, werden abgebaut.

(A2)

1: (0.0) NEB should be REL

1: (0.0) NEB without conjunction

(B1)*

Eine Bürgerfahrt nach Straßburg veranstaltet die SPD, um den Tag der Einheit mit dem Europagedanken zu verknüpfen.

(B2)

1: (0.0) OBJI not allowed

1:2: (0.0) OBJI should be NEB

(C1)*

Aus Angst vor dem Vorwurf der Wirtschaft, "Wohnungszwangsbewirtschaftung" zu wollen, unterblieben bisher die Maßnahmen.

(C2)

2: (0.0) object definition

2: (0.0) OBJI not allowed

1:2: (0.0) projectivity

1: (0.01) GMOD without genitive

1:3: (0.1) double GMOD

Figure 5.5: Some errors of the MSTParser on NEGRA test set and the WCDG constraints violated by the incorrect edges. (A1) NEGRA sentence 18904: "wollte" should be a **`REL`** of "Zeltstädte"; (B1) NEGRA sentence 18672: edge *2* should be **`NEB`**; (C1) non-projectivity based on false attachment in NEGRA sentence 18649: "wollen" should be an **`OBJI`** of "Vorwurf" and "Wohnungszwangsbewirtschaftung" should be an **`OBJA`** of "wollen" (A2), (B2), (C2) — the WCDG constraints violated at the incorrect edges.

Chapter 6

MSTParser as Predictor for WCDG

2 + 2 = 5 for extremely large values of 2.
Old wisdom

The previous chapter confirmed that the MSTParser is a very promising predictor for the WCDG as the accuracy it achieves is almost domain independent, it is not very sensitive to POS tagging errors and it can deal with the non-projective data. Moreover, it incorporates information from a complementary source in comparison to that of WCDG. All these are motivating aspects for integrating the MSTParser as a predictor for WCDG. In comparison to the previously used shift-reduce predictor, it is essential that non-projective structures can be parsed by the MSTParser, even while the performance of the combined approach is to be investigated first as the MSTParser has shown to be not very precise about the non-projective edges in comparison to WCDG. Besides, the MSTParser, when used together with the present POS tagger, outperforms the shift-reduce predictor by 6% and 7.3%, structural and labeled accuracy respectively. The last fact also hides a potential danger that WCDG might not be flexible enough to deal with such a good input, because in an attempt to fix some errors WCDG might not only correct a few, but also introduce additional ones.

6.1 Constraint Weights for MST Predictor

To evaluate the MSTParser as a predictor component for WCDG, the plug-in architecture of the WCDG system can be used efficiently. The most important thing to decide on is how valuable the information the predictor provides relative to the already present information in the system is. This gradation is expressed in WCDG with a constraint weight. It is assumed that if the information is considered reliable enough then low constraint weights could be used, but in this case the chance that the errors committed by the predictor would get corrected is lower than in the case that higher weights have been chosen since the predictor input was not considered reliable enough from the very beginning. Typically for WCDG, the best suitable weights are to be determined experimentally.

For each experiment a separate WCDG driver file including the parameter settings is prepared (see E.1 for details). The driver loads the `MST.cdg` file extending the WCDG grammar with three additional constraints and switches the predictor (referred from now on as the MST predictor) on:

```
#pragma predict MST 'mst.pl -v 3 -' cat

{X!SYN} : 'MST:regent' : stat : W :
predict(X@id, MST, gov) = X^to;

{X|SYN} : 'MST:null' : stat : W :
predict(X@id, MST, gov) = 0;

{X:SYN} : 'MST:label' : stat : W :
predict(X@id, MST, lab) = X.label;
```

The first two constraints advise WCDG on the structural information whereby the second deals with the elements modifying the root and the first with all the others; the third fetches the edge label predicted. W, $0 \leq W \leq 1$, stays for the constraint weight chosen for the experiment. These constraints do not differ from those used for the shift-reduce predictor in the previous experiments [Fot06].

The first set of the experiments uses the same weights for the label and the structural constraints. Actually, the MSTParser is not so reliable in edge labeling as in the structural information, but this difference can be tuned in a successive experiment. For easier comparison to the previous results, the timeout value has been set to a very large value of $10\,ms$, the time during which the search terminates on its own for over 90 percent of the sentences. All the experiments described in this chapter were done with the POS tagger of German that is embedded into WCDG. The other predictors are switched on only when explicitly mentioned.

	Correct (of 16687)		Accuracy, %		
Weights	head	label	structural	labeled	Time
0.1	15335	15011	91.9	90.0	85.26
0.3	15411	15117	92.4	90.6	110.86
0.5	15480	15228	92.8	91.3	160.59
0.7	15526	15270	93.0	91.5	174.81
0.75	15533	15275	**93.1**	91.5	173.95
0.8	15516	15279	93.0	91.6	174.50
0.85	15512	15294	93.0	91.7	180.43
0.9	15523	15313	93.0	**91.8**	189.11
0.95	15516	15302	93.0	91.7	216.00
1.0	14934	14668	89.5	87.9	199.38

Table 6.1: Search for MST constraint weights.

Table 6.1 summarizes the results of the experiments. The highest structural accuracy of 93.1% is achieved for the MST constraint weights set to 0.75 and the highest labeled accuracy of 91.8% is reached for value of 0.9. Constraint weights equal 1.0 actually switch the constraints off making the experiment correspond to the one without them with the difference that the grammar is three constraints longer, more time is needed to load it, and less time is available for the search before the timeout is reached which is perceptible by a slight difference to the result referenced in

Weight for	Correct (of 16687)		Accuracy, %		
label constraint	head	label	structural	labeled	Time
0.8	15535	15294	93.1	91.7	160.61
0.85	15533	15319	93.1	91.8	156.23
0.90	**15538**	**15326**	93.1	**91.8**	154.78
0.95	15533	15311	93.1	91.8	171.09

Table 6.2: Search for MST label constraint weights (with structural constraints at 0.75).

Chapter 4. Slight variations in the performance are also possible due to the heuristic nature of the search algorithm.

The difference in the constraint values for which the optimal structural and labeled accuracy is achieved suggests another experiment which has the purpose to find the optimal combination of both. The results in Table 6.2 show that a slight improvement of the labeled accuracy to 91.8% is possible if the structural constraints are left at 0.75 and the label constraint weight is changed to 0.9 whereby the optimal structural accuracy value of 91.1% is preserved. In fact, the number of correctly analyzed edges is even greater than in the previous experiments with the same constraint values achieved: for 5 more edges the correct regent is found (15538 vs. 15533) and 13 more get a correct label (15326 vs. 15313).

Best solutions are also paid off with time reduction. Both 0.75-weight experiment in Table 6.1 and 0.9-weight experiment in Table 6.2 have spent the least time to find the solution parse among all the alternative experiments referenced in the corresponding table.

		With punctuation, %		Without punctuation, %	
Sentence length	Instances	structural	labeled	structural	labeled
1 – 10	340	96.0	93.5	95.1	92.3
11 – 20	323	94.1	92.8	93.2	91.7
21 – 30	229	93.1	92.0	92.0	90.8
31 – 40	76	91.0	90.0	89.5	88.3
≥ 40	32	90.3	89.1	88.6	87.3
overall	1000	93.1	91.8	92.0	90.5

Table 6.3: Dependency on the sentence length (MST as predictor for WCDG, real POS tagger, constraint weights 0.75/0.90).

A better reliability of the MSTParser as a predictor than that of the shift-reduce parser is also reflected in the determined constraint values. With the shift-reduce predictor, the optimal performance was achieved when the corresponding constraints were set to a greater value of 0.9. Besides, the difference in the constraint weights for the regent vs. that for the label determined for the MST predictor confirms the previously made assumption about the difference in the reliability of these two types of information provided by it.

The details about the values for the structural and parsing accuracy for the best combined experiment are summarized in Table 6.3. It also shows that even under the exclusion of the punctuation the structural accuracy is well over 90 percent and the labeled accuracy steps over the 90 percent boundary, too.

6.2 Results Analysis

The performance of the combined experiment is well above the values that each of the participating parsers could reach alone. This section will outline the main cases in which the synergy has brought good improvements and in which probably not.

In the absolute figures, the output of the MSTParser has 1509 errors of which 902 are fixed by WCDG. While fixing, it adds 542 of its own errors, so that the result still has 1149 errors. There are only 220 different constraints (by name, not by amount of edges) violated by the output of WCDG and all hard conflicts are resolved from the viewpoint of WCDG. This is how the top of the list looks like now (compare to Table 5.13):

fragment	(0.05)	124
isolated subordinate clause	(0.4)	9
noun as ethic dative	(0.4)	6
absent conjunction	(0.2)	6
number of subject	(0.1)	6
category of co-subordination	(0.1)	5
isolated relative clause	(0.1)	4
case of DET-object	(0.19)	4
parenthesis around object	(0.3)	4
shortened subordinate clause	(0.4)	3
OBJI without zu	(0.1)	3
absent antecedent	(0.1)	3

This does not in any case mean that there are no errors left in the output, even the constraints with the weight of 0.1 are rather prohibitive and identify most probable cases of errors. But those cases that were explicitly forbidden by the grammar (see Table 5.13) have been fixed, at least a solution was found that the grammar allows, although in some cases not the correct one.

