A Dynamic Oracle for Linear-Time 2-Planar Dependency Parsing

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 propose an efficient dynamic oracle for training the 2-Planar transition-based parser, a linear-time parser with over 99% coverage on non-projective syntactic corpora. This novel approach outperforms the static training strategy in the vast majority of languages tested and scored better on most datasets than the arc-hybrid parser enhanced with the Swap transition, which can handle unrestricted non-projectivity.

1 Introduction

Linear-time greedy transition-based parsers such as *arc-eager*, *arc-standard* and *arc-hybrid* (Nivre, 2003, 2004; Kuhlmann et al., 2011) are widely used for dependency parsing due to their efficiency and performance, but they cannot deal with non-projective syntax. To address this, various extensions have been proposed, involving new transitions (Attardi, 2006; Nivre, 2009; Fernández-González and Gómez-Rodríguez, 2012), data structures (Gómez-Rodríguez and Nivre, 2010; Pitler and McDonald, 2015) or pre and postprocessing (Nivre and Nilsson, 2005). Among these extensions, the 2-Planar parser (Gómez-Rodríguez and Nivre, 2010) has attractive properties, as it (1) keeps the original worst-case linear time, (2) has close to full coverage of non-projective phenomena, and (3) needs no pre- or post-processing.

Dynamic oracles (Goldberg and Nivre, 2012) are known to improve the accuracy of greedy parsers by enabling more robust training, by exploring configurations beyond the gold path. While dynamic oracles have been defined for many transition-based algorithms (Goldberg and Nivre, 2013; Goldberg et al., 2014; Gómez-Rodríguez et al., 2014; Gómez-Rodríguez and Fernández-González, 2015; de Lhoneux et al., 2017), none is available so far for the 2-Planar system. The

lack of the arc-decomposability property, which can be used to derive dynamic oracles for parsers that have it, makes the obtention of one non-trivial.

To fill this gap, we define an efficient dynamic oracle for the 2-Planar transition-based parser, using similar loss calculation techniques as described in (Gómez-Rodríguez and Fernández-González, 2015) for the non-arc-decomposable Covington parser (Covington, 2001). Training the 2-Planar parser with this novel strategy achieves accuracy gains in the vast majority of datasets tested. In addition, we empirically compare our novel approach to the most similar existing alternative:[1] the arc-hybrid parser with a swap transition trained with a static-dynamic oracle, recently introduced by de Lhoneux et al. (2017); which can handle unrestricted non-projective dependencies in $O(n^2)$ worst-case time in theory, but expected linear time in practice (Nivre, 2009). Our approach outperforms this swap-based system on average over a standard set of dependency treebanks.

2 The 2-Planar parser

We briefly sketch the 2-Planar transition system, which was defined by Gómez-Rodríguez and Nivre (2010, 2013) under the transition-based parsing framework (Nivre, 2008) and is based on the arc-eager algorithm (Nivre, 2003), keeping its linear time complexity. It works by building, in a single pass, two non-crossing graphs (called *planes*) whose union provides a dependency parse in the set of 2-planar (or pagenumber-2) graphs, which is known to cover over 99% of parses in a large number of real treebanks (Gómez-Rodríguez and Nivre, 2010; Gómez-Rodríguez, 2016).

Parser configurations have the form $c =$

---

[1] Although the two-register parser by Pitler and McDonald (2015) is even closer to ours in terms of the transition system (being based on arc-eager and running in linear time), no dynamic oracle is known for it, to the best of our knowledge.

$\langle \Sigma_1, \Sigma_2, B, A \rangle$, where Σ_1 and Σ_2 are, respectively, the *active stack* and *inactive stack* and are tied to each of the planes or pages of the output graph, B is a *buffer* of unread words, and A the set of dependency arcs built so far. For an input string $w_1 \cdots w_n$, the parser starts at configuration $c_s(w_1 \ldots w_n) = \langle [\,], [\,], [w_1 \ldots w_n], \emptyset \rangle$, applying transitions until a terminal configuration $\langle \Sigma_1, \Sigma_2, [\,], A \rangle$ is reached, and A yields the output.

