Exploiting Dynamic Oracles to Train Projective Dependency Parsers on Non-Projective Trees

Lauriane Aufrant, Guillaume Wisniewski, François Yvon

Exploiting Dynamic Oracles to Train Projective Dependency Parsers on Non-Projective Trees

Lauriane Aufrant[1,2], **Guillaume Wisniewski**[1] and **François Yvon**[1]
[1]LIMSI, CNRS, Univ. Paris-Sud, Université Paris-Saclay, F-91405 Orsay, France
[2]DGA, 60 boulevard du Général Martial Valin, 75 509 Paris, France
{lauriane.aufrant,guillaume.wisniewski,francois.yvon}@limsi.fr

Abstract

Because the most common transition systems are projective, training a transition-based dependency parser often implies to either ignore or rewrite the non-projective training examples, which has an adverse impact on accuracy. In this work, we propose a simple modification of dynamic oracles, which enables the use of non-projective data when training projective parsers. Evaluation on 73 treebanks shows that our method achieves significant gains (+2 to +7 UAS for the most non-projective languages) and consistently outperforms traditional projectivization and pseudo-projectivization approaches.

1 Introduction

Because of their efficiency and ease of implementation, transition-based parsers are the most common systems for dependency parsing. However, efficiency comes at a price, namely a loss in expressivity: while graph-based parsers are able to produce any tree spanning the input sentence, many transition-based systems are restricted to projective trees. Informally, a dependency tree is *non-projective* if at least one dependency crosses another arc (see Figure 1).

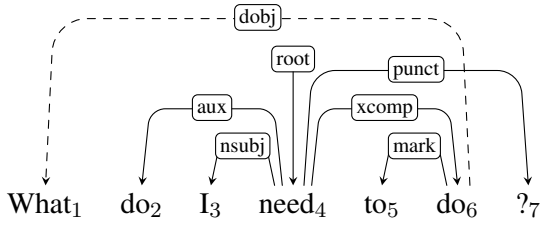

Figure 1: An example of non-projectivity: the tree is made non-projective by the dependency *What* ↶ *do*.

The inability to generate non-projective trees is an obvious issue for accuracy: at test time, a projective parser is guaranteed to be wrong for all the non-projective dependencies, a limitation already pointed out several times (Nivre, 2009; Lacroix and Béchet, 2014). In this paper, we show that the impact can also be severe *at training time*. This is because the standard training procedure assumes that the reference tree is within reach of the parser, which is not the case for non-projective examples. Therefore, projective parsers cannot make any use of such samples and common practice is to filter them out, thereby wasting potentially valuable training material. Depending on the annotation schemes and languages, between 5 and 10% of the training set are typically discarded.[1]

Several strategies have been proposed to overcome the projectivity constraint. One line of research is to sacrifice parsing efficiency and introduce special transition systems capable to build non-projective dependencies (Covington, 2001; Nivre, 2009). Another approach introduces non-projective dependencies by post-processing the output projective trees. This is the case of the pseudo-projectivization method (Nivre and Nilsson, 2005), which encodes crossings in augmented relation labels and makes all examples projective. The accuracy on projective dependencies alone can also be maximized by projectivizing all training examples prior to training, using Eisner (1996)'s decoder.

In this work, we propose an alternative strategy: we show (§3) that it is possible, with a small modification of the dynamic oracle of Goldberg and Nivre (2012), to directly train a projective parser with non-projective examples. While our approach remains unable to produce non-projective trees, it still results in significant improvements

[1]In UD 1.2, examples with non-projective dependencies represent 4.96% of trees (0.48% of dependencies) for English, 12.45% (0.83%) for French, 8.19% (0.33%) for Arabic. However, the numbers rise for some languages such as Dutch (30.87%, 4.10%) or Ancient Greek (63.22%, 9.78%). See Straka et al. (2015) for complete data on 37 treebanks.

Proceedings of NAACL-HLT 2018, pages 413–419
New Orleans, Louisiana, June 1 - 6, 2018.

on the overall UAS (§4), and consistently outperforms the (pseudo-)projectivization approaches.

