Non-Projective Dependency Parsing with Non-Local Transitions

Daniel Fernández-González and **Carlos Gómez-Rodríguez**
Universidade da Coruña
FASTPARSE Lab, LyS Research Group, Departamento de Computación
Campus de Elviña, s/n, 15071 A Coruña, Spain
d.fgonzalez@udc.es, carlos.gomez@udc.es

Abstract

We present a novel transition system, based on the Covington non-projective parser, introducing non-local transitions that can directly create arcs involving nodes to the left of the current focus positions. This avoids the need for long sequences of No-Arc transitions to create long-distance arcs, thus alleviating error propagation. The resulting parser outperforms the original version and achieves the best accuracy on the Stanford Dependencies conversion of the Penn Treebank among greedy transition-based parsers.

1 Introduction

Greedy transition-based parsers are popular in NLP, as they provide competitive accuracy with high efficiency. They syntactically analyze a sentence by greedily applying transitions, which read it from left to right and produce a dependency tree.

However, this greedy process is prone to error propagation: one wrong choice of transition can lead the parser to an erroneous state, causing more incorrect decisions. This is especially crucial for long attachments requiring a larger number of transitions. In addition, transition-based parsers traditionally focus on only two words of the sentence and their local context to choose the next transition. The lack of a global perspective favors the presence of errors when creating arcs involving multiple transitions. As expected, transition-based parsers build short arcs more accurately than long ones (McDonald and Nivre, 2007).

Previous research such as (Fernández-González and Gómez-Rodríguez, 2012) and (Qi and Manning, 2017) proves that the widely-used projective *arc-eager* transition-based parser of Nivre (2003) benefits from shortening the length of transition sequences by creating non-local attachments. In particular, they augmented the original transition system with new actions whose behavior entails more than one arc-eager transition and involves a context beyond the traditional two focus words. Attardi (2006) and Sartorio et al. (2013) also extended the *arc-standard* transition-based algorithm (Nivre, 2004) with the same success.

In the same vein, we present a novel unrestricted non-projective transition system based on the well-known algorithm by Covington (2001) that shortens the transition sequence necessary to parse a given sentence by the original algorithm, which becomes linear instead of quadratic with respect to sentence length. To achieve that, we propose new transitions that affect non-local words and are equivalent to one or more Covington actions, in a similar way to the transitions defined by Qi and Manning (2017) based on the arc-eager parser. Experiments show that this novel variant significantly outperforms the original one in all datasets tested, and achieves the best reported accuracy for a greedy dependency parser on the Stanford Dependencies conversion of the WSJ Penn Treebank.

2 Non-Projective Covington Parser

The original non-projective parser defined by Covington (2001) was modelled under the transition-based parsing framework by Nivre (2008). We only sketch this transition system briefly for space reasons, and refer to (Nivre, 2008) for details.

Parser configurations have the form $c = \langle \lambda_1, \lambda_2, B, A \rangle$, where λ_1 and λ_2 are lists of partially processed words, B a list (called buffer) of unprocessed words, and A the set of dependency arcs built so far. Given an input string $w_1 \cdots w_n$, the parser starts at the initial configuration $c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$ and runs transitions until a terminal configuration of the

Proceedings of NAACL-HLT 2018, pages 693–700
New Orleans, Louisiana, June 1 - 6, 2018.

