IMS at the CoNLL 2017 UD Shared Task: CRFs and Perceptrons Meet Neural Networks

Anders Björkelund* and **Agnieszka Falenska*** and **Xiang Yu*** and **Jonas Kuhn**
Institute for Natural Language Processing
University of Stuttgart
{anders,falenska,xiangyu,jonas}@ims.uni-stuttgart.de

Abstract

This paper presents the IMS contribution to the CoNLL 2017 Shared Task. In the preprocessing step we employed a CRF POS/morphological tagger and a neural tagger predicting supertags. On some languages, we also applied word segmentation with the CRF tagger and sentence segmentation with a perceptron-based parser. For parsing we took an ensemble approach by blending multiple instances of three parsers with very different architectures. Our system achieved the third place overall and the second place for the surprise languages.

1 Introduction

This paper presents the IMS contribution to the CoNLL 2017 UD Shared Task (Zeman et al., 2017). Our submission to the Shared Task (ST) ranked third. Our overall approach relies on established techniques for improving accuracies of dependency parsers, including strong preprocessing, supertagging and parser combination.

The task was to predict dependency trees from raw text. To make the ST more accessible to participants, the organizers provided baseline predictions for all preprocessing steps (including word and sentence segmentation and POS/morphological feature predictions) using the baseline UDPipe system (Straka et al., 2016). We scrutinized the baseline and considered where we could improve over it. It turns out that, although the UDPipe baseline is a strong one, considerable parsing accuracy improvements can be gained by improving the preprocessing steps. In particular, we applied our own POS/morphology tagging using a CRF tagger and supertagging (Ouchi et al., 2014) with a neural tagger. Additionally, we performed our own word and/or sentence segmentation on a subset of the test sets.

For the parsing step we applied an ensemble approach using three different parsers, sometimes using multiple instances of the same parser: one graph-based parser trained with the perceptron; one transition-based beam search parser also trained with the perceptron; and one greedy transition-based parser trained with neural networks. The parser outputs were combined through blending (also known as reparsing; Sagae and Lavie, 2006) using the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967).

The final test runs were carried out on the TIRA platform (Potthast et al., 2014) where participants were assigned a virtual machine. To ensure that our final test run would finish on time on the VM, we established a time budget for each treebank and set a goal that a full test run should finish within 24 hours. Thus we applied a combination search under a time constraint to limit the maximal number of instances of the individual parsers.

An interesting aspect of the ST was the introduction of four surprise languages. These languages were only announced one week before the test phase at which point the participants were provided with roughly 20 gold standard sentences for each language. Unfortunately, among the allowed external resources the amount of parallel data for the surprise languages was rather limited. This prevented us from using cross-lingual techniques or multilingual word vectors. We therefore resorted to blending models trained on the small samples as well as delexicalized models trained on other source languages.

Another challenge of the ST were 14 parallel new test domains for the known languages. Since the UD annotation scheme is applied on all of the treebanks, this suggests that the training data of

*All three authors contributed equally.

Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pages 40–51,
Vancouver, Canada, August 3-4, 2017.

the same language from different domains could be combined. We made several experiments in this direction and trained models on merged treebanks for most of the parallel test sets (Section 7).

The remainder of this paper is organized as follows. Section 2 discusses our preprocessing steps, including word and sentence segmentation, POS and morphological tagging, and supertagging. In Section 3 we describe the three baseline parsers, while blending is reviewed in Section 4. In Section 5 we go through our pipeline and show results on the development data. Sections 6 and 7 describe our approaches to the surprise languages and parallel test sets, respectively. Our official test set results are shown in Section 8 and Section 9 concludes.

2 Preprocessing

For most data sets word and sentence segmentation plays a minimal role, as it is delivered almost for free by means of whitespaces, sentence-final punctuation and capital letters. Therefore our overall architecture applies word/sentence segmentation pipeline only on treebanks for which this task is non-trivial (see Figure 1). These test sets can roughly be grouped into two categories: Languages where tokenization is challenging, e.g., Chinese and Japanese, but also languages such as Arabic and Hebrew, where many orthographic tokens are segmented into smaller syntactic words with transformations. The second category comprises the treebanks where the detection of sentence boundaries is difficult, mostly classical texts.

2.1 Word Segmentation

We applied our own word segmentation on six languages: Arabic, French, Hebrew, Japanese, Vietnamese, and Chinese. We selected them by analyzing the UDPipe baseline and picking out cases where we potentially could surpass it.

For Arabic, French and Hebrew, the difficulty lies in splitting *orthographic* words (i.e., multiword tokens) into several *syntactic* words (e.g., clitics). Additionally the orthographic words are often not the simple concatenation of their components. For example in French, the multiword token *des* would be split into two syntactic words *de* and *les*. We cast this problem as classification by predicting the Levenshtein edit script to transform a multiword token into its components.

Concretely with the French example, we take the multiword token *des* as input, and predict *de&les*, where *&* is an artificial delimiter to split the token. To reduce the tag set, we used the Levenshtein edit script "`=2+&le=1`" instead of *de&les* as the target class, which means keeping the first 2 characters, adding "`&le`", then keeping 1 character, so that *des* can be transformed into *de&les* (thus split into *de* and *les*). Using edit scripts reduced the tag set size from about 12,000 to 1,000 for Arabic and from 14,000 to 600 for Hebrew.

For Japanese, Vietnamese and Chinese, we simply applied a standard chunking method: for each character (or phoneme in Vietnamese), we predicted the chunk boundary, jointly with the POS tag of the word.

In both cases, we used the state-of-the-art morphological CRF tagger[1] MarMoT (Müller et al., 2013) to predict the tags (edit scripts or chunk boundaries). We used second order models for Arabic, French and Hebrew, and third order models for Japanese, Vietnamese and Chinese.