Thus, errors in sentences 18602 and 18718 shown in Figures 5.2 and 5.3 respectively that concern double verb compliments and **GMOD** without genitive have been correctly fixed. But, for example, an attempt to deprive sentence 18700 (Figure 5.4) of a second subject has chosen the wrong subject to remove: it changed the previously correct **SUBJC** to **NEB** so that **EXPL** was never identified and remained as a subject. The relative clause in sentence 18904 (Figure 5.5 (A1)) has been identified correctly (but incorrectly attached in the part of the sentence that is not shown). Changing **OBJI** for **NEB** as suggested by the constraints in sentence 18672 (Figure 5.5 (B1)) has worked quite properly. The projectivity issue in sentence 18649 (Figure 5.5 (C1)) has been removed, but the attachment difficulty remained: thus, "Angst" and not "Vorwurf" erroneously became the regent of "wollen" although the edge has been assigned an **OBJI** correctly, only "Wohnungszwangsbewirtschaftung" has been misattached to "Vorwurf" with an **APP** label at the edge.

The values of the per label accuracy of WCDG, the MSTParser and both combined (all with the same POS tagger whereas all the other statistical enhancements of WCDG were not working)

	Gold,	WCDG (POS)		MSTParser (POS)		WCDG (POS + MST)	
Label	(*items*)	str. pr.	lab. rec.	str. pr.	lab. rec.	str. pr.	lab. rec.
DET	2015	98.4	99.3	98.7	99.5	99.3	99.5
PN	1723	97.4	97.4	98.0	98.0	98.0	98.7
PP	1697	67.6	98.1	78.3	97.4	80.1	98.5
ADV	1234	76.6	94.7	79.4	95.4	82.2	97.2
SUBJ	1210	94.0	90.9	91.3	86.4	95.8	94.0
ATTR	1156	95.2	95.8	97.7	98.2	98.3	98.4
S	1126	89.2	90.1	89.3	90.5	90.5	91.0
AUX	635	95.9	94.2	98.6	97.8	98.7	97.6
OBJA	604	87.9	83.9	83.8	72.5	92.5	88.7
APP	470	85.1	88.5	88.9	90.9	90.9	94.0
KON	436	78.9	88.1	78.9	88.3	86.0	89.2
CJ	430	85.6	86.5	90.9	91.4	93.0	93.5
GMOD	354	90.7	90.7	89.0	85.3	96.3	95.8
KONJ	185	88.6	91.9	91.9	95.7	95.1	95.7
PRED	144	90.3	75.0	85.4	60.4	91.7	76.4
NEB	122	68.9	82.8	73.0	66.4	79.5	90.2
REL	122	64.8	77.9	59.0	77.0	68.9	86.9
OBJD	88	92.0	86.4	76.1	20.5	89.8	85.2
PART	87	96.6	98.8	96.6	96.6	96.6	96.6
KOM	86	77.9	93.0	79.1	98.8	81.4	96.5
AVZ	85	100.0	95.3	98.8	92.9	100.0	94.1
OBJI	57	91.2	94.7	82.5	86.0	82.5	86.0
OBJP	53	79.2	66.0	94.3	32.1	94.3	83.0
OBJC	53	73.6	73.6	56.6	37.7	62.3	60.4
ZEIT	50	76.0	86.0	78.0	66.0	82.0	80.0
SUBJC	36	83.3	69.4	94.4	38.9	94.4	77.8
PAR	23	43.5	43.5	8.7	0.0	39.1	34.8
GRAD	16	50.0	43.8	81.2	62.5	81.2	75.0
ETH	15	86.7	86.7	73.3	13.3	93.3	93.3
EXPL	14	100.0	50.0	100.0	42.9	100.0	78.6

Table 6.4: Per label accuracy in comparison. Structural precision (str. pr.) vs. label recall (lab. rec.).

are shown in comparison in Table 6.4. The most typical result suggested by this table is that the performance in the combined experiment reaches a higher level than in each of the parsers alone. Especially large difference between the label recall WCDG could reach alone and that became

possible with the predictor is observed for **APP**, **CJ**, **NEB**, **REL** and **OBJD**; the same is true for the structural precision of **PP** and **NEB**.

In this respect, the result about the increase of the structural precision of the **PP**-attachment seems interesting as the MSTParser attaches 79.3 percent of **PP**s correctly which is even more than WCDG combined with the **PP**-attacher that achieves 78.7 percent structural precision for **PP** edges. (Cf.: if MSTParser is trained on NEGRA itself it achieves an even greater performance of 80.4%). This result is not very surprising. As the MSTParser is itself a statistical parser trained on a full corpus there seem to be no reasons why it should deal **PP**s worse that a **PP**-attacher that has been trained on restricted four-tuples input.

As for the errors in the output of the MSTParser that are most often corrected in the hybrid experiment, this happens for both the structural precision and label recall of the most verb compliments, such as **OBJA**, **OBJD**, **PRED**, **OBJC**, or for such subordinate clauses like **NEB** and **REL**.

GMOD is an interesting case: structural precision and label recall increase for it by around 6% although the MSTParser has predicted it correctly quite more seldom than WCDG alone (the label recall of the MSTParser for **GMOD** was over 5% below that of WCDG).

The cases in which WCDG would perform worse with the predictor than its predictor alone can be hardly found. The only example of a very slight decline is the label recall of **KOM**.

Still, one may observe many cases in which the predictor had a negative influence on the performance of WCDG, such as for different kinds of objects (**OBJD**, **OBJC** and **OBJI**) and **PAR**. For all, the result of the MSTParser was below that of WCDG with no other enhancements with the exception of the POS tagger. Same can be said about the labeled accuracy for **AVZ**, **PART** and **ZEIT**. This worsening effect can be attributed to the lower values of the WCDG constraints for the corresponding labels and edges than for the MST predictor. Thus, the search could not find a decision scoring better than that when the MST prediction has been followed.

One area where the synergy of both approaches was not so successful is that of non-projectivity. The WCDG grammar specially includes hard constraints for the violations of non-projectivity so that the search algorithm does not lose its time in vain on non-projective paths (and these would present a substantial increase to the number of paths).

	Non-projective edges				Non-projective sentences			
Algorithm	recall		precision		recall		precision	
WCDG (POS)	377	(37%)	138	(53%)	178	(51%)	91	(63%)
MSTParser (POS)	413	(23%)	94	(36%)	182	(35%)	64	(44%)
WCDG (POS + MST)	289	(48%)	139	(53%)	145	(61%)	89	(61%)
WCDG (POS + SR)	304	(41%)	123	(47%)	140	(57%)	80	(55%)
Gold	261 (100%)				145 (100%)			

Table 6.5: Precision and recall of the non-projective edges for different parsing runs.

Table 6.5 shows the same measures of non-projective edges and sentences precision and recall as in the previous chapter. The experiment WCDG (POS + SR) shows the values of the WCDG experiment with the shift-reduce predictor for comparison. The table makes clear that WCDG

	Non-projective sentences		Projective sentences	
Algorithm	structural	labeled	structural	labeled
WCDG (POS)	87.2	85.9	90.2	88.5
MSTParser (POS)	88.2	85.7	91.7	88.7
WCDG (POS + MST)	91.3	90.0	93.6	92.4
WCDG (POS + SR)	88.7	87.4	92.2	90.6
Gold	261 (100%)		145 (100%)	

Table 6.6: Structural and labeled accuracy for different parsing runs for non-projective vs. projective sentences.

using only the POS tagger is preciser than the MSTParser finding the cases of non-projectivity. Although the comparison of the overall structural and labeled accuracy on non-projective vs. projective sentences that is generally done to evaluate the performance of the parser on the non-projective sentences speaks clearly for the combination of the two (on the non-projective sentences the accuracy increases by 4% percent over that of WCDG and by 3% over that of the MSTParser), the absolute number of correctly identified non-projective edge pairs increases only by one, and of the correctly identified sentences with non-projectivity even reduces by 2. Still, in comparison to the accuracy improvements on the projective sentences that is equal 2 and 3 in the respective cases, the benefits for parsing non-projective sentences with the combination of the two parsers becomes obvious.

Not the same non-projective sentence or edges of the WCDG (POS) experiment are present in the result of the WCDG (POS + MST), but while a quarter of new non-projectivity cases is added to the non-projective set, around the same number of cases resolved without the external predictor is lost.

In any case, the overall result is better than when the shift-reduce parser is used as a predictor. For the WCDG (POS + SR) experiment the number of identified non-projective sentences is reduced by 11 and the number of correctly identified non-projective edge pairs is reduced by 15. The overall accuracy on the non-projective sentences (Table 6.6) is increased only slightly when the shift-reduce parser is used as a predictor in comparison to the WCDG (POS) experiment, both structural and labeled accuracy increase by 1.5%.

6.3 Combining Different Predictors

At last, the MST predictor was experimentally evaluated in combination with other predictors available in WCDG. The results of the experiments are shown in Table 6.7. Every combination of the MST with other predictors (first four experiments) improves the accuracy further by additional tenths of a percent. In combination with the supertagger, the increase is the highest and equals 0.4%. In the previous experiments in which different predictors were combined, the supertagger also brought the highest gains.

