A Dependency-Driven Parser for German Dependency and Constituency Representations

Johan Hall
Växjö University
Sweden
johan.hall@vxu.se

Joakim Nivre
Växjö University and
Uppsala University
Sweden
joakim.nivre@vxu.se

Abstract

We present a dependency-driven parser that parses both dependency structures and constituent structures. Constituency representations are automatically transformed into dependency representations with complex arc labels, which makes it possible to recover the constituent structure with both constituent labels and grammatical functions. We report a labeled attachment score close to 90% for dependency versions of the TIGER and TüBa-D/Z treebanks. Moreover, the parser is able to recover both constituent labels and grammatical functions with an F-Score over 75% for TüBa-D/Z and over 65% for TIGER.

1 Introduction

Is it really that difficult to parse German? Kübler et al. (2006) point out three grammatical features that could make parsing of German more difficult: finite verb placement, flexible phrase ordering and discontinuous constituents. Earlier studies by Dubey and Keller (2003) and Dubey (2005) using the Negra treebank (Skut et al., 1997) reports that lexicalization of PCFGs decrease the parsing accuracy when parsing Negra's flat constituent structures. However, Kübler et al. (2006) present a comparative study that suggests that it is not harder to parse German than for example English. By contrast, Rehbein and van Genabith (2007) study different parser evaluation metrics by simulating parser errors on two German treebanks (with different treebank annotation schemes) and they claim that the question whether German is harder to parse than English is still undecided.

This paper does not try to answer the question above, but presents a new way of parsing constituent structures that can output the whole structure with all grammatical functions. The shared task on parsing German was to parse both the constituency version and the dependency version of the two German treebanks: TIGER (Brants et al., 2002) and TüBa-D/Z (Telljohann et al., 2005). We present a dependency-driven parser that parses both dependency structures and constituent structures using an extended version of MaltParser 1.0.[1] The focus of this paper is how MaltParser parses the constituent structures with a dependency-based algorithm.

This paper is structured as follows. Section 2 briefly describes the MaltParser system, while section 3 continues with presenting the dependency parsing. Section 4 explains how a transition-based dependency-driven parser can be turned into a constituency parser. Section 5 presents the experimental evaluation and discusses the results. Finally section 6 concludes.

2 MaltParser

MaltParser is a transition-based parsing system which was one of the top performing systems on multilingual dependency parsing in the CoNLL 2006 shared task (Buchholz and Marsi, 2006; Nivre et al., 2006) and the CoNLL shared task 2007 (Nivre et al., 2007; Hall et al., 2007). The basic idea of MaltParser is to derive dependency graphs using a greedy parsing algorithm that approximates a glob-

[1] MaltParser is distributed with an open-source license and can be downloaded free of charge from following page: http://www.vxu.se/msi/users/jha/maltparser/

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

ally optimal solution by making a sequence of locally optimal choices. The system is equipped with several parsing algorithms, but we have chosen to only optimize Nivre's parsing algorithm for both the dependency track and the constituency track. Nivre's algorithm is a deterministic algorithm for building labeled projective dependency structures in linear time (Nivre, 2006). There are two essential parameters that can be varied for this algorithm. The first is the arc order and we selected the arc-eager order that attaches the right dependents to their head as soon as possible. The second is the stack initialization and we chose to use an empty stack initialization that attaches root dependents with a default root label after completing the left-to-right pass over the input.

The algorithm uses two data structures: a stack to store partially processed tokens and a queue of remaining input tokens. The arc-eager transition-system has four parser actions:

1. LEFT-ARC(r): Adds an arc labeled r from the next input token to the top token of the stack, the top token is popped from the stack because it must be complete with respect to left and right dependents at this point.

2. RIGHT-ARC(r): Adds an arc labeled r from the top token of the stack to the next input token and pushes the next input token onto the stack (because it may have dependents further to the right).

3. REDUCE: Pops the top token of the stack. This transition can be performed only if the top token has been assigned a head and is needed for popping a node that was pushed in a RIGHT-ARC(r) transition and which has since found all its right dependents.

4. SHIFT: Pushes the next input token onto the stack. This is correct when the next input token has its head to the right or should be attached to the root.

MaltParser uses history-based feature models for predicting the next parser action at nondeterministic choice points. Previously, MaltParser combined the prediction of the transition with the prediction of the arc label r into one complex prediction with one feature model. The experiments presented in this paper use another prediction strategy, which divide the prediction of the parser action into several predictions. First the transition is predicted; if the transition is SHIFT or REDUCE the nondeterminism is resolved, but if the predicted transition is RIGHT-ARC or LEFT-ARC the parser continues to predict the arc label r. This prediction strategy enables the system to have three different feature models: one for predicting the transition and two for predicting the arc label r (RIGHT-ARC and LEFT-ARC). We will see in section 4 that this change makes it more feasible to encode the inverse mapping into complex arc labels for an arbitrary constituent structure without losing any information.

All symbolic features were converted to numerical features and we use the quadratic kernel $K(x_i, x_j) = (\gamma x_i^T x_j + r)^2$ of the LIBSVM package (Chang and Lin, 2001) for mapping histories to parser actions and arc labels. All results are based on the following settings of LIBSVM: $\gamma = 0.2$ and $r = 0$ for the kernel parameters, $C = 0.5$ for the penalty parameter, and $\epsilon = 1.0$ for the termination criterion. We also split the training instances into smaller sets according to the fine-grained part-of-speech of the next input token to train separate one-versus-one multi-class LIBSVM-classifiers.

3 Dependency Parsing

Parsing sentences with dependency structures like the one in Figure 1 is straightforward using MaltParser. During training, the parser reconstructs the correct transition sequence needed to derive the gold standard dependency graph of a sentence. This involves choosing a label r for each arc, which in a pure dependency structure is an atomic symbol. For example, in Figure 1, the arc from *hat* to *Beckmeyer* is labeled SUBJ. This is handled by training a separate labeling model for RIGHT-ARC and LEFT-ARC. During parsing, the sentence is processed in the same way as during training except that the parser requests the next transition from the transition classifier. If the predicted transition is an arc transition (RIGHT-ARC or LEFT-ARC), it then asks the corresponding classifier for the arc label r.

One complication when parsing the dependency version of the two German treebanks is that they

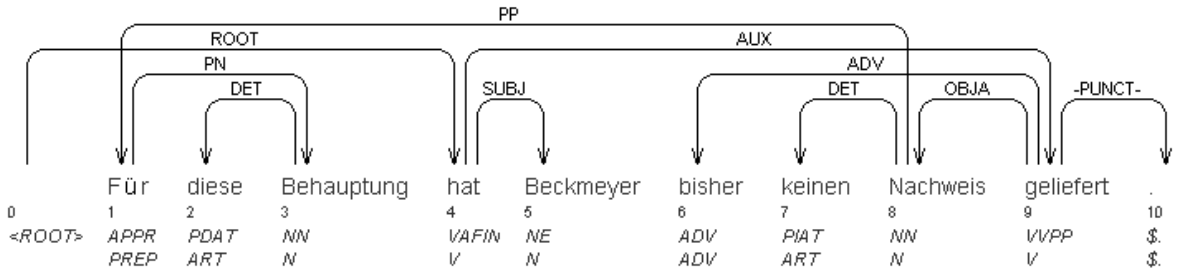

Figure 1: The sentence "For this statement has Beckmeyer until now not presented any evidence." is taken from dependency version of TüBa-D/Z treebank.

contain non-projective structures, such as the dependency graph illustrated in Figure 1. Nivre's parsing algorithm only produces projective dependency structures, and therefore we used pseudo-projective parsing for recovering non-projective structures. The training data are projectivized and information about these transformations is encoded into the arc labels to enable deprojectivizition of the parser output (Nivre and Nilsson, 2005).

4 Constituency Parsing

This section explains how a transition-based dependency parser can be used for parsing constituent structures. The basic idea is to use the common practice of transforming a constituent structure into a dependency graph and encode the inverse mapping with complex arc labels. Note that the goal is not to create the best dependency representation of a constituent structure. Instead the main objective is to find a general method to transform constituency to dependency so that is easy to do the inverse transformation without losing any information. Moreover, another goal is to transform the constituent structures so that it is feasible for a transition-based dependency parser to induce a parser model based on the resulting dependency graphs and during parsing use this parser model to derive constituent structures with the highest accuracy possible. Hence, the transformation described below is not designed with the purpose of deriving a linguistically sound dependency graph from a constituent structure.

Our strategy for turning a dependency parser into a constituency parser can be summarized with the following steps:

1. Identify the lexical head of every constituent in the constituent structure.
2. Identify the head of every token in the dependency structure.
3. Build a labeled dependency graph that encodes the inverse mapping in the arc labels.
4. Induce a parser model based on the labeled dependency graphs.
5. Use the induced parser model to parse new sentences into dependency graphs.
6. Derive the constituent structure by performing the inverse mapping encoded in the dependency graph produced in step 5.

4.1 Identify the Heads

The first steps are basically the steps that are used to convert a constituent structure to a dependency structure. One way of doing this is to traverse the constituent structure from the root node and identify the head-child and the lexical head of all constituent nodes in a recursive depth-first search. Usually this process is governed by pre-defined head-finding rules that define the direction of the search for each distinct constituent label. Moreover, it is quite common that the head-finding rules define some kind of priority lists over which part of speech or grammatical function is the more preferable head-child.

For our experiment on German we have kept this search of the head-child and lexical head very simple. For the TIGER treebank we perform a left-to-right search to find the leftmost lexical child. If no lexical child can be found, the head-child of the

constituent will be the leftmost constituent child and the lexical head will be the lexical child of the head child recursively. For the TüBa-D/Z treebank we got higher accuracy if we varied the direction of search according to the label of the target constituent.[2] We also tried more complex and linguistically motivated head rules, but unfortunately no improvement in accuracy could be found. We want to stress that the use of more complex head rules was done late in the parser optimization process and it would not be a surprise if more careful experiments resulted in the opposite conclusion.

Given that all constituents have been assigned a lexical head it is a straightforward process to identify the head and the dependents of all input tokens. The algorithm investigates, for each input token, the containing constituent's lexical head, and if the token is not the lexical head of the constituent it takes the lexical head as its head in the dependency graph; otherwise the head will be assigned the lexical head of a higher constituent in the structure. The root of the dependency graph will be the lexical head of the root of the constituent structure.

4.2 Build a Labeled Dependency Graph

The next step builds a labeled dependency representation that encodes the inverse mapping in the arc labels of the dependency graph. Each arc label is a quadruple consisting of four sublabels (*dependency relation*, *head relations*, *constituent labels*, *attachment*). The meaning of each sublabel is following:

- The *dependency relation* is the grammatical function of the highest constituent of which the dependent is the lexical head.
- The *head relations* encode the path of function labels from the dependent to the highest constituent of which is the lexical head (with path elements separated by |).
- The *constituent labels* encode the path of constituent labels from the dependent to the highest constituent of which is the lexical head (with path elements separated by |).
- The *attachment* is a non-negative integer i that encodes the attachment level of the highest constituent of which it is the lexical head.

4.3 Encoding Example

Figure 2 illustrates the procedure of encoding the constituency representation as a dependency graph with complex arc labels for a German sentence. The constituent structure is shown above the sentence and below we can see the resulting dependency graph after the transformation. We want to stress that the resulting dependency graph is not linguistically sound, and the main purpose is to demonstrate how a constituent structure can be encoded in a dependency graph that have all information need for the inverse transformation.

For example, the constituent MF has no lexical child and therefore the head-child is the leftmost constituent NX. The lexical head of MF is the token *Beckmeyer* because it is the lexical head of NX. For the same reason the lexical head of the constituent SIMPX is the token *Für* and this token will be the head of the token *Beckmeyer*, because SIMPX dominates MF. In the dependency graph this is illustrated with an arc from the head *Für* to its dependent *Beckmeyer*.

The arc *Für* to *Beckmeyer* is labeled with a complex label (??, HD|ON, NX|MF, 2), which consists of four sublabels. The first sublabel is the grammatical function above MF and because this is missing a dummy label ?? is used instead. The sublabel HD|ON encodes a sequence of head relations from the lexical head *Beckmeyer* to MF. The constituent labels are encoded in the same way in the third sublabel NX|MF. Finally, the fourth sublabel indicates the attachment level of the constituent MF. In this case, MF should be attached to the constituent two levels up in the structure with respect to the head *Für*.[3]

The two arcs *diese* to *Behauptung* and *keinen* to *Nachweis* both have the complex arc label (HD, *, *, 0), because the tokens *Behauptung* and *Nachweis* are attached to a constituent without being a lexical head of any dominating constituent. Consequently, there are no sequences of head relations and constituent

[2] It was beneficial to make a right-to-left search for the following labels: ADJX, ADVX, DM, DP, NX, PX

[3] If the fourth sublabel had an attachment level of 1, then the constituent MF would be attached to the constituent VF instead of the constituent SIMPX.