2 Training transition-based parsers

2.1 Parsing with a transition system

In transition-based parsing (Nivre, 2003), dependency trees are built incrementally, in a shift-reduce manner: to parse a sentence, a sequence of transitions (the *derivation*) is applied on the internal state of the parser (the *configuration*), consisting typically in a stack and a buffer of unprocessed words. In the ARCEAGER transition system (Nivre, 2004), four actions are available, one for each possible transition (see Table 1).

At parsing time, each transition to follow is predicted in turn by a classifier, typically an averaged perceptron (Collins and Roark, 2004) or a neural network (Chen and Manning, 2014; Dyer et al., 2015; Andor et al., 2016), based on features extracted from the current parser configuration.

Compared to other parsing frameworks, such as graph-based parsing (McDonald et al., 2005), a major advantage of transition-based parsing is its computational efficiency: the processing time of a sentence is linear in its length.

2.2 Training with dynamic oracles

The classifier used to predict the actions is typically trained in an online fashion, using a dataset consisting of input sentences and reference parse trees. Various strategies have been envisioned to generate, based on that data, pairs of positive (gold) and negative (predicted) parser configurations with which to update the model. A recent and successful proposal uses so-called *dynamic oracles* (Goldberg and Nivre, 2012): the training example is parsed by the model, and for each predicted configuration in the resulting derivation, the dynamic oracle computes a reference action, tailored to the current configuration.

In practice, the reference is defined as an action which does not degrade the accuracy on that sentence: if a transition prevents a gold arc from being produced later (such as attaching a token to the wrong head), it is incorrect, but no error is flagged if that arc was already unreachable (for instance if the true head was already removed from the stack).

Formally, if each configuration is associated with a UAS_{max}, the maximum UAS value that can be achieved by any of its successor derivations, then the *action cost* is defined as the difference between the current UAS_{max} and the future UAS_{max} (once the corresponding action has been applied). The best decision in that situation is the one which ensures the best future UAS, ie. which leaves UAS_{max} unchanged and has zero cost. By definition of UAS_{max}, in all configurations at least one action has zero cost; there may even be several in case of spurious ambiguities. Hence, when asked for a reference action, the dynamic oracle simply returns the set of zero-cost actions.

The core of the method is thus the computation of action costs. In order to simplify it, Goldberg and Nivre (2013) introduce the concept of *arc decomposition*: a transition system is *arc-decomposable* if in every configuration, all the gold dependencies that are still reachable can be reached simultaneously by the same derivation. It ensues that for such systems, the action cost is simply the number of gold arcs that the action explicitly forbids, which are in general straightforward to enumerate. For instance in ARCEAGER, the REDUCE action (which pops the topmost stack element s) has a cost of 1 for each child of s still in the buffer, since they no longer can get their true head (Goldberg and Nivre, 2013).

Arc-decomposability does not always hold, however, in which case there are extra costs to take into account: by definition of non-arc-decomposable systems, some arcs are incompatible (they are not unreachable, they can simply not be reached together). Therefore, at some point, adding a gold arc will imply renouncing to another gold arc, thereby inserting an error. It is however incorrect to assign this cost to the given action, since it is due to a much earlier action which introduced the incompatibility.[2] As exemplified by Goldberg et al. (2014) and Gómez-Rodríguez and Fernández-González (2015), it is not impossible to derive dynamic oracles for non-arc-decomposable systems, but taking this kind of incompatibilities into account makes the computation of their action

[2] A straightforward example of non-arc-decomposable system is the ARCSTANDARD system (Nivre, 2003). It consists in three actions: SHIFT (the same as in ARCEAGER), and LEFT and RIGHT actions, which link the two topmost stack elements in either direction, and then remove the child from the stack. For the reference tree 'The ↶ bag ↶ fell', in the configuration where the buffer is empty and the stack is [*The bag fell*], the only way to output the *bag* ↶ *fell* dependency is consequently to attach (and pop) '*bag*' immediately, thereby renouncing to its child '*The*', even tough it is currently reachable (with RIGHT+LEFT). In this configuration all decisions seem to degrade the UAS, but the actual error was done when SHIFTing '*fell*' before attaching '*The*'.