Covington:	Shift:	$\langle \lambda_1, \lambda_2, j\|B, A\rangle \Rightarrow \langle \lambda_1 \cdot \lambda_2\|j, [], B, A\rangle$
	No-Arc:	$\langle \lambda_1\|i, \lambda_2, B, A\rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, B, A\rangle$
	Left-Arc:	$\langle \lambda_1\|i, \lambda_2, j\|B, A\rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{j \rightarrow i\}\rangle$ only if $\nexists x \mid x \rightarrow i \in A$ (single-head) and $i \rightarrow^* j \notin A$ (acyclicity).
	Right-Arc:	$\langle \lambda_1\|i, \lambda_2, j\|B, A\rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{i \rightarrow j\}\rangle$ only if $\nexists x \mid x \rightarrow j \in A$ (single-head) and $j \rightarrow^* i \notin A$ (acyclicity).
NL-Covington:	Shift:	$\langle \lambda_1, \lambda_2, j\|B, A\rangle \Rightarrow \langle \lambda_1 \cdot \lambda_2\|j, [], B, A\rangle$
	Left-Arc$_k$:	$\langle \lambda_1\|i_k\|...\|i_1, \lambda_2, j\|B, A\rangle \Rightarrow \langle \lambda_1, i_k\|...\|i_1\|\lambda_2, j\|B, A \cup \{j \rightarrow i_k\}\rangle$ only if $\nexists x \mid x \rightarrow i_k \in A$ (single-head) and $i_k \rightarrow^* j \notin A$ (acyclicity).
	Right-Arc$_k$:	$\langle \lambda_1\|i_k\|...\|i_1, \lambda_2, j\|B, A\rangle \Rightarrow \langle \lambda_1, i_k\|...\|i_1\|\lambda_2, j\|B, A \cup \{i_k \rightarrow j\}\rangle$ only if $\nexists x \mid x \rightarrow j \in A$ (single-head) and $j \rightarrow^* i_k \notin A$ (acyclicity).

Figure 1: Transitions of the non-projective Covington (top) and NL-Covington (bottom) dependency parsers. The notation $i \rightarrow^* j \in A$ means that there is a (possibly empty) directed path from i to j in A.

form $\langle \lambda_1, \lambda_2, [], A\rangle$ is reached: at that point, A contains the dependency graph for the input.[1]

The set of transitions is shown in the top half of Figure 1. Their logic can be summarized as follows: when in a configuration of the form $\langle \lambda_1|i, \lambda_2, j|B, A\rangle$, the parser has the chance to create a dependency involving words i and j, which we will call left and right focus words of that configuration. The Left-Arc and Right-Arc transitions are used to create a leftward ($i \leftarrow j$) or rightward arc ($i \rightarrow j$), respectively, between these words, and also move i from λ_1 to the first position of λ_2, effectively moving the focus to $i - 1$ and j. If no dependency is desired between the focus words, the No-Arc transition makes the same modification of λ_1 and λ_2, but without building any arc. Finally, the Shift transition moves the whole content of the list λ_2 plus j to λ_1 when no more attachments are pending between j and the words of λ_1, thus reading a new input word and placing the focus on j and $j + 1$. Transitions that create arcs are disallowed in configurations where this would violate the single-head or acyclicity constraints (cycles and nodes with multiple heads are not allowed in the dependency graph). Figure 3 shows the transition sequence in the Covington transition system which derives the dependency graph in Figure 2.

The resulting parser can generate arbitrary non-projective trees, and its complexity is $O(n^2)$.

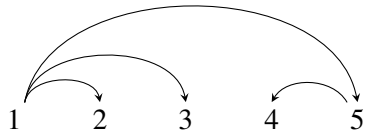

Figure 2: Dependency tree for an input sentence.

Tran.	λ_1	λ_2	Buffer	Arc
	[ ]	[ ]	[1, 2, 3, 4, 5]	
SH	[1]	[ ]	[2, 3, 4, 5]	
RA	[ ]	[1]	[2, 3, 4, 5]	$1 \rightarrow 2$
SH	[1, 2]	[ ]	[3, 4, 5]	
NA	[1]	[2]	[3, 4, 5]	
RA	[ ]	[1, 2]	[3, 4, 5]	$1 \rightarrow 3$
SH	[1, 2, 3]	[ ]	[4, 5]	
SH	[1, 2, 3, 4]	[ ]	[5]	
LA	[1, 2, 3]	[4]	[5]	$4 \leftarrow 5$
NA	[1, 2]	[3, 4]	[5]	
NA	[1]	[2, 3, 4]	[5]	
RA	[ ]	[1, 2, 3, 4]	[5]	$1 \rightarrow 5$
SH	[1, 2, 3, 4, 5]	[ ]	[ ]	

Figure 3: Transition sequence for parsing the sentence in Figure 2 using the Covington parser (LA=LEFT-ARC, RA=RIGHT-ARC, NA=NO-ARC, SH=SHIFT).

