Bracketing Encodings for 2-Planar Dependency Parsing

Michalina Strzyz **David Vilares** **Carlos Gómez-Rodríguez**
Universidade da Coruña, CITIC
FASTPARSE Lab, LyS Research Group
Departamento de Ciencias de la Computación y Tecnologías de la Información
Campus de Elviña, s/n, 15071 A Coruña, Spain
{michalina.strzyz,david.vilares,carlos.gomez}@udc.es

Abstract

We present a bracketing-based encoding that can be used to represent any 2-planar dependency tree over a sentence of length n as a sequence of n labels, hence providing almost total coverage of crossing arcs in sequence labeling parsing. First, we show that existing bracketing encodings for parsing as labeling can only handle a very mild extension of projective trees. Second, we overcome this limitation by taking into account the well-known property of 2-planarity, which is present in the vast majority of dependency syntactic structures in treebanks, i.e., the arcs of a dependency tree can be split into two planes such that arcs in a given plane do not cross. We take advantage of this property to design a method that balances the brackets and that encodes the arcs belonging to each of those planes, allowing for almost unrestricted non-projectivity ($\sim 99.9\%$ coverage) in sequence labeling parsing. The experiments show that our linearizations improve over the accuracy of the original bracketing encoding in highly non-projective treebanks (on average by 0.4 LAS), while achieving a similar speed. Also, they are especially suitable when PoS tags are not used as input parameters to the models.

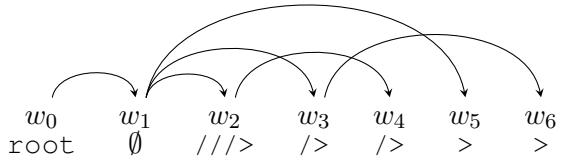
1 Introduction

In the last few years, approaches that cast syntactic parsing as the task of finding a sequence have gained traction for both dependency and constituency parsing. In sequence-to-sequence (seq2seq) parsing (Vinyals et al., 2015; Li et al., 2018), parse trees are represented as arbitrary-length sequences, where the attention mechanism can be seen as an abstraction of the stack and the buffer in transition-based systems that decides what words are relevant to make a decision at a given time step. In sequence labeling parsing (Gómez-Rodríguez and Vilares, 2018; Strzyz et al., 2019b), the tree for a sentence of length n is represented as a sequence of n labels, one per word, so the parsing process is word-synchronous (Kitaev and Klein, 2019) and can be addressed by frameworks traditionally used for other natural language processing tasks, such as part-of-speech tagging or named-entity recognition. Current sequence labeling parsers combine competitive accuracy with high computational efficiency, while providing extra simplicity using off-the-shelf sequence labeling software without the need for ad-hoc parsing algorithms.

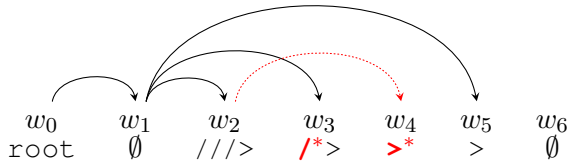
In the realm of dependency parsing, pioneering work dates back to Spoustová and Spousta (2010), who used a relative PoS-tag based encoding to represent trees as label sequences, but the resulting accuracy was not practical even for the standards of the time, probably due to the inability of pre-deep-learning architectures to successfully learn the representation. Using more modern architectures with the ability to contextualize words based on the sentence, and various tree encodings, Strzyz et al. (2019b) were the first to show that competitive accuracy could be reached. Subsequently, this accuracy has been improved further by techniques like the use of multi-task learning to parse dependencies and constituents together (Strzyz et al., 2019a) and of contextualized embeddings (Vilares et al., 2020).

While parsing as sequence labeling does not need specific parsing algorithms or data structures, as in graph-based or transition-based parsing, the responsibility of providing suitable parsing representations

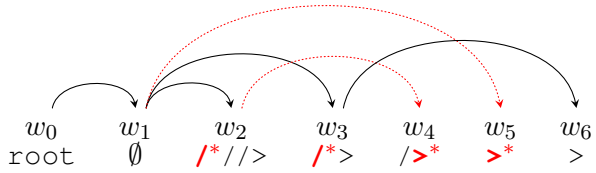
Proceedings of the 28th International Conference on Computational Linguistics, pages 2472–2484
Barcelona, Spain (Online), December 8-13, 2020



(a) Projective encoding restricted to a single plane. Infeasible to reconstruct a non-projective sentence.