SHIFT	[S]	$(\sigma,$	$b\|\beta,$	$P)$	$\Rightarrow$	$(\sigma\|b,$	$\beta,$	$P)$	if b is a word
LEFT	[L]	$(\sigma\|s,$	$b\|\beta,$	$P)$	$\Rightarrow$	$(\sigma,$	$b\|\beta,$	$P + (b \rightarrow s))$	if s is a word and s is unattached
RIGHT	[R]	$(\sigma\|s,$	$b\|\beta,$	$P)$	$\Rightarrow$	$(\sigma\|s\|b,$	$\beta,$	$P + (s \rightarrow b))$	
REDUCE	[RE]	$(\sigma\|s,$	$\beta,$	$P)$	$\Rightarrow$	$(\sigma,$	$\beta,$	$P)$	if s is attached

Table 1: The ARCEAGER transition system: semantics and preconditions of each action. σ stands for the stack, s for the topmost stack element, β stands for the buffer, b for the first buffer element, and P is the partially built tree.

costs much more complex.

3 Using dynamic oracles to train on non-projective data

The reason why dynamic oracles can help solving the non-projectivity issue is that non-projective examples, in this framework, are not different from projective ones: the cost is well-defined for any action, and by definition there is always at least one zero-cost action. So, all training examples are usable by design, regardless of their projectivity. The issue rather resides in deriving a sound definition of the cost, which covers non-projective cases;[3] in the current state of the art, the dynamic oracles derived for projective systems are only sound for projective examples, though (see Figure 2).

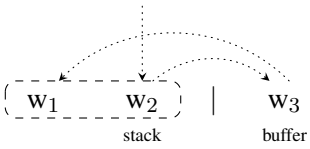

Figure 2: A configuration where all actions have non-zero cost (thereby contradicting its definition), for a non-projective reference tree (see dotted edges), when using the standard ARCEAGER dynamic oracle. The action cost is 2 for SHIFT (only w_2 can be correctly attached, by applying L+L+L afterwards), 2 for LEFT (L+S+L correctly attaches w_1 only) and 1 for RIGHT (RE+L+L correctly attaches both w_2 and w_3).

One way to look at non-projective examples is as a set of configurations containing arc incompatibilities: when two crossing edges are reachable, only one can actually belong to the final output. Yet, this is a known setting: with non-arc-decomposable systems, some erroneous configurations face the same issue. Hence, from the oracle point of view, the initial empty configuration already comes with embedded 'past errors' (the incompatibilities due to edge crossings). As in non-arc-decomposable systems, the cost incurred by these incompatibilities is not due to actions to come, but should be attributed to previous actions, taken in a fictive history before the initial configuration. As such, the natural behavior of dynamic oracles is to ignore this cost. Hence, using the same methodology as for non-arc-decomposable systems, it is formally possible to define dynamic oracles for non-projective examples. But it implies enumerating all non-projective arc incompatibilities in an arbitrary parser configuration (and proving exhaustivity), which is a difficult task and remains, to date, an open question.

Instead of deriving exact costs, we propose here a straightforward strategy which *approximates* the action costs for non-projective examples: using the usual cost computation, but defining the oracles as *minimum-cost* actions instead of zero-cost ones. Indeed, when the parser ends up in a configuration where all decisions appear erroneous, the part of the cost which is common to all actions should in fact have been taken care of in the past; with this minimum-cost approach, it is ignored. In Figure 2, the RIGHT action is chosen as reference by the minimum-cost criterion, thereby acknowledging the fact that non-projectivity by itself incurs a cost of 1. This oracle generalizes the zero-cost one, as they are equivalent on projective trees.

Compared to an exact oracle, this approximated cost biases the oracle towards delaying the resolution of incompatibilities (like the SHIFT action in Figure 3). A few updates are consequently unsound, but empirically their impact remains small compared to the benefits of making more examples usable, as will be assessed in the next section.