The accuracy reaches its maximum at $93.9\%/92.6\%$ (if finer comparison about the absolute number of the correct edges and labels is done) when all the available predictors of WCDG excluding

	Correct (of 16687)		Accuracy, %		
Experiment	head	label	structural	labeled	Time
PP + MST	15540	15331	93.1	91.9	178.10
CP + MST	15549	15335	93.2	91.9	169.97
SR + MST	15569	15341	93.3	91.9	190.49
ST + MST	15598	15381	93.5	92.2	291.56
CP + SR + MST	15576	15350	93.3	92.0	203.19
CP + ST + MST	15618	15402	93.6	92.3	272.91
ST + SR + MST	15658	15448	93.8	92.6	316.27
PP + CP + ST + MST	15632	15413	93.7	92.4	294.86
PP + ST + SR + MST	15657	15443	93.8	92.5	316.46
CP + ST + SR + MST	**15664**	**15454**	**93.9**	**92.6**	309.14
PP + CP + ST + SR + MST	15663	15451	93.9	92.6	311.59

Table 6.7: Combinations of predictors.

the **PP**-attacher are involved: the chunker, the supertagger as well as the shift-reduce and MST predictors. Unforunately, the **PP**-attacher brings accuracy reductions in those cases when it is working parallel to the shift-reduce predictor. This effect has already been observed in the experiments that combined the two alone. In the experiment in which the MST was combined with the **PP**-attacher, the increase of the performance was also below a tenth of a percent. The possible reasons why the use of an additional information source does not improve the performance in this case may be the disadvantages of the **PP**-attacher compared to a full parser that were already outlined in the previous section.

6.4 Summary

The experiments that were described in this chapter and the analysis done have proved that integrating the MSTParser in WCDG as a full predictor is beneficial for both parsers. Since they take their decisions based on completely different sources of knowledge, combining both helps avoid many mistakes each of the parsers would do alone as well as find solutions in situations in which every of them has been unable to do it. Moreover, the accuracy grows to $93.1\%/92.9\%$. These accuracy values are greater than any previous parsing experiments on the used NEGRA test set could achieve.

Still, the expected benefits for the non-projective sentences have not yet been observed to the full extent. The precision of the combined system to find non-projective sentences and edges remained limited by the value that WCDG could achieve alone. While the MSTParser in many cases predicts non-projectivity correctly — although this amount could be sufficiently greater for an external predictor — WCDG is seldom capable of accepting this external evidence. On the contrary, WCDG often accepts an incorrect projective solution of the predictor instead of relying on its own cues. In its interaction with external predictors WCDG should typically decide about the alternatives. The difference for the described case of non-projectivity disambiguation is that

the existing framework apparently does not allow to take decisions about non-projectivity based on various sources of knowledge. Whether the reasons are incorrect relations between different constraint weights that control the decisions made about non-projectivity or whether this is a grave lack of the system that is a topic of separate research. However, it should be mentioned that the accuracy achieved on the non-projective sentences has improved in general when all the edges and not only non-projective ones are considered (but so did the accuracy on the projective sentences, too).

Chapter 7

Concluding Remarks

Tempus fugit (time flies)
Ovid

7.1 Summary

This work has one more time showed that hybrid parsing methods outperform parsers built according to only one parsing paradigm. In this work, a combination of two different sources of knowledge was investigated, one that is coming from a rule-based system and the other that is based on a novel statistical solution in combination with maximum spanning tree parsing algorithms. Weighted Constraint Dependency Grammar presented in Chapter 4 is a system of the first type that allows for efficient integration of external sources of knowledge due to its flexible constraint grammar.

The external predictor the integration of which into WCDG has been investigated in Chapter 6, is a dependency parser, the MSTParser, developed by R. McDonald. The reasons for the efficiency of the MSTParser lie in the successful combination of online learning explored in Chapter 2 and maximum spanning tree search as a solution method for the parsing problem that allows to formulate efficient algorithms for both projective and non-projective cases that were presented in Chapter 3.

The high accuracy, the MSTParser achieved previously on German data, has been confirmed on the data annotated according to the internal guidelines of WCDG. The results of the experiments have been reported in Chapter 5. The errors that the parser makes on German data have been systematized and analyzed. An important role during this analysis was played by the WCDG system that provides comfortable options to analyze output of external parsing solutions.

The integration of the MSTParser into WCDG has improved the parsing results previously achievable by each of the parsers alone and reached 93.1% structural and 91.8% labeled accuracy. Both types of accuracy have improved by further 0.8% to $93.9\%/92.6\%$ after the MSTParser was used in combination with other external predictors of WCDG.

The main experiment results are presented Table 7.1 that shows them both with and without punctuation. Table 7.1 (A) shows the experimental results obtained with a real POS tagger that were reported in Chapters 5 and 6 in comparison to the previous WCDG result under similar

conditions. Table 7.1 (B) summarizes the results of similar experiments in ideal conditions, i.e., when the POS tags from the annotation and not a real POS tagger is used. In this case, the accuracy increases further to $94.2\%/93.1\%$.

			With punctuation, %		Without punctuation, %	
Experiment		Components	structural	labeled	structural	labeled
(A)	1	WCDG (POS only)	89.6	88.0	87.8	86.0
	2	MSTParser	91.0	88.0	89.5	86.0
	3	WCDG + MST	93.1	91.8	92.0	90.5
	4	WCDG + all	93.9	92.6	92.9	91.4
(B)	5	WCDG (POS only)	90.4	89.1	88.9	87.3
	6	MSTParser	91.9	89.3	90.5	87.5
	7	WCDG + MST	93.8	92.5	92.9	91.3
	8	WCDG + five	94.2	93.1	93.3	92.0

Table 7.1: Overview of the main results with and without punctuation. (A) with a real POS tagger; (B) with tagging from the gold standard.

A special emphasis during the analysis of the results has been made on exploring the benefits of parsing non-projective sentences with an external predictor as this was the first time a predictor has been used that allows to parse non-projective input.

7.2 Outlook

Two perspectives on further research can be pointed out. One deals with the improvements of the MSTParser and the other with its integration into WCDG.

The analysis of the errors that the MSTParser does on German data that was carried out in Chapter 5 can be used as a basis for the German specific extensions of it. This area of further research was already mentioned by the author in [McD06] who pointed out that the features the MSTParser considers were tuned with the help of English data and new features would be helpful for other languages, especially if these languages, like German, have a more flexible word order than English. For the simplification of this task, the MSTParser also provides an interface to add new features.

On the other hand, it seems promising to use WCDG specially to fix the errors that occur in the MSTParser output most often, such as verb compliments confusion or projectivity violations. Such corrections would probably be possible on the restricted grammar variant having much less constraints than in the present constraint collection.

Although the evaluation has shown that the MSTParser is a very promising predictor for WCDG to have, the integration of it into WCDG for broad coverage tests was prevented by the memory usage issues of the MSTParser. It needs at least 1800 GB memory for the Java heap and this is something not every computer has at disposal nowadays. Presently, the whole parsing model is stored in one file and even to parse only one sentence, the MSTParser still loads the whole model,

i.e., hundreds of megabytes, into operational memory. The memory usage of the MSTParser could be surely reduced much if it separated the parsing model among many files and the search management for the file where the features needed at the moment lie was added. Most of the features saved in the model are lexicalized featured that store information about a given word and so they do not have to be loaded for every sentence. Besides, the number of the POS features for a given language is restricted and they do not take so much place in the model as the lexicalized features. Having a model partitioned between many files would make it necessary to load only the files important for the given sentence and ignore the rest. This model management could also be easily enhanced with a simple lexicon storing, e.g., the base forms of the words on the presence of which the MSTParser is so dependent as it was stated in the experiments. Surely, all these changes would come at the expense of the time lost for searching the necessary files — probably this suggestion should only refer to the testing phase as time losses during training are not desired since it already lasts rather long — but it would allow to deploy the parser at more computer systems, besides, the proposed changes would also enable the parser to use more training data, to get larger parsing models and integrate different data sources in one model.

Although the algorithm of the MSTParser works on non-projective data, in the combination with WCDG the gains were only perceived for the non-projective sentences on the whole, but not in the number of the recognized non-projective edges. The reasons why this improvement was not achieved should be investigated.

Now that at least two full predictors, the shift-reduce predictor and the MST are available for WCDG and their strengths and weaknesses have been investigated in detail one should attempt to formulate more precise rules for WCDG that would allow to choose the predictor whose input is more reliable in the concrete context. An attempt to differentiate the weights of the predictions according to the label of the edge with the MST predictor alone has unfortunately reduced and not improved the performance. A possible reason is that for no label does the MST achieve the absolute performance and such rules are better formulated not absolutely prohibitively, but relative to some other information sources so that the heuristic search is guided about other alternatives.

Appendix A

Some Mathematical Definitions

The definitions A.1 to A.5 in this appendix are drawn from [SS02].