(b) Non-projective 2-planar encoding with second-plane-averse greedy plane assignment. The arc $w_3 \rightarrow w_6$ is not assigned a plane because it would cross arcs belonging to both planes, which is forbidden by the 2-planar constraint.



(c) Non-projective 2-planar encoding with plane assignment based on restriction propagation on the crossings graph.

Figure 1: Bracketing-based encodings with their plane assignment strategies for a non-projective sentence. The red, dotted lines refer to the arcs represented in the second plane, denoted by * in the encoding label.

with reasonable coverage and learnability falls instead on the encoding used to represent trees as sequences of labels. Strzyz et al. (2019b) used four different encodings that obtained substantially different parsing accuracies in the experiments, with two encodings achieving competitive accuracy: the relative PoS tag (rel-PoS) encoding of Spoustová and Spousta (2010) and a new encoding based on balanced brackets, inspired by Yli-Jyrä and Gómez-Rodríguez (2017). While the encoding of Spoustová and Spousta (2010) achieved a good accuracy, and it has full coverage of non-projective dependency trees, it requires PoS tags to encode the dependency arcs. This can be seen as a weakness, not just because computing and feeding PoS tags increases the latency, but also because the traditional assumption that PoS tagging is needed for parsing is being increasingly called into question (de Lhoneux et al., 2017; Smith et al., 2018; Kitaev and Klein, 2018; Anderson and Gómez-Rodríguez, 2020). Low-frequency PoS tags can cause sparsity in the encoding, and low-quality PoS tags could be a potential source of errors in low-resource languages. For this reason, Lacroix (2019) proposed two alternative encodings with the same relative indexing philosophy, but without using PoS tags. However, these encodings require a composition of two sequence labeling processes instead of one.

On the other hand, the bracketing encoding inspired in (Yli-Jyrä and Gómez-Rodríguez, 2017) represents the trees independently of PoS tags or any other previous tagging step, but it has the limitation of being restricted to a very mild extension of projective trees.

Contribution. In this paper, we extend the idea of the bracketing-based encoding to non-projective parsing by defining a variant that can encode all 2-planar dependency trees (Yli-Jyrä, 2003). 2-planar dependency trees have been shown to cover the vast majority of non-projective trees in attested sentences (Gómez-Rodríguez, 2016) and have been used in transition-based parsing (Gómez-Rodríguez and Nivre, 2013; Fernández-González and Gómez-Rodríguez, 2018). We show that our encoding provides better parsing accuracy than the original bracketing-based encoding on highly non-projective UD treebanks; and than the rel-PoS encoding when assuming PoS tags are not fed as input parameters to the models. The source code is available at `https://github.com/mstrise/dep2label`.

2 Preliminaries

Given a sentence $w_1 \dots w_n$, we associate the words with nodes $0, 1, \dots, n$, where 0 is a dummy root node. Then, a dependency graph is an edge-labeled graph (V, E) with $V = \{0, 1, \dots, n\}$ and E a set of edges of the form (h, d, l) where $h \in V$ is the head, $d \in V \setminus \{0\}$ is the dependent, and l is the dependency label. The goal of a dependency parser is to find a dependency graph that is a tree (i.e. without cycles, and with no dependent having more than one head) rooted at node 0.

2.1 Bracketing encoding

Dependency arcs are encoded through a sequence of bracket elements from a set $B = \{<, \backslash, /, >\}$. A balanced pair of brackets $(<, \backslash)$ in the labels of the words w_i and w_j represents a left arc from word w_j to w_{i-1}. A balanced pair of brackets $(/, >)$ in the labels of the words w_i and w_j represents a right arc from word w_{i-1} to w_j. A token can have one incoming arc and several outgoing arcs, resulting on labels composed of several such brackets, following the regular expression `(<)?((\)*|(/)*)(>)?`. As shown in Figure 1a, the token w_2 is assigned a label `///>` that can be interpreted as: the previous token w_1 has three outgoing arcs to the right and one of them matches the left incoming arc of w_2 (`/>`) meaning that w_1 is the head of w_2. The remaining two dependents will be given by the matching `>` in the labels of the following words.