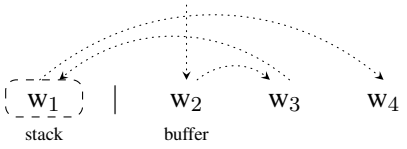

Figure 3: A configuration where action cost is poorly approximated, for a non-projective reference tree (see dotted edges). All gold arcs are reachable, but at most two can be reached simultaneously. The cost computed for LEFT is 2, 1 for RIGHT and 0 for SHIFT, even though all actions lead to a tree with two gold arcs.

[3] Exhaustive search would be a straightforward strategy to compute exact action costs in any setting, but it is computationally too expensive.

		% non-projective sentences				# training sentences	
# corresponding treebanks:	μ [73]	$>50\%$ [1]	25-50% [9]	10-25% [32]	$<10\%$ [31]	>500 [57]	<500 [16]
PANPARSER – greedy ARCEAGER							
Static oracle (only projective snt.)	78.28	56.23	76.22	75.49	82.47	81.34	67.37
Static oracle + projectivized snt.	+0.31	+2.12	+1.54	+0.15	+0.06	+0.50	-0.36
Dynamic oracle (only projective snt.)	78.94	57.75	76.99	76.26	82.97	81.92	68.34
Dynamic oracle + projectivized snt.	+0.32	+1.06	+1.04	+0.52	-0.11	+0.24	+0.61
Dynamic oracle + pseudo-proj. snt.	+0.26	+2.01	+1.49	+0.19	-0.07	+0.47	-0.45
Dynamic oracle + non-projective snt.	+0.48	+2.43	+1.83	+0.44	+0.07	+0.52	+0.36
PANPARSER – **beam** ARCEAGER							
Global dynamic oracle (only projective snt.)	79.77	57.12	78.28	76.83	83.96	83.12	67.84
Global dynamic oracle + non-projective snt.	80.40	61.49	80.11	77.54	84.04	83.63	68.87
	+0.63	+4.38	+1.84	+0.71	+0.07	+0.51	+1.03
PANPARSER – greedy **ARCHYBRID**							
Static oracle (only projective snt.)	75.71	53.08	73.66	73.19	79.63	78.29	66.51
Dynamic oracle (only projective snt.)	76.51	54.23	74.61	73.96	80.41	79.23	66.81
Dynamic oracle + non-projective snt.	+0.55	+3.08	+2.16	+0.33	+0.22	+0.53	+0.61
MALTPARSER – greedy ARCEAGER							
Baseline (only projective snt.)	72.88	57.88	71.74	69.99	76.68	76.82	58.87
+ pseudo-projectivized sentences	+0.37	+5.84	+1.41	+0.19	+0.08	+0.48	-0.01
+ non-proj. output (pseudo-proj. + deproj.)	+0.45	+6.83	+1.69	+0.25	+0.09	+0.59	-0.04
UDPIPE – **tuned** hyperparameters							
Baseline (proj. and non-proj. parsers)	79.47	66.99	81.20	75.51	83.45	83.67	64.48

Table 2: Comparison on Universal Dependencies 2.0 of various strategies to handle non-projective training examples, depending on the non-projectivity rate and on treebank size. We report the average UAS over the corresponding sets of languages. All UAS gains are computed with respect to their '*only projective snt.*' baseline.

4 Experiments

The benefits of non-projective examples for training projective parsers are evaluated on the 73 treebanks of the UD 2.0 (Nivre et al., 2017b,a). Three methods to exploit non-projective trees (instead of discarding them) are contrasted: learning on the trees projectivized using Eisner (1996)'s algorithm, learning on pseudo-projectivized examples (Nivre and Nilsson, 2005) and learning on the non-projective trees, with the minimum-cost oracle described in §3. Projectivization is based on Yoav Goldberg's code.[4] For pseudo-projectivization, the MALTPARSER 1.9 implementation is used, with the `head` encoding scheme. For parsing, we use PANPARSER (Aufrant and Wisniewski, 2016), our own open source[5] implementation of a greedy ARCEAGER parser (using an averaged perceptron and a dynamic oracle).

As shown in Table 2, it is empirically better to handle non-projective sentences with minimum-cost dynamic oracles than to discard them all; but this strategy also outperforms projectivization and pseudo-projectivization. As expected, the gains of all methods increase when the proportion of non-projectivity increases, i.e. when more examples would have been discarded.