2.2 Sentence boundary detection

We applied our own sentence segmentation on nine languages (see Figure 1). For some of them, like Gothic or Latin PROIEL, typical orthographic features (e.g., punctuation or capitalization) that indicate sentence boundaries are not present and UDPipe was achieving extremely low scores (23.51 and 19.76 F1 respectively). The others were selected empirically by tests on the development data.

We employed a beam-search transition-based parser extended to predict sentence boundaries (Björkelund et al., 2016). This parser (referred to as **TPSeg**) is an extension of our transition-based parser (see Section 3.2) using the perceptron and is trained using DLaSO updates (Björkelund and Kuhn, 2014; Björkelund et al., 2016). It marks sentence boundaries with an additional transition. For this parser the input is not just a pre-tokenized sentence, but a pre-tokenized document. As documents during test-time we used paragraphs from the raw input text, assuming that no sentence would span across a paragraph break.

A training instance for the parser is a document (rather than a sentence). Some treebanks have the entire training set represented as a single paragraph (document). Initial experiments showed that

[1] `http://cistern.cis.lmu.de/marmot/`

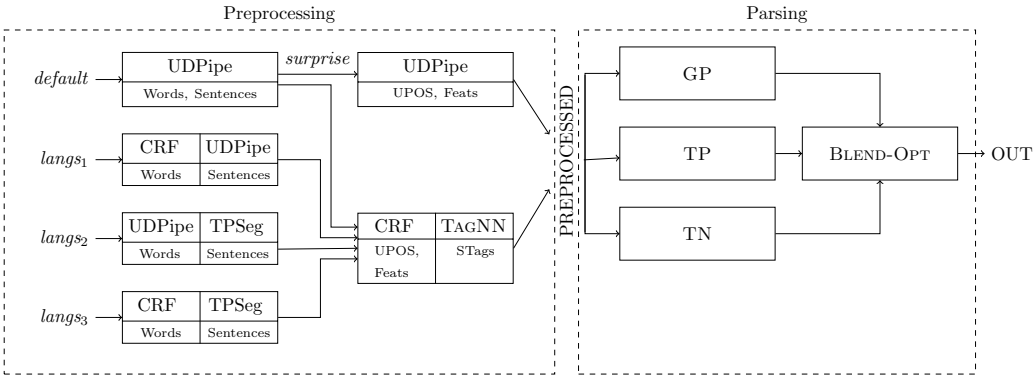

Figure 1: System architecture, where *langs*$_1$: he, ja, fr, fr_sequoia, fr_partut, fr_pud, vi, zh; *langs*$_2$: cu, en, et, got, grc_proiel, la, la_ittb, la_proiel, nl_lassysmall, sl_sst; *langs*$_3$: ar, ar_pud.

training the parser on a single document took considerable time and also did not perform very well. Instead, we created artificial documents for training by taking chunks of 10 sentences from the training set and treating them as documents (irrespective of whether they went across paragraphs).

We trained the parser using gold word segmentation and POS/morphology information. At test time we relied on UDPipe predictions in most cases. However, for Arabic, the only language where we did both word and sentence segmentation, we applied our own POS/morphology tagger since the word segmentation had changed. Additionally, we applied our tagger on Old Church Slavonic, Estonian, Gothic, Ancient Greek PROIEL and Dutch LassySmall since we found that this lead to better sentence segmentation results on the development sets.

2.3 Part-of-Speech and Morphological Tagging

We used MarMoT to jointly predict POS tags and morphological features. We annotated the training sets via 5-fold jackknifing. All parsers for all languages except the surprise ones were trained on jackknifed features. We did not use XPOS tags and lemmas. We used MarMoT with default hyperparameters.

2.4 Supertags

Supertags (Joshi and Bangalore, 1994) are labels for tokens which encode syntactic information, e.g., the head direction or the subcategorization frame. Supertagging has recently been proposed to provide syntactic information to the feature model of statistical dependency parsers (Ambati et al. (2013; 2014), Ouchi et al. (2014)).

We follow the definition of supertagging from Ouchi et al. (2014) and extract supertag tag sets from the treebanks. We use their Model 1 to design our supertags. That is, we encode the dependency relation (*label*), the relative head direction (*hdir*) and the presence of left and right dependents (*hasLdep*, *hasRdep*) and follow the template *label/hdir+hasLdep_hasRdep*.

We used an in-house neural-based tagger (**TAGNN**) to predict the supertags (Yu et al., 2017). It takes the context of a word within a window size of 15. The input word representations are concatenations of three components: output of a character-based Convolutional Neural Network (CNN), pretrained word embeddings provided by the ST organizers, and a binary code indicating the target word. The word representations of the whole context-window are then fed into another CNN to predict the supertag of the target word. We used TAGNN instead of CRF for supertagging, since it performed considerably better in the preliminary experiments.

3 Baseline parsers

Surdeanu and Manning (2010) show that combining a set of parsers with a simple voting scheme can improve parsing performance. Martins et al. (2013) demonstrate that self-application, i.e., stacking a parser on its own output, only leads to minuscule improvements.[2] Therefore to profit from combining components one of the most significant factor is their diversity. Thus we experimented with three parsers with quite different ar-

[2]In fact, even supertagging can be regarded as a form of stacking. Also in this case, the key ingredient is that the suppertagger is architecturally sufficiently different from the parser (Faleńska et al., 2015).

chitectures and additionally varied their settings.

3.1 Graph-based perceptron parser

As the graph-based parser we used *mate*[3] (Bohnet, 2010), henceforth referred to as **GP**. This is a state-of-the-art graph- and perceptron-based parser. The parser uses the Carreras (2007) extension of the Eisner (1997) decoding algorithm to build a projective parse tree. It then applies the non-projective approximation algorithm of McDonald and Pereira (2006) to recover non-projective dependencies. We train the parser using the default number of training epochs (10).