Since each opening bracket is always matched to the closest same-direction closing bracket, this encoding is unable to handle crossing arcs in the same direction. An attempt of encoding such crossing arcs will result in decoding into non-crossing arcs. However, the encoding can handle crossing arcs in opposite directions, as long as left and right brackets are balanced independently (e.g. by using separate stacks for each kind of bracket). The paper by Strzyz et al. (2019b) erroneously describes the encoding as only supporting projective trees. In fact, the implementation in that paper is supporting this mild extension of projectivity where crossing arcs in opposite directions are allowed.

2.2 2-Planarity

A dependency graph (V, E) is said to be k-planar, for $k \geq 1$, if there is a partition of the edges into sets $E_1, \dots, E_k$, called planes, in such a way that edges that are in the same plane do not cross. For $k = 1$, this corresponds to the concept of a noncrossing dependency graph (Kuhlmann and Jonsson, 2015) or planar linear arrangement (Chao and Sha, 1992) (not to be confused with a planar graph). Under the assumption of trees rooted at the dummy root node 0, 1-planar trees are equivalent to the well-known projective trees. For $k \geq 2$, this means that the dependency graph (together with the linear order of the words) is a k-page book embedding of a graph (see (Pitler et al., 2013)). Intuitively, a k-planar graph is one where each arc can be assigned one out of k colors in such a way that arcs with the same color do not cross (see Figure 1).

2-planarity has been shown to be particularly relevant for parsing, as the overwhelming majority of syntactic structures in syntactic treebanks has been shown to be 2-planar (Gómez-Rodríguez and Nivre, 2013; Gómez-Rodríguez, 2016) and efficient transition-based parsers have been proposed for this set of structures (Gómez-Rodríguez and Nivre, 2010; Fernández-González and Gómez-Rodríguez, 2018).

3 2-Planar bracketing encodings

In order to support the extended non-projective coverage provided by 2-planarity in the bracketing system, we balance a different set of brackets for each plane. We introduce a set of "star" bracket elements denoting arcs belonging to the second plane, $B^* = \{<^*, \backslash^*, /^*, >^*\}$. A token w_i can be assigned elements from both B and B^*. Brackets only match when they are on the same plane, i.e., $(<, \backslash)$, $(/, >)$ are matching pairs of brackets that encode arcs in the first plane, and $(<^*, \backslash^*)$, $(/^*, >^*)$ are matching pairs of brackets that encode arcs in the second plane. The decoding process is implemented by operating on separate stacks for the first-plane brackets and the second-plane brackets.

3.1 Plane assignment strategies

According to the definition in Section 2.2, a tree is 2-planar if its edges can be partitioned into two planes, E_1 and E_2, such that edges in the same plane do not cross. However, often this partition is not unique

(for example, in the case of trees that are also 1-planar, *any* partition satisfies the condition). Thus, for the encoding in Section 3 to provide a single sequence of labels for each gold tree during training, we need to fix a *plane assignment strategy*, i.e., a canonical way of assigning each arc to a plane to obtain such a partition. While the number of possible partitions is exponential in the size of the tree, desirable partitions should be easily learnable, i.e., follow predictable patterns. Given that the amount of crossing dependencies in treebanks is scarce (Ferrer-i-Cancho et al., 2018), it makes sense to look for partitions that do not make use of an extra plane when not needed, so that the parsing of sentences or fragments without crossing arcs does not become more difficult or need more output labels than in the basic bracketing encoding (as they will only use one plane and thus one set of brackets). Following this general principle, we define the following plane assignment strategies:

Second-Plane-Averse Greedy Plane Assignment Arcs in the gold tree are traversed in left-to-right order of their right endpoint, with shortest arcs first when they share a right endpoint (this is the order in which arcs will be decoded using a stack, see Section 4). For each arc a, we assign the first plane if possible (i.e., if no arc crossing a has already been assigned the first plane). Otherwise, we assign the second plane if possible, or no plane if the arc a crosses arcs assigned to both planes. The process is formally described with pseudocode in Algorithm 1.

Algorithm 1: 2p-greedy

```
Input: A set of arcs T, and input length n
Result: Two sets (planes) of arcs P_1, P_2
P_1 ← ∅;
P_2 ← ∅;
for x_r ← 1 to n do
    for x_l ← x_r − 1 downto 0 do
        if ∃a ∈ T | a = (x_l, x_r, l) ∨ a = (x_r, x_l, l) then
            nextArc ← a;
            C ← {b ∈ (P_1 ∪ P_2) | b crosses a};
            if C ∩ P_1 = ∅ then
                P_1 ← P_1 ∪ {nextArc};
            else if C ∩ P_2 = ∅ then
                P_2 ← P_2 ∪ {nextArc};
            else
                do nothing (failed to assign nextArc to a plane);
        end
    end
end
return P_1, P_2;
```

Second-Plane-Averse Plane Assignment based on Restriction Propagation on the Crossings Graph While the greedy approach is very simple, it has the disadvantage that it may make suboptimal decisions leading to reduced coverage: assigning an arc to a given plane may seem like a good local decision, but depending on how arcs cross each other in the whole tree, it may lead to a subsequent situation where an arc cannot be assigned a plane even if the tree is actually 2-planar.