Definition A.1. (*Real vector space*) A set $\mathcal{H}$ is called a *vector* space (or *linear* space) over $\mathbb{R}$ if addition and scalar multiplication are defined, and satisfy for all $\boldsymbol{x}, \boldsymbol{x'}, \boldsymbol{x''} \in \mathcal{H}$, and $\lambda, \lambda' \in \mathbb{R}$

$$\begin{aligned}
&\boldsymbol{x} + (\boldsymbol{x'} + \boldsymbol{x''}) = (\boldsymbol{x} + \boldsymbol{x'}) + \boldsymbol{x''},\\
&\boldsymbol{x} + \boldsymbol{x'} = \boldsymbol{x'} + \boldsymbol{x} \in \mathcal{H},\\
&\mathbf{0} \in \mathcal{H}, \boldsymbol{x} + \mathbf{0} = \boldsymbol{x},\\
&-\boldsymbol{x} \in \mathcal{H}, -\boldsymbol{x} + \boldsymbol{x} = \mathbf{0},\\
&\lambda\boldsymbol{x} \in \mathcal{H},\\
&1\boldsymbol{x} = \boldsymbol{x},\\
&\lambda(\lambda')\boldsymbol{x}) = (\lambda\lambda')\boldsymbol{x},\\
&\lambda(\boldsymbol{x} + \boldsymbol{x'}) = \lambda\boldsymbol{x} + \lambda\boldsymbol{x'},\\
&(\lambda + \lambda')\boldsymbol{x} = \lambda\boldsymbol{x} + \lambda'\boldsymbol{x}.
\end{aligned}$$

Definition A.2. (*Linear combination of vectors*) Given a set of vectors $\boldsymbol{x}_i \in \mathcal{H}$ and a set of scalars $\lambda_i \in \mathbb{R}$, then the *linear combination* of those vectors with those scalars as coefficients is

$$\sum_{i=1}^{m} \lambda_i \boldsymbol{x}_i.$$

Definition A.3. (*Basis*) A set of vectors $\boldsymbol{x}_i$ that allows us to uniquely write each element of $\mathcal{H}$ as a linear combination is called a *basis* of $\mathcal{H}$. For the uniqueness to hold, the vectors have to be *linearly independent*, i.e., none of the vectors $\boldsymbol{x}_i$ can be written as a linear combinations of the other vectors from the set. E.g., the *canonical basis* of $\mathbb{R}^N$ is $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_N\}$, where for $j = 1, \ldots, N, [e_j]_i = \delta_{ij}$. And δ_{ij} is the Kronecker symbol defined as

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j,\\ 0 & \text{otherwise.} \end{cases}$$

Definition A.4. (*Inner product*) An *inner product* (dot product) on a vector space $\mathcal{H}$ is a symmetric bilinear function

$$\langle ., . \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{R} \qquad (\boldsymbol{x}, \boldsymbol{x'}) \mapsto \langle \boldsymbol{x}, \boldsymbol{x'} \rangle,$$

that is *strictly positive definite*, i. e., it has the property that for all $\boldsymbol{x} \in \mathcal{H}, \langle \boldsymbol{x}, \boldsymbol{x'} \rangle \geq 0$ with equality only for $\boldsymbol{x} = \mathbf{0}$.

Definition A.5. (*Norm*) Any dot product defines the corresponding *norm* via

$$\|x\| := \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}.$$

Definition A.6. (*Signum function*) The *signum function* is defined as

$$sgn(x) = \begin{cases} -1 & : x < 0, \\ 0 & : x = 0, \\ 1 & : x > 0. \end{cases}$$

Definition A.7. (*Infimum*) The *infimum*, or the *greatest lower bound*, of a subset of some set is the greatest element, not necessarily in the subset, that is less than or equal to all other elements of the subset. In analysis, the infimum of a subset S of real numbers is denoted by $\inf(S)$ and is defined to be the biggest real number that is smaller than or equal to every number in S. It is defined that if no such number exists (because S is not bounded below), $\inf(S) = -\infty$ and if S is empty, $\inf(S) = \infty$.

Definition A.8. (*Almost cyclic sequence*) $\{i(\nu)\}_{\nu=0}^{\infty}$ is *almost cyclic* on $I = \{1, 2, \ldots, m\}$ if $i(\nu) \in I$ for all $\nu \geq 0$, and there exists an integer $C \geq m$ such that for all $\nu \geq 0$, $I \subseteq \{i(\nu+1), i(\nu+2), \ldots, i(\nu+C)\}$ [CZ97].

Definition A.9. (*Hamming loss*) The *Hamming loss* is a standard loss function which measures the number of places that the hypothesized output $\boldsymbol{y'}$ differs from the true output $\boldsymbol{y}$.

Appendix B

Optimization Theory Fundamentals

This appendix is based on [CST03, SS02].

Definition B.1. (*Convex set*) A set $\Omega \subseteq \mathbb{R}^n$ is called *convex* if, $\forall \boldsymbol{w}, \boldsymbol{u} \in \Omega$, and for any $\theta \in (0,1)$, the point $(\theta \boldsymbol{w} + (1-\theta)\boldsymbol{u}) \in \Omega$.

Definition B.2. (*Convex function*) A real-valued function $f(\boldsymbol{w})$ is called *convex* for $\boldsymbol{w} \in \mathbb{R}^n$, and for any $\theta \in (0,1)$, if

$$f(\theta \boldsymbol{w} + (1-\theta)\boldsymbol{u}) \leq \theta f(\boldsymbol{w}) + (1-\theta) f(\boldsymbol{u}).$$

If a strict inequality holds, the function is said to be *strictly convex*.

Definition B.3. (*Affine function*) An *affine function* is one that can be expressed in the form

$$f(\boldsymbol{w}) = \mathbf{A}\boldsymbol{w} + \boldsymbol{b},$$

for some matrix $\mathbf{A}$ and vector $\boldsymbol{b}$. Affine functions are convex.

Theorem B.1. *(*Fermat*) A necessary condition for $\boldsymbol{w}^*$ to be a minimum of $f(\boldsymbol{w})$, $f \in C^1$, is $\frac{\partial f(\boldsymbol{w}^*)}{\partial \boldsymbol{w}} = \mathbf{0}$. This condition, together with convexity of f, is also a sufficient condition.*

Definition B.4. (*Lagrangian function*) Given an optimization problem with domain $\Omega \subseteq \mathbb{R}^n$,

$$\begin{array}{lll} \text{minimize} & f(\boldsymbol{w}), & \boldsymbol{w} \in \Omega \\ \text{subject to} & g_i(\boldsymbol{w}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\boldsymbol{w}) = 0, & i = 1, \ldots, m, \end{array}$$

the *Lagrangian function* is defined as

$$L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\boldsymbol{w}) + \sum_{i=1}^{k} \alpha_i g_i(\boldsymbol{w}) + \sum_{i=1}^{m} \beta_i h_i(\boldsymbol{w}) \tag{B.1}$$

$$= f(\boldsymbol{w}) + \boldsymbol{\alpha}' \mathbf{g}(\boldsymbol{w}) + \boldsymbol{\beta}' \mathbf{h}(\boldsymbol{w}), \tag{B.2}$$

where the coefficients α_i and β_i are called *Lagrange multipliers*.

Theorem B.2. *(*Lagrange*) A necessary condition for a normal point $\boldsymbol{w}^*$ to be a minimum of $f(\boldsymbol{w})$ subject to $h_i(\boldsymbol{w}) = 0, i = 1, \ldots, m$, with $f, h_i \in C^1$, is*

$$\frac{\partial L(\boldsymbol{w}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{w}} = \mathbf{0},$$
$$\frac{\partial L(\boldsymbol{w}^*, \boldsymbol{\beta}^*)}{\partial \beta} = 0,$$

for some values $\boldsymbol{\beta}^*$. The above conditions are also sufficient provided that $L(\boldsymbol{w}, \boldsymbol{\beta}^*)$ is a convex function of $\boldsymbol{w}$.

Definition B.5. (*Lagrangian dual problem*) The *Lagrangian dual problem* of the primal problem of Definition 2.2 in Section 2.1.2 is the following problem:

$$\begin{aligned} \text{maximize} \quad & \theta(\boldsymbol{\alpha}, \boldsymbol{\beta}), \\ \text{subject to} \quad & \boldsymbol{\alpha} \geq 0, \end{aligned}$$

where $\theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_{\boldsymbol{w} \in \Omega} L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$.

Theorem B.3. *(*Kuhn-Tucker*) Given an optimization problem with convex domain $\Omega \subseteq \mathbb{R}^n$,*

$$\begin{aligned} \textit{minimize} \quad & f(\boldsymbol{w}), && \boldsymbol{w} \in \Omega \\ \textit{subject to} \quad & g_i(\boldsymbol{w}) \leq 0, && i = 1, \ldots, k, \\ & h_i(\boldsymbol{w}) = 0, && i = 1, \ldots, m, \end{aligned}$$

with $f \in C^1$ convex and g_i, h_i affine, necessary and sufficient conditions for a normal point $\boldsymbol{w}^*$ to be an optimum are the existence of $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ such that

$$\begin{aligned} \frac{\partial L(\boldsymbol{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{w}} &= \mathbf{0}, \\ \frac{\partial L(\boldsymbol{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \beta} &= 0, \\ \alpha_i^* g_i(\boldsymbol{w}^*) &= 0, i = 1, \ldots, k, \\ g_i(\boldsymbol{w}^*) &\leq 0, i = 1, \ldots, k, \\ \alpha_i^* &\geq 0, i = 1, \ldots, k. \end{aligned}$$

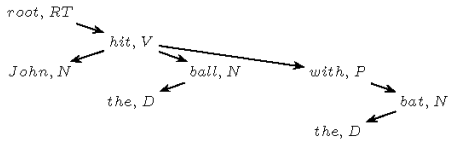
The third relation is known as *Karush-Kuhn-Tucker* (KKT) complimentary condition.

The solution point in the above theorems can be in one of two positions with respect to an inequality constraint, either in the interior of the feasible region, with the constraint inactive, or on the boundary defined by that constraint with the constraint active. In the first case, the conditions for the optimality for that constraint are given by Fermat's theorem, so the α_i need to be zero. In the second case, one can use Lagrange's theorem with a non-zero α_i. So the KKT conditions say that either a constraint is active, meaning $g_i(\boldsymbol{w}^*) = 0$, or the corresponding multiplier satisfies $\alpha_i^* = 0$. This is summarized in the equation $g_i(\boldsymbol{w}^*)\alpha_i^* = 0$.

