Stacking or Supertagging for Dependency Parsing – What's the Difference?

Agnieszka Faleńska[§,◇], **Anders Björkelund**[§], **Özlem Çetinoğlu**[§] and **Wolfgang Seeker**[§]

[§]Institute for Natural Language Processing, University of Stuttgart, Germany
[◇]Institute of Computer Science, University of Wrocław, Poland
{falensaa,anders,ozlem,seeker}@ims.uni-stuttgart.de

Abstract

Supertagging was recently proposed to provide syntactic features for statistical dependency parsing, contrary to its traditional use as a disambiguation step. We conduct a broad range of controlled experiments to compare this specific application of supertagging with another method for providing syntactic features, namely stacking. We find that in this context supertagging is a form of stacking. We furthermore show that (i) a fast parser and a sequence labeler are equally beneficial in supertagging, (ii) supertagging/stacking improve parsing also in a cross-domain setting, and (iii) there are small gains when combining supertagging and stacking, but only if both methods use different tools. The important consideration is therefore not the method but rather the diversity of the tools involved.

1 Introduction

We present a systematic comparison of two methods that have been proposed to improve statistical dependency parsers: *supertagging* and *stacking*.

Supertags are labels for tokens much like POS tags but they also encode syntactic information, e.g. the head direction or the subcategorization frame. Supertagging was developed for deep grammar formalisms as the disambiguation of supertag assignment *prior to parsing* (Bangalore and Joshi, 1999; Clark and Curran, 2004; Ninomiya et al., 2006). Recently, it was presented as a method to provide syntactic information to the feature model of a statistical dependency parser. Ambati et al. (2013; 2014) provide CCG supertags to a dependency parser, whereas Ouchi et al. (2014) extract their supertag tag set from a dependency treebank (see Figure 1). In this paper, we adopt this *particular definition* and take supertagging as a way of incorporating syntactic features instead of the traditional use for disambiguation.

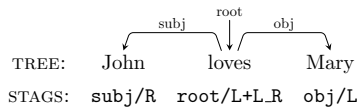

Figure 1: Supertags that are derived from dependency trees for each token. They encode the label, the head direction, and the presence of left and right dependents.

Parser stacking was introduced by Nivre and McDonald (2008) and Martins et al. (2008). In stacking, two parsers are run in sequence so that the second parser can use the output of the first parser as features, for example, whether a particular arc was already predicted by the first parser.

When supertags were first proposed by Joshi and Bangalore (1994), they called supertagging *almost parsing*, because supertags anticipate a lot of syntactic disambiguation. In stacking, the first step is running a parser, or in other words: *real parsing*. In this paper, we investigate the difference between *almost* and *real parsing* for improving a statistical dependency parser.

We conduct an extensive number of comparative experiments with two state-of-the-art dependency parsers and a state-of-the-art sequence labeler on 10 different data sets. In the first set of experiments, we use only the two parsers and compare both methods in artificial and realistic settings. In the second set of experiments, we control for the methods and compare different ways of realizing them. In the last set, we evaluate the benefit of combining both methods.

Intuitively, stacking should give higher improvements than the version of supertagging defined by Ouchi et al. (2014), since trees in stacking are more informative than supertag sequences in supertagging. However, our experiments show

Proceedings of the 14th International Conference on Parsing Technologies, pages 118–129,
Bilbao, Spain; July 22–24, 2015.

that both methods perform on par. Based on an in-depth analysis of these findings, we argue that supertagging is a form of stacking.

One apparent advantage of supertagging is the fact that one can predict supertags without a parser and thus possibly faster. However, greedy transition-based parsers are extremely fast as well. We show that the output of a CRF sequence labeler and a greedy transition-based parser are of equal usefulness when used in supertagging. This setup suggests application to large-scale (e.g. web) data. We test both methods on the English Web Treebank (Bies et al., 2012) and show that they also improve parsing in a cross-domain setting.

Our experiments on combining supertagging and stacking show small gains only when supertags and trees are predicted by different tools. Surdeanu and Manning (2010) demonstrate that diversity of algorithms is important when stacking parsers. Since supertagging is a form of stacking, this also holds for supertagging, and we argue that this is a more important factor than the choice between the two methods.

We give background on supertagging and stacking in Section 2 and describe our experimental setup in Section 3. We present our experiments in Sections 4 to 6 and conclude with Section 7.

2 Background

The term supertag originated in Joshi and Bangalore (1994) as an elementary structure associated with a lexical item. These elementary structures carry more information than POS tags, hence the name super POS tags or supertags. Within Lexicalized Tree Adjoining Grammar (LTAG) (Schabes et al., 1988) supertags correspond to trees that localize dependencies. A supertagger assigns supertags to each word of a sentence. A parser then combines these structures into a full parse (Bangalore and Joshi, 1999) that leads to simplified and faster parsing. The same approach applied to Combinatory Categorial Grammar (CCG) (Clark and Curran, 2004) and Head-Driven Phrase Structure Grammar (HPSG) (Ninomiya et al., 2006) speeds up the parser dramatically.

Foth et al. (2006) were the first to utilize supertags in a dependency parsing context by incorporating them as soft constraints into their rule-based parser (Foth et al., 2004). In LTAG, CCG, or HPSG supertags are the elementary components of the framework in question. In Foth et al. (2006), supertags are specifically designed to capture syntactic properties.

Ouchi et al. (2014) use supertags as features in a statistical dependency parser for English. Ambati et al. (2014) instead utilize CCG categories for English and Hindi. Both demonstrate significant improvements. Björkelund et al. (2014) extend the positive results to nine other languages.

Another way of exploiting one parser's output as features in another parser is stacking. Nivre and McDonald (2008) define a simple set of local features that mark whether an arc is present in the input tree. They demonstrate that stacking parsers leads to higher parsing accuracy than a non-stacked baseline. Martins et al. (2008) extend this feature set to include non-local information, e.g. information about siblings and grandparents of dependents. However, the additional non-local features provide only minor further gains over the local ones if the parser itself already uses non-local features.