We modified the publicly available sources of this parser in two ways. First, we extended the feature set with features based on the supertags following Faleńska et al. (2015). Second, we changed the perceptron implementation to shuffle the training instances between epochs.[4] Shuffling enables us to obtain different instances of the parser trained with different random seeds, which are used in the blending step.

Since the time complexity of the Carreras (2007) decoder is quite high ($O(n^4)$) this parser required a considerable amount of time to parse long sentences. Therefore, while applying this parser in the blending scenario, we skipped all sentences longer than 50 tokens.[5] We additionally made sure that for each treebank we had at least one parser that was not GP, so that all sentences would be parsed.

3.2 Transition-based beam-perceptron parser

We apply an in-house transition-based beam search parser trained with the perceptron (Björkelund and Nivre, 2015), henceforth referred to as **TP**.[6] We have previously extended this parser to accommodate features from supertags (Faleńska et al., 2015). It uses the ArcStandard system extended with a Swap transition (Nivre, 2009) and is trained using the improved oracle by Nivre et al. (2009).

The parser is trained with a globally optimized structured perceptron (Zhang and Clark, 2008) using max-violation updates (Huang et al., 2012). We use the default settings for beam size (20) and number of training epochs (also 20). Similarly to GP, we employ different seeds for the random number generator used during shuffling of the training instances in order to obtain multiple different models.

3.3 Transition-based greedy neural parser

We use an in-house transition-based greedy parser with neural networks (Yu and Vu, 2017), henceforth referred to as **TN**.[7]

The parser uses a CNN to compose word representations from characters, it also takes the embeddings of word forms, universal POS tags and supertags and concatenates all of them as input features. The input is then fed into two hidden layers with ReLU non-linearity, and finally predicts the transition with a softmax layer. The parser uses the same Swap transition system and oracle as TP. We use the default hyperparameters during training and testing.

During training the parser additionally predicts the supertag of the top token in the stack and includes the tagging cross-entropy into the cost function. This approach is similar to stack-propagation (Zhang and Weiss, 2016), where the tagging task is only used as a regularizer.

4 Blending

To enhance the performance of the baseline single parsers we combined them using blending (Sagae and Lavie, 2006). We trained multiple instances of each baseline parser using different random seeds. We parsed every sentence and assigned scores to arcs depending on how frequent they were in the predicted trees. We used the Chu-Liu-Edmonds algorithm to decode the maximum spanning tree from the resultant graph. This way we obtained the majority decision of the parser instances under the tree constraint.

As a baseline for blending (**BLEND-BL**), we took four instances from each of the baseline parsers: The four GP instances were trained with different random seeds. The four TP instances further split into two groups: two parse from left to right (TP-l2r) and two parse from right to left (TP-r2l). The four TN instances differ not only in the parsing direction, but also in the word embeddings, two use pretrained embeddings

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

[4] The publicly available version does not shuffle.

[5] For the baseline results on the development sets (Tables 3 and 4), the parser was applied to all sentences.

[6] This parser as well as the variant that we applied for sentence segmentation (TPSeg) is available on the first author's website.

[7] This parser as well as the neural tagger used for supertagging (TAGNN) is available on the third author's website.

from the organizers (TN-l2r-vec, TN-r2l-vec) and two use randomly initialized embeddings (TN-l2r-rand, TN-r2l-rand).

The 4+4+4 combination was rather arbitrary and simply based on the intuition that different parsers should be equally represented and as diverse as possible. However, this might not be the optimal combination since different parsers are better at different treebanks. Also, given the relatively limited computing resources on the VM, we needed to optimize the number of blended instances in terms of speed.

We thus applied a combination search under a time constraint. First we measured time needed by each parser to parse every development treebank on the VM as an estimation of time usage for the test run. We then defined a time budget of 1,000 seconds for each treebank, and checked all combinations of the parsers on the development set under the time budget. We took the combinations from a pool of 24 individual instances, divided into seven groups: 8×GP; 4×TP-l2r; 4×TP-r2l; 2×TN-l2r-rand; 2×TN-l2r-vec; 2×TN-r2l-rand; 2×TN-r2l-vec.

Note that enumerating all combinations of individual instances is not feasible (2^{24} combinations). Thus we applied a two-step heuristic search. First we searched for the optimal number of instances from the 7 groups, by drawing samples from the pool of instances with only different random seed (at most $9 \times 5 \times 5 \times 3 \times 3 \times 3 \times 3 = 18,225$ possibilities). Once the optimal numbers of instances were found, we then searched exhaustively for the optimal instances (**BLEND-OPT**).

5 Evaluation

In this section we evaluate the aforementioned methods on the 55 treebanks for which development data was available.

5.1 Word and sentence segmentation

As discussed in Section 2, we applied our own word and/or sentence segmentation to a subset of languages. The corresponding results on the development sets are shown in Tables 1 and 2.

For word tokenization both our methods (predicting edit script and tagging with chunk boundaries) outperform the UDPipe baseline by 2.64 F1-score points on average. The biggest gains are achieved for Hebrew (4.57 points) and Vietnamese (4.67 points).

Using the TPSeg parser to predict sentence boundaries results in an average improvement of 9.32 points on sentence segmentation F1-score over the UDPipe baseline. Especially the difficult data sets that do not use orthographic features to indicate sentence boundaries improve by a big margin, for example Latin PROIEL by 18.76 and Gothic by 15.73.

Most importantly, the improvements in word and sentence segmentation F1-score roughly translate into LAS improvements with a 1:1 and a 5:1 ratio, respectively.

	UDPipe	CRF	Δ LAS
ar	93.86	95.53	2.04
fr	99.18	99.66	0.60
fr_sequoia	98.65	99.35	0.90
he	88.15	92.72	4.82
ja	89.53	92.10	5.08
vi	83.99	88.66	5.57
zh	88.95	92.76	5.47
average	91.76	94.40	3.50

Table 1: F1 scores for word segmentation and gains in LAS for TP.