3 Non-Projective NL-Covington Parser

The original logic described by Covington (2001) parses a sentence by systematically traversing every pair of words. The Shift transition, introduced by Nivre (2008) in the transition-based version, is an optimization that avoids the need to apply a sequence of No-Arc transitions to empty the list λ_1 before reading a new input word.

However, there are still situations where sequences of No-Arc transitions are needed. For example, if we are in a configuration C with focus words i and j and the next arc we need to create

[1] Note that, in general, A is a forest, but it can be converted to a tree by linking headless nodes as dependents of an artificial root node at position 0.

goes from j to $i-k$ $(k>1)$, then we will need $k-1$ consecutive No-Arc transitions to move the left focus word to i and then apply Left-Arc. This could be avoided if a non-local Left-Arc transition could be undertaken directly at C, creating the required arc and moving k words to λ_2 at once. The advantage of such approach would be twofold: (1) less risk of making a mistake at C due to considering a limited local context, and (2) shorter transition sequence, alleviating error propagation.

We present a novel transition system called *NL-Covington* (for "non-local Covington"), described in the bottom half of Figure 1. It consists in a modification of the non-projective Covington algorithm where: (1) the Left-Arc and Right-Arc transitions are parameterized with k, allowing the immediate creation of any attachment between j and the kth leftmost word in λ_1 and moving k words to λ_2 at once, and (2) the No-Arc transition is removed since it is no longer necessary.

This new transition system can use some restricted global information to build non-local dependencies and, consequently, reduce the number of transitions needed to parse the input. For instance, as presented in Figure 4, the NL-Covington parser will need 9 transitions, instead of 12 traditional Covington actions, to analyze the sentence in Figure 2.

In fact, while in the standard Covington algorithm a transition sequence for a sentence of length n has length $O(n^2)$ in the worst case (if all nodes are connected to the first node, then we need to traverse every node to the left of each right focus word); for NL-Covington the sequence length is always $O(n)$: one Shift transition for each of the n words, plus one arc-building transition for each of the $n-1$ arcs in the dependency tree. Note, however, that this does not affect the parser's time complexity, which is still quadratic as in the original Covington parser. This is because the algorithm has $O(n)$ possible transitions to be scored at each configuration, while the original Covington has $O(1)$ transitions due to being limited to creating local leftward/rightward arcs between the focus words.

The completeness and soundness of NL-Covington can easily be proved as there is a mapping between transition sequences of both parsers, where a sequence of $k-1$ No-Arc and one arc transition in Covington is equivalent to a Left-Arc$_k$ or Right-Arc$_k$ in NL-Covington.

Tran.	λ_1	λ_2	Buffer	Arc
	[ ]	[ ]	[1, 2, 3, 4, 5]	
SH	[1]	[ ]	[2, 3, 4 , 5]	
RA_1	[ ]	[1]	[2, 3, 4 , 5]	$1 \rightarrow 2$
SH	[1, 2]	[ ]	[3, 4, 5]	
RA_2	[ ]	[1, 2]	[3, 4, 5]	$1 \rightarrow 3$
SH	[1, 2, 3]	[ ]	[4, 5]	
SH	[1, 2, 3, 4]	[ ]	[5]	
LA_1	[1, 2, 3]	[4]	[5]	$4 \leftarrow 5$
RA_3	[ ]	[1, 2, 3, 4]	[5]	$1 \rightarrow 5$
SH	[1, 2, 3, 4, 5]	[ ]	[ ]	

Figure 4: Transition sequence for parsing the sentence in Figure 2 using the NL-Covington parser (LA=LEFT-ARC, RA=RIGHT-ARC, SH=SHIFT).