An example of this can be seen in Figure 1b: the greedy strategy will assign the arcs $w_1 \rightarrow w_3$ and $w_1 \rightarrow w_5$ to the first plane, which in a local context is the simplest thing to do. However, the fact that $w_1 \rightarrow w_3$ crosses $w_2 \rightarrow w_4$ (which is thus assigned to the second plane) and $w_3 \rightarrow w_6$ crosses both $w_1 \rightarrow w_5$ (first plane) and $w_2 \rightarrow w_4$ (second plane) then means that it is impossible to assign a plane to the arc $w_3 \rightarrow w_6$. This could have been prevented by assigning arc $w_1 \rightarrow w_5$ to the second plane, but a greedy algorithm has no way to anticipate this. To deal with this problem, we propagate restrictions by traversing the crossings graph, i.e., a graph where its nodes represent the edges in the gold tree and two

nodes are linked if the corresponding edges cross (Gómez-Rodríguez and Nivre, 2013). Whenever we assign a given arc to plane 1, then we forbid plane 1 for its neighbors in the crossings graph (i.e. the arcs that cross it), we forbid plane 2 for the neighbors of its neighbors, plane 1 for the neighbors of those, and so on. For arcs assigned to plane 2, we proceed symmetrically.

Thus, the traversal order of arcs is the same as in the previous strategy, but for each new arc a, we look at the restrictions and assign it to the first plane if allowed, otherwise to the second plane if allowed, and finally to no plane if neither are allowed. In this case, the latter will only happen for non-2-planar trees: it is easy to show that situations where both planes are forbidden for the same arc can only happen if the crossings graph has a cycle of odd length, which is equivalent to the tree not being 2-planar (see (Gómez-Rodríguez and Nivre, 2013)). Thus, this strategy guarantees full coverage of 2-planar structures. The pseudocode of the strategy can be seen in Algorithm 2, where $\overline{P_1}$ and $\overline{P_2}$ represent the arcs forbidden from planes 1 and 2, respectively.

Algorithm 2: 2p-prop

Input: A set of arcs T, and input length n
Result: Two sets (planes) of arcs P_1, P_2
function `Propagate` (*Edge sets* $T, \overline{P_1}, \overline{P_2}$*, Edge* e*, Plane* i) **:**
  $\overline{P_i} \leftarrow \overline{P_i} \cup \{e\}$;
  // e forbidden from plane i
  for ($e' \in T \mid e'$ crosses e) **do**
    if $e' \notin \overline{P_{3-i}}$ **then**
      $(\overline{P_1}, \overline{P_2}) \leftarrow$ `Propagate` $(T, \overline{P_1}, \overline{P_2}, e', 3-i)$;
  end
  return $\overline{P_1}, \overline{P_2}$;
$P_1 \leftarrow \varnothing$, $P_2 \leftarrow \varnothing$, $\overline{P_1} \leftarrow \varnothing$, $\overline{P_2} \leftarrow \varnothing$;
for $x_r \leftarrow 1$ ***to*** n **do**
  for $x_l \leftarrow x_r - 1$ ***downto*** 0 **do**
    if $\exists a \in T \mid a = (x_l, x_r, l) \vee a = (x_r, x_l, l)$ **then**
      nextArc $\leftarrow a$;
      if nextArc $\notin \overline{P_1}$ **then**
        $P_1 \leftarrow P_1 \cup \{$nextArc$\}$;
        `Propagate` $(T, \overline{P_1}, \overline{P_2}$,*nextArc*,$2)$;
      else if nextArc $\notin \overline{P_2}$ **then**
        $P_2 \leftarrow P_2 \cup \{$nextArc$\}$;
        `Propagate` $(T, \overline{P_1}, \overline{P_2}$,*nextArc*,$1)$;
      else
        do nothing (failed to assign nextArc to a plane);
    end
  end
end
return P_1, P_2;

Switch-averse plane assignment strategies Another possibility is to implement variants of the previous two strategies that are *switch-averse*, rather than *second-plane-averse*. These variants work like the previous strategies, except for the difference than when both planes can be assigned to the current arc, we assign the last plane used, instead of always preferring to assign the first plane.