	UDPipe	TPSeg	Δ LAS
ar	77.99	94.01	0.83
cu	37.09	48.03	3.16
en	76.35	78.69	0.66
et	84.91	86.40	0.54
got	23.51	39.24	4.01
grc_proiel	41.95	54.38	1.91
la_ittb	77.38	80.55	0.47
la_proiel	19.76	38.52	4.00
nl_lassysmall	79.31	82.35	0.82
average	57.58	66.91	1.82

Table 2: F1 score for sentence segmentation and gains in LAS for TP.

5.2 Preprocessing and Supertags

To see the improvements stemming from our preprocessing steps we run the baseline parsers in four incremental settings: (1) using only the UDPipe baseline predictions, (2) replacing POS and morphological features with CRF predictions, (3) adding supertags, and (4) applying our own word and sentence segmentation. Table 3 shows the average LAS for each parser across the 55 development sets for the consecutive experiments. For each set of experiments the parsers were trained on corresponding jackknifed annotations for POS, morphology, and supertags. Gold word

and sentence segmentation was used while training parsers in all settings.

The table shows that replacing the POS and morphological tagging with the CRF instead of baseline UDPipe predictions improves the parsers by 0.66 on average.[8] The introduction of supertags brings an additional 0.88 points which demonstrates that supertags are a useful source of syntactic features for dependency parsers, irrespective of architecture. Replacing the word and sentence segmentation from UDPipe with our own improves on average by 0.74 points. It is worth noting that this improvement stems only from the 15 treebanks where we applied our own segmentation, although the averages in Table 3 are computed across all 55 treebanks.

	UDPipe	CRF	+STags	+segm.
GP	75.46	76.01	+1.12	+0.74
TP (l2r)	74.69	75.49	+0.97	+0.78
TN (l2r-vec)	74.95	75.58	+0.54	+0.71
average	75.03	75.69	+0.88	+0.74

Table 3: Average (across 55 treebanks) gains in parsing accuracies (LAS) for incremental changes to UDPipe preprocessing baseline.

5.3 Development Results

Our overall results on the development sets are shown in Table 4. The table shows the performance of the preprocessing steps, the individual baseline parsers, and the results of the two blends. The 15 treebanks where we applied our own word and/or sentence segmentation are marked explicitly in the table, for the other cases we used the UDPipe baseline.

The three single baseline parsers achieved similar average performances. Each one of them performed the highest on some of the treebanks, but not on all. It is worth noting that the strongest baseline parser, GP, is perceptron-based rather than a neural model. That is not to say that perceptrons generally are stronger than neural models (our neural TN parser is a greedy parser, and other participants in the Shared Task present considerably stronger neural models), however it indicates that perceptrons are not miles behind the more recent neural-based parsers.

Blending parsers yield a strong boost over the baselines. BLEND-BL improves roughly 2-3 points depending on the choice of baseline. By searching for optimal combinations under the time budget, this can be further improved by 0.49 on average (BLEND-OPT). The search reduced the number of models from 660 to 438. In particular, there were 221 instances of GP, 79 of TP, and 138 of TN.

6 Surprise languages

The implementation of the surprise languages in the Shared Task was done in a rather peculiar way with respect to preprocessing. The test sets were annotated by the organizers through cross-validation. That is, the test sets were provided with predicted (by UDPipe) POS and morphological tags. Participants were provided with a small sample (about 20 sentences) for each surprise language, however only with gold standard preprocessing. This meant that it was difficult to use the samples for tuning/development since we would either have to use gold standard preprocessing, or apply cross-validation on the samples ourselves which most likely would have resulted in considerably worse preprocessing than that which was delivered for the test sets. We chose to consistently use gold preprocessing for all development experiments on the surprise languages.

A straightforward approach to the surprise languages is to use delexicalized parser transfer (Zeman and Resnik, 2008). The idea is to train a parser on a source treebank using only non-lexical features (in our case universal POS tags and morphological features) and apply it on sentences from the target language. We followed Rosa and Zabokrtský (2015) and performed multi-source delexicalized transfer by blending together models trained on several languages. Contrary to them, we treat the source languages equally and blend them with the same weight.

We trained delexicalized TP and GP parsers for 40 source languages (we took the 40 biggest treebanks, excluding the domain specific ones). We refrained from training TN since the main motivation of this parser is that it operates on characters. Therefore, using it in the delexicalized setting does not make sense.

To narrow down the number of possible source language parsers, we used TP to select the best six source languages for each surprise language using the sample data. We then searched for the optimal

[8] The actual improvements on the POS and morphological tagging tasks amount to 0.68 and 1.17, respectively.