The minimum-cost technique is notably effective on the least projective treebanks, which correspond to ancient languages like Ancient Greek (63% of non-projective examples, with a gain of +2.4 UAS, compared to +1.1 and +2.0 for projectivization and pseudo-projectivization) and Latin (41%, +2.0 vs +0.4/+2.8); but it also achieves large improvements for modern languages with less non-projectivity, such as Dutch-LassySmall (30%, +7.0 vs +5.7/+3.3), Belarusian (17%, +5.2 vs +2.2/+4.4) and Turkish (11%, +2.3 vs +1.9/+1.3).

Apart from higher gains on average, the advantage of the minimum-cost strategy is that

[4] https://www.cs.bgu.ac.il/~yoavg/software/projectivize

[5] https://perso.limsi.fr/aufrant

it is consistently beneficial, whereas pseudo-projectivization is detrimental for small treebanks. A plausible explanation is that arbitrarily rewriting the trees introduces inconsistencies in the training material, which are only alleviated when data is large enough. In that regard, the opposite effects of projectivization (detrimental with a static oracle, beneficial with a dynamic one) highlight the limited reliability of such transformations.

The minimum-cost strategy is also applied to an improved version of PANPARSER, using beam search and a dynamic oracle extended to global training (Aufrant et al., 2017), with a beam of size 8 and the max-violation strategy. The minimum-cost criterion appears particularly fit for that setting, with even larger gains (+0.63 UAS on average) despite a higher baseline.

Comparison with other parsers For illustrative purposes, similar experiments are conducted with other parsing systems: the ARCHYBRID version of PANPARSER, MALTPARSER and UDPIPE. MALTPARSER is the original implementation of the ARCEAGER system, but differs from ours in several ways, notably feature templates and the oracle (which is not dynamic, but precomputed statically); to help comparison, additional results are reported for PANPARSER without dynamic oracles. UDPIPE is a state-of-the-art neural parser including both projective and non-projective parsing systems; we use version 1.1 (Straka and Straková, 2017) with Straka (2017)'s set of tuned hyperparameters, but without their pre-trained word embeddings, for fair comparison.

The ARCHYBRID results show that the gains achieved by the minimum-cost criterion are not specific to the ARCEAGER system: despite different baseline scores, the proposed strategy yields similar improvements.

Compared to MALTPARSER, our ARCEAGER baseline appears much stronger (+5.4 UAS) on the downsized datasets; but the gains achieved when exploiting the non-projective trees (with pseudo-projectivization) are similar in both implementations. There is one exception, Ancient Greek (the only treebank with more than 50% non-projective sentences), for which the MALTPARSER gains are way larger than those of PANPARSER; but this treebank seems particular in several regards[6] and consequently does not question the superiority of the minimum-cost oracle over the pseudo-projectivization strategy, measured even in Ancient Greek for PANPARSER.

Table 2 also reports the gains achieved by MALTPARSER when pseudo-projectivization is followed by deprojectivization of the output. Plain comparison of this line with the minimum-cost strategy is delicate, because it does not result from better training only, but also from a gain in expressivity: it is able to retrieve even non-projective dependencies. But it is interesting to see that deprojectivization only marginally improves over pseudo-projectivization alone: most of the gain actually resides in the treebank augmentation rather than in retrieving non-projective dependencies. Besides, the minimum-cost strategy outperforms even the deprojectivized results.

Finally, measures with UDPIPE reveal that, even though it benefits a lot from its higher expressivity (as it uses non-projective systems for the most non-projective treebanks), it achieves low accuracies on small treebanks and is thus outperformed on average by the beam version of PANPARSER (+0.30 UAS) – and the minimum-cost criterion significantly widens that gap (+0.97 UAS).

5 Conclusion

This work has addressed the restriction of projective parsers to train only on projective examples. We have explained how the dynamic oracle framework can help overcoming this issue, and shown that a simple modification of the framework (using minimum-cost actions as references instead of zero-cost ones) enables a seamless use of non-projective examples. Compared to the traditional (pseudo-)projectivization approaches, this method provides higher and more reliable improvements over the filtering baseline.