The implementation of the 2-planar transition-based parser by Gómez-Rodríguez and Nivre (2010) used a switch-averse restriction-propagation strategy. This is a reasonable choice because in their transition-based parser it minimizes the number of transitions used: the algorithm's state holds the "current" plane being used, and switching to the other plane costs one transition. In our sequence labeling context, where

Language	% non-projective sentences	% non-projective dependencies	Language	% non-projective sentences	% non-projective dependencies
Ancient Greek$_{\text{Perseus}}$	63.87	10.14	Korean$_{\text{Kaist}}$	21.70	2.55
Basque$_{\text{BDT}}$	33.17	4.69	Danish$_{\text{DDT}}$	21.50	1.74
Hungarian$_{\text{Szeged}}$	27.11	1.97	Gothic$_{\text{PROIEL}}$	17.57	2.53
Portuguese$_{\text{Bosque}}$	23.31	1.85	Lithuanian$_{\text{HSE}}$	17.49	1.27
Urdu$_{\text{UDTB}}$	22.57	1.32	*Japanese$_{GSD}$*	*0*	*0*
Afrikaans$_{\text{AfriBooms}}$	22.34	1.62	*Galician$_{CTG}$*	*0*	*0*

Table 1: Percentage of non-projective sentences and dependencies of the selected UD treebanks, where Japanese$_{\text{GSD}}$ and Galician$_{\text{CTG}}$ are control treebanks.

this is no longer true (the model always makes n predictions for a sequence of length n), we made some initial experiments with switch-averse strategies but we found that they performed consistently (albeit slightly) worse than second-plane-averse strategies, so we discarded them for our experiments.

4 Bracketing decoding

When a sentence is represented with the bracketing encoding in a single plane, a valid left arc is associated with a pair of matching brackets < and \ while a right arc is associated with a pair of / and >. For each sentence we create two initially empty stacks, σ_L and σ_R, in order to keep the elements separate with respect to the arc direction. Thus, the output labels generated by the system are read from left to right, decomposed into their brackets, and then brackets corresponding to left arcs are processed in σ_L and those that encode right arcs are processed in σ_R. In order to handle a second plane with brackets represented as ($<^*$, $\backslash^*$) and ($/^*$, $>^*$), we simply use additional stacks: σ_L^* and σ_R^*.

More particularly, decoding proceeds by reading a label for each token and pushing each opening bracketing element to the corresponding stack while preserving the token's index. For instance, when reading a new label that contains <, the bracket element is pushed into the σ_L stack and can only be popped once there is a later matching label with a closing bracketing element \ that will be used to create a left arc, by recovering the index stored together with the < bracket. Analogously, right arcs are processed in the same way, but in a different stack.

Postprocessing Decoded labels do not ensure creating a well-formed tree. For that reason, we adapt some common heuristics for all encodings in order to postprocess them. In case some of the brackets in any of the stacks are unbalanced, the outermost bracket elements are discarded. Tokens that are not assigned any head are recovered by attaching them to the word that is attached to the dummy root (i.e., the syntactic head of the sentence). Cycles are also solved by removing the leftmost arc in the cycle.

5 Experiments

Data We extracted the most non-projective treebanks from UDv2.4 (Nivre and others, 2019) based on the percentage of non-projective sentences, and discarded some of them due to the lack of a pre-trained UDpipe model or due to the lack of a development set. The selected treebanks were: Ancient Greek$_{\text{Perseus}}$, Basque$_{\text{BDT}}$, Hungarian$_{\text{Szeged}}$, Portuguese$_{\text{Bosque}}$, Urdu$_{\text{UDTB}}$, Afrikaans$_{\text{AfriBooms}}$, Korean$_{\text{Kaist}}$, Danish$_{\text{DDT}}$, Gothic$_{\text{PROIEL}}$, Lithuanian$_{\text{HSE}}$. In addition, two fully projective treebanks (Galician$_{\text{CTG}}$ and Japanese$_{\text{GSD}}$) were included as control treebanks. Table 1 shows the selected treebanks with their percentage of non-projective sentences and dependencies. For all of them, we ran UDPipe models (Straka and Straková, 2017) to obtain predicted segmentation and tokenization. We also computed predicted PoS tags, but they were not used (nor gold PoS tags were) to train *any* of the models, but just to decode the labels from the rel-PoS encoding (Strzyz et al., 2019b). In addition, we included dummy beginning- and end-of-sentence tokens (BOS,EOS) as in previous work in parsing as labeling.

Model For our experiments we use bidirectional long short-term memory networks (Hochreiter and Schmidhuber, 1997; Schuster and Paliwal, 1997) as implemented in the NCRF++ framework (Yang and

Zhang, 2018).[1] Each input word w_i is represented as a vector which comes from a concatenation of (i) an external pre-trained word embedding, which is further fine-tuned during training, and (ii) a second word embedding which results from the output of a char-LSTM, which is trained end-to-end together with the rest of the network.

In this context, let $\text{LSTM}_\theta(\vec{x})$ be a black-box long short-term memory network that processes the sequence of vectors $\vec{x} = [\vec{x}_1, ..., \vec{x}_{|\vec{x}|}]$, then the output for $\vec{x}_i$ is a hidden vector $\vec{h_i}$ which represents the word based on its left and right sentence context:

$$\vec{h}_i = \text{BiLSTM}_\theta(\vec{x}, i) = \text{LSTM}^l_\theta(\vec{x}_{[1:i]}) \circ \text{LSTM}^r_\theta(\vec{x}_{[|\vec{x}|:i]}). \quad (1)$$

More particularly, we stack 2 BiLSTMs before computing the output layer. For this, we consider a simple hard-sharing multi-task learning architecture, where each $\vec{h_i}$ is sent to three separate layers in order to generate the classifications through regular softmaxes: two labels predicted for each plane (one label per plane)[2] and another one for the word's dependency relation. Afterwards, label decoding is followed by a postprocessing step with some heuristics to ensure a valid dependency tree (as described in §4).

5.1 Analysis and results

Next, we compare the performance of the original bracketing encoding (1p-brackets), 2-planar with greedy plane assignment (2p-greedy) and 2-planar with restriction propagation (2p-prop) with respect to their theoretical arc coverage, as well as their empirical recall and precision. For UAS/LAS, we also report results for models trained on the rel-PoS encoding.

Language	1p-brackets	2p-greedy	2p-prop	Language	1p-brackets	2p-greedy	2p-prop
Ancient Greek$_{\text{Perseus}}$	89.53	99.27	**99.33**	Afrikaans$_{\text{AfriBooms}}$	98.65	**99.99**	**99.99**
Basque$_{\text{BDT}}$	94.85	**99.85**	99.62	Korean$_{\text{Kaist}}$	98.42	**100.00**	**100.00**
Hungarian$_{\text{Szeged}}$	97.57	99.96	**99.98**	Danish$_{\text{DDT}}$	98.10	**99.97**	99.96
Portuguese$_{\text{Bosque}}$	98.10	**99.95**	99.88	Gothic$_{\text{PROIEL}}$	97.58	99.94	**99.98**
Urdu$_{\text{UDTB}}$	98.68	**99.95**	99.94	Lithuanian$_{\text{HSE}}$	98.35	99.97	**100.00**

Table 2: Percentage of arcs covered by the proposed encodings on the gold training set from highly non-projective treebanks.

Theoretical advantage Table 2 compares the dependency arc coverage by the encodings on the gold training sets. It is easy to conclude that the 2-planar encodings almost fully succeed to reconstruct highly non-projective datasets, while the bracketing encoding suffers more. When comparing both plane assignments for the 2-planar encodings, we see that the coverage of 2p-greedy is already so high (99.9% or more in all but two treebanks) that the extra coverage provided by 2p-prop is not large in absolute terms. In fact, in some treebanks, 2p-prop even has slightly less measured coverage than 2p-greedy, even though (as explained earlier) the former guarantees full coverage of 2-planar trees while the latter does not. This can be explained because there are *non-2-planar* trees where 2p-greedy happens to cover more arcs. In such trees, the theoretical guarantee provided by 2p-prop does not apply.

With respect to the number of labels that each encoding generates (which will directly impact the output size of the softmax layers), Table 3 shows the comparison of the output vocabulary sets for each of the tasks in the multi-task learning setup. We can see that, for most languages, bracketing encodings generate a smaller tag set than rel-PoS[3]; and in general, the 2-planar encodings do not produce increases in tagset size with respect to the 1-planar bracketing encoding. In fact, for the most non-projective languages (like Ancient Greek or Basque), the 2-planar encodings clearly compress the tag set as, in spite of having a larger variety of brackets, they appear distributed among the two planes so that the bracket strings in each label will tend to be shorter.

[1] We omit however the CRF on top of the BiLSTMs.

[2] If a given word has no arcs associated to that plane, we generate an empty label $\emptyset$.

[3] In the rel-PoS encoding, each label word represents the head based on a PoS-tag offset. See also (Strzyz et al., 2019b).

Language	Encoding	Task 1 (1st plane)	Task 2 (2nd plane)	Task 3 (deprel)	Language	Encoding	Task 1 (1st plane)	Task 2 (2nd plane)	Task 3 (deprel)	Language	Encoding	Task 1 (1st plane)	Task 2 (2nd plane)	Task 3 (deprel)
Ancient Greek	rel-PoS	166	–	27	Urdu	rel-PoS	190	–	27	Gothic	rel-PoS	121	–	34
	1p-brackets	210	–	27		1p-brackets	95	–	27		1p-brackets	114	–	34
	2p-greedy	108	37	27		2p-greedy	80	22	27		2p-greedy	78	18	34
	2p-prop	109	39	27		2p-prop	80	22	27		2p-prop	78	19	34
Basque	rel-PoS	132	–	32	Afrikaans	rel-PoS	110	–	28	Lithuanian	rel-PoS	89	–	38
	1p-brackets	134	–	32		1p-brackets	77	–	28		1p-brackets	57	–	38
	2p-greedy	84	25	32		2p-greedy	62	15	28		2p-greedy	46	11	38
	2p-prop	83	25	32		2p-prop	62	15	28		2p-prop	46	12	38
Hungarian	rel-PoS	128	–	56	Korean	rel-PoS	134	–	32	*Japanese*	*rel-PoS*	*77*	–	*27*
	1p-brackets	101	–	56		1p-brackets	89	–	32		*1p-brackets*	*45*	–	*27*
	2p-greedy	71	19	56		2p-greedy	73	14	32		*2p-greedy*	*45*	*3*	*27*
	2p-prop	71	21	56		2p-prop	73	14	32		*2p-prop*	*45*	*3*	*27*
Portuguese	rel-PoS	192	–	43	Danish	rel-PoS	150	–	38	*Galician*	*rel-PoS*	*132*	–	*26*
	1p-brackets	110	–	43		1p-brackets	128	–	38		*1p-brackets*	*82*	–	*26*
	2p-greedy	88	25	43		2p-greedy	97	23	38		*2p-greedy*	*82*	*3*	*26*
	2p-prop	88	27	43		2p-prop	96	25	38		*2p-prop*	*82*	*3*	*26*

Table 3: Label size of each encoding based on the training and dev set for each treebank. Each task contains three additional labels: BOS, EOS and $\emptyset$. Hence the Japanese and Galician treebanks have three labels for the second plane, although they are fully projective.

Language	1p-brackets		2p-greedy		2p-prop	
	P	R	P	R	P	R
Ancient Greek$_{\text{Perseus}}$	85.74	54.34	86.33	63.85	**87.58**	**66.23**
Basque$_{\text{BDT}}$	69.87	45.80	70.14	**52.97**	**72.77**	52.80
Hungarian$_{\text{Szeged}}$	37.17	66.98	35.51	71.70	**37.80**	**74.53**
Portuguese$_{\text{Bosque}}$	52.94	24.77	55.84	39.45	**61.64**	**41.28**
Urdu$_{\text{UDTB}}$	36.63	**36.63**	38.10	31.68	**39.78**	**36.63**
Afrikaans$_{\text{AfriBooms}}$	40.99	65.35	46.72	63.37	**46.94**	**68.32**
Korean$_{\text{Kaist}}$	59.45	**49.54**	62.24	47.03	**62.80**	47.03
Danish$_{\text{DDT}}$	45.54	**48.57**	**46.36**	**48.57**	45.37	46.67
Gothic$_{\text{PROIEL}}$	50.50	26.42	**58.88**	32.64	56.00	**36.27**
Lithuanian$_{\text{HSE}}$	**34.38**	**91.67**	27.59	66.67	33.33	83.33
Average	51.32	51.01	52.77	51.79	**54.40**	**55.31**

Table 4: Models' precision and recall of non-projective sentences on the test set.