	Preprocessing (F1)				Baseline parsers (LAS)			Blending (LAS)	
	Words	Sentences	UPOS	Feats	GP	TP	TN	BLEND-BL	BLEND-OPT
ar$^{\otimes\odot}$	95.53	94.01	90.73	86.40	70.88	70.54	71.35	72.59	72.99
bg	99.84	92.41	97.88	96.22	84.81	84.65	83.35	85.93	86.09
ca	99.96	98.77	98.16	97.50	87.04	86.75	85.11	87.59	87.90
cs	99.96	92.41	98.68	93.65	87.36	86.75	83.74	87.42	87.45
cs_cac	100.00	99.09	99.07	91.01	87.37	86.52	85.41	87.56	88.17
cs_cltt	98.65	74.11	90.30	79.69	73.01	72.83	73.94	76.84	77.30
cu$^{\odot}$	100.00	48.03	94.92	89.26	73.35	72.45	73.02	75.27	75.86
da	99.68	84.36	95.22	94.59	78.01	76.98	74.59	79.63	80.01
de	99.91	92.25	92.95	84.72	78.72	77.83	75.10	80.02	80.54
el	99.87	88.67	95.77	91.03	81.53	80.78	80.06	83.45	84.16
en$^{\odot}$	98.69	78.69	93.09	94.03	78.37	77.27	76.93	79.11	79.41
en_lines	99.93	87.36	94.92	99.93	76.51	76.23	76.41	78.59	79.35
en_partut	99.46	97.62	94.23	93.35	76.58	76.15	77.35	79.41	80.12
es	99.80	98.07	96.12	96.97	84.91	84.25	83.00	85.08	85.25
es_ancora	99.94	96.33	98.10	97.57	86.66	86.18	85.16	87.07	87.30
et$^{\odot}$	99.79	86.40	89.20	84.13	63.56	62.15	63.87	66.49	67.76
eu	99.99	99.00	94.14	89.83	74.99	73.73	74.47	76.96	77.59
fa	99.69	97.14	96.16	96.24	82.86	82.43	81.97	84.21	84.42
fi	99.69	86.47	95.49	93.01	80.08	78.99	78.34	81.71	82.13
fi_ftb	99.93	82.52	92.85	93.15	79.97	79.03	80.35	81.17	82.02
fr$^{\otimes}$	99.66	97.09	96.90	96.78	87.24	86.93	86.25	87.76	87.92
fr_sequoia$^{\otimes}$	99.35	90.20	96.58	95.68	84.30	83.42	83.01	85.54	86.17
gl	99.93	98.04	96.80	99.80	80.33	79.31	78.89	81.74	81.87
got$^{\odot}$	100.00	39.24	94.75	87.90	68.29	67.59	67.35	70.65	71.01
grc	99.98	99.17	88.34	89.58	65.77	64.06	64.28	68.14	69.09
grc_proiel$^{\odot}$	100.00	54.38	96.64	90.57	75.07	74.29	73.98	77.30	77.80
he$^{\otimes}$	92.72	98.57	89.24	87.21	68.89	68.75	68.31	70.94	71.02
hi	100.00	98.46	96.12	90.89	88.92	88.83	89.73	90.09	90.45
hr	99.98	97.23	96.70	87.57	80.54	79.91	79.15	82.41	82.99
hu	99.91	94.55	93.84	72.72	72.06	72.00	70.84	76.13	76.50
id	99.99	90.83	93.33	99.56	75.26	74.80	73.68	77.38	77.43
it	99.70	93.20	97.14	97.31	85.70	84.80	84.48	86.35	86.77
ja$^{\otimes}$	92.10	99.71	89.82	92.08	78.52	78.66	79.58	79.71	80.04
ko	99.45	91.10	93.10	99.17	70.47	71.13	74.37	74.03	76.41
la_ittb$^{\odot}$	99.88	80.55	96.79	92.41	76.10	75.40	75.67	78.21	79.16
la_proiel$^{\odot}$	99.99	38.52	95.83	89.46	68.99	67.05	67.82	71.45	71.99
lv	98.91	96.48	91.29	85.40	67.08	65.03	65.31	69.19	70.25
nl	99.87	92.11	94.38	92.92	79.04	78.29	76.36	80.18	80.72
nl_lassysmall$^{\odot}$	99.90	82.35	96.01	95.74	79.03	77.85	76.27	80.41	80.98
no_bokmaal	99.89	96.91	97.55	96.36	86.17	85.90	84.89	86.79	87.01
no_nynorsk	99.92	93.05	97.05	96.01	84.56	84.19	83.55	85.36	85.50
pl	99.87	99.56	96.12	85.71	84.67	83.78	83.66	85.89	86.89
pt	99.74	89.27	96.86	95.05	86.04	85.71	84.53	87.20	87.56
pt_br	99.83	96.65	97.39	99.71	86.99	86.49	86.33	87.88	87.87
ro	99.55	95.16	96.68	96.13	82.05	81.61	80.12	83.19	83.37
ru	99.92	96.18	95.44	87.50	80.22	79.60	78.89	81.89	82.37
ru_syntagrus	99.68	97.67	98.15	94.40	89.10	88.45	86.79	89.37	89.19
sk	100.00	77.85	95.04	80.13	78.49	77.92	76.71	80.10	80.70
sl	99.94	99.59	97.13	90.54	85.92	84.33	82.42	87.05	87.40
sv	99.77	95.59	95.52	94.80	77.11	76.34	75.28	79.18	79.96
sv_lines	99.97	87.28	94.50	99.97	76.10	75.65	75.93	78.00	78.80
tr	97.88	96.98	91.35	86.17	58.05	58.07	57.04	61.42	62.16
ur	99.99	98.37	93.46	80.24	79.02	78.45	79.78	80.70	80.96
vi$^{\otimes}$	88.66	96.28	80.64	88.56	47.01	47.62	47.76	49.66	49.77
zh$^{\otimes}$	92.76	97.60	86.23	91.57	62.93	63.28	63.37	66.17	66.80
average	99.07	89.45	94.56	91.78	77.87	77.24	76.83	79.52	80.01

Table 4: Development results. The treebanks for which we did our own word and/or sentence segmentation are marked with $^{\otimes}$ and $^{\odot}$ respectively. The TP and TN models correspond to TP-l2r TN-l2r-vec, respectively.

combination (across both parsers) where, instead of a time budget, we arbitrarily set the maximum number of parsers to eight.

sme_{lex}	51.02
fi_{delex}	51.02
$\text{no\_bokmaal}_{delex}$	47.62
et_{delex}	47.62
BLEND-OPT	60.54

(a) Target language: sme

hsb_{lex}	65.00
cs_{delex}	78.04
sl_{delex}	76.74
sk_{delex}	75.43
BLEND-OPT	78.70

(c) Target language: hsb

kmr_{lex}	48.76
fa_{delex}	42.15
el_{delex}	33.88
uk_{delex}	29.75
BLEND-OPT	44.63

(b) Target language: kmr

bxr_{lex}	41.83
eu_{delex}	37.25
ur_{delex}	35.95
hi_{delex}	35.29
BLEND-OPT	44.44

(d) Target language: bxr

Table 5: Parsing accuracy (LAS) for surprise languages: the three best delexicalized TP-l2r parsers and lexicalized parser obtained by leave-one-out jackknifing.