Appendix C

MSTParser Resources

Here we show a concrete example of the feature representation of an edge in a dependency tree. The tree is given below and the edge of interest is the dependency between the main verb *hit* and its argument headed preposition *with*. We use simplified part-of-speech tags for illustrative purposes only.



f(hit,with)

Basic Features

p-word="hit", p-pos="V", c-word="with", c-pos="P"
p-pos="V", c-word="with", c-pos="P"
p-word="hit", c-word="with", c-pos="P"
p-word="hit", p-pos="V", c-pos="P"
p-word="hit", p-pos="V", c-word="with"
p-word="hit", c-word="with"
p-pos="V", c-pos="P"
p-word="hit", p-pos="V"
c-word="with", c-pos="P"
p-word="hit"
p-pos="V"
c-word="with"
c-pos="P"

Extended Features

p-pos="V", b-pos="D", c-pos="P"
p-pos="V", b-pos="N", c-pos="P"
p-pos="V", p-pos+1="D", c-pos-1="N", c-pos="P"
p-pos="V", c-pos-1="N", c-pos="P"
p-pos="V", p-pos+1="D", c-pos="P"
p-pos-1="N", p-pos="V", c-pos-1="N", c-pos="P"
p-pos="V", c-pos-1="N", c-pos="P"
p-pos-1="N", p-pos="V", c-pos="P"
p-pos="V", p-pos+1="D", c-pos="P", c-pos+1="D"
p-pos="V", c-pos="P", c-pos+1="D"
p-pos="V", p-pos+1="D", c-pos="P"
p-pos-1="N", p-pos="V", c-pos="P", c-pos+1="D"
p-pos="V", c-pos="P", c-pos+1="D"
p-pos-1="N", p-pos="V", c-pos="P"

Note that since *hit* and *with* are not longer than 5 characters we do not have any additional 5-gram back-off features. If, however, the verb was *smashed*, we could have the feature,

p-word:5="smash", c-word="with"

along with other 5-gram back-off features.

All features are also conjoined with the direction of attachment and the distance between the words. So, in addition to the feature,

p-word="hit", c-word="with"

the system would also have the feature,

p-word="hit", c-word="with", dir=R, dist=2

to indicate that the child *with* is to the right of the parent *hit* and that they are separated by 2 words. Distances were calculated into buckets with thresholds of 1, 2, 3, 4, 5 and 10.

Figure C.1: Feature example from [McD06]. In this example, w_i-pos+1 stays for w_{i+1}-pos.

(a)

Basic Uni-Gram Features
x_i-word, x_i-pos
x_i-word
x_i-pos
x_j-word, x_j-pos
x_j-word
x_j-pos

(b)

Basic Bi-Gram Features
x_i-word, x_i-pos, x_j-word, x_j-pos
x_i-pos, x_j-word, x_j-pos
x_i-word, x_j-word, x_j-pos
x_i-word, x_i-pos, x_j-pos
x_i-word, x_i-pos, x_j-word
x_i-word, x_j-word
x_i-pos, x_j-pos

(c)

In Between POS Features
x_i-pos, b-pos, x_j-pos
Surrounding Word POS Features
x_i-pos, x_{i+1}-pos, x_{j-1}-pos, x_j-pos
x_{i-1}-pos, x_i-pos, x_{j-1}-pos, x_j-pos
x_i-pos, x_{i+1}-pos, x_j-pos, x_{j+1}-pos
x_{i-1}-pos, x_i-pos, x_j-pos, x_{j+1}-pos

(d)

Second-Order Features
x_i-pos, x_k-pos, x_j-pos
x_k-pos, x_j-pos
x_k-word, x_j-word
x_k-word, x_j-pos
x_k-pos, x_j-word

Table C.1: Features used by the MSTParser (cf.: [McD06]).

Features as $f(i, j)$ in (a), (b), and (c) or $f(i, k, j)$ in (d), where

- x_i — the head in the dependency relation
- x_j — the modifier in the dependency relation;
- x_i-word/-pos — the word/POS of the head in the dependency edge;
- x_j-word/-pos — the word/POS of the modifier;
- x_{i+1}-pos — the POS of the word to the right of the head in the sentence;
- x_{i-1}-pos — the POS of the word to the left of the head;
- x_{j+1}-pos — the POS of the word to the right of the modifier;
- x_{j-1}-pos — the POS of the word to the left of the modifier;
- b-pos — the POS of the word between the head and modifier;
- x_k-word/-pos — the word/POS of the middle modifier (cf. Section 3.2).

- **Edge Features:** Word/pre-suffix/POS feature identity of the head and the modifier (suffix length 2 and 3).
  Same for the morphological feature identity (M).
  Does the head and its modifier share a prefix/suffix? Attachment direction.
  Is the modifier the first/last word in the sentence?
  What morphological features do head and modifier have the same value for? (M)
- **Sibling Feattures:** Word/POS/pre-suffix feature identity of the modifiers left/right siblings in the tree?
  Same with the morphological feature identity (M).
  Do any of the modifier siblings share its POS?
- **Context Features:** POS tag of each intervening word between head and modifier.
  Do any of the words between the head and the modifier have a head other than the head?
  Are any of teh words between the head and the modifier not descedent of the head?
- **Non-Local Features:** How many modifiers does the modifier have?
  Is this the left/right-most modifier for the head?
  Is this the first modifier to the left/right of the head?
  What morphological features of the grandparent and the modifier have identical values?(M)

Figure C.2: Second-stage labeling features (cf.: [McD06]). Features labeled with (M) are only used if the data has extended morphological features.

Initialization: $C[s][s][d][c] = 0.0 \quad \forall s, d, c$
for $k : 1..n$
  for $s : 1..n$
    $t = s + k$
    if $t > n$ then break

    $\%$ *First: create incomplete items*
    $C[s][t][\leftarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(t, s))$
    $C[s][t][\rightarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(s, t))$

    $\%$ *Second: create complete items*
    $C[s][t][\leftarrow][1] = \max_{s \leq r < t} (C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0])$
    $C[s][t][\rightarrow][1] = \max_{s < r \leq t} (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1])$

  end for
end for

Figure C.3: Pseudo-code for bottom-up first-order Eisner parsing algorithm (from [McD06]).

$C[s][t][d][c]$ is a dynamic programming table to store the score of the best subtree from position s to t, with $s \leq t$. The direction d, $d \in \{\leftarrow, \rightarrow\}$, indicates whether the subtree is gathering left or right dependents and if $d = \leftarrow$ then t must be the head of the subtree and if $d = \rightarrow$ then s is the head. The comlete value c, $c \in \{0, 1\}$ means that the subtree is complete and there are no more dependents if $c = 1$, and the value $c = 0$ means that it is incomplete and needs to be completed.

Then, to find the best score for an incomplete left subtree $C[s][t][\leftarrow][0]$, one needs to find the index $s \leq r \leq t$ that gives the best score for joining two complete subtrees $C[s][r][\rightarrow][1]$ and $C[r+1][t][\leftarrow][1]$, the score of joining these two complete subtrees being the score of these subtrees plus the score of creating an edge from word x_t to word x_s. Besides, it is guaranteed to be the score of the best subtree as by enumerating over all values of r all possible combinations are considered. With a unique root at the left-hand side of the sentence, the score of the best tree for the entire sentence is $C[1][n][\rightarrow][1]$.

Chu-Liu-Edmonds(G, s) Graph $G = (V, E)$
Edge weight function $s\colon E \rightarrow \mathbb{R}$

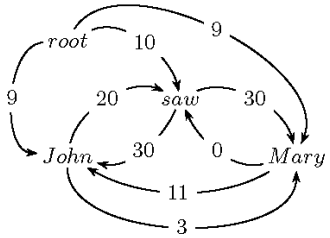
1. Let $M = \{(x^*, x) : x \in V, x^* = \arg\max_{x'} s(x', x)\}$
2. Let $G_M = (V, M)$
3. If G_M has no cycles, then it is an MST: return G_M
4. Otherwise, find a cycle C in G_M
5. Let $< G_C, c, ma >= contract(G, C, s)$
6. Let $\boldsymbol{y} =$ Chu-Liu-Edmonds(G_C, s)
7. Find vertex $x \in C$
   such that $(x', c) \in \boldsymbol{y}$ and $ma(x', c) = x$
8. Find edge $(x'', x) \in C$
9. Find all edges $(c, x''') \in \boldsymbol{y}$
10. $\boldsymbol{y} = \boldsymbol{y} \cup \{(ma(c, x'''), x''')\}_{\forall (c,x''') \in \boldsymbol{y}}$
    $\cup\, C \cup \{(x', x)\} - \{(x'', x)\}$
11. Remove all vertices and edges in $\boldsymbol{y}$ containing c
12. Return $\boldsymbol{y}$

contract$(G = (V, E), C, s)$

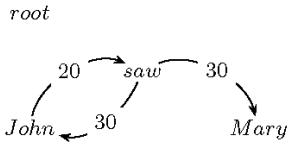
1. Let G_C be the subgraph of G excluding nodes in C
2. Add a node c to G_C representing cycle C
3. For $x \in V - C\colon \exists_{x' \in C}(x', x) \in E$
   Add edge (c, x) to G_C with
   $ma(c, x) = \arg\max_{x' \in C} s(x', x)$
   $x' = ma(c, x)$
   $s(c, x) = s(x', x)$
4. For $x \in V - C\colon \exists_{x' \in C}(x, x') \in E$
   Add edge (x, c) to G_C with
   $ma(x, c) = \arg\max_{x' \in C} [s(x, x') - s(a(x'), x')]$
   $x' = ma(x, c)$
   $s(x, c) = [s(x, x') - s(a(x'), x') + s(C)]$
   where $a(v)$ is the predecessor of v in C
   and $s(C) = \sum_{v \in C} s(a(v), v)$
5. Return $< G_C, c, ma >$

Figure C.4: Chu-Liu-Edmonds algorithm for finding maximum spanning trees in directed graphs (from [McD06]).