Acknowledgments

This work has been partly funded by the French *Direction générale de l'armement* and by the *Agence Nationale de la Recherche* under ParSiTi project (ANR-16-CE33-0021) and MultiSem project (ANR-16-CE33-0013).

[6]The baseline MALTPARSER already behaves differently for that language: it slightly outperforms the baseline PANPARSER instead of underperforming it by a large margin. The explanation may simply lie in other differences between implementations, for instance MALTPARSER's feature templates may be particularly suited to Ancient Greek.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally Normalized Transition-Based Neural Networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany. Association for Computational Linguistics.

Lauriane Aufrant and Guillaume Wisniewski. 2016. PanParser: a Modular Implementation for Efficient Transition-Based Dependency Parsing. Technical report, LIMSI-CNRS.

Lauriane Aufrant, Guillaume Wisniewski, and François Yvon. 2017. Don't Stop Me Now! Using Global Dynamic Oracles to Correct Training Biases of Transition-Based Dependency Parsers. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 318–323, Valencia, Spain. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Michael Collins and Brian Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain.

Michael A. Covington. 2001. A Fundamental Algorithm for Dependency Parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China. Association for Computational Linguistics.

Jason M Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2012. A Dynamic Oracle for Arc-Eager Dependency Parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India. The COLING 2012 Organizing Committee.

Yoav Goldberg and Joakim Nivre. 2013. Training Deterministic Parsers with Non-Deterministic Oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414.

Yoav Goldberg, Francesco Sartorio, and Giorgio Satta. 2014. A Tabular Method for Dynamic Oracles in Transition-Based Parsing. *Transactions of the Association for Computational Linguistics*, 2:119–130.

Carlos Gómez-Rodríguez and Daniel Fernández-González. 2015. An Efficient Dynamic Oracle for Unrestricted Non-Projective Parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 256–261, Beijing, China. Association for Computational Linguistics.

Ophélie Lacroix and Denis Béchet. 2014. A Three-Step Transition-Based System for Non-Projective Dependency Parsing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 224–232, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Joakim Nivre. 2003. An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies*, IWPT 2003, Nancy, France.

Joakim Nivre. 2004. Incrementality in Deterministic Dependency Parsing. In *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain. 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. Association for Computational Linguistics.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Cristina Bosco, Gosse Bouma, Sam Bowman, Aljoscha Burchardt, Marie Candito, Gauthier Caron, Gülşen

Cebirolu Eryiit, Giuseppe G. A. Celano, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Silvie Cinková, Çar Çöltekin, Miriam Connor, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Marhaba Eli, Ali Elkahky, Tomaž Erjavec, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökrmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà M, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Petter Hohle, Radu Ion, Elena Irimia, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşkara, Hiroshi Kanayama, Jenna Kanerva, Tolga Kayadelen, Václava Kettnerová, Jesse Kirchner, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Tatiana Lando, Phng Lê Hng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Shunsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, Pinkey Nainwani, Anna Nedoluzhko, Lng Nguyn Th, Huyn Nguyn Th Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Martin Popel, Lauma Pretkalnia, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Livy Real, Siva Reddy, Georg Rehm, Larissa Rinaldi, Laura Rituma, Rudolf Rosa, Davide Rovati, Shadi Saleh, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Antonio Stella, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Gertjan van Noord, Viktor Varga, Veronika Vincze, Jonathan North Washington, Zhuoran Yu, Zdeněk Žabokrtský, Daniel Zeman, and Hanzhi Zhu. 2017a. 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.

Joakim Nivre, Željko Agić, Lars Ahrenberg, et al. 2017b. Universal Dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan. Association for Computational Linguistics.

Milan Straka. 2017. CoNLL 2017 Shared Task - UDPipe Baseline Models and Supplementary Materials. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University.

Milan Straka, Jan Hajič, Jana Straková, and Jan Hajič jr. 2015. Parsing Universal Dependency Treebanks Using Neural Networks and Search-Based Oracle. In *International Workshop on Treebanks and Linguistic Theories (TLT14)*, pages 208–220.

Milan Straka and Jana Straková. 2017. Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics.