Arc-Hybrid Non-Projective Dependency Parsing
with a Static-Dynamic Oracle

Miryam de Lhoneux   Sara Stymne   Joakim Nivre

Department of Linguistics and Philology
Uppsala University

Abstract

We extend the arc-hybrid transition system for dependency parsing with a SWAP transition that enables reordering of the words and construction of non-projective trees. Although this extension potentially breaks the arc-decomposability of the transition system, we show that the existing dynamic oracle can be modified and combined with a static oracle for the SWAP transition. Experiments on five languages with different degrees of non-projectivity show that the new system gives competitive accuracy and is significantly better than a system trained with a purely static oracle.

1   Introduction

Non-projective sentences are a notorious problem in dependency parsing. Traditional algorithms like those developed by Nivre (2003, 2004) for transition-based parsing only allow the construction of projective trees. These algorithms make use of a stack, a buffer and a set of arcs, and parsing consists of performing a sequence of transitions on these structures. Traditional algorithms have been extended in different ways to allow the construction of non-projective trees (Nivre and Nilsson, 2005; Attardi, 2006; Nivre, 2007; Gómez-Rodríguez and Nivre, 2010). One method proposed by Nivre (2009) is based on the idea of word reordering. This is achieved by adding a transition that swaps two items in the data structures used, enabling the construction of arbitrary non-projective trees while still only adding arcs between adjacent words (after possible reordering). This technique was previously used in the arc-standard transition system (Nivre, 2004). The first contribution of this paper is to show that it can also be combined with the arc-hybrid system

proposed by Kuhlmann et al. (2011).

Recent advances in dependency parsing have demonstrated the benefit of using dynamic oracles for training dependency parsers (Goldberg and Nivre, 2013). Traditionally, parsers were trained in a static way and were only exposed to configurations resulting from optimal transitions during training. Dynamic oracles define optimal transition sequences for any configuration in which the parser may be. The use of dynamic oracles enables training with exploration of errors, which mitigates the problem of error propagation at prediction time.

In order to define a dynamic oracle, we need to be able to compute the cost of any transition in any configuration, where cost is usually defined as minimum Hamming loss with respect to the best tree reachable from that configuration. Goldberg and Nivre (2013) showed that this computation is straightforward for transition systems that satisfy the property of *arc-decomposability*, meaning that a tree is reachable from a configuration if and only if every arc in the tree is reachable in itself. Based on this result, they defined dynamic oracles for the arc-eager (Nivre, 2003), arc-hybrid (Kuhlmann et al., 2011) and easy-first (Goldberg and Elhadad, 2010) systems.

Transition systems that allow non-projective trees are in general not arc-decomposable and therefore require different methods for constructing dynamic oracles (Gómez-Rodríguez and Fernández-González, 2015). The online reordering system of Nivre (2009) is furthermore based on the arc-standard system, which is not even arc-decomposable in itself (Goldberg and Nivre, 2013). The second contribution of this paper is to show that we can take advantage of the arc-decomposability of the arc-hybrid transition system and extend the existing dynamic oracle to deal with the added swap transition. The resulting or-

acle is static with respect to the new transition but remains dynamic for all other transitions. We show experimentally that this static-dynamic oracle gives a significant advantage over the alternative static oracle and results in competitive results for non-projective parsing.

2 An Extended Transition System

The arc-hybrid system has configurations of the form $c = (\Sigma, B, A)$, where

- Σ is a stack (represented as a list with the head to the right),

- B is a buffer (represented as a list with the head to the left),

- A is a set of dependency arcs (represented as (h, d) pairs).[1]

Given a sentence $W = w_1, \ldots, w_n$, the system is initialized to:

$$c_0 = ([\,], [1, \ldots, n, n{+}1], \{\ \})$$

where $n{+}1$ is a special root node, denoted r from now on. Terminal configurations have the form:

$$c = ([\,], [r], A)$$

and the parse tree is given by the arc set A.

The original arc-hybrid system from Kuhlmann et al. (2011) has three transitions:[2]

- LEFT$[(\sigma|s_0,\ b|\beta,\ A)] = (\sigma,\ b|\beta,\ A \cup \{(b, s_0)\})$

- RIGHT$[(\sigma|s_1|s_0,\ \beta,\ A)] = (\sigma|s_1,\ \beta,\ A \cup \{(s_1, s_0)\})$

- SHIFT$[(\sigma,\ b|\beta,\ A)] = (\sigma|b,\ \beta,\ A)$

There are preconditions such that SHIFT is legal only if $b \neq r$, RIGHT only if $|\Sigma| > 1$ and LEFT only if $|\Sigma| > 0$. In order to enforce that r has exactly one dependent, as required by some dependency grammar frameworks, we add a precondition such that LEFT is legal only if $|\Sigma| = 1$ or $b \neq r$.

In the extended system, we add a SWAP transition to be able to construct non-projective trees using online reordering:

- SWAP$[(\sigma|s_0,\ b|\beta,\ A)] = (\sigma,\ b|s_0|\beta,\ A)$

There is a precondition making SWAP legal only if $|\Sigma| > 0$, $|B| > 1$ and $s_0 < b$.[3]

The SWAP transition reorders nodes by moving the item on top of the stack (s_0) to the second position in the buffer, thus inverting the order of s_0 and b. The SHIFT and SWAP transitions together implement a simple sorting algorithm, which allows us to permute the order of nodes arbitrarily. As shown by (Nivre, 2009), this implies that we can construct any non-projective tree by reordering and adding arcs between adjacent nodes using LEFT and RIGHT.

As already observed, the main advantage of the arc-hybrid system over the arc-standard system is that it is arc-decomposable, which allows us to construct a simple and efficient dynamic oracle. The arc-eager system (Nivre, 2003) is also arc-decomposable but cannot be combined with SWAP because items on the stack in that system do not necessarily represent disjoint subtrees.

3 A Static-Dynamic Oracle

The dynamic oracle for arc-hybrid parsing defined by Goldberg and Nivre (2013) computes the cost of a transition by counting the number of gold arcs that are made unreachable by applying that transition. This presupposes that the system is arc-decomposable, a result that is proven in the same paper. Our construction of an oracle for arc-hybrid parsing with online ordering is based on the conjecture that we can retain arc-decomposition by only making SWAP transitions that are necessary to make non-projective arcs reachable and by enforcing all such transitions. Proving this conjecture is, however, outside the scope of this paper.

3.1 Auxiliary Functions and Notation

We assume that gold trees are preprocessed at training time to compute the following information for each input node i:

- PROJ(i) = the position of node i in the projective order.[4]

- RDEPS(i) = the set of reachable dependents of i, initially all dependents of i.

- LEFT:
$$\mathcal{C}(\textsc{Left}) = |\text{RDEPS}(s_0)| + [\![h(s_0) \neq b \text{ and } s_0 \in \text{RDEPS}(h(s_0))]\!]$$

  Updates: Set $\text{RDEPS}(s_0) = [\,]$ and remove s_0 from $\text{RDEPS}(h(s_0))$.

- RIGHT:
$$\mathcal{C}(\textsc{Right}) = |\text{RDEPS}(s_0)| + [\![h(s_0) \neq s_1 \text{ and } s_0 \in \text{RDEPS}(h(s_0))]\!]$$

  Updates: Set $\text{RDEPS}(s_0) = [\,]$ and remove s_0 from $\text{RDEPS}(h(s_0))$.

- SHIFT:
  1. If there exists a node $i \in B_{-b}$ such that $b < i$ and $\text{PROJ}(b) > \text{PROJ}(i)$:
$$\mathcal{C}(\textsc{Shift}) = 0$$
  2. Else:
$$\mathcal{C}(\textsc{Shift}) = |\{d \in \text{RDEPS}(b) \,|\, d \in \Sigma\}| + [\![h(b) \in \Sigma_{-s_0} \text{ and } b \in \text{RDEPS}(h(b))]\!]$$

  Updates: Remove b from $\text{RDEPS}(h(b))$ if $h(b) \in \Sigma_{-s_0}$ and remove $d \in \Sigma$ from $\text{RDEPS}(b)$.

Figure 1: Transition costs and updates. Expressions of the form $[\![\Phi]\!]$ evaluate to 1 if Φ is true, 0 otherwise. We use s_0 and s_1 to refer to the top and second top item of the stack respectively and we use b to denote the first item of the buffer. Σ refers to the stack and Σ_{-s_0} to the stack excluding s_0 (if Σ is not empty). B refers to the buffer and B_{-b} to the buffer excluding b.

We use $h(i)$ to denote the head of a node i in the gold tree.

3.2 Static Oracle for SWAP

Our oracle is fully dynamic with respect to SHIFT, LEFT and RIGHT but basically static with respect to SWAP. This means that only optimal (zero cost) SWAP transitions are allowed during training and that we force the parser to apply the SWAP transition when needed.

Optimal: To prevent non-optimal SWAP transitions, we add a precondition so that SWAP is legal only if $\text{PROJ}(s_0) > \text{PROJ}(b)$.

Forced: To force necessary SWAP transitions, we bypass the oracle whenever $\text{PROJ}(s_0) > \text{PROJ}(b)$.[5]

3.3 Dynamic Oracle

Since we use a static oracle for SWAP transitions, these will always have zero cost. The dynamic oracle thus only needs to define costs for the remaining three transitions. To construct the oracle, we start from the old dynamic oracle for the projective

system and extend it to account for the added flexibility introduced by SWAP. Figure 1 defines the transition costs as well as the necessary updates to RDEPS after applying a transition.

- LEFT: Adding the arc (b, s_0) and popping s_0 from the stack means that s_0 will not be able to acquire a head different from b nor any of its reachable dependents. In the old projective case, the loss was limited to a head in $s_0|\beta$ and dependents in $b|\beta$, but because s_0 can potentially be swapped back to the buffer, we instead define reachability explicitly through $\text{RDEPS}(s_0)$ (for dependents) and $\text{RDEPS}(h(s_0))$ (for the head) and update these accordingly after applying the transition.

- RIGHT: Adding the arc (s_1, s_0) and popping s_0 from the stack means that s_0 will not be able to acquire a head different from s_1 nor any of its reachable dependents. In the old projective case, the loss was limited to a head and dependents in $b|\beta$, but because s_0 can potentially be swapped back to the buffer, we again define reachability explicitly through $\text{RDEPS}(s_0)$ (for dependents)

---

[5]This is equivalent to an *eager* static oracle for SWAP in the sense of Nivre et al. (2009).
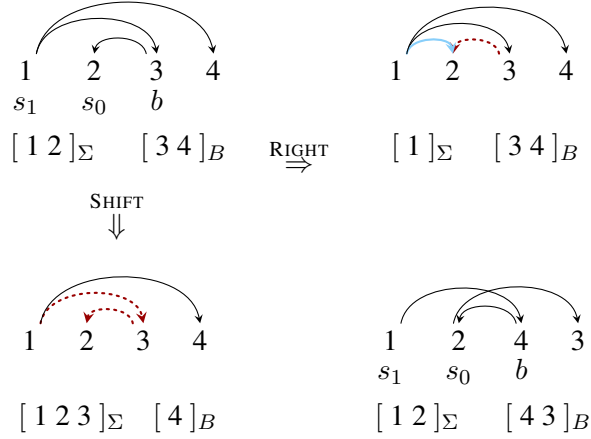
Figure 2: Top left: Configuration with all nodes in projective order and gold tree displayed above the nodes. Top right: Gold arc lost (the red dotted arc) when applying a RIGHT transition from the top left configuration. The arc added by the transition is in blue, it is not in the gold tree. Bottom left: Gold arcs lost (the red dotted arcs) when applying a SHIFT transition from the top left configuration. Bottom right: Configuration where b is higher in the projective order than a following node in the buffer.

and RDEPS($h(s_0)$) (for the head) and update these accordingly after applying the transition.

- SHIFT: In the projective case, shifting b onto the stack means that b will not be able to acquire a head in Σ other than the top item s_0 nor any dependents in Σ. With the SWAP transition and a static oracle, we also have to consider the case where b can later be swapped back to the buffer, in which case SHIFT has zero cost. We therefore have two cases in Figure 1. In the first case, no updates are needed. In the second case, the updates are analogous to the old projective case.

To illustrate how the oracle works, let us look at some hypothetical configurations. First, we can have a situation as in the top left corner of Figure 2, where all nodes are in projective order given the gold tree displayed above the nodes. For simplicity, the nodes are named after their projective order.

Applying a RIGHT transition in this configuration makes it impossible for s_0 (2) to be attached to its head (3) and therefore makes us lose the arc $3 \rightarrow 2$, as shown in the top right corner. If we instead apply a SHIFT transition, we lose the arc between b (3) and its head (1) as well as the arc $3 \rightarrow 2$, as shown in the bottom left corner. By contrast, a LEFT transition has zero cost, because no arcs are lost so the best tree reachable in the orig-

inal configuration is still reachable after applying the LEFT transition.

Consider now a configuration, like the one in the bottom right corner of Figure 2, where the nodes are not in projective order. Here we can shift b (4) onto the stack without cost, because it will later be swapped back to the buffer to restore the projective order between 4 and 3. A RIGHT transition makes us lose the head and dependent of s_0 ($4 \rightarrow 2$ and $2 \rightarrow 3$). A LEFT transition makes us lose the dependent of s_0 ($2 \rightarrow 3$).

The combination of a dynamic oracle for LEFT, RIGHT and SHIFT with a static oracle for SWAP allows us to benefit from training with exploration in most situations and at the same time capture nonprojective dependencies.

4 Experiments

We extend the parser we used in de Lhoneux et al. (2017), a greedy transition-based parser that predicts the dependency tree given the raw words of a sentence. That parser is itself an extension of the parser developed by Kiperwasser and Goldberg (2016). It relies on a BiLSTM to learn informative features of words in context and a feed-forward network for predicting the next parsing transition. It learns vector representations of the words as well as characters. Contrary to parsing tradition, it makes no use of part-of-speech tags.

We first compare our system, which uses our static-dynamic oracle, with the same system using a static oracle. This is to find out if we can benefit from error exploration using our partially dynamic oracle. We use the same set of hyperparameters as in that paper in all our experiments.

We additionally compare our method to a different approach to handling non-projectivity, pseudo-projective parsing, as performed in de Lhoneux et al. (2017). Pseudo-projective parsing was developed by Nivre and Nilsson (2005). In a pre-processing step, the training data is projectivised: some nodes get reattached to a close parent. Parsed data are then 'deprojectivised' in a post-processing step. In order for information about non-projectivity to be recoverable after parsing, when projectivising, arcs are renamed to encode information about the original parent of dependents which get re-attached. Note that hyperparameters were tweaked for the pseudo-projective system, possibly giving an unfair advantage.

Lastly, we compare to a *projective* baseline, using a dynamic oracle but no SWAP transition.[7] This is to find out the extent to which dealing with non-projectivity is important.

We selected a sample of 5 treebanks from the Universal Dependencies CoNLL 2017 shared task data (Nivre et al., 2017). We selected languages to have different frequencies of non-projectivity, both at the sentence level and at the level of individual arcs, ranging from a very high frequency (Ancient-Greek) to a low frequency (English), as well as some typological variety. Non-projective frequencies were taken from Straka et al. (2015) and are included in Table 1, which report our results on the development sets (best out of 20 epochs).

Comparing to the static baseline, we can verify that our static-dynamic oracle really preserves the benefits of training with error exploration, with improvements ranging from 0.5 to 3.5 points. (All differences here are statistically significant with p<0.01, except for Portuguese, where the difference is statistically significant with p<0.05 according to the McNemar test).

The new system achieves results largely on par

Language	%NP	S-Dy	Static	PProj	Proj
A.Greek	9.8 / 63.2	**59.53**	56.04	59.22	46.98
Arabic	0.3 / 8.2	**77.08**	76.61	76.96	76.55
Basque	5.0 / 33.7	72.27	70.98	**74.16**	68.85
English	0.5 / 5.0	81.97	81.00	82.21	**82.37**
Portuguese	1.3 / 18.4	**87.34**	86.60	87.20	85.39

Table 1: LAS on dev sets with gold tokenization for our static-dynamic system (S-Dy), the static and projective baselines (Static, Proj) and the pseudo-projective system of de Lhoneux et al. (2017) (PProj). %NP = percentage of non-projective arcs/sentences.

with the pseudo-projective parser. Our method is better by a small margin for 3 out of 5 languages and worse by a large margin for 1. Overall, these results are encouraging given that our method is simpler and more efficient to train, with no need for pre- or post-processing and no extension of the dependency label set.[8]

Comparing to the projective baseline, we see that strictly projective parsing can be slightly better than both online reordering and pseudo-projective parsing for a language with few non-projective arcs/sentences like English. For all other languages, we see small (Arabic) to big (Ancient Greek) improvements from dealing with non-projectivity in some way.

5 Conclusion

We have shown how the SWAP transition for online reordering can be integrated into the arc-hybrid transition system for dependency parsing in such a way that we still benefit from training with exploration using a static-dynamic oracle. In the future, we intend to test this further by evaluating our model on more languages with proper tuning of hyperparameters. We are also interested in the question of whether it is possible to define a fully dynamic oracle for our system and allow exploration for the SWAP transition too.

Acknowledgments

---

[6]We released our system as UUparser 2.0, available at https://github.com/UppsalaNLP/uuparser.

[7]When training the projective baseline, we removed non-projective sentences from the training data.

[8]We made no systematic study of training time but observed that it took roughly half the time to train our parser compared to the pseudo-projective one.

References

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.

Miryam de Lhoneux, Yan Shao, Ali Basirat, Eliyahu Kiperwasser, Sara Stymne, Yoav Goldberg, and Joakim Nivre. 2017. From raw text to universal dependencies - look, no tags! In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Association for Computational Linguistics, Vancouver, Canada, pages 207–217.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*. pages 742–750.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics* 1:403–414.

Carlos Gómez-Rodríguez and Daniel Fernández-González. 2015. An efficient dynamic oracle for unrestricted non-projective parsing. In *ACL-15-SHORT*. pages 256–261.

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 (ACL)*. pages 1492–1501.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics* 4:313–327.

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 (ACL)*. pages 673–682.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*. 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. 2007. Incremental non-projective dependency parsing. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*. pages 396–403.

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, Željko Agić, Lars Ahrenberg, et al. 2017. Universal Dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague. http://hdl.handle.net/11234/1-1983.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*. pages 73–76.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*. pages 99–106.

Milan Straka, Jan Hajič, Jana Straková, and Jan Hajič jr. 2015. Parsing universal dependency treebanks using neural networks and search-based oracle. In *Proceedings of the 14th Workshop on Treebanks and Linguistic Theories (TLT)*.