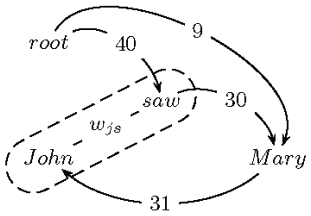
We illustrate here the application of the Chu-Liu-Edmonds algorithm to dependency parsing on the simple example $\boldsymbol{x}$ = *John saw Mary*. We assume that we know $\mathbf{w}$. The directed graph representation G_x of sentence $\boldsymbol{x}$ is



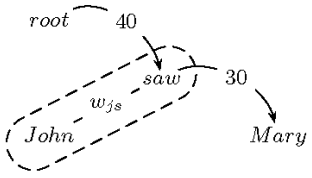
The first step of the algorithm is to find, for each word, the highest scoring incoming edge



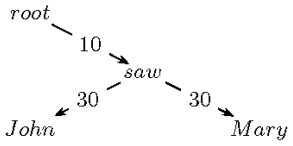
If the result were a tree, it would have to be the maximum spanning tree. However, in this case we have a cycle, so we will contract it into a single node and recalculate edge weights according to Figure 3.



The new vertex w_{js} represents the contraction of vertices *John* and *saw*. The edge from w_{js} to *Mary* is 30 since that is the highest scoring edge from any vertex in w_{js}. The edge from *root* into w_{js} is set to 40 since this represents the score of the best spanning tree originating from *root* and including only the vertices in w_{js}. The same leads to the edge from *Mary* to w_{js}. The fundamental property of the Chu-Liu-Edmonds algorithm is that a MST in this graph can be transformed into an MST in the original graph. Thus, we recursively call the algorithm on this graph. Note that we need to keep track of the real endpoints of the edges into and out of w_{js} for reconstruction later. Running the algorithm, we must find the best incoming edge to all words,



This is a tree and thus the MST of this graph. We now need to go up a level and reconstruct the graph. The edge from w_{js} to *Mary* originally was from the word *saw*, so we include that edge. Furthermore, the edge from *root* to w_{js} represented a tree from *root* to *saw* to *John*, so we include all those edges to get the MST,



This is obviously the MST for this graph.

Figure C.5: Chu-Liu-Edmonds algorithm example (from [McD06]).

Initialization: $C[s][s][d][c] = 0.0 \quad \forall s, d, c$
for $k : 1..n$
  for $s : 1..n$
    $t = s + k$
    if $t > n$ then break

    % *Create Sibling Items*
    $C[s][t][\text{-}][2] = \max_{s \leq r < t} \{C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1]\}$

    % *First Case: head picks up first modifier*
    $C[s][t][\leftarrow][0] = C[s][t-1][\rightarrow][1] + C[t-1][t][\leftarrow][1] + s(t, \text{-}, s)$
    $C[s][t][\rightarrow][0] = C[s][s][\rightarrow][1] + C[s+1][t][\leftarrow][1] + s(s, \text{-}, t))$

    % *Second Case: head picks up a pair of modifiers (through a sibling item)*
    $C[s][t][\leftarrow][0] = \max\{C[s][t][\leftarrow][0], \max_{s \leq r < t} \{C[s][r][\text{-}][2] + C[r][t][\leftarrow][0] + s(t, r, s)\}\}$
    $C[s][t][\rightarrow][0] = \max\{C[s][t][\rightarrow][0], \max_{s < r \leq t} \{C[s][r][\rightarrow][0] + C[r][t][\text{-}][2] + s(s, r, t)\}\}$

    % *Ceate complete items*
    $C[s][t][\leftarrow][1] = \max_{s \leq r < t} \{C[s][r][\rightarrow][0] + C[r+1][t][\leftarrow][0] + s(t, s)\}$
    $C[s][t][\rightarrow][1] = \max_{s \leq r < t} \{C[s][r][\rightarrow][0] + C[r+1][t][\leftarrow][0] + s(s, t)\}$

  end for
end for

Figure C.6: Pseudo-code for bottom-up second-order Eisner parsing algorithm (from [McD06]).

$C[s][t][d][c]$ is a dynamic programming table to store the score of the best subtree from position s to position t, $s \leq t$, with direction d and complete value c (see C.3). In the second-order case, $c \in \{0, 1, 2\}$ with $c = 1$ to represent that a subtree is complete (has no more dependents), $c = 0$ to indicate an incomplete tree (that has to be completed), and $c = 2$ to mark sibling subtrees. As sibling subtrees have no inherent direction, it is assumed that for $c = 2$ $d =$ null, represented by $(-)$ in the listing.

2-order-non-proj-approx$(\boldsymbol{x}, s)$
Sentence $\boldsymbol{x} = x_0 \dots x_n, x_0 = root$
Weight function $s : (i, k, j) \rightarrow \mathbb{R}$

1. Let $\boldsymbol{y} =$ **2-order-proj**$(\boldsymbol{x}, s)$
2. while $true$
3. $\quad m = -\infty, c = -1, p = -1$
4. $\quad$ for $j : 1 \cdots n$
5. $\quad\quad$ for $i : 0 \cdots n$
6. $\quad\quad\quad \boldsymbol{y'} = \boldsymbol{y}[i \rightarrow j]$
7. $\quad\quad\quad$ if $\neg$tree$(\boldsymbol{y'})$ or $\exists k : (i, k, j) \in \boldsymbol{y}$ continue
8. $\quad\quad\quad \delta = s(\boldsymbol{x}, \boldsymbol{y'}) - s(\boldsymbol{x}, \boldsymbol{y})$
9. $\quad\quad\quad$ if $\delta > m$
10. $\quad\quad\quad\quad m = \delta, c = j, p = i$
11. $\quad\quad$ end for
12. $\quad$ end for
13. $\quad$ if $m > 0$
14. $\quad\quad \boldsymbol{y} = \boldsymbol{y}[p \rightarrow c]$
15. $\quad$ else *return* $\boldsymbol{y}$
16. end while

Figure C.7: Second-order non-projective approximate algorithm (from [McD06]).

In line 1 of the above listing, $\boldsymbol{y}$ is set to the highest scoring second-order projective tree that can be found with the Eisner algorithm (Figure C.6). The loop between lines 2 – 16 exits only when no further score improvement is possible. Within each iteration, the single highest-scoring change in dependency graph of $\boldsymbol{y}$ that does not break the tree constraint is searched for. The test $\text{tree}(\boldsymbol{y})$ is true if and only if the dependency graph $\boldsymbol{y}$ satisfies the tree constraint. The nested loop in lines 4 and 5 enumerates all (i, j) pairs and line 6 sets $\boldsymbol{y'}$ to the dependency graph identical to $\boldsymbol{y}$ except that x_j's head is x_i instead of what it was in $\boldsymbol{y}$. Line 7 checks the validity of that change and line 8 computes the score change for the new graph. If this change is greater than the previous best change, lines 9 – 10 record that a new tree was created. After considering all valid edge changes to the tree, the algorithm checks that the new tree does have a higher score and if this is the case, the change is saved and the loop is re-entered. Otherwise, the algorithm ends since no single edge change can improve the score.

Appendix D

WCDG Resources

Label	Function	Label	Function
<empty>	used for punctuation	**OBJA2**	second direct object
ADV	adverbial modification	**OBJD**	indirect object
APP	apposition	**OBJG**	genitive object
ATTR	prenominal attribute	**OBJC**	clausal object
AUX	auxiliary phrases	**OBJI**	infinitive object
AVZ	split verb prefixes	**OBJP**	prepositional object
CJ	co-ordinated element	**PAR**	parenthetic matrix clause
DET	determiner	**PART**	discontinuous morphemes
ETH	ethical dative	**PN**	PP kernel
EXPL	expletive pronoun	**PP**	prepositional phrase
GMOD	possesive modifications	**PRED**	predicate
GRAD	nominal degree expression	**REL**	relative clause
KOM	comparison	**S**	main clause
KON	co-ordinating conjunction	**SUBJ**	surface nominal subject
KONJ	subordinating conjunction	**SUBJC**	subject clause
NEB	modal subclause	**VOK**	vocative
NP2	stranded NP in co-ordination	**ZEIT**	nominal time expression
OBJA	direct object		

Table D.1: Dependency labels used in the WCDG grammar of German.

$A :=$ the set of level of analysis
$W :=$ the set of all lexical readings of words in the sentence
$L :=$ the set defined dependency labels
$E := A \times W \times W \times L =$ the base of set dependency edges
$D := A \times W =$ the set of domains $d_{a,w}$ of all constraint variables
$B := \varnothing =$ the best analysis found
$C := \varnothing =$ the current analysis

{ Create the search space }
for $e \in E$
  if eval$(e) > 0$
  then $d_{a,w} := d_{a,w} \cup \{e\}$

{ Build initial analysis }
for $d_(a, w) \in D$
  $e_0 = \arg\max_{e \in d_{a,w}}$ score$(C \cup \{e\})$
  $C := C \cup \{e_0\}$
$B := C$
$T := \varnothing =$ tabu set of conflicts removed so far
$U := \varnothing =$ set of removable conflicts
$i :=$ the penalty threshhold above which conflicts are ignored
$n := 0$

{ Remove conflicts }
while $\exists c \in$ eval$(C) \backslash U$: penalty$(c) > i$
  and no interruption occurred

  { Determine which conflict to resolve }
  $c_n = \arg\max_{c \in \text{eval}(C) \backslash U}$ penalty(c)
  $T = T \cup \{c\}$

  { Find the best resolution set }
  $R_n := \arg\max_{R \in \times \text{domains}(c_n)}$ score(replace(C, R))
    where replace(C, R) does not cause any $c \in T$
    and $|R \backslash C| \leq 2$

  if no R_n can be found
    { Consider c_0 unremovable }
    $n := 0, C := B, T := \varnothing, U := U \cup \{c_0\}$
  else
    { Take a step }
    $n := n + 1, C :=$ replace(C, R_n)
    if score$(C) >$ score(B)
      $n := 0, B := C, T := \varnothing, U := U \cap$ eval(C)
return B

Figure D.1: Basic algorithm for heuristic transformational search (from [Fot06]).