4 Experiments

4.1 Data and Evaluation

We use 9 datasets[2] from the CoNLL-X (Buchholz and Marsi, 2006) and all datasets from the CoNLL-XI shared task (Nivre et al., 2007). To compare our system to the current state-of-the-art transition-based parsers, we also evaluate it on the Stanford Dependencies (de Marneffe and Manning, 2008) conversion (using the Stanford parser v3.3.0)[3] of the WSJ Penn Treebank (Marcus et al., 1993), hereinafter PT-SD, with standard splits. Labelled and Unlabelled Attachment Scores (LAS and UAS) are computed excluding punctuation only on the PT-SD, for comparability. We repeat each experiment with three independent random initializations and report the average accuracy. Statistical significance is assessed by a paired test with 10,000 bootstrap samples.

4.2 Model

To implement our approach we take advantage of the model architecture described in Qi and Manning (2017) for the *arc-swift* parser, which extends the architecture of Kiperwasser and Goldberg (2016) by applying a biaffine combination during the featurization process. We implement both the Covington and NL-Covington parsers under this architecture, adapt the featurization process with biaffine combination of Qi and Manning (2017) to these parsers, and use their same training

[2]We excluded the languages from CoNLL-X that also appeared in CoNLL-XI, i.e., if a language was present in both shared tasks, we used the latest version.

[3]https://nlp.stanford.edu/software/lex-parser.shtml

	Covington		NL-Covington	
Language	UAS	LAS	UAS	LAS
Arabic	66.67	53.24	**68.69**	**54.59**
Basque	74.31	66.18	**75.45**	**67.61**
Catalan	91.93	86.12	**92.60**	**86.99**
Chinese	83.87	76.19	**85.25**	**77.56**
Czech	84.27	77.91	**86.26**	**79.95**
English	89.94	88.74	**91.51**	**90.47**
Greek	79.91	72.65	**80.61**	**73.41**
Hungarian	76.80	65.21	**78.57**	**67.51**
Italian	82.03	75.87	**83.63**	**78.03**
Turkish	80.29	70.68	**81.30**	**71.28**
Bulgarian	81.78	76.23	**83.65**	**78.40**
Danish	86.56	81.18	**88.40**	**82.77**
Dutch	86.19	82.24	**87.45**	**83.76**
German	85.72	82.28	**87.24**	**83.92**
Japanese	92.20	90.41	**93.63**	**91.65**
Portuguese	86.69	82.19	**87.89**	**83.69**
Slovene	76.07	66.81	**77.83**	**69.74**
Spanish	74.67	69.41	**76.58**	**71.60**
Swedish	74.65	64.67	**75.62**	**65.95**
Average	81.82	75.17	**83.27**	**76.78**

Table 1: Parsing accuracy (UAS and LAS, including punctuation) of the Covington and NL-Covington non-projective parsers on CoNLL-XI (first block) and CoNLL-X (second block) datasets. Best results for each language are shown in bold. All improvements in this table are statistically significant ($\alpha = .05$).

setup. More details about these model parameters are provided in Appendix A.

Since this architecture uses batch training, we train with a static oracle. The NL-Covington algorithm has no spurious ambiguity at all, so there is only one possible static oracle: canonical transition sequences are generated by choosing the transition that builds the shortest pending gold arc involving the current right focus word j, or Shift if there are no unbuilt gold arcs involving j.

We note that a dynamic oracle can be obtained for the NL-Covington parser by adapting the one for standard Covington of Gómez-Rodríguez and Fernández-González (2015). As NL-Covington transitions are concatenations of Covington ones, their loss calculation algorithm is compatible with NL-Covington. Apart from error exploration, this also opens the way to incorporating non-monotonicity (Fernández-González and Gómez-Rodríguez, 2017). While these approaches have shown to improve accuracy under online training settings, here we prioritize homogeneous comparability to (Qi and Manning, 2017), so we use batch training and a static oracle, and still obtain state-of-the-art accuracy for a greedy parser.

Parser	Type	UAS	LAS
(Chen and Manning, 2014)	gs	91.8	89.6
(Dyer et al., 2015)	gs	93.1	90.9
(Weiss et al., 2015) greedy	gs	93.2	91.2
(Ballesteros et al., 2016)	gd	93.5	91.4
(Kiperwasser and Goldberg, 2016)	gd	93.9	91.9
(Qi and Manning, 2017)	gs	94.3	92.2
This work	gs	**94.5**	**92.4**
(Weiss et al., 2015) beam	b(8)	94.0	92.1
(Alberti et al., 2015)	b(32)	94.2	92.4
(Andor et al., 2016)	b(32)	94.6	92.8
(Shi et al., 2017)	dp	94.5	-
(Kuncoro et al., 2017) (constit.)	c	95.8	94.6

Table 2: Accuracy comparison of state-of-the-art transition-based dependency parsers on PT-SD. The "Type" column shows the type of parser: *gs* is a greedy parser trained with a static oracle, *gd* a greedy parser trained with a dynamic oracle, *b(n)* a beam search parser with beam size n, *dp* a parser that employs global training with dynamic programming, and *c* a constituent parser with conversion to dependencies.

4.3 Results

Table 1 presents a comparison between the Covington parser and the novel variant developed here. The NL-Covington parser outperforms the original version in all datasets tested, with all improvements statistically significant ($\alpha = .05$).

Table 2 compares our novel system with other state-of-the-art transition-based dependency parsers on the PT-SD. Greedy parsers are in the first block, beam-search and dynamic programming parsers in the second block. The third block shows the best result on this benchmark, obtained with constituent parsing with generative re-ranking and conversion to dependencies. Despite being the only non-projective parser tested on a practically projective dataset,[4] our parser achieves the highest score among greedy transition-based models (even above those trained with a dynamic oracle).

We even slightly outperform the arc-swift system of Qi and Manning (2017), with the same model architecture, implementation and training setup, but based on the projective arc-eager transition-based parser instead. This may be because our system takes into consideration any permissible attachment between the focus word j and any word in λ_1 at each configuration, while their approach is limited by the arc-eager logic: it al-

[4]Only 41 out of 39,832 sentences of the PT-SD training dataset present some kind of non-projectivity.

Language	Arc-swift UAS	Arc-swift LAS	NL-Covington UAS	NL-Covington LAS
Arabic	67.54	53.65	**68.69***	**54.59***
Basque	74.88	67.44	**75.45**	**67.61**
Catalan	**92.98**	**87.51***	92.60	86.99
Chinese	84.96	77.34	**85.25**	**77.56**
Czech	85.92	79.82	**86.26**	**79.95**
English	91.41	90.43	**91.51**	**90.47**
Greek	**81.64***	**74.56***	80.61	73.41
Hungarian	**78.70**	**69.27***	78.57	67.51
Italian	83.29	**78.60***	**83.63**	78.03
Turkish	79.56	70.22	**81.30***	**71.28***
Bulgarian	83.28	78.19	**83.65**	**78.40**
Danish	87.86	82.58	**88.40***	**82.77**
Dutch	83.27	80.14	**87.45***	**83.76***
German	86.28	82.97	**87.24***	**83.92***
Japanese	**93.64**	**91.92**	93.63	91.65
Portuguese	87.01	83.09	**87.89***	**83.69***
Slovene	**77.89**	69.37	77.83	**69.74**
Spanish	75.55	70.62	**76.58***	**71.60***
Swedish	75.00	65.66	**75.62**	**65.95**
Average	82.67	76.49	**83.27**	**76.78**

Table 3: Parsing accuracy (UAS and LAS, with punctuation) of the arc-swift and NL-Covington parsers on CoNLL-XI (1st block) and CoNLL-X (2nd block) datasets. Best results for each language are in bold. * indicates statistically significant improvements ($\alpha = .05$).