Surdeanu and Manning (2010) present a study on parser stacking for English. They find that one important factor is the diversity of the parsing algorithms involved. Specifically, stacking a parser on itself does not lead to gains. This effect was also observed by Martins et al. (2008).

3 Experimental Setup

3.1 Data Sets and Preprocessing

We perform experiments on the data from the SPMRL 2014 Shared Task (Seddah et al., 2014), which consists of data sets for 9 languages (see Table 1). To these 9, we add the English Penn Treebank converted to Stanford Dependencies.[1] We use sections 2-21 for training, 24 as development set and 23 as test set.

Contrary to most previous work, we use automatically predicted preprocessing in all the parsing experiments. POS tags and morphological features are jointly assigned using MarMoT[2] (Müller et al., 2013), a state-of-the-art morphological CRF tagger. To improve tagging accuracy we integrate the analyses of language-specific morphological analyzers as additional features into MarMoT (see Table 1). We use the mate-tools[3] for lemmatiza-

[1] We use version 3.4.1 of the Stanford Parser from http://nlp.stanford.edu/software/lex-parser.shtml

[2] https://code.google.com/p/cistern/

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

tion. We annotate the training sets via 5-fold jackknifing.

ISO	Language	Morphological Analyzer
ar	Arabic	AraMorph, a re-impl. of Buckwalter (2002)
eu	Basque	Apertium (Forcada et al., 2011)
fr	French	An extension of Zhou (2007)
he	Hebrew	Analyzer from Goldberg and Elhadad (2013)
de	German	SMOR (Schmid et al., 2004)
hu	Hungarian	Magyarlanc (Zsibrita et al., 2013)
ko	Korean	HanNanum (Park et al., 2010)
pl	Polish	Morfeusz (Woliński, 2006)
sv	Swedish	Granska (Domeij et al., 2000)

Table 1: Analyzers used in the tagger.

3.2 Supertag Design

Foth et al. (2006) experiment with different tag set designs and show that richer supertags improve their parser's accuracy more. However, richer tags increase the tag set size considerably and make it more difficult to predict them automatically.

Ouchi et al. (2014) test two models for English. Model 1 includes the relative head position of a word (**hdir**), its dependency relation (**label**), and information about dependents to the left or right (**hasLdep, hasRdep**). The tag set is derived from the treebank, an example is shown in Figure 1. Model 2 additionally uses dependency relations of obligatory dependents of verbs. The difference between the two models has no impact on the performance of a parser, however. Björkelund et al. (2014) find the same effect for the same models on nine other languages.

ar	eu	fr	de	he	hu	ko	pl	sv	en
42	179	128	239	196	280	74	113	253	222

Table 2: Tag set sizes for training sets.

Based on these results we decided to use Model 1 in all of the experiments. The supertags are extracted from the respective training sets and follow the template **label/hdir+hasLdep_hasRdep**. Table 2 gives the tag set sizes for each data set.

3.3 Notation

We denote stacking and supertagging by STACK and STAG, respectively. When a tool *y* uses the output of another tool *x*, we mark this by superscript and subscript. For example, STACK^{y}_{x} means that tool *y* uses the output of tool *x* in stacking. Similarly, STAG^{y}_{x} means that tool *y* uses the supertags predicted by tool *x*. We follow Martins et al. (2008)'s terminology and call *x* the Level 0 tool and *y* the Level 1 tool.

3.4 Parsers and Feature Models

In the experiments, Level 1 tools will always be dependency parsers since we are interested in the effect of supertagging and stacking on parsing performance. We experiment with one graph-based and one transition-based parser to cover the two major paradigms in dependency parsing.

We extend the parsers' baseline feature sets in two directions: (1) to extract features for stacking, i.e., to extract features from a provided dependency tree, and (2) to extract features from a sequence of supertags. For stacking, features are taken from Nivre and McDonald (2008) and slightly adapted to our setting. For supertagging, we mirror the features from stacking to the best extent possible given the more limited information that is contained in the supertags to begin with.

We note that feature engineering can be done more elaborately both for stacking (Martins et al., 2008) and supertagging (Ouchi et al., 2014). However, since not all types of features that can be extracted necessarily carry over from one method to the other, a simpler feature set is more useful for a comparison. Moreover, both of the aforementioned papers only demonstrate minor performance gains with more elaborate features.

Transition-based Parser. We use the parser by Björkelund and Nivre (2015) as our transition-based parser. It uses the arc-standard decoding algorithm extended with a SWAP transition (Nivre, 2009) to handle non-projective structures.[4] The system applies arc transitions between the two topmost items of the stack (denoted s_0 and s_1). The lazy SWAP oracle by Nivre et al. (2009) is used during training. The parser is globally trained using beam-search and early update (Zhang and Clark, 2008). The implementation uses the passive-aggressive perceptron (Crammer et al., 2006) and a hash kernel for feature mapping following Bohnet (2010). The parser is trained for 25 iterations using a beam size of 20. We omit the definition of the baseline feature set of the transition-based parser, however it is primarily based on that of Zhang and Nivre (2011) with adaptations to the arc-standard setting.

Table 3 outlines the feature templates used for stacking and supertagging. The predicates

[4]This parser is available on the second author's website.