Appendix E

Details of Experiment Results

```
load MST.cdg
set timelimit 600000
set cache off
set debug on
set edges off
set profile on
set progress on
set showdeleted on
set usenonspec off

newnet SENTENCE
frobbing method=dynamic execute=q
renewnet
frobbing method=combined execute=v,zSENTENCE.mst,q
quit
```

Figure E.1: A sample WCDG driver file.

Corpus	Length	Instances	Accuracy, % structural	labeled
Grundgesetz:	1 – 10	409	97.0	93.8
	11 – 20	327	93.9	90.5
	21 – 30	221	91.8	89.3
	31 – 40	95	89.4	86.8
	≥ 40	102	86.7	84.0
overall	18.4	$1,154$	91.1	88.2
Genesis:	1 – 10	1106	96.5	90.8
	11 – 20	873	92.5	87.3
	21 – 30	456	90.6	85.9
	31 – 40	169	89.2	84.5
	≥ 40	105	88.4	83.9
overall	15.9	$2,709$	91.7	86.7
wyvern:	1 – 10	3905	95.3	91.2
	11 – 20	3966	93.6	89.2
	21 – 30	1299	91.7	87.5
	31 – 40	295	89.8	85.0
	≥ 40	82	88.3	84.4
overall	13.8	$9,547$	93.0	88.7
EU:	1 – 10	763	98.7	97.1
	11 – 20	606	95.0	92.7
	21 – 30	462	93.0	90.7
	31 – 40	330	91.6	89.0
	≥ 40	403	89.5	87.3
overall	23.9	$2,564$	92.0	89.7
azure:	1 – 10	4186	96.1	91.7
	11 – 20	4487	93.8	89.4
	21 – 30	1580	92.0	87.8
	31 – 40	371	90.1	86.0
	≥ 40	82	88.2	84.4
overall	14.1	$10,706$	93.3	89.0

Table E.1: Testing the `heiseticker` model on other corpora (including punctuation).

			Accuracy, %	
Corpus	Length	Instances	structural	labeled
Grundgesetz:	1 – 10	409	96.6	92.9
	11 – 20	327	93.2	89.5
	21 – 30	221	91.0	88.2
	31 – 40	95	88.3	85.4
	≥ 40	102	85.2	82.3
overall	18.4	$1,154$	90.0	86.9
Genesis:	1 – 10	1106	95.8	88.8
	11 – 20	873	91.3	85.2
	21 – 30	456	89.1	83.7
	31 – 40	169	87.4	82.1
	≥ 40	105	86.6	81.4
overall	15.9	$2,709$	90.3	84.5
wyvern:	1 – 10	3905	93.7	88.0
	11 – 20	3966	92.1	86.6
	21 – 30	1299	90.1	85.1
	31 – 40	295	87.9	82.2
	≥ 40	82	86.0	81.3
overall	13.8	$9,547$	91.4	86.0
EU:	1 – 10	763	98.5	96.7
	11 – 20	606	94.5	92.0
	21 – 30	462	92.4	89.9
	31 – 40	330	90.8	88.0
	≥ 40	403	88.5	86.1
overall	23.9	$2,564$	91.2	88.7
azure:	1 – 10	4186	94.7	88.9
	11 – 20	4487	92.4	87.1
	21 – 30	1580	90.5	85.6
	31 – 40	371	88.3	83.4
	≥ 40	82	86.1	81.6
overall	14.1	$10,706$	91.8	86.5

Table E.2: Testing the `heiseticker` model on other corpora (excluding punctuation).

			Accuracy, %	
Corpus	Length	Instances	structural	labeled
Grundgesetz:	1 – 10	409	96.9	93.5
	11 – 20	327	93.5	90.4
	21 – 30	221	91.4	88.8
	31 – 40	95	89.3	87.0
	≥ 40	102	86.0	83.7
overall	18.4	$1,154$	90.7	88.0
Genesis:	1 – 10	1106	96.9	91.8
	11 – 20	873	92.6	87.7
	21 – 30	456	90.7	86.2
	31 – 40	169	89.7	85.1
	≥ 40	105	88.9	84.1
overall	15.9	$2,709$	92.0	87.2
wyvern:	1 – 10	3905	95.4	91.8
	11 – 20	3966	93.4	89.4
	21 – 30	1299	91.6	87.5
	31 – 40	295	89.3	84.8
	≥ 40	82	87.6	83.4
overall	13.8	$9,547$	92.9	88.9
EU:	1 – 10	763	94.7	93.2
	11 – 20	606	94.3	92.0
	21 – 30	462	92.3	90.0
	31 – 40	330	91.0	88.7
	≥ 40	403	88.3	86.4
overall	23.9	$2,564$	90.9	88.8
azure:	1 – 10	4186	96.1	92.1
	11 – 20	4487	93.7	89.6
	21 – 30	1580	91.7	87.6
	31 – 40	371	90.5	86.5
	≥ 40	82	87.7	84.1
overall	14.1	$10,706$	93.2	89.2

Table E.3: Testing the NEGRA model on other corpora (including punctuation).

			Accuracy, %	
Corpus	Length	Instances	structural	labeled
Grundgesetz:	1 – 10	409	96.5	92.6
	11 – 20	327	92.8	89.4
	21 – 30	221	90.4	87.6
	31 – 40	95	88.2	85.6
	≥ 40	102	84.5	82.0
overall	18.4	$1,154$	89.6	86.7
Genesis:	1 – 10	1106	96.2	90.0
	11 – 20	873	91.4	85.6
	21 – 30	456	89.1	84.0
	31 – 40	169	88.0	82.7
	≥ 40	105	87.2	81.7
overall	15.9	$2,709$	90.6	85.0
wyvern:	1 – 10	3905	93.7	88.8
	11 – 20	3966	92.0	87.0
	21 – 30	1299	90.0	85.1
	31 – 40	295	87.3	81.9
	≥ 40	82	85.2	80.2
overall	13.8	$9,547$	91.2	86.2
EU:	1 – 10	763	94.0	92.3
	11 – 20	606	93.7	91.2
	21 – 30	462	91.6	89.1
	31 – 40	330	90.2	87.6
	≥ 40	403	87.1	85.0
overall	23.9	$2,564$	90.0	87.7
azure:	1 – 10	4186	94.7	89.4
	11 – 20	4487	92.4	87.4
	21 – 30	1580	90.2	85.3
	31 – 40	371	88.9	84.1
	≥ 40	82	85.5	81.2
overall	14.1	$10,706$	91.7	86.7

Table E.4: Testing the NEGRA model on other corpora (excluding punctuation).

Bibliography

[Abn96] ABNEY, S.: Statistical Methods and Linguistics. In: J. Klavans and P. Resnik, editors, *The Balancing Art: Combining Symbolic Approaches to Language*, 1996

[Als96] ALSHAWI, H.: Head automata for speech translation. In: *Proceedings of the fourth International Conference on Spoken Language Processing*, 1996

[BDH+02] BRANTS, S. ; DIPPER, S. ; HANSEN, S. ; LEZIUS, W. ; SMITH, G.: The TIGER treebank. In: *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*, 2002

[Blo62] BLOCK, H. D.: The Perceptron: A Model of Brain Functioning. I. In: *Reviews of Modern Physics* (1962)

[BMDK06] BUCHHOLZ, S. ; MARSI, E. ; DUBEY, A. ; KRYMOLOWSKI, Y.: CoNLL-X shared task on multilingual dependency parsing. In: *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, 2006

[Bra00] BRANTS, T.: TnT — A Statistical Part-of-Speech Tagger. In: *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000)*, 2000

[BVZ98] BOYKOV, Y. ; VEKSLER, O. ; ZABIH, R.: Markov random fields with efficient approximations. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1998

[CC04] CLARK, S. ; CURRAN, J. R.: The Importance of Supertagging for Wide-Coverage CCG Parsing. In: *Proceedings of the 20th International Conference on Computational Linguistics*, 2004

[Cho55] CHOMSKY, N.: *Syntactic Structures*. Mouton, 1955

[CL65] CHU, Y. J. ; LIU, T. H.: On the shortest arborescence of a directed graph. In: *Science Sinica* (1965)

[Col96] COLLINS, M.: A new statistical parser based on bigram lexical dependencies. In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, 1996

[Col02] COLLINS, M.: Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In: *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, 2002

[CR04] COLLINS, M. ; ROARK, B.: Incremental parsing with the perceptron algorithm. In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004

[Cra04] CRAMMER, Y.: *Online Learning of Complex Categorial Problems*, Hebrew University, Diss., 2004

[CST03] CHRISTIANINI, N. ; SHAWE-TAYLOR, J.: *An Introduction to Support Vector Machines and other kernel-based methods*. Cambridge University Press, 2003

[CZ97] CENSOR, Y. ; ZENIOS, S.: *Parallel Optimization: Theory, Algorithms, and Applications*. Oxford University Press, 1997