Language	Covington trans./sent.	NL-Covington trans./sent.
Arabic	194.80	**78.22**
Basque	46.74	**30.13**
Catalan	117.35	**60.07**
Chinese	19.12	**14.95**
Czech	60.62	**33.03**
English	78.01	**46.75**
Greek	89.23	**48.77**
Hungarian	68.54	**37.66**
Italian	63.67	**40.93**
Turkish	53.53	**30.08**
Bulgarian	51.35	**29.81**
Danish	66.77	**36.34**
Dutch	42.78	**28.93**
German	61.16	**31.89**
Japanese	24.30	**16.11**
Portuguese	76.14	**40.74**
Slovene	56.15	**31.79**
Spanish	109.70	**55.28**
Swedish	48.59	**29.07**
PTB-SD	81.65	**46.92**
Average	70.51	**38.37**

Table 4: Average transitions executed per sentence (trans./sent.) when analyzing each dataset by the original Covington and NL-Covington algorithms.

lows all possible rightward arcs (possibly fewer than our approach as the arc-eager stack usually contains a small number of words), but only one leftward arc is permitted per parser state. It is also worth noting that the arc-swift and NL-Covington parsers have the same worst-case time complexity, ($O(n^2)$), as adding non-local arc transitions to the arc-eager parser increases its complexity from linear to quadratic, but it does not affect the complexity of the Covington algorithm. Thus, it can be argued that this technique is better suited to Covington than to arc-eager parsing.

We also compare NL-Covington to the arc-swift parser on the CoNLL datasets (Table 3). For fairness of comparison, we projectivize (via maltparser[5]) all training datasets, instead of filtering non-projective sentences, as some of the languages are significantly non-projective. Even doing that, the NL-Covington parser improves over the arc-swift system in terms of UAS in 14 out of 19 datasets, obtaining statistically significant improvements in accuracy on 7 of them, and statistically significant decreases in just one.

Finally, we analyze how our approach reduces the length of the transition sequence consumed by the original Covington parser. In Table 4 we report the transition sequence length per sentence used by the Covington and the NL-Covington algorithms to analyze each dataset from the same benchmark used for evaluating parsing accuracy. As seen in the table, NL-Covington produces notably shorter transition sequences than Covington, with a reduction close to 50% on average.

5 Conclusion

We present a novel variant of the non-projective Covington transition-based parser by incorporating non-local transitions, reducing the length of transition sequences from $O(n^2)$ to $O(n)$. This system clearly outperforms the original Covington parser and achieves the highest accuracy on the WSJ Penn Treebank (Stanford Dependencies) obtained to date with greedy dependency parsing.

Acknowledgments

This work has received funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from the TELEPARES-UDC (FFI2014-51978-C2-2-R) and ANSWER-ASAP (TIN2017-85160-C2-1-R) projects from MINECO, and from Xunta de Galicia (ED431B 2017/01).

[5] http://www.maltparser.org/

References

Chris Alberti, David Weiss, Greg Coppola, and Slav Petrov. 2015. Improved transition-based parsing and tagging with neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. pages 1354–1359. http://aclweb.org/anthology/D/D15/D15-1159.pdf.

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, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. http://aclweb.org/anthology/P/P16/P16-1231.pdf.

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*. pages 166–170.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack-lstm parser. *CoRR* abs/1603.03793. http://arxiv.org/abs/1603.03793.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606* .

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*. pages 149–164. http://www.aclweb.org/anthology/W06-2920.

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)*. Doha, Qatar, pages 740–750. http://www.aclweb.org/anthology/D14-1082.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*. ACM, New York, NY, USA, pages 95–102.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The stanford typed dependencies representation. In *Coling 2008: Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*. Association for Computational Linguistics, Stroudsburg, PA, USA, CrossParser '08, pages 1–8. http://dl.acm.org/citation.cfm?id=1608858.1608859.

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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*. pages 334–343. http://aclweb.org/anthology/P/P15/P15-1033.pdf.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2012. Improving transition-based dependency parsing with buffer transitions. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, pages 308–319. http://aclweb.org/anthology/D/D12/D12-1029.pdf.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2017. A full non-monotonic transition system for unrestricted non-projective parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 288–298. http://aclweb.org/anthology/P17-1027.