In addition to the delexicalized models, we also trained lexicalized TP and GP models[9] on the sample data and applied leave-one-out jackknifing.[10] A comparison between the three best delexicalized TP models and the lexicalized TP parser is shown in Table 5. Only for Upper Sorbian were transferred models able to surpass the model trained on the very small training data. Interestingly, the blended models were much better than any of the models for all languages except Kurmanji. Therefore we decided not to use any of the delexicalized models for this language. For the other three surprise languages we used ultimately blended eight delexicalized (selected as described above) and eight lexicalized models, the intuition being that this would give equal weight to lexicalized and delexicalized models.

7 Parallel datasets

For the 14 additional parallel datasets (PUD) we used parsers trained on their corresponding languages. For several languages there were more than one treebank in the training data for the corresponding PUD test set. This begs the question as to whether the models used for the PUD test sets should be trained only on the primary treebank, or on the combination of all training sets corresponding to that language. For the main treebanks, initial experiments indicated that this was a bad idea and parsers performed better when training sets were not combined. However, for the PUD test sets we had no information on the annotation scheme nor the domain, which made it difficult to decide whether to use only the primary training set or all available.

For each language with multiple training sets, we trained one parser on each training set as well as on their concatenation. We applied these models on the development sets and created a confusion matrix. Without prior knowledge about the PUD treebanks, we estimated the expected LAS as the average LAS of the development sets and chose the model that maximizes the estimation.

Table 6 shows such a confusion matrix for Swedish using the TN parser. The expected LAS for PUD (right-most column) is highest when trained on the concatenation of the two treebanks. This observation held for all the languages with multiple treebanks that we tested and we therefore used models trained on the concatenation of all training data with two exceptions: For Czech time prevented us from training models and creating a confusion matrix and we only used models trained on the primary treebank. For Finish FTB the README distributed with the treebank states that this treebank is a conversion that tries to approximate the primary Finish treebank. This suggests that it does not entirely conform to the Finish UD standard. We assumed that the Finish PUD test set would be closer to the primary treebank and therefore chose to use only the model trained on the primary treebank.

	sv	sv_lines	exp. sv_pud
sv	75.28	63.54	69.41
sv_lines	67.78	75.93	71.86
sv_concat	75.31	75.23	75.27

Table 6: Confusion matrix for Swedish with expected LAS on Swedish PUD.

8 Test Set results

Our final results on the test sets are shown in Table 7. Overall we ranked third in the Shared Task with a macro-average LAS of 74.42 behind two teams: Stanford and Cornell. Both of them used

[9] We did not train lexicalized TN models since it had problems with exploding gradients and convergence due to the small size of the sample data.

[10] That is, for a sample set of 20 sentences this boils down to 20-fold cross-validation.

state-of-the-art neural-based parsers (Zeman et al., 2017).

Our efforts to improve the preprocessing scores paid off. On most of the languages where we applied our word and/or sentence segmentation we achieved the best parsing results. On the secondary evaluation metrics we ranked first for word segmentation and sentence segmentation, second for POS tagging, and first for morphological tagging. Additionally we were second on parsing the surprise languages.

As it turns out, all PUD treebanks were presumably annotated following the guidelines of the primary treebanks. This most likely lowered our results a little bit for some of the PUD treebanks. However, for Russian PUD our results are abnormally low compared to many other participants. We scored about 13 points behind the top result, in comparison to an average distance of less than 2 points. This is most likely an artifact of how the non-primary (SynTagRus) Russian treebank is considerably larger than the primary Russian treebank, which means that a parser trained on the concatenation is mostly dominated by the SynTagRus annotation style and domains.

9 Conclusion

We have presented the IMS contribution to the CoNLL 2017 UD Shared Task. We have shown that tuning the preprocessing methods is a way to achieve competitive parsing performance. We made use of a CRF tagger for POS and morphological features and very strong word and sentence segmentation tools.

None of our baseline parsers alone would rank third. We therefore used blending to combine them. In general, we can confirm the observation of Surdeanu and Manning (2010) that the diversity of parsers is important. Additionally, we observed that both the diversity of parser architectures and number of instances of the same parser can improve performance. Furthermore, our automatized combination search method could be seen as a case of a "sparsely" weighted voting scheme.

We confirmed two of our previous findings on a larger scale. (1) Syntactic information can help sentence segmentation (Björkelund et al., 2016). (2) Supertags improve parsing performance across all languages (Faleńska et al., 2015).

For the surprise languages we blended delexicalized models from other languages with lexicalized models trained on the small in-language sample data. This approach seems to have been robust and rendered us second rank for surprise languages. However, further analysis would be required in order to understand whether the lexicalized or delexicalized models in general fare better in this setting.

As for the PUD treebanks we found that, although the UD annotation scheme should be consistent across treebanks, combining training sets for one language is not useful for parsing the PUD test sets. Whether this depends on annotation idiosyncrasies or domain differences is an open question and deserves further attention.

Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) via the SFB 732, project D8. We express our gratitude to the organizers of the Shared Task (Zeman et al., 2017), the treebank providers (Nivre et al., 2017) and baseline system authors (Straka et al., 2016). We also thank our colleagues Özlem Çetinoğlu and Kyle Richardson for the contribution of ideas.

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)*. Association for Computational Linguistics, Sofia, Bulgaria, pages 604–609. http://www.aclweb.org/anthology/P13-2107.