Figure 1 shows the parser's transitions. The Shift transition pops the first (leftmost) word in the buffer, and pushes it to both stacks; the Reduce transition pops the top word from the active stack, implying that we have added all arcs to or from it on the plane tied to that stack; and the Left-Arc and Right-Arc transitions create a leftward or rightward dependency arc connecting the first word in the buffer to the top of the active stack. Finally, the Switch transition makes the active stack inactive and vice versa, changing the plane the parser is working with. Transitions that violate the single-head or acyclicity constraints are disallowed, so that the output is a forest. Finally, to guarantee the termination of the parsing process, two consecutive Switch transitions are not allowed.

3 A dynamic oracle

We now define an efficient dynamic oracle to train the 2-Planar algorithm, which operates under the assumption of a fixed assignment of arcs to planes.

Following Goldberg and Nivre (2013), if the Hamming loss ($\mathcal{L}$) between trees t and t_G is the amount of words with a different head in t and t_G, then implementing a dynamic oracle reduces to defining a loss function $\ell(c)$ which, given a parser configuration c and a gold tree t_G, computes the minimum loss between t_G and a tree reachable from c.[2] We call this the minimum loss of configuration c, $\ell(c) = \min_{t \mid c \rightsquigarrow t} \mathcal{L}(t, t_G)$. A correct dynamic oracle will return the set of transitions τ that do not increase this loss (i.e., $\ell(\tau(c)) - \ell(c) = 0$), thus leading to the best parse reachable from c.

For parsers that are *arc-decomposable*[3], $\ell(c)$ can be obtained by counting gold arcs that are *not* individually reachable from c, which is trivial in most parsers. Unfortunately, the 2-Planar parser

---

[2]We say that an arc set X is *reachable* from configuration c, and we write $c \rightsquigarrow X$, if there is some (possibly empty) path of transitions from c to some configuration $c' = \langle \Sigma_1, \Sigma_2, B, A' \rangle$, with $X \subseteq A'$.

[3]i.e., if every individual arc of X is reachable from a given configuration c, the set X as a whole is reachable from c.

is non-arc-decomposable. To show this, it suffices to consider any configuration where an incorrect arc created in A forms a cycle together with a set of otherwise reachable gold arcs, just as in the proof of non-arc-decomposability for Covington provided by Gómez-Rodríguez and Fernández-González (2015). In fact, the same counterexample provided there also works for this parser.

Note, however, that non-arc-decomposability in the 2-Planar parser not only comes from cycles (as in Covington) but also from situations where, due to a poor assignment of planes to already-built arcs, no possible plane assignment allows building a set of pending gold arcs. Thus, the loss calculation technique of the Covington dynamic oracle is not directly applicable to the 2-Planar parser.

However, if we statically choose a canonical plane assignment and we calculate loss with respect to that assignment (i.e., creating a correct arc in the non-canonical plane incurs loss), then the Covington technique, based on counting individually unreachable arcs and then correcting for the presence of cycles, works for the 2-Planar parser. This is the idea of our dynamic oracle, which therefore is a correct dynamic oracle only with respect to a preset criterion for plane assignment, and not for all the possible plane assignments that would produce the gold dependency structure.

In particular, given a 2-planar gold dependency tree whose set of arcs is t_G, we need to divide it into two gold arc sets t_G^1 and t_G^2, associated with each plane.[4] In this paper, we take as canonical the division provided by the static oracle of Gómez-Rodríguez and Nivre (2010), which prefers to build arcs in the active plane to minimize the number of Switch transitions needed.[5]

Once the plane assignment is set, we can associate individually unreachable arcs to a plane. Then, we can calculate configuration loss as:

$$\ell(c) = |\mathcal{U}_1(c, t_G^1) \cup \mathcal{U}_2(c, t_G^2)| \\ + n_c(A \cup (\mathcal{I}_1(c, t_G^1) \cup \mathcal{I}_2(c, t_G^2)))$$

---

[4]In practice, for gold parses that are not 2-planar, some arcs will need to be discarded, so that $t_G^1 \cup t_G^2$ will be a 2-planar subset of t_G. Note that in this case, our oracle is correct with respect to this 2-planar subset, but it does not guarantee minimum loss with respect to the original non-2-planar graph (in the same way as existing projective dynamic oracles do not guarantee it with respect to non-projective trees).

[5]Our dynamic oracle can work with any plane assignment criterion. We chose this one for simplicity, for direct comparability to the existing static oracle, and because it has been shown to be learnable in practice.

Shift:	$\langle \Sigma_1, \Sigma_2, w_i | B, A \rangle \Rightarrow \langle \Sigma_1 | w_i, \Sigma_2 | w_i, B, A \rangle$
Reduce:	$\langle \Sigma_1 | w_i, \Sigma_2, B, A \rangle \Rightarrow \langle \Sigma_1, \Sigma_2, B, A \rangle$
Left-Arc:	$\langle \Sigma_1 | w_i, \Sigma_2, w_j | B, A \rangle \Rightarrow \langle \Sigma_1 | w_i, \Sigma_2, w_j | B, A \cup \{w_j \rightarrow w_i\} \rangle$
	only if $\nexists w_k \mid w_k \rightarrow w_i \in A$ (single-head) and $w_i \rightarrow^* w_j \notin A$ (acyclicity).
Right-Arc:	$\langle \Sigma_1 | w_i, \Sigma_2, w_j | B, A \rangle \Rightarrow \langle \Sigma_1 | w_i, \Sigma_2, w_j | B, A \cup \{w_i \rightarrow w_j\} \rangle$
	only if $\nexists w_k \mid w_k \rightarrow w_j \in A$ (single-head) and $w_j \rightarrow^* w_i \notin A$ (acyclicity).
Switch:	$\langle \Sigma_1, \Sigma_2, B, A \rangle \Rightarrow \langle \Sigma_2, \Sigma_1, B, A \rangle$

Figure 1: Transitions of the 2-Planar dependency parser. The notation $w_i \rightarrow^* w_j \in A$ means that there is a (possibly empty) directed path from w_i to w_j in A.

where for $i \in \{1, 2\}$, each set $\mathcal{I}_i(c, t_G^i) = \{x \rightarrow y \in t_G^i \mid c \rightsquigarrow (x \rightarrow y)\}$ is the set of *individually reachable arcs* of t_G^i from configuration c; $\mathcal{U}_i(c, t_G^i)$ is the set of *individually unreachable arcs* of t_G^i from c, defined as $t_G^i \setminus \mathcal{I}_i(c, t_G^i)$; and $n_c(G)$ denotes the number of cycles in a graph G.

To compute the sets of individually unreachable arcs $\mathcal{U}_i(c, t_G^i)$ from a configuration $c = \langle \Sigma_1, \Sigma_2, B, A \rangle$, we examine gold arcs. A gold arc $x \rightarrow y \in t_G^i$ will be in $\mathcal{U}_i(c, t_G^i)$ if it is not in $A \cap t_G^i$ (the set of already-built arcs from the plane of interest) and at least one of the following holds:

- $\min(x, y) \notin \Sigma_i \cup B \vee \max(x, y) \notin B$, (i.e., $\min(x, y)$ must be in plane i's stack or in the buffer, and $\max(x, y)$ must be in the buffer so that the arc $x \rightarrow y$ can still be built),

- there is some $z \neq 0, z \neq x$ such that $z \rightarrow y \in A$, (i.e., we cannot create $x \rightarrow y$ because it would violate the single-head constraint),

- x and y are on the same weakly connected component of A (i.e., we cannot create $x \rightarrow y$ due to the acyclicity constraint).

- $x \rightarrow y \in A \cap t_G^{3-i}$ (i.e., the arc was already erroneously created in the other plane and, therefore, is unreachable in plane i).

Once we have $\mathcal{U}_i(c, t_G^i)$ for each of the two planes, $\mathcal{I}_i(c, t_G^i)$ can be obtained as $t_G^i \setminus \mathcal{U}_i(c, t_G^i)$. Finally, since the graph $A \cup \mathcal{I}_1(c, t_G^1) \cup \mathcal{I}_2(c, t_G^2)$ has in-degree 1, the algorithm by Tarjan (1972) can be used to implement the function n_c to count its cycles in $O(n)$ time. For this reason, the full loss calculation runs in linear time as well.[6]

Given a plane assignment, $\ell(c)$ is an exact expression of the loss of a configuration of the 2-Planar parser as expressed in Figure 1, without the control constraint that forbids two consecutive Switch transitions. This can be proven using the same reasoning as for the Covington loss expression of (Gómez-Rodríguez and Fernández-González, 2015). Thus, the computation of $\ell(c)$ provides a complete and correct dynamic oracle for this parser under a given plane assignment, by directly evaluating $\ell(\tau(c)) - \ell(c)$ for each transition τ. However, to make the oracle correct for the practical version, where consecutive Switch transitions are disallowed, we need to modify the cost calculation for the Switch transition.

In particular, applying a Switch transition does not affect the loss, so $\ell(\text{Switch}(c)) - \ell(c)$ is always 0. Indeed, if Switch transitions are always allowed, their cost is zero because they can always be undone and thus never affect the reachability of any arcs. However, when consecutive Switch transitions are banned to ensure parser termination, choosing to Switch can have consequences as, in the resulting configuration, the parser will be forced to take one of the other four transitions, which may lead to suboptimal outcomes compared to not having switched.

To address this, we compute the cost of Switch transitions instead as $min(\{\ell(\tau(\text{Switch}(c))) - \ell(c) | \tau \neq \text{Switch}\})$, i.e., the minimum number of gold arcs missed after being forced to apply one of the other four transitions after Switch (if this cost is 0, then switching stacks is an optimal choice). Adding this modification makes the dynamic oracle correct for the practical version of the parser

---

[6]The check for acyclicity using weakly connected components has no impact on the complexity: when weakly connected components are represented using path compression

and union by rank, the relevant operations run in amortized inverse Ackermann time, meaning that they behave as constant time for all practical purposes, like in (Gómez-Rodríguez and Nivre, 2013)

that disallows consecutive Switch transitions.

Regularization While the above dynamic oracle is theoretically correct, we noticed experimentally that the Switch transition tends to switch stacks very frequently during training, due to exploration. This leads the parser to learn unnecessarily long and complex transition sequences that change planes more than needed, harming accuracy.

To avoid this, we add a regularization term to $\ell(c)$ representing the transition sequence length from c to its minimum-loss reachable tree(s), to discourage unnecessarily long sequences. This amounts to penalizing the Switch transition if there is any zero-cost transition available in the active plane and changing planes will delay its application. Thus, arcs assigned to the currently active plane will be built before switching if possible, enforcing a global arc creation order. This is similar to the prioritization of monotonic paths in (Honnibal et al., 2013, §6), as they also penalize unneeded actions that will need to be undone later.

4 Experiments

4.1 Data and Evaluation

We conduct our experiments on the commonly-used non-projective benchmark compounded of nine datasets from the CoNLL-X shared task (Buchholz and Marsi, 2006) and all datasets from the CoNLL-XI shared task (Nivre et al., 2007).[7] We also use the Stanford Dependencies (de Marneffe and Manning, 2008) conversion (using the Stanford parser v3.3.0)[8] of the WSJ Penn Treebank (PTB-SD) (Marcus et al., 1993) with standard splits. Labelled and Unlabelled Attachment Scores (LAS and UAS) are computed including punctuation for all datasets except for the PTB where, following common practice, the punctuation is excluded. We train our system for 15 iterations and choose the best model according to development set accuracy. Statistical significance is calculated using a paired test with 10,000 bootstrap samples.

4.2 Model

We implement both the static oracle and the dynamic oracle with aggressive exploration for the

---

2-Planar parser under the neural network architecture proposed by Kiperwasser and Goldberg (2016). We also add the static-dynamic arc-hybrid parser with Swap transition (de Lhoneux et al., 2017), implemented under the same framework to perform a fair comparison.

The neural network architecture used in this paper is taken from Kiperwasser and Goldberg (2016). We use the same BiLSTM-based featurization method that concatenates the representations of the top 3 words on the active stack and the leftmost word in the buffer for the arc-hybrid and 2-Planar algorithms, and we add the top 2 words on the inactive stack for the latter. Following Kiperwasser and Goldberg (2016), we also include the BiLSTM vectors of the rightmost and leftmost modifiers of words from the stacks, as well as the leftmost modifier of the first word in the buffer. 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 as in (Kiperwasser and Goldberg, 2016).

4.3 Results

Table 1 shows that the 2-Planar parser trained with a dynamic oracle outperforms the static training strategy in terms of UAS in 15 out of 20 languages, with 8 of these improvements statistically significant ($\alpha = .05$), and one statistically significant decrease. When comparing with the enhanced arc-hybrid system in Table 2, our approach provides a better UAS in 12 out of 20 datasets tested, achieving statistically significant ($\alpha = .05$) gains in accuracy on 7 of them, and significant losses on 3 of them.

We could not find a clear pattern to explain why the 2-Planar algorithm outperforms arc-hybrid plus Swap in some languages and vice versa. The latter seems to work better on treebanks with less non-projectivity such as the English, Chinese and Japanese datasets, and worse on those with higher amounts like Turkish, Dutch or Basque. However, some cases like Czech or Catalan go against this trend. From (Gómez-Rodríguez and Nivre, 2010), we also know that the Dutch and German treebanks have a relatively high proportion of non-2-planar trees, but the 2-Planar parser seems to be a better option on them than the extended arc-

	2-Planar static		2-Planar dynamic	
Language	UAS	LAS	UAS	LAS
Arabic	**83.20**	**73.48**	82.96	73.24
Basque	77.61	69.94	**78.11**	**70.11**
Catalan	92.50	87.92	**93.70***	**88.48***
Chinese	85.95	80.97	**87.08***	**81.73***
Czech	84.67	78.56	**85.29**	**79.40**
English	89.57	88.69	**90.87***	**90.03***
Greek	81.93	74.23	**82.06**	**74.79**
Hungarian	81.50	75.94	**82.48***	**76.97***
Italian	86.85	82.36	**87.24**	**82.38**
Turkish	**81.68**	73.59	81.48	**73.61**
Bulgarian	93.14	89.74	**93.23**	**89.97**
Danish	88.31	84.23	**88.57**	**84.76**
Dutch	85.51	81.63	**86.50***	**82.70***
German	**90.80**	**88.71**	90.71	88.51
Japanese	92.51	90.56	**93.19***	**90.65***
Portugue.	88.68	85.12	**89.02***	**85.92***
Slovene	78.67	70.49	**79.30**	**70.86**
Spanish	**83.63***	**79.79***	82.42	78.68
Swedish	**89.92**	**85.48**	89.83	85.40
PTB-SD	93.59	91.60	**93.96***	**92.06***
Average	86.51	81.65	**86.90**	**82.01**

	2-Planar dynamic		AHybrid_Swap static-dynamic	
Language	UAS	LAS	UAS	LAS
Arabic	**82.96***	**73.24***	80.74	70.69
Basque	**78.11***	**70.11***	75.60	68.70
Catalan	**93.70***	**88.48**	93.12	88.06
Chinese	87.08	81.73	**87.31**	**82.02**
Czech	85.29	79.40	**85.71***	**80.08***
English	90.87	90.03	**91.37**	**90.37**
Greek	82.06	74.79	**83.72***	**76.33***
Hungarian	**82.48**	**76.97**	82.20	76.88
Italian	**87.24***	**82.38***	86.24	81.48
Turkish	**81.48***	**73.61***	77.44	69.22
Bulgarian	**93.23**	**89.97**	93.17	89.62
Danish	88.57	**84.76**	**88.65**	84.60
Dutch	**86.50***	**82.70***	84.24	81.04
German	**90.71**	**88.51**	90.60	88.39
Japanese	93.19	90.65	**93.40**	**91.47***
Portugue.	**89.02**	**85.92**	88.78	85.48
Slovene	**79.30***	**70.86**	77.68	70.61
Spanish	82.42	78.68	**83.98***	**80.15***
Swedish	89.83	**85.40**	**89.92**	85.18
PTB-SD	**93.96**	**92.06**	93.83	91.93
Average	**86.90**	**82.01**	86.39	81.62

Table 1: Parsing accuracy of the 2-Planar parser trained with static and dynamic oracles on CoNLL-XI (first block), CoNLL-X (second block) and PTB-SD (third block) datasets. Best results for each language are shown in boldface. Statistically significant improvements ($\alpha = .05$) are marked with *.

Table 2: Parsing accuracy of the 2-Planar parser trained with the dynamic oracle and the arc-hybrid parser with the Swap transition trained with a static-dynamic oracle on CoNLL-XI (first block), CoNLL-X (second block) and PTB-SD (third block) datasets. Best results for each language are in boldface. Statistically significant improvements ($\alpha = .05$) are marked with *.

hybrid system that can handle unrestricted non-projectivity. The reasons, beyond the scope of this research, might be related to different dependency length distributions or non-projective topologies.

We noticed that, in general, the 2-Planar parser has higher precision on non-projective arcs and the enhanced arc-hybrid parser has a better recall.

5 Conclusion

We present an efficient dynamic oracle to train the 2-Planar transition-based parser, which is correct with respect to a given plane assignment, and results in notable gains in accuracy. The parser trained with this dynamic oracle performs better on average than an expected linear-time parser supporting unrestricted non-projectivity.

References

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

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.

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.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017. Arc-hybrid non-projective dependency parsing with a static-dynamic oracle. In *Proceedings of the The 15th International Conference on Parsing Technologies (IWPT).*. Pisa, Italy.

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.

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.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*. Association for Computational Linguistics, Mumbai, India, pages 959–976. http://www.aclweb.org/anthology/C12-1059.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics* 1:403–414. http://anthology.aclweb.org/Q/Q13/Q13-1033.pdf.

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. 2016. Restricted non-projectivity: Coverage vs. efficiency. *Comput. Linguist.* 42(4):809–817. https://doi.org/10.1162/COLI_a_00267.

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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*. pages 256–261. http://aclweb.org/anthology/P/P15/P15-2042.pdf.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, USA, ACL '10, pages 1492–1501. http://dl.acm.org/citation.cfm?id=1858681.1858832.

Carlos Gómez-Rodríguez and Joakim Nivre. 2013. Divisible transition systems and multiplanar dependency parsing. *Comput. Linguist.* 39(4):799–845. https://doi.org/10.1162/COLI_a_00150.

Carlos Gómez-Rodríguez, Francesco Sartorio, and Giorgio Satta. 2014. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pages 917–927. http://aclweb.org/anthology/D14-1099.

Matthew Honnibal, Yoav Goldberg, and Mark Johnson. 2013. A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning, CoNLL 2013, Sofia, Bulgaria, August 8-9, 2013*. pages 163–172. http://aclweb.org/anthology/W/W13/W13-3518.pdf.

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.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. Association for Computational Linguistics, Stroudsburg, PA, USA, HLT '11, pages 673–682. http://dl.acm.org/citation.cfm?id=2002472.2002558.

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.

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. 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 (ACL-IJCNLP)*. pages 351–359.

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.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *ACL '05: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Morristown, NJ, USA, pages 99–106.

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.

Emily Pitler and Ryan McDonald. 2015. A linear-time transition system for crossing interval trees. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Denver, Colorado, pages 662–671. http://www.aclweb.org/anthology/N15-1068.

Robert Endre Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM J. Comput.* 1(2):146–160. http://dblp.uni-trier.de/db/journals/siamcomp/siamcomp1.html.