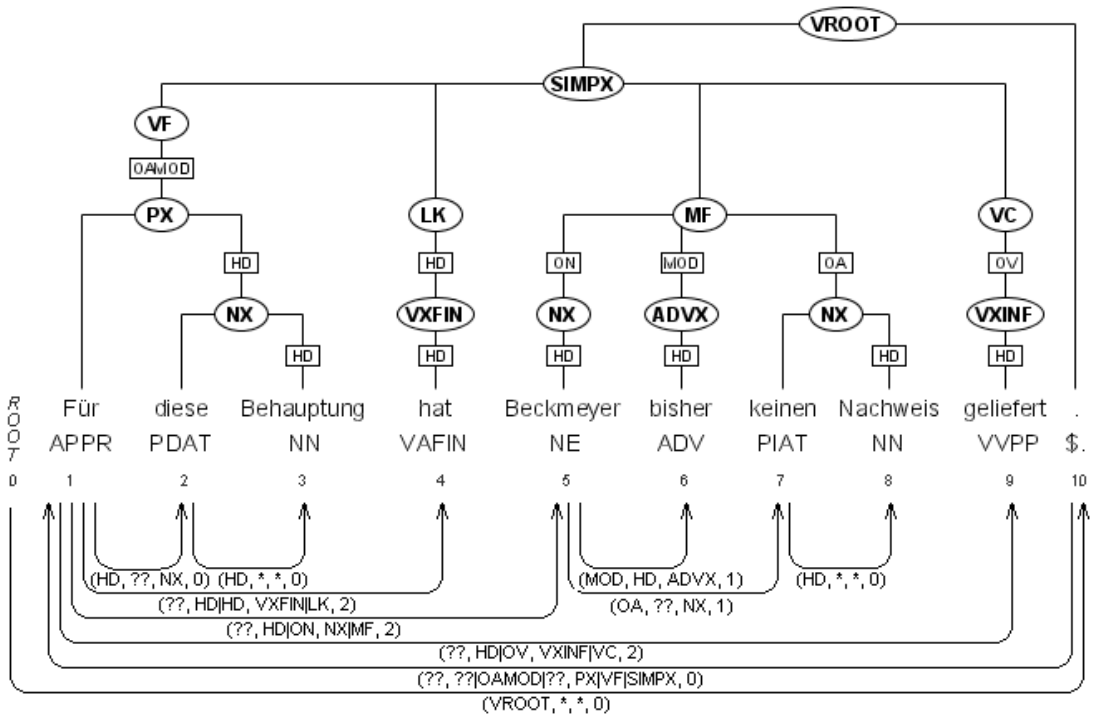

Figure 2: The sentence "For this statement has Beckmeyer until now not presented any evidence." is taken from TüBa-D/Z treebank and show the encoding of a constituent structure as a dependency graph.

labels to encode, and these are therefore marked *. The encoding of the virtual root VROOT is treated in a special way and the label VROOT is regarded as a dependency relation instead of a constituent label.

If we compare the dependency graphs in Figure 1 and Figure 2, we can see large differences. The more linguistically motivated dependency graph (LDG) in Figure 1 has a completely difference structure and different arc labels compared to the automatically generated dependency graph (ADG) in Figure 2. There are several reasons, some of which are listed here:

- Different conversions strategies: LDG is based on a conversion that sometimes leads to non-projective structures for non-local dependencies. For example, in Figure 2, the extracted PP *Für diese Behauptung* has the grammatical function OAMOD, which indicates that it is a modifier (MOD) of a direct object (OA) elsewhere in the structure (in this case *keinen Nachweis*). In LDG, this is converted to a non-projective dependency from *Nachweis* to *Für* (with the label PP). No such transformtion is attempted in ADC, which simply attaches *Für* to the lexical head of the containing constituent.

- Different head-finding rules: ADG are derived without almost no rules at all. Most likely, the conversion of LDG makes use of several linguistically sound head-finding rules. A striking difference is the root of the dependency graph, where LDG has its root at the linguistically motivated token *hat*. Whereas ADG has its root at the end of the sentence, because the leftmost lexical child of the virtual root VROOT is the punctuation.

- Different arc labels: ADG encodes the constituent structure in the complex arc labels to be able to recover the constituent structure, whereas LDG have linguistically motivated dependency relations that are not present in the constituent structure.

We believe that our simplistic approach can be further improved by using ideas from the conversion process of LDG.

4.4 Inverse Mapping

The last step of our presented strategy is to make the inverse transformation from a dependency graph to a constituent structure. This is done by a bottom-up and top-down process of the dependency graph. First we iterate over all tokens in the dependency graph and restore the sequence of constituent nodes with constituent labels and grammatical functions for each individual token using the information of the sublabels *head relations* and *constituent labels*. After this bottom-up process we have the lineage of constituents for each token where the token is the lexical head. The top-down process then traverse the dependency graph recursively from the root with pre-order depth-first search. For each token, the highest constituent of the lineage of the token is attached to its head lineage at an attachment level according to the sublabel *attachment*. Finally, the edge between the dominating constituent and the highest constituent of the lineage is labeled with a grammatical function according to the sublabel *dependency relation*.

4.5 Parsing

For the constituency versions of both TIGER and TüBa-D/Z we can recover the constituent structure without any loss of information, if we transform from constituency to dependency and back again to constituency. During parsing we predict the sublabels separately with separate feature models for RIGHT-ARC and LEFT-ARC. Moreover, the parsed constituent structure can contain discontinuous constituency because of wrong attachment levels of constituents. To overcome this problem, the structure is post-processed and the discontinuous constituents are forced down in the structure so that the parser output can be represented in a nested bracketing format.

5 Experiments

The shared task on parsing German consisted of parsing either the dependency version or the constituency version of two German treebanks, although we chose to parse both versions. This section first presents the data sets used. We continue with a brief overview of how we optimized the four different parser models. Finally, the results are discussed.

5.1 Data Sets

The prepared training and development data distributed by the organizers were based on the German TIGER (Brants et al., 2002) and TüBa-D/Z (Telljohann et al., 2005) treebanks, one dependency and one constituency version for each treebank. Both treebanks contain German newspaper text and the prepared data sets were of the same size. The development set contained 2611 sentences and the training set contained 20894 sentences. The dependency and constituency versions contained the same set of sentences.

The dependency data were formated according to the CoNLL dependency data format.[4] The LEMMA, FEATS, PHEAD and PDEPREL columns of the CoNLL format were not used at all.

The constituency data have been converted into a bracketing format similar to the Penn Treebank format. All trees are dominated by a VROOT node and all constituents are continuous. The test data consisted of sentences with gold-standard part-of-speech tags and also the gold-standard grammatical functions attached to the part-of-speech tags. Unfortunately, we were not aware of that the grammatical functions attached to the part-of-speech tags should be regarded as input to the parser and therefore our presented results are based on not using the grammatical functions attached to the part-of-speech tags as input to the parser.

We divided the development data into two sets, one set used for parser optimization (80%) and the other 20% we saved for final preparation before the release of the test data. For the final test run we trained parser models on all the data, both the training data and the development data.

5.2 Parser optimization

We ran several experiments to optimize the four different parser models. The optimization of the dependency versions was conducted in a way similar to the parser optimization of MaltParser in the CoNLL shared tasks (Nivre et al., 2006; Hall et al., 2007). A new parameter for the extended version

[4] More information about the CoNLL dependency data format can be found at: http://nextens.uvt.nl/ conll/#dataformat. Yannick Versley has done work of converting both treebanks to a dependency annotation that is similar to the Hamburg dependency format.

of MaltParser 1.0 is the prediction strategy, where we could choose between combining the prediction of the transition with the prediction of the arc label into one complex prediction or dividing the prediction of the parser action into two predictions (one model for predicting the transition and two models for predicting the arc label depending on the outcome of the transition-model). It was beneficial to use the divided predication strategy for all four data sets. In the next step we performed a feature optimization with both forward and backward selection, starting from a model extrapolated from many previous experiments on different languages. Because we chose to use the divided predication strategy this step was more complicated compared to using the combined strategy, because we needed to optimize three feature models (one transition-model and two arc-label models, one for RIGHT-ARC and one for LEFT-ARC).

The optimization of the constituency versions was even more complex because each parser model contained nine feature models (one transition-model, two models for each sublabel). Another problem for the parser optimization was the fact that we tried out new ideas and for example changed the encoding a couple of times. Due to the time constraints of the shared task it was not possible to start parser optimization all over again for every change. We also performed some late experiments with different head-finding rules to make the intermediate dependency graphs more linguistically sound, but unfortunately these experiments did not improve the parsing accuracy. We want to emphasize that the time for developing the extended version of MaltParser to handle constituency was severely limited, especially the implementation of head-finding rules, so it is very likely that head-finding rules can improve parsing accuracy after more careful testing and experiments.

5.3 Results and Discussion

The results based on the prepared test data for the dependency and constituency tracks are shown in table 1. The label attachment score (LAS) was used by the organizer for evaluating the dependency versions, that is, the proportion of tokens that are assigned the correct head and the correct arc label (punctuation included). We can see that the dependency results

	Dependency	**Constituency**		
Treebank	**LAS**	**LP**	**LR**	**LF**
TIGER	90.80	67.06	63.40	65.18
TüBa-D/Z	88.64	76.44	74.79	75.60

Table 1: The results for the extended version of MaltParser 1.0 in the shared task on parsing German dependency and constituency representations.

are close to 90% for both the treebanks, 90.80 for TIGER and 88.64 for Tüba-D/Z, which were the unchallenged best scores in the shared task. The highest score on parsing German in the CoNLL-X shared task was obtained by the system of McDonald et al. (2006) with a LAS of 87.34 based on the TIGER treebank, but we want to stress that these results are not comparable due to different data sets (and a different policy regarding the inclusion of punctuation).

The constituency versions were evaluated according to the labeled recall (LR), labeled precision (LP) and labeled F-score (LF). Labeled in this context means that both the constituent label and the grammatical function should agree with the gold-standard, but grammatical functions labeling the edge between a constituent and a token were not included in the evaluation. The labeled F-scores are 75.60 for Tüba-D/Z and 65.18 for TIGER and these results are the second best results in the shared task out of three systems. We want to emphasize that the results may not be strictly comparable because of different use of the grammatical functions attached to the parts of speech in the bracketing format. We did not use these grammatical functions as input, instead these were assigned by the parser. Our results are competitive if we compare with Kübler et al. (2006), who report 51.41 labeled F-score on the Negra treebank and 75.33 on the TüBa-D/Z treebank using the unlexicalized, markovized PCFG version of the Stanford parser.

We believe that our results for the constituency representations can be improved upon by investigating different methods for encoding the inverse mapping in the complex arc labels and performing a more careful evaluation of head-finding rules to derive a more linguistically sound dependency representation. Another interesting line of future work is to try to parse discontinuous constituents by using

a non-projective parsing algorithm like the Covington algorithm (Covington, 2001) or using pseudo-projective parsing for discontinuous constituency parsing (Nivre and Nilsson, 2005).

6 Conclusion

We have shown that a transition-based dependency-driven parser can be used for parsing German with both dependency and constituent representations. We can report state-of-the-art results for parsing the dependency versions of two German treebanks, and we have demonstrated, with promising results, how a dependency parser can parse full constituent structures by encoding the inverse mapping in complex arc labels of the dependency graph. We believe that this method can be improved by using, for example, head-finding rules.

Acknowledgments

We want to thank the treebank providers for making the data available for the shared task and the organizers for their efforts in organizing it. Thanks also to two reviewers for useful comments.

References

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER Treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories Sozopol*, pages 1–18.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164.

Chih-Chung Chang and Chih-Jen Lin. 2001. LIBSVM: A Library for Support Vector Machines.

Michael A. Covington. 2001. A Fundamental Algorithm for Dependency Parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Amit Dubey and Frank Keller. 2003. Probabilistic Parsing for German using Sister-Head Dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 96–103.

Amit Dubey. 2005. What to do when Lexicalization fails: Parsing German with Suffix Analysis and Smoothing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 314–321.

Johan Hall, Jens Nilsson, Joakim Nivre, Gülşen Eryiğit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single Malt or Blended? A Study in Multilingual Parser Optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 933–939.

Sandra Kübler, Erhard W. Hinrichs, and Wolfgang Maier. 2006. Is it Really that Difficult to Parse German. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP 2006)*, pages 111–119.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual Dependency Analysis with a Two-Stage Discriminative Parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 216–220.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled Pseudo-Projective Dependency Parsing with Support Vector Machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre. 2006. *Inductive Dependency Parsing*. Springer.

Ines Rehbein and Josef van Genabith. 2007. Treebank Annotation Schemes and Parser Evaluation for German. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007)*, pages 630–639.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An Annotation Scheme for Free Word Order Languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)*, pages 314–321.

Heike Telljohann, Erhard W. Hinrichs, Sandra Kübler, and Heike Zinsmeister. 2005. Stylebook for the Tübingen Treebank of Written German (TüBa-D/Z). Seminar für Sprachwissenschaft, Universität Tübingen, Germany.