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*. Association for Computational Linguistics, Gothenburg, Sweden, pages 159–163. http://www.aclweb.org/anthology/E14-4031.

Anders Björkelund, Agnieszka Faleńska, Wolfgang Seeker, and Jonas Kuhn. 2016. How to train dependency parsers with inexact search for joint sentence boundary detection and parsing of entire documents. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 1924–1934. http://www.aclweb.org/anthology/P16-1181.

Anders Björkelund and Jonas Kuhn. 2014. Learning structured perceptrons for coreference resolu-

	Words	Sent.	UPOS	Feats	LAS	Rank
ar⊗⊙	95.53	77.71	90.62	87.15	72.90	1
bg	99.91	92.83	97.95	96.47	87.65	3
ca	99.97	98.95	97.97	97.16	87.74	4
cs	99.90	92.03	98.44	93.14	86.39	5
cs_cac	99.99	100.0	98.76	90.72	86.99	3
cs_cltt	99.35	95.06	96.79	87.88	80.67	4
cu⊙	99.96	50.44	94.94	88.90	76.84	1
da	99.69	79.36	95.27	94.83	79.52	3
de	99.65	79.11	92.24	83.11	75.47	3
el	99.88	90.79	96.53	91.37	84.96	3
en⊙	98.67	74.72	93.29	94.40	78.71	5
en_lines	99.94	85.84	95.08	99.94	78.25	4
en_partut	99.49	97.51	93.32	92.69	79.37	5
es	99.69	94.15	95.53	96.34	83.15	9
es_ancora	99.95	97.05	98.19	97.72	87.12	5
et⊙	99.77	85.21	89.50	84.62	67.60	3
eu	99.96	99.58	94.01	89.57	77.97	3
fa	99.64	98.00	96.21	96.34	83.34	3
fi	99.63	84.56	95.15	92.43	81.21	3
fi_ftb	99.88	83.83	92.80	93.43	81.33	3
fr⊗	99.46	93.59	96.22	96.12	83.82	3
fr_sequoia⊗	99.49	83.75	96.61	96.10	85.40	4
gl	99.92	96.15	96.87	99.75	81.60	3
got⊙	100.0	41.65	95.03	88.36	71.36	1
grc	99.95	98.43	87.59	88.00	68.23	2
grc_proiel⊙	100.0	51.38	96.48	90.24	75.28	1
he⊗	91.37	99.39	87.34	85.06	68.16	1
hi	100.0	99.20	96.28	91.03	90.41	2
hr	99.93	96.92	96.48	85.82	82.51	3
hu	99.82	93.85	92.63	72.61	73.55	3
id	99.99	91.15	93.42	99.45	77.70	3
it	99.73	97.10	97.43	97.37	87.85	3
ja⊗	91.68	94.92	89.07	91.66	78.21	5
ko	99.73	93.05	93.74	99.34	79.51	3
la_ittb⊙	99.99	93.37	97.41	94.27	84.09	2
la_proiel⊙	100.0	40.63	95.63	89.22	71.55	1
lv	98.91	98.59	89.72	84.14	68.03	3
nl	99.88	77.14	91.38	90.04	75.07	3
nl_lassysmall⊙	99.93	84.59	97.61	97.55	86.86	2
no_bokmaal	99.75	95.76	97.12	95.56	85.98	5
no_nynorsk	99.85	91.23	96.80	95.25	85.05	4
pl	99.88	98.91	96.35	86.53	86.75	3
pt	99.52	89.79	96.58	94.62	85.11	2
pt_br	99.84	96.84	97.36	99.73	87.10	7
ro	99.64	93.42	96.86	96.24	83.50	3
ru	99.91	96.42	95.45	87.27	81.49	3
ru_syntagrus	99.57	97.81	98.18	94.55	89.80	3
sk	100.0	83.53	94.60	81.23	80.53	3
sl	99.96	99.24	96.90	90.08	85.86	4
sv	99.84	96.37	96.10	95.15	82.28	3
sv_lines	99.98	86.44	94.40	99.98	78.88	3
tr	97.89	96.63	91.54	86.82	62.39	3
ur	100.0	98.32	92.98	81.03	80.93	3
vi⊗	86.67	92.59	77.88	86.33	47.51	1
zh⊗	92.81	98.19	86.33	91.71	68.56	1
average	99.01	88.96	94.45	91.75	79.60	3

(a) Big treebanks.

	Words	Sent.	UPOS	Feats	LAS	Rank
bxr	99.35	91.81	84.12	81.65	32.24	1
hsb	99.84	90.69	90.30	74.20	61.67	2
kmr	98.85	97.02	90.04	81.61	46.70	2
sme	99.88	98.79	86.81	81.93	40.67	2
average	99.48	94.58	87.82	79.85	45.32	2

(b) Surprise treebanks. All preprocessing by UDPipe.

	Words	Sent.	UPOS	Feats	LAS	Rank
ar_pud⊗⊙	93.32	96.79	73.89	24.66	49.94	1
cs_pud	99.29	96.43	95.76	89.89	81.00	5
de_pud	98.00	86.49	84.53	31.67	71.88	3
en_pud	99.66	97.13	93.95	89.27	81.55	4
es_pud	99.47	93.42	88.48	40.47	78.63	9
fi_pud	99.61	93.67	96.77	95.35	85.21	3
fr_pud⊗	98.87	92.32	58.62	36.83	77.60	3
hi_pud	97.81	90.83	83.68	16.84	53.57	4
it_pud	99.17	96.58	93.22	57.34	86.16	4
ja_pud	93.44	94.89	91.07	54.44	81.98	3
pt_pud	99.42	95.65	88.36	43.51	75.53	5
ru_pud	97.18	98.95	87.19	33.35	62.72	21
sv_pud	98.26	90.20	90.32	53.47	74.41	3
tr_pud	96.62	93.91	71.39	23.62	38.23	1
average	97.87	94.09	87.69	50.89	71.31	3

(c) PUD treebanks.