[D'E59] D'ESOPO, D. A.: A convex programming procedure. In: *Naval Research Logistics Quarterly* (1959)

[DFM04] DAUM, M. ; FOTH, K. ; MENZEL, W.: Automatic transformation of phrase structure treebanks to dependency trees. In: *Proceedings of the 4th International Conference on Language Resources and Evaluation*, 2004

[DHS00] DUDA, R. O. ; HART, P.E. ; STORK, D. G.: *Pattern Classification*. 2nd edition. Wiley, 2000

[Edm67] EDMONDS, J.: Optimum branchings. In: *Journal of Research of the National Bureau of Standards* (1967)

[Eis96] EISNER, J.: Three new probabilistic models for dependency parsing: An exploration. In: *Proceedings of the International Conference on Computational Linguistics (COLING)*, 1996

[FBM06] FOTH, K. ; BY, T. ; MENZEL, W.: Guiding a constraint dependency parser with supertags. In: *Proceedings of the 21st International Conference on Computational Linguistics*, 2006

[FDM05] FOTH, K. ; DAUM, M. ; MENZEL, W.: Parsing unrestricted German text with defeasible constraints. In: H. Christiansen, P. R. Skadhauge, and J. Villadsen, editors, *Constraint Solving and Language Processing*, 2005

[FHS+05] FOTH, K. ; HAMERICH, S. ; SCHRÖDER, I. ; SCHULZ, M. ; BY, T.: *[X]CDG User Guide*. Hamburg University, 2005

[Fis52] FISHER, R.: *Contributions to Mathematical Statistics*. Wiley, 1952

[FM06a] FOTH, K. ; MENZEL, W.: The benefit of stochastic PP-attachment to a rule-based parser. In: *Proceedings of the EACL Workshop, Robust Methods in Analysis of Natural Language Data*, 2006

[FM06b] FOTH, K. ; MENZEL, W.: Hybrid Parsing: Using Models as Predictors for a Symbolic Parser. In: *Proceedings of the 21st International Conference on Computational Linguistics*, 2006

[FM06c] FOTH, K. ; MENZEL, W.: Robust parsing: More with less. In: *Proceedings of the 21st International Conference on Computational Linguistics*, 2006

[FMS00] FOTH, K. ; MENZEL, W. ; SCHRÖDER, I.: A transformation-based parsing technique with anytime properties. In: *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 2000

[Fot06] FOTH, K.: *Hybrid Methods of Natural Language Analysis*, Hamburg University, Doctoral Thesis, 2006

[FS98] FREUND, Y. ; SCHAPIRE, R. E.: Large margin classification using the perceptron algorithm. In: *Proceedings of the 11th Annual Conference on Computational Learning Theory*, 1998

[Geo03] GEORGIADIS, L.: Arborescence optimization problems solvable by Edmonds' algorithm. In: *Theoretical Computer Science* (2003)

[HB02] HENDERSON, J. ; BRILL, E.: Exploiting Diversity in Natural Language Processing: Combining Parsers. In: *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 2002

[HF02] HAGENSTRÖM, J. ; FOTH, K.: Tagging for robust parsers. In: *Proceedings of the 2nd International Workshop, Robust Methods in Analysis of Natural Language Data*, 2002

[Hil57] HILDRETH, C.: A quadratic programming procedure. In: *Naval Research Logistics Quarterly* (1957)

[Hir01] HIRAKAWA, H.: Sematic dependency analysismethod for Japanese based on optimum tree search algorithm. In: *Proceeding of the Pacific Association for Computational Linguistics*, 2001

[JH99] JAAKKOLA, T. ; HAUSSLER, D.: Exploiting generative models in discriminative classifiers. In: *Advances in Neural Information Processing Systems*, 1999

[JM00] JURAFSKY, D. ; MARTIN, J. H.: *Speech and Language Processing*. Prentice Hall, 2000

[Kel00] KELLER, F.: *Gradience in Grammar*, University of Edinburgh, PhD thesis, 2000

[Kle04] KLEIN, D.: *The Unsupervised Learning of Natural Language Structure*, Stanford University, PhD Dissertation, 2004

[LC91] LENT, A. ; CENSOR, Y.: The primal-dual algorithm as a constraint-set manipulation device. In: *Mathematical Programming* (1991)

[LS06] LONG, P. M. ; SERVEDIO, R. A.: Discriminative Learning can Succeed where Generative Learning Fails. In: *Information Processing Letters* (2006)

[Mar90] MARUYAMA, H.: Structural disambiguation with constraint propagation. In: *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL)*, 1990

[McD06] MCDONALD, R.: *Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing*, University of Pennsylvania, PhD Dissertation, 2006

[McK03] MCKAY, D. J. C.: *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003

[MCP05] MCDONALD, R. ; CRAMMER, K. ; PEREIRA, F.: Online Large-Margin Training of Dependency Parsers. In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005

[MLP06] MCDONALD, R. ; LERMAN, K. ; PEREIRA, F.: Multilingual Dependency Analysis with a Two-Stage Discriminative Parser. In: *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, 2006

[Moo05] MOORE, R.: A discriminative framework for bilingual word alignment. In: *Proceedings of the Joint Conference on Human Language Technology and Empirical Methods on Natural Language Processing (HLT/EMNLP)*, 2005

[MP69] MINSKY, M. ; PAPERT, S.: *Perceptrons — Expanded Edition: An Introduction to Computational Geometry*. MIT Press, 1969

[MP06] MCDONALD, R. ; PEREIRA, F.: Online learning of approximate dependency parsing algorithms. In: *Proceedings of the Annual Meeting of the European American Chapter of the Association for Computational Linguistics (ACL)*, 2006

[MSM93] MARCUS, M. ; SANTORINI, B. ; MARCINKIEWICZ, M.: Building a large annotated corpus of English: the Penn Treebank. In: *Computational Linguistics* (1993)

[Niv03] NIVRE, J.: An Efficient Algorithm for Projective Dependency Parsing. In: *Proceeding of the 4th International Workshop on Parsing Technologies (IWPT-2003)*, 2003

[Nov62] NOVIKOFF, A. B. J.: On convergence proofs on perceptrons. In: *Proceedings of the Symposium on the Mathematical Theory of Automata*, 1962

[Ras06] RASMUSSEN, C. E.: *Gaussian processes for machine learning*. MIT Press, 2006

[Rat99] RATNAPARKHI, A.: Learning to parse natural language with maximum entropy models. In: *Machine Learning* (1999)

[Rib04] RIBAROV, K.: *Automatic building of a dependency tree*, Chrles University, PhD thesis, 2004

[Ric94] RICHARDSON, S.: Bootstrapping Statistical Processing into a Rule-based Natural Language Parser. In: *The Balancing Act: Combining Symbolic and Statistical Approaches to Language. Proceedings of the Workshop*, 1994

[Rij79] VAN RIJSBERGEN, C. J.: *Information Retrieval*. 2nd edition. University of Glasgow, 1979

[RM90] R. MALOUF, G. van N.: Wide Coverage Parsing with Stochastic Attribute Value Grammars. In: *The 1st International Joint Conference on Natural Language Processing*

Workshop Beyond Shallow Analyses — Formalisms and statistical modeling for deep analyses, 1990

[RSCJ04] ROARK, B. ; SARACLAR, M. ; COLLINS, M. ; JOHNSON, M.: Discriminative language modeling with conditional random fields and the perceptron algorithm. In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004

[RSNM04] RAINA, R. ; SHEN, Y. ; NG, A. Y. ; MCCALLUM, A.: Classification with hybrid generative/discriminative models. In: *Neural Information Processing* (2004)

[Sch94] SCHMID, H.: Probabilistic part-of-speech tagging using decision trees. In: *International Conference on New Methods in Language Processing*, 1994

[Sch02] SCHRÖDER, I.: *Natural Language Parsing with Graded Constraints*, University of Hamburg, PhD Thesis, 2002

[SL06] SAGAE, K. ; LAVIE, A.: Parser Combinations by Reparsing. In: *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings (HLT-NAACL)*, 2006

[SPMF01] SCHRÖDER, I. ; POP, H. F. ; MENZEL, W. ; FOTH, K.: Learning grammar weights using genetic algorithms. In: *Proceedings of the Euroconference Recent Advances in Natural Language Processing*, 2001

[SS02] SCHÖLKOPF, B. ; SMOLA, A.: *Learning with Kernels*. MIT Press, 2002

[Tar77] TARJAN, R. E.: Finding optimum branchings. In: *Networks* (1977)

[Tas04] TASKAR, B.: *Learning Structured Prediction Models: A large Margin Approach*, Stanford University, PhD Dissertation, 2004

[TGK04] TASKAR, B. ; GUESTRIN, C. ; KOLLER, D.: Max-Margin Markov Networks. In: *Advances in Neural Information Processing Systems (NIPS 2003)*, 2004

[Vap00] VAPNIK, V.: *The Nature of Statistical Learning Theory*. Springer, 2000

[WH02] WANG, W. ; HARPER, M. P.: The SuperARV language model: Investigating the effectiveness of tightly integrating multiple knowledge sources. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-2002)*, 2002

[YM03] YAMADA, H. ; MATSUMOTO, Y.: Statistical dependency analysis with support vector machines. In: *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 2003

[You67] YOUNGER, D.H.: Recognition and parsing of context-free languages in time n^3. In: *Information and Control* (1967)

[Zu05] ZEMAN, D. ; ŽABOKRTSKÝ, Z.: Improving Parsing Accuracy by Combining Diverse Dependency Parsers. In: *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, 2005