Stacking features	
$\text{head}_G(s_0) = s_1$	$\text{head}_G(s_1) = s_0$
$\text{hdir}_G(s_0)$	$\text{hdir}_G(s_1)$
$\text{label}_G(s_0)$	$\text{label}_G(s_1)$
$\text{hasL}_G(s_0) \oplus \text{pos}(\text{ldep}(s_0))$	$\text{hasR}_G(s_0) \oplus \text{pos}(\text{rdep}(s_0))$
$\text{hasL}_G(s_1) \oplus \text{pos}(\text{ldep}(s_1))$	$\text{hasR}_G(s_1) \oplus \text{pos}(\text{rdep}(s_1))$
Supertag features	
$\text{stag}_S(s_0)$	$\text{stag}_S(s_1)$
$\text{label}_S(s_0)$	$\text{label}_S(s_1)$
$\text{hdir}_S(s_0)$	$\text{hdir}_S(s_1)$
$\text{hasL}_S(s_0)$	$\text{hasR}_S(s_1)$
$\text{stag}_S(s_0) \oplus \text{pos}(\text{ldep}(s_0))$	$\text{stag}_S(s_0) \oplus \text{pos}(\text{rdep}(s_0))$
$\text{stag}_S(s_1) \oplus \text{pos}(\text{ldep}(s_1))$	$\text{stag}_S(s_1) \oplus \text{pos}(\text{rdep}(s_1))$
$\text{hasL}_S(s_0) \oplus \text{hdir}_S(s_1)$	$\text{hasR}_S(s_1) \oplus \text{hdir}_S(s_0)$
$\text{hasL}_S(s_0) \oplus \text{pos}(\text{ldep}(s_0))$	$\text{hasR}_S(s_1) \oplus \text{pos}(\text{rdep}(s_1))$
$\text{hasR}_S(s_0) \oplus \text{pos}(\text{rdep}(s_0))$	$\text{hasL}_S(s_1) \oplus \text{pos}(\text{ldep}(s_1))$

Table 3: Feature templates used for stacking and supertagging in the transition-based parser. $\oplus$ denotes conjunctions of basic templates. All templates are conjoined with the POS tag of the topmost stack items s_0 and s_1.

$\text{head}_X(d)$, $\text{hdir}_X(d)$, $\text{label}_X(d)$, $\text{hasL/R}_X(d)$, and $\text{stag}_X(d)$ extract the head of d, the direction of d's head, the arc label of d, whether d has left/right dependents, and the supertag according to the Level 0 prediction X. X is either a dependency graph (in stacking) or a supertag assignment (in supertagging), denoted G and S in Table 3, respectively. The predicates $\text{l/rdep}(d)$ extract the leftmost/rightmost dependent of d given the current parser state. $\text{pos}(d)$ extracts the POS tag of d, with a special placeholder if d is undefined.

The stacking features are mostly taken from Nivre and McDonald (2008) with the exception of the last two rows. These features encode whether d should have left/right dependents according to G conjoined with whether d has left/right dependents in the current configuration. We added these features because existence of left/right dependents is also encoded in the supertags. Conjoining the existence of left/right dependents according to the Level 0 predictions with the POS tag of left/right dependents in the current parser state thus encodes whether dependents were attached or not. Since the arc-standard algorithm works bottom-up, every token needs to collect all its dependents before it can be attached to its own head.

The supertag features mimic the information provided by stacking. For instance, in stacking the Level 0 predictions explicitly include whether s_0 is the head of s_1. In supertagging this is approximated by combining the direction of the head of s_1 with whether s_0 expects dependents on the left.

Stacking features	
$\text{head}_G(d) = h$	
$\text{label}_G(d)$	
$\text{head}_G(d) = h \oplus \text{label}_G(d)$	
Supertag features	
$\text{stag}_S(h)$	$\text{stag}_S(d)$
$\text{label}_S(d)$	$\text{hdir}_S(d)$
$\text{label}_S(d) \oplus \text{hdir}_S(d)$	
$\text{hasL}_S(h)$	$\text{hasR}_S(h)$
$\text{hdir}_S(d) \oplus \text{hasL}_S(h)$	$\text{hdir}_S(d) \oplus \text{hasR}_S(h)$
$\text{label}_S(d) \oplus \text{hasL}_S(h)$	$\text{label}_S(d) \oplus \text{hasR}_S(h)$
$\text{label}_S(d) \oplus \text{hdir}_S(d) \oplus \text{hasL}_S(h)$	
$\text{label}_S(d) \oplus \text{hdir}_S(d) \oplus \text{hasR}_S(h)$	

Table 4: Feature templates used for stacking and supertagging in the graph-based parser. $\oplus$ denotes conjunctions of basic templates. All templates are conjoined with the direction of that arc and with the POS tag of the head and the dependent.

Graph-based Parser. The graph-based parser we use is TurboParser,[5] which solves the parsing task by doing global inference using a dual decomposition algorithm and outputs non-projective structures natively (Martins et al., 2013).

Table 4 shows the stacking and supertagging features as we implemented them in TurboParser. They are synchronized with the features for the transition-based parser where possible. We extract these features only on first-order factors, with d and h denoting the dependent and the head, respectively. Unlike in Nivre and McDonald (2008), the features cannot access the label of the current arc during feature extraction, as it is automatically combined with the features after the extraction.

Like in the transition-based parser, supertag and stacking features are modeled to capture similar information. However, features that combine information about dependents of dependents with information about the head are not included since these would require higher-order factors.

3.5 Evaluation

We evaluate the parsing experiments using *Labeled Attachment Score* (LAS).[6] We mark statistical significance against respective baselines by † and ‡, denoting p-value < 0.05 and p-value < 0.01 respectively. Significance testing is carried out using the Wilcoxon signed-rank test. Averages and oracle experiments are not tested for significance.

[5] We use version 2.0.1 from `http://www.ark.cs.cmu.edu/TurboParser/`. We train TurboParser with MODEL-TYPE=FULL which uses third-order features.

[6] The ratio of tokens with a correct head and label to the total number of tokens in the test data.