	Words	Sent.	UPOS	Feats	LAS	Rank
fr_partut⊗	99.56	98.00	95.92	93.50	83.82	4
ga	99.29	95.81	89.99	80.05	69.22	3
gl_treegal	98.62	81.63	92.32	91.31	71.30	3
kk	94.91	81.38	58.48	45.49	25.29	5
la⊙	99.99	93.75	84.41	75.56	51.82	5
sl_sst⊙	99.82	21.41	90.91	84.82	55.88	3
ug	98.52	63.55	74.96	98.52	43.51	1
uk	99.81	92.59	89.68	74.88	69.27	3
average	98.81	78.52	84.58	80.52	58.76	3

(d) Small treebanks.

Table 7: Test results. The treebanks for which we did our own word and/or sentence segmentation are marked with ⊗ and ⊙ respectively.

tion with latent antecedents and non-local features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Baltimore, Maryland, pages 47–57. http://www.aclweb.org/anthology/P14-1005.

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*. Association for Computational Linguistics, Bilbao, Spain, pages 76–86. http://www.aclweb.org/anthology/W15-2210.

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)*. Coling 2010 Organizing Committee, Beijing, China, pages 89–97. http://www.aclweb.org/anthology/C10-1011.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*. Association for Computational Linguistics, Prague, Czech Republic, pages 957–961. http://www.aclweb.org/anthology/D/D07/D07-1101.

Yoeng-jin Chu and Tseng-hong Liu. 1965. On the shortest aborescence of a directed graph. *Science Sinica* 14:1396–1400.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards* 71(B):233–240.

Jason Eisner. 1997. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 5th International Workshop on Parsing Technologies (IWPT)*. MIT, Cambridge, MA, pages 54–65. http://cs.jhu.edu/ jason/papers/#eisner-1997-iwpt.

Agnieszka Faleńska, Anders Björkelund, Özlem Çetinoğlu, and Wolfgang Seeker. 2015. Stacking or supertagging for dependency parsing – what's the difference? In *Proceedings of the 14th International Conference on Parsing Technologies*. Association for Computational Linguistics, Bilbao, Spain, pages 118–129. http://www.aclweb.org/anthology/W15-2215.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Montréal, Canada, pages 142–151. http://www.aclweb.org/anthology/N12-1015.

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*. Association for Computational Linguistics, Stroudsburg, PA, USA, COLING '94, pages 154–160. https://doi.org/10.3115/991886.991912.

A. Martins, M. Almeida, and N. A. Smith. 2013. "turning on the turbo: Fast third-order non-projective turbo parsers". In *Annual Meeting of the Association for Computational Linguistics - ACL*. volume -, pages 617–622.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the ACL (EACL 2006)*. Association for Computational Linguistics, Trento, Italy, pages 81–88. http://www.aclweb.org/anthology-new/E/E06/E06-1011.pdf.

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

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*. Association for Computational Linguistics, Suntec, Singapore, pages 351–359. http://www.aclweb.org/anthology/P/P09/P09-1040.

Joakim Nivre, Željko Agić, Lars Ahrenberg, et al. 2017. Universal dependencies 2.0 CoNLL 2017 shared task development and test data. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University. http://hdl.handle.net/11234/1-2184.

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)*. Association for Computational Linguistics, Paris, France, pages 73–76. http://www.aclweb.org/anthology/W09-3811.

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*. Association for Computational Linguistics, Gothenburg, Sweden, pages 154–158. http://www.aclweb.org/anthology/E14-4030.

Martin Potthast, Tim Gollub, Francisco Rangel, Paolo Rosso, Efstathios Stamatatos, and Benno Stein. 2014. Improving the reproducibility of PAN's shared tasks: Plagiarism detection, author identification, and author profiling. In Evangelos Kanoulas, Mihai Lupu, Paul Clough, Mark Sanderson, Mark Hall, Allan Hanbury, and Elaine Toms, editors, *Information Access Evaluation meets Multilinguality, Multimodality, and Visualization. 5th*

International Conference of the CLEF Initiative (CLEF 14). Springer, Berlin Heidelberg New York, pages 268–299. https://doi.org/10.1007/978-3-319-11382-1_22.

Rudolf Rosa and Zdenek Zabokrtský. 2015. Klcpos3 - a language similarity measure for delexicalized parser transfer. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*. pages 243–249. http://aclweb.org/anthology/P/P15/P15-2040.pdf.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*. Association for Computational Linguistics, New York City, USA, pages 129–132. http://www.aclweb.org/anthology/N/N06/N06-2033.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, pos tagging and parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. European Language Resources Association (ELRA), Paris, France. http://www.lrec-conf.org/proceedings/lrec2016/pdf/873_Paper.pdf.

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*. Association for Computational Linguistics, pages 649–652.

Xiang Yu, Agnieszka Falenska, and Ngoc Thang Vu. 2017. A general-purpose tagger with convolutional neural networks. In *arXiv preprint arXiv:1706.01723*.

Xiang Yu and Ngoc Thang Vu. 2017. Character composition model with convolutional neural networks for dependency parsing on morphologically rich languages. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Vancouver, Canada.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gökırmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Hěctor Martínez Alonso, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadova, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Association for Computational Linguistics.

Daniel Zeman and Philip Resnik. 2008. Cross-language parser adaptation between related languages. In *Third International Joint Conference on Natural Language Processing, IJCNLP 2008, Hyderabad, India, January 7-12, 2008*. pages 35–42. http://aclweb.org/anthology/I/I08/I08-3008.pdf.

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1557–1566. https://doi.org/10.18653/v1/P16-1147.

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*. Association for Computational Linguistics, Honolulu, Hawaii, pages 562–571. http://www.aclweb.org/anthology/D08-1059.