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 (ACL-IJCNLP 2015). Volume 2: Short Papers*. Association for Computational Linguistics, Beijing, China, pages 256–261. http://www.aclweb.org/anthology/P15-2042.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks* pages 5–6.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL* 4:313–327. https://transacl.org/ojs/index.php/tacl/article/view/885.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Association for Computational Linguistics, pages 1249–1258. http://aclanthology.coli.uni-saarland.de/pdf/E/E17/E17-1117.pdf.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated

corpus of English: The Penn Treebank. *Computational Linguistics* 19:313–330.

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.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*. ACL/SIGPARSE, pages 149–160.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*. pages 50–57.

Joakim Nivre. 2008. Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics* 34(4):513–553. https://doi.org/10.1162/coli.07-056-R1-07-027.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*. pages 915–932. http://www.aclweb.org/anthology/D/D07/D07-1096.pdf.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. pages 1532–1543. http://www.aclweb.org/anthology/D14-1162.

Peng Qi and Christopher D. Manning. 2017. Arc-swift: A novel transition system for dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*. pages 110–117. https://doi.org/10.18653/v1/P17-2018.

Francesco Sartorio, Giorgio Satta, and Joakim Nivre. 2013. A transition-based dependency parser using a dynamic parsing strategy. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 135–144. http://aclanthology.coli.uni-saarland.de/pdf/P/P13/P13-1014.pdf.

Tianze Shi, Liang Huang, and Lillian Lee. 2017. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. *CoRR* abs/1708.09403. http://arxiv.org/abs/1708.09403.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. 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 1: Long Papers*. pages 323–333. http://aclweb.org/anthology/P/P15/P15-1032.pdf.

A Model Details

We provide more details of the neural network architecture used in this paper, which is taken from Qi and Manning (2017).

The model consists of two blocks of 2-layered bidirectional long short-term memory (BiLSTM) networks (Graves and Schmidhuber, 2005) with 400 hidden units in each direction. The first block is used for POS tagging and the second one, for parsing. As the input of the tagging block, we use words represented as word embeddings, and BiLSTMs are employed to perform feature extraction. The resulting output is fed into a multi-layer perceptron (MLP), with a hidden layer of 100 rectified linear units (ReLU), that provides a POS tag for each input token in a 32-dimensional representation. Word embeddings concatenated to these POS tag embeddings serve as input of the second block of BiLSTMs to undertake the parsing stage. Then, the output of the parsing block is fed into a MLP with two separate ReLU hidden layers (one for deriving the representation of the head, and the other for the dependency label) that, after being merged and by means of a softmax function, score all the feasible transitions, allowing to greedily choose and apply the highest-scoring one.

Moreover, we adapt the featurization process with biaffine combination described in Qi and Manning (2017) for the arc-swift system to be used on the original Covington and NL-Covington parsers. In particular, arc transitions are featurized by the concatenation of the representation of the head and dependent words of the arc to be created, the No-Arc transition is featurized by the rightmost word in λ_1 and the leftmost word in the buffer B and, finally, for the Shift transition only the leftmost word in B is used. Unlike Qi and Manning (2017) do for baseline parsers, we do not use the featurization method detailed in Kiperwasser and Goldberg (2016)[6] for the original Covington parser, as we observed that this results in lower

[6]For instance, Kiperwasser and Goldberg (2016) featurize all transitions of the arc-eager parser in the same way by concatenating the representations of the top 3 words on the stack and the leftmost word in the buffer.

scores and then the comparison would be unfair in our case. We implement both systems under the same framework, with the original Covington parser represented as the NL-Covington system plus the No-Arc transition and with k limited to 1. A thorough description of the model architecture and featurization mechanism can be found in Qi and Manning (2017).

Our training setup is exactly the same used by Qi and Manning (2017), training the models during 10 epochs for large datasets and 30 for small ones. In addition, we initialize word embeddings with 100-dimensional GloVe vectors (Pennington et al., 2014) for English and use 300-dimensional Facebook vectors (Bojanowski et al., 2016) for other languages. The other parameters of the neural network keep the same values.

The parser's source code is freely available at `https://github.com/danifg/Non-Local-Covington`.