		avg.	ar	eu	fr	de	he	hu	ko	pl	sv	en
①	BL^{GB}	84.45	84.99	82.79	83.95	88.53	79.41	83.98	86.18	84.96	79.59	90.12
②	$\text{STAG}^{\text{GB}}_{\text{TB}}$	85.15	$85.54^{\ddagger}$	$83.24^{\ddagger}$	$84.33^{\ddagger}$	$89.14^{\ddagger}$	$80.10^{\dagger}$	$85.52^{\ddagger}$	86.48	85.48	$80.63^{\ddagger}$	$90.99^{\ddagger}$
③	$\text{STACK}^{\text{GB}}_{\text{TB}}$	85.16	$85.65^{\ddagger}$	$83.32^{\ddagger}$	$84.29^{\ddagger}$	$89.15^{\ddagger}$	$80.03^{\ddagger}$	$85.46^{\ddagger}$	$86.59^{\ddagger}$	$85.52^{\dagger}$	$80.66^{\ddagger}$	$90.95^{\ddagger}$
④	BL^{TB}	84.37	85.09	81.77	83.47	87.89	79.70	85.25	85.71	84.34	79.97	90.54
⑤	$\text{STAG}^{\text{TB}}_{\text{GB}}$	85.01	$85.58^{\ddagger}$	$82.99^{\ddagger}$	$83.88^{\ddagger}$	$88.88^{\ddagger}$	80.04	85.37	$86.31^{\ddagger}$	85.03	$81.04^{\ddagger}$	$90.98^{\ddagger}$
⑥	$\text{STACK}^{\text{TB}}_{\text{GB}}$	85.08	$85.60^{\ddagger}$	$83.14^{\ddagger}$	$84.16^{\ddagger}$	$88.91^{\ddagger}$	80.30	85.38	$86.06^{\dagger}$	$85.06^{\dagger}$	$81.32^{\ddagger}$	$90.88^{\ddagger}$

Table 5: Parsing results (LAS) on test sets.

4 Comparing Supertagging and Stacking

The purpose of the following experiments is to compare supertagging and stacking and to derive some conclusions about their relationship to each other. We use one parser as the Level 0 parser and the other one as Level 1 parser. In stacking, the Level 1 parser exploits the tree produced by the Level 0 parser as additional features. In supertagging, we derive the supertag of each token from the tree that is output by the Level 0 parser. The Level 1 parser then uses these supertags as additional features. Although supertags are normally predicted with sequence labelers, using a parser on Level 0 in both cases ensures that the only difference between the two settings is the means by which the information is given to the Level 1 parser, i.e. as a tree or as a sequence of supertags. Figure 2 illustrates this setup.

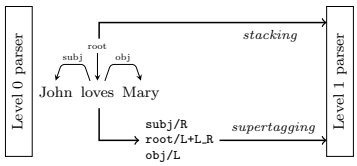

Figure 2: Setup for comparing stacking and supertagging.

The training sets are annotated with predicted dependency trees or supertags via 5-fold jackknifing. In the tables, **GB** stands for the graph-based parser and **TB** for the transition-based parser.

4.1 Supertagging and Stacking Accuracy

First of all we convince ourselves that both strategies, supertagging and stacking, indeed improve over the baseline. Table 5 gives the performance of the Level 1 parser on the test sets: In the baseline setting (**BL**) the parser is run without any additional information. STAG and STACK show the performance of the Level 1 parser when provided with supertags or a tree from the Level 0 parser.

As demonstrated by previous work, both stacking and supertagging consistently improve the parsing performance of the Level 1 parser. Moreover, both methods improve the parsing accuracies to the same extent, with the average improvements about 0.7% points absolute for both the graph-based and the transition-based parser. Almost all of the improvements are statistically significant, with a few exceptions, most notably Polish. For supertagging, our results confirm the findings by Ouchi et al. (2014) and Ambati et al. (2014). The stacking results are in line with Nivre and McDonald (2008) and Martins et al. (2008). Here, it is worth noting that even though dependency parsers have markedly advanced since 2008, the fact remains that stacking parsers improves performance.

We now continue with a more in-depth analysis to find out where the improvements are coming from. We perform the analysis on the development sets in order to not compromise our test sets. The corresponding accuracies for the development sets can be found in Tables 6 and 7 in rows ① to ③.

4.2 In-Depth Analysis

The overall improvements with supertagging and stacking are similar, but they might still come about in different ways. To investigate this, we follow McDonald and Nivre (2007) and look into accuracy distributions of comparable systems relative to sentence length and dependency length, i.e. the distance between the dependent and the head.

We present the analysis on the concatenation of all the development sets. We also looked at the corresponding plots for the individual treebanks. While the absolute numbers vary across the different data sets, the relative differences between the baseline, supertagging, and stacking models are consistent with the concatenation.

Figure 3 gives the accuracy of both parsers relative to sentence length in bins of size 10. Bin sizes are represented as grey bars.[7]

[7]Note that if there are fewer items in a bin, the curves are more sensitive to small absolute changes.

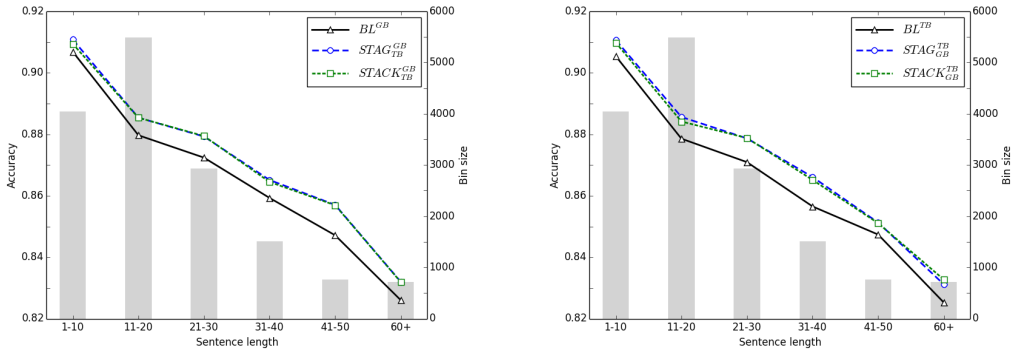

Figure 3: The accuracy of the graph-based (left) and transition-based (right) parser relative to sentence length.

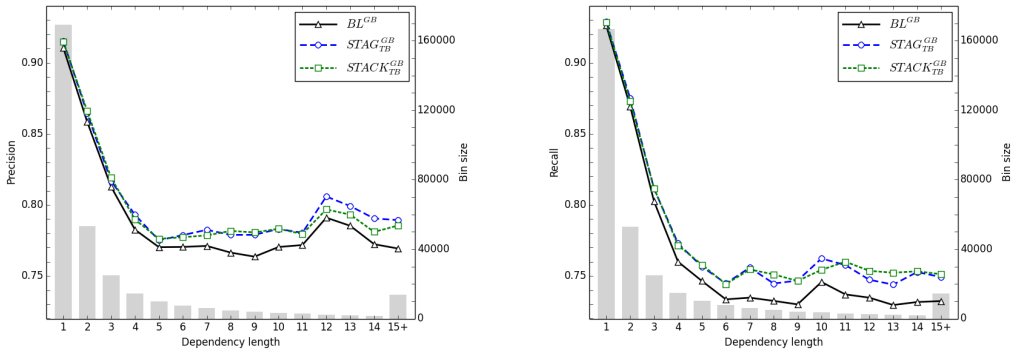

Figure 4: The dependency precision and recall of the graph-based parser relative to dependency length.

		avg.	ar	eu	fr	de	he	hu	ko	pl	sv	en
①	BL^{GB}	84.16	85.64	83.37	84.78	91.46	78.71	82.60	86.08	85.17	75.24	88.59
②	$STAG^{GB}_{TB}$	84.81	86.25‡	83.96‡	85.05†	92.10‡	79.36†	83.92‡	86.24	85.52†	76.27†	89.47‡
③	$STACK^{GB}_{TB}$	84.79	86.25‡	83.78‡	85.14‡	92.01‡	79.56‡	83.72‡	86.34†	85.29	76.44‡	89.41‡
④	$STAG^{GB}_{oracle}$	95.73	93.78	96.66	96.43	98.60	94.43	94.37	93.48	97.41	94.22	97.91
⑤	$STACK^{GB}_{oracle}$	96.43	98.67	96.87	98.38	98.90	92.62	94.21	96.46	96.55	93.33	98.34
⑥	$STAG^{GB}_{GB}$	84.44	85.83‡	83.68‡	84.91	91.64‡	79.31‡	82.87‡	86.28‡	85.18	75.89‡	88.77‡
⑦	$STACK^{GB}_{GB}$	84.23	85.69†	83.44	84.80	91.49	78.90†	82.64	86.08	85.24	75.39	88.62

Table 6: Results (LAS) for the graph-based parser for different experiments on development sets.

		avg.	ar	eu	fr	de	he	hu	ko	pl	sv	en
①	BL^{TB}	84.04	85.69	82.22	84.07	91.15	78.80	83.27	85.97	84.51	75.65	89.06
②	$STAG^{TB}_{GB}$	84.94	86.06‡	84.02‡	84.68‡	91.98‡	79.82	83.83‡	86.56‡	85.73†	77.20‡	89.48‡
③	$STACK^{TB}_{GB}$	84.86	86.19‡	83.86‡	84.55‡	91.98‡	79.76	83.87‡	86.25	85.65	77.16‡	89.30
④	$STAG^{TB}_{oracle}$	96.66	94.09	97.65	96.64	98.80	95.70	96.46	94.90	98.50	95.65	98.16
⑤	$STACK^{TB}_{oracle}$	96.83	98.80	97.08	98.62	98.65	92.96	96.14	97.27	97.21	93.18	98.42
⑥	$STAG^{TB}_{TB}$	84.16	85.77	82.49	84.05	91.44‡	78.69	83.61‡	85.92	84.84	75.78	89.01
⑦	$STACK^{TB}_{TB}$	84.12	85.73	82.24	84.29†	91.35‡	78.67	83.45†	85.86	84.76	75.66	89.22

Table 7: Results (LAS) for the transition-based parser for different experiments on development sets.

Figure 4 displays the graph-based parser's performance relative to dependency length in terms of precision and recall.[8] Precision is defined as the percentage of correct predictions among all predicted arcs of length l and recall is the percentage of correct predictions among gold standard arcs of length l.[9] In all graphs, the stacked and

[8]The corresponding curves for the transition-based parser look very similar.

[9]For precision, the bin sizes shown as grey bars are averages over all three systems, as the number of predicted arcs of a certain length can vary.

supertagged systems show a consistent improvement over the baseline. Moreover, the curves of the stacked and supertagged systems are mostly parallel and close to each other.

Supertagging and stacking thus do not just appear similar at the macro level in terms of LAS. The analysis shows that their contributions are also very similar when broken down by dependency or sentence length and the improvements are not restricted to sentences or arcs of particular lengths. We therefore conclude that both methods are indeed doing the same thing.

4.3 Oracle Experiments

In order to assess the potential utility of supertags we provided the parsers with gold supertags. We expect the gold supertags to give a considerable boost to accuracy as they encode correct syntactic information. Intuitively, we would expect the corresponding stacking experiment (providing gold trees) to reach 100% accuracy since the parser receives the full solution as features. However, this assumption turns out not to hold.

Row ④ in Tables 6 and 7 shows the results for the supertag experiments. Comparing row ④ with row ②, we find big jumps (between 7 and 20% absolute) in performance. For German, English, and Polish performance goes up even to 97/98%. These huge jumps are due to the amount of syntactic information encoded in the supertags, which is much higher than in POS tags for example.

Row ⑤ in Tables 6 and 7 shows the results for the stacking experiments. Surprisingly, stacking with gold dependency trees does not reach 100% accuracy. Moreover, comparing rows ④ and ⑤ we find that on average supertagging and stacking improve performance of a parser to the same extent.

The fact that gold supertags do not yield maximum accuracy is not so surprising since a supertag sequence does not encode the full dependency tree, but merely indicates direction of heads and dependents. However, it is puzzling that stacking with gold trees does not lead to perfect parsing results. In case of the transition-based parser, the reason might be that the parser does not do exhaustive search but uses beam search to explore only a fraction of the search space. That is, the gold solution can get pruned early enough that the parser never considers it. For the graph-based parser this result is more unexpected since this parser does exact search. We currently do not have any explanation for this, however we hypothesize that the lack of regularization during training might assign enough weight on the regular features such that they can override the few stacking features that convey the correct solution.

4.4 Self-Application

Rows ⑥ and ⑦ in Tables 6 and 7 show experiments where we use the same parser at Level 0 and Level 1. We know from Martins et al. (2008) that self-application, i.e. stacking a parser on its own output, leads to at most tiny improvements, especially compared to a setting with different parsers. Our results corroborate these findings. More interestingly, we find a similar effect for supertagging.[10] This effect demonstrates that it is important that Level 0 and Level 1 use different ways of modeling the data in order to benefit from the combination (cf. Surdeanu and Manning (2010)).

5 Supertagging Without Parsers

One potential advantage of supertagging over stacking is the fact that one can predict supertags without a parser. Most previous work predicts supertags using classifiers or sequence models, which is the standard for tagging problems. As tagging is commonly considered an "easier" task than parsing, one could assume that supertags can be predicted very efficiently using standard sequence labeling algorithms. But sequence labelers would not be able to predict the dependency tree in a stacking setup.

The two parsers that we use in the experiments are indeed unlikely to outperform standard sequence labelers in terms of speed. However, greedy arc-standard parsers are very fast. In the next experiment, we therefore compare a greedy arc-standard parser, which is the transition-based parser without beam search, with MarMoT (see Section 3.1). We follow Ouchi et al. (2014) in adding POS tags and morphological information to the feature model of the sequence labeler.

The purpose of this experiment is two-fold: So far, we predicted supertags by predicting a tree first and then deriving the supertags from that tree. Now we test how our previous results compare to supertags predicted by a sequence labeler, which is

[10] Note that most of the improvements in $\text{STAG}^{\text{GB}}_{\text{GB}}$ are actually statistically significant. However, the difference to BL^{GB} is considerably smaller than in the predicted setting in row ② (avg. difference is 0.28% vs. 0.65% points absolute).

		avg.	ar	eu	fr	de	he	hu	ko	pl	sv	en
①	BL^{GB}	84.16	85.64	83.37	84.78	91.46	78.71	82.60	86.08	85.17	75.24	88.59
②	$\text{STAG}^{\text{GB}}_{\text{SL}}$	84.91	86.24‡	84.33‡	84.89	91.89‡	79.82‡	83.54‡	86.85‡	85.89†	76.62‡	89.06‡
③	$\text{STAG}^{\text{GB}}_{\text{GTB}}$	84.65	86.16‡	83.56	85.02†	91.97‡	79.41‡	83.36‡	86.07	85.29	76.56‡	89.12‡
④	$\text{STACK}^{\text{GB}}_{\text{GTB}}$	84.66	86.24‡	83.57	85.11†	91.97‡	79.50‡	83.22‡	86.22	85.45	76.26†	89.09‡
⑤	BL^{TB}	84.04	85.69	82.22	84.07	91.15	78.80	83.27	85.97	84.51	75.65	89.06
⑥	$\text{STAG}^{\text{TB}}_{\text{SL}}$	84.63	85.81	83.58‡	84.07	91.37†	79.86‡	83.91‡	86.98‡	84.93	76.87†	88.91
⑦	$\text{STAG}^{\text{TB}}_{\text{GTB}}$	84.16	85.70	82.44	84.16	91.31	79.38	83.16	85.91	84.59	76.12	88.81
⑧	$\text{STACK}^{\text{TB}}_{\text{GTB}}$	84.17	85.84†	82.46	84.19	91.22	78.93	83.30	85.77	84.92	76.12	88.95

Table 8: Parsing results (LAS) with a sequence labeler and a greedy transition-based parser on development sets.

the common way of predicting supertags. But furthermore, we want to see how supertagging with a sequence labeler compares to supertagging and stacking with a parser that is equally efficient.

Table 8 gives the result of the experiment. We denote the sequence labeler by **SL** and the greedy parser by **GTB**. Rows ② and ⑥ show that, on average, the parsing performance is not harmed by predicting supertags with the sequence labeler instead of one of the parsers (compare to row ② in Tables 6 and 7). It depends on the individual data set whether the sequence labeler is more useful than one of the parsers or not. The supertags predicted by the sequence labeler improve parsing performance to a similar extent.

The experiments with the greedy parser yield different results for the graph-based and the transition-based parser on Level 1: When the graph-based parser acts as Level 1, the greedy parser is slightly behind the sequence labeler. This holds both for supertagging and stacking experiments (compare row ② to rows ③ and ④), which again suggests that supertagging and stacking are interchangeable. However, when Level 1 is the transition-based parser, we find a self-application effect for the greedy parser, both in supertagging and stacking (rows ⑤ vs. ⑦ and ⑧). This is not surprising since the decoding algorithms in the beam-search and greedy transition-based parser are identical. It simply underlines the importance of having different algorithms in the setup.

5.1 Out-of-Domain Application

The previous experiment shows that the greedy parser at Level 0 gives competitive results compared to a sequence labeler. Having fast predictors available for stacking or supertagging suggests an application where speed matters, e.g. Ambati et al. (2014) propose supertags to improve the performance of fast parsers in a web scale scenario.

As web data can be any kind of text, the question is whether the positive effects of supertagging and stacking are actually preserved in such an out-of-domain setting. To test this, we conduct experiments on the English Web Treebank (Bies et al., 2012) converted to Stanford Dependency format. Models are trained on sections 2-21 from the English Penn Treebank.

	avg.	answ.	email	news.	review	blog
BL^{GB}	76.28	74.09	75.06	76.16	76.32	79.78
$\text{STAG}^{\text{GB}}_{\text{SL}}$	76.82	74.52‡	75.75‡	76.88‡	76.99‡	79.98
$\text{STACK}^{\text{GB}}_{\text{TB}}$	76.93	74.88‡	75.72‡	76.49	77.10‡	80.44‡
BL^{TB}	76.51	74.41	75.16	76.09	76.76	80.13
$\text{STAG}^{\text{TB}}_{\text{SL}}$	76.83	74.37	75.85‡	76.61†	77.06	80.28
$\text{STACK}^{\text{TB}}_{\text{GB}}$	76.83	74.64	75.68‡	76.96‡	77.14‡	81.05‡
BL^{GTB}	74.42	72.32	73.25	74.00	74.73	77.79
$\text{STAG}^{\text{GTB}}_{\text{SL}}$	75.01	72.75	73.86‡	74.88‡	75.33‡	78.24‡

Table 9: Results (LAS) on the English Web Treebank.

The results in Table 9 show consistent improvements on the five genres of the data set both for supertagging and stacking. Both are thus good methods to improve parsing accuracies when parsing out-of-domain data. Since parsing speed also depends on the Level 1 parser, a greedy transition-based parser would be preferable in such an application. Using supertagging with a sequence labeler to provide syntactic information to the greedy parser is then a good choice because it avoids a self-application effect.

The last two rows in Table 9 show the performance when the greedy parser is acting as Level 1. Supertagging improves over the baseline significantly on 4 out of 5 data sets. However, the baseline for the greedy parser is on average about 2% points absolute behind the other two parsers. This loss in accuracy buys a significant speed-up though. The greedy parser is about 29 times faster[11] than the graph-based parser on the English

[11] We report parsing time. Exact runtimes depend on im-

		avg.	ar	eu	fr	de	he	hu	ko	pl	sv	en
①	BL^{GB}	84.16	85.64	83.37	84.78	91.46	78.71	82.60	86.08	85.17	75.24	88.59
②	$\max(\text{STAG}^{\text{GB}}_{\text{SL}}, \text{STACK}^{\text{GB}}_{\text{TB}})$	85.00	86.25‡	84.33‡	85.14‡	92.01‡	79.82‡	83.72‡	86.85‡	85.89†	76.62‡	89.41‡
③	$(\text{STAG}_{\text{SL}}+\text{STACK}_{\text{TB}})^{\text{GB}}$	85.20	86.62‡	84.51‡	85.35‡	92.22‡	79.90‡	84.23‡	86.67‡	85.78†	76.98‡	89.74‡
④	BL^{TB}	84.04	85.69	82.22	84.07	91.15	78.80	83.27	85.97	84.51	75.65	89.06
⑤	$\max(\text{STAG}^{\text{TB}}_{\text{SL}}, \text{STACK}^{\text{TB}}_{\text{GB}})$	84.94	86.19‡	83.86‡	84.55‡	91.98‡	79.86‡	83.91‡	86.98‡	85.65	77.16‡	89.30
⑥	$(\text{STAG}_{\text{SL}}+\text{STACK}_{\text{GB}})^{\text{TB}}$	85.13	86.43‡	84.09‡	84.59‡	92.04‡	79.75†	84.08‡	87.32‡	86.01†	77.30‡	89.67‡

Table 10: Results (LAS) on development sets for combining supertags and stacking.

data set and even 80 times faster on the Arabic data set. As the Arabic data set has very long sentences, the higher complexity of the graph-based parser has a notable effect on its performance. Compared to the beam-search transition-based parser, the greedy parser is about 10 times faster on English and 5 times faster on Arabic.

6 Combining Supertagging and Stacking

We now explore whether the combination of supertagging and stacking yields even better parsers.

In rows ③ and ⑥ in Table 10, we show results when supertag and stacking features come from different sources, i.e. they were predicted by different tools[12]. For both parsers, the sequence labeler predicts the supertags and the respective other parser provides the tree for the stacking features. The combinations are better than the baseline. Rows ② and ⑤ give results from the best single source, i.e. either $\text{STAG}^{\text{Y}}_{\text{SL}}$ or $\text{STACK}^{\text{Y}}_{\text{X}}$.

For most of the languages the difference between the combination and the best single component is statistically not significant, except Arabic, German, Hungarian, and English for $(\text{STAG}_{\text{SL}}+\text{STACK}_{\text{TB}})^{\text{GB}}$, and Arabic for $(\text{STAG}_{\text{SL}}+\text{STACK}_{\text{GB}})^{\text{TB}}$. The increment goes up to 0.51 in case of Hungarian. On average, the gains are, however, marginal – the graph-based parser's accuracy increases by 0.2% absolute and the transition-based parser improves by 0.18% absolute. Although these differences denote improvements, they are not nearly as high as the improvements over the baseline for the single components and it depends on the actual data set whether it is worth the effort.

In Section 4, we argued that supertagging and stacking are similar and the diversity of tools is the more important factor. The improvements by the combination can also be interpreted along these lines: They are caused by using different tools rather than the fact that we are combining the two methods. It is like stacking onto two parsers instead of one.

7 Conclusion

In this paper, we have shown that supertagging as a method for providing syntactic features for statistical dependency parsing (Ambati et al., 2014; Ouchi et al., 2014) is a form of stacking. Although supertags do not convey as much information as full trees, they improve dependency parsers to an equal amount. The two methods are thus in principle interchangeable.

Combining stacking and supertagging only gives improvements if different tools are used. In this case, the improvements come from the involvement of different tools rather than their combination. Furthermore, using supertags in a parser that predicted them itself does not lead to improvements. This is in line with findings by Surdeanu and Manning (2010) on stacking, of which supertagging is a variant. Therefore, while it is not so important which method is used, it is important to use different algorithms in these setups.

Finally, we have shown that sequence labelers can be replaced by greedy parsers in supertagging without compromising quality or speed. We applied them in a cross-domain parsing scenario and demonstrated that supertagging and stacking improve parsing also in this setting.

However, there are circumstances where one method might be preferable over the other, for example, when one wants to stack on a slow parser (cf. Øvrelid et al. (2009)). Rather than running the slow parser on every sentence in a stacking setup, it can be run once on some training data. A supertagger can then be trained on this data to provide syntactic information at a fraction of the cost (see Ambati et al. (2014) for CCG).

plementation and hardware. We therefore give relative numbers so the reader gets an impression of the magnitude.

[12] We did experiments with combining supertags and stacking from the same Level 0 tool, however since the features were derived from the same tree the differences compared to stacking only were negligible as expected.

Acknowledgements

The authors would like to thank the anonymous reviewers for their comments. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) via the SFB 732, projects D2 and D8 (PI: Jonas Kuhn), and the Integrated Research Training Group (MGK) of the SFB 732. Agnieszka Faleńska was additionally funded by the Project "International computer science and applied mathematics for business" at the University of Wrocław co-financed with EU funds within the European Social Fund (POKL.04.01.01-00-005/13).

References

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. 2013. Using CCG categories to improve Hindi dependency parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 604–609, Sofia, Bulgaria, August. Association for Computational Linguistics.

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. 2014. Improving Dependency Parsers using Combinatory Categorial Grammar. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 159–163, Gothenburg, Sweden, April. Association for Computational Linguistics.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265, June.

Ann Bies, Justin Mott, Colin Warner, and Seth Kulick. 2012. English Web Treebank LDC2012T13.

Anders Björkelund and Joakim Nivre. 2015. Non-deterministic oracles for unrestricted non-projective transition-based dependency parsing. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 76–86, Bilbao, Spain, July. Association for Computational Linguistic.

Anders Björkelund, Özlem Çetinoğlu, Agnieszka Faleńska, Richárd Farkas, Thomas Müller, Wolfgang Seeker, and Zsolt Szántó. 2014. The IMS-Wrocław-Szeged-CIS entry at the SPMRL 2014 Shared Task: Reranking and Morphosyntax meet Unlabeled Data. In *Notes of the SPMRL 2014 Shared Task on Parsing Morphologically-Rich Languages*, Dublin, Ireland, August.

Bernd Bohnet. 2010. Top Accuracy and Fast Dependency Parsing is not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China, August. Coling 2010 Organizing Committee.

Tim Buckwalter. 2002. Buckwalter Arabic Morphological Analyzer Version 1.0. *Linguistic Data Consortium, University of Pennsylvania, 2002. LDC Catalog No.: LDC2002L49.*

Stephen Clark and James R. Curran. 2004. The Importance of Supertagging for Wide-coverage CCG Parsing. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online Passive–Aggressive Algorithms. *Journal of Machine Learning Reseach*, 7:551–585, March.

Rickard Domeij, Ola Knutsson, Johan Carlberger, and Viggo Kann. 2000. Granska-an efficient hybrid system for Swedish grammar checking. In *In Proceedings of the 12th Nordic Conference in Computational Linguistics*.

Mikel L. Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim ORegan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M Tyers. 2011. Apertium: A free/open-source platform for rule-based machine translation. *Machine Translation*.

Kilian A. Foth, Michael Daum, and Wolfgang Menzel. 2004. Interactive grammar development with WCDG. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*.

Kilian A. Foth, Tomas By, and Wolfgang Menzel. 2006. Guiding a Constraint Dependency Parser with Supertags. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 289–296, Sydney, Australia, July. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2013. Word Segmentation, Unknown-word Resolution, and Morphological Agreement in a Hebrew Parsing System. *Computational Linguistics*, 39(1):121–160, March.

Aravind K. Joshi and Srinivas Bangalore. 1994. Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing. In *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, pages 154–160, Stroudsburg, PA, USA. Association for Computational Linguistics.

André Filipe Torres Martins, Dipanjan Das, Noah A. Smith, and Eric P. Xing. 2008. Stacking Dependency Parsers. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 157–166, Honolulu, Hawaii, October. Association for Computational Linguistics.

André Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August. Association for Computational Linguistics.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131, Prague, Czech Republic, June. Association for Computational Linguistics.

Thomas Müller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient Higher-Order CRFs for Morphological Tagging. In *In Proceedings of EMNLP*.

Takashi Ninomiya, Takuya Matsuzaki, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. 2006. Extremely Lexicalized Models for Accurate and Fast HPSG Parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 155–163, Sydney, Australia, July. Association for Computational Linguistics.

Joakim Nivre and Ryan McDonald. 2008. Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio, June. Association for Computational Linguistics.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An Improved Oracle for Dependency Parsing with Online Reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76, Paris, France, October. Association for Computational Linguistics.

Joakim Nivre. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore, August. Association for Computational Linguistics.

Hiroki Ouchi, Kevin Duh, and Yuji Matsumoto. 2014. Improving Dependency Parsers with Supertags. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 154–158, Gothenburg, Sweden, April. Association for Computational Linguistics.

Lilja Øvrelid, Jonas Kuhn, and Kathrin Spreyer. 2009. Improving data-driven dependency parsing using large-scale LFG grammars. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 37–40, Suntec, Singapore, August. Association for Computational Linguistics.

Sangwon Park, Donghyun Choi, Eunkyung Kim, and Keysun Choi. 2010. A plug-in component-based Korean morphological analyzer. In *Proceedings of HCLT 2010*.

Yves Schabes, Anne Abeille, and Aravind K. Joshi. 1988. Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 2*, COLING '88, pages 578–583, Stroudsburg, PA, USA. Association for Computational Linguistics.

Helmut Schmid, Arne Fitschen, and Ulrich Heid. 2004. SMOR: A German Computational Morphology Covering Derivation, Composition, and Inflection. In *Proceedings of the IVth International Conference on Language Resources and Evaluation (LREC 2004*, pages 1263–1266.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho Choi, Matthieu Constant, Richárd Farkas, Iakes Goenaga, Koldo Gojenola, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiorkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clérgerie. 2014. Overview of the spmrl 2014 shared task on parsing morphologically rich languages. In *Notes of the SPMRL 2014 Shared Task on Parsing Morphologically-Rich Languages*, Dublin, Ireland.

Mihai Surdeanu and Christopher D. Manning. 2010. Ensemble Models for Dependency Parsing: Cheap and Good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 649–652, Los Angeles, California, June. Association for Computational Linguistics.

Marcin Woliński. 2006. Morfeusz - A practical tool for the morphological analysis of Polish. In *Intelligent information processing and web mining*, pages 511–520. Springer.

Yue Zhang and Stephen Clark. 2008. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii, October. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based Dependency Parsing with Rich Non-local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA, June. Association for Computational Linguistics.

Zhenxia Zhou. 2007. Entwicklung einer französischen Finite-State-Morphologie. Diplomarbeit, Institute for Natural Language Processing, University of Stuttgart.

János Zsibrita, Veronika Vincze, and Richárd Farkas. 2013. Magyarlanc 2.0: Szintaktikai elemzés és felgyorsított szófaji egyértelműsítés. In *IX. Magyar Számítógépes Nyelvészeti Konferencia*.