Language	1p-brackets		2p-greedy		2p-prop	
	P	R	P	R	P	R
Ancient Greek$_{\text{Perseus}}$	20.82	10.32	**32.40**	18.65	31.40	**19.16**
Basque$_{\text{BDT}}$	18.41	11.80	28.11	19.83	**31.76**	**20.40**
Hungarian$_{\text{Szeged}}$	1.57	4.05	3.13	9.25	**4.06**	**10.98**
Portuguese$_{\text{Bosque}}$	10.87	5.18	14.50	9.84	**20.18**	**11.92**
Urdu$_{\text{UDTB}}$	3.26	**3.92**	0.69	0.65	**3.41**	**3.92**
Afrikaans$_{\text{AfriBooms}}$	11.04	18.09	**13.09**	**19.15**	12.59	18.62
Korean$_{\text{Kaist}}$	28.12	**21.68**	**32.26**	21.37	31.06	21.53
Danish$_{\text{DDT}}$	7.66	11.35	**12.96**	**19.86**	9.45	13.48
Gothic$_{\text{PROIEL}}$	11.11	5.17	**19.63**	11.03	17.46	**11.38**
Lithuanian$_{\text{HSE}}$	0	0	0	0	**1.64**	**6.25**
Average	11.29	9.16	15.68	12.96	**16.30**	**13.76**

Table 5: Models' precision and recall of non-projective dependencies on the test set.

Results To investigate how the coverage in Table 2 translates into non-projective performance in actual parsing, we report models' precision and recall. In Table 4, the precision and recall on *non-projective sentences*[4] increase across the treebanks with 2-planar models, suggesting that they are capable of identifying non-projective sentences to a greater extent than the original bracketing model. Table 5 shows that 2p-greedy and 2p-prop models improve the recall and precision of *non-projective dependencies* in

[4]Precision and recall on non-projective sentences are computed by looking whether a given sentence is identified as non-projective (i.e. given a non-projective parse), disregarding the correctness of the predicted non-projective dependencies for that sentence.

the majority of treebanks.[5] Again, 2-planar encodings outperform the original bracketing baseline, even though the latter is able to cover non-projectivity to some degree (crossing arcs pointing in opposite directions). Both 2p-greedy and 2p-prop obtain similar scores, showing that their coverage is comparable.

Language	Encoding	dev		test		Language	Encoding	dev		test	
		UAS	LAS	UAS	LAS			UAS	LAS	UAS	LAS
Ancient Greek$_{\text{Perseus}}$	rel-PoS	65.29	58.27	62.91	55.07	Korean$_{\text{Kaist}}$	rel-PoS	81.47	78.50	77.25	73.92
	1p-brackets	64.70	57.21	63.36	54.80		1p-brackets	84.54	81.54	**82.37**	**79.03**
	2p-greedy	67.10	59.97	**65.90**	**57.15**		2p-greedy	85.01	82.01	82.33	78.91
	2p-prop	67.06	59.84	65.11	56.55		2p-prop	84.65	81.73	82.32	**79.03**
Basque$_{\text{BDT}}$	rel-PoS	77.48	72.91	75.28	70.19	Danish$_{\text{DDT}}$	rel-PoS	78.28	74.93	77.07	73.45
	1p-brackets	80.13	75.37	78.37	72.95		1p-brackets	80.60	76.59	78.25	73.94
	2p-greedy	79.98	75.18	78.13	72.63		2p-greedy	80.68	76.80	78.49	74.07
	2p-prop	80.44	75.56	**78.58**	**73.08**		2p-prop	81.15	77.27	**78.87**	**74.42**
Hungarian$_{\text{Szeged}}$	rel-PoS	72.58	67.13	66.19	59.32	Gothic$_{\text{PROIEL}}$	rel-PoS	65.25	58.58	**67.14**	**59.72**
	1p-brackets	75.09	69.13	67.80	60.50		1p-brackets	65.26	57.92	66.63	59.02
	2p-greedy	75.47	69.33	**68.07**	**60.74**		2p-greedy	65.26	58.05	66.84	59.26
	2p-prop	75.26	69.05	67.95	60.63		2p-prop	65.29	57.91	66.25	58.41
Portuguese$_{\text{Bosque}}$	rel-PoS	87.10	84.28	84.74	81.02	Lithuanian$_{\text{HSE}}$	rel-PoS	39.04	26.37	31.05	19.70
	1p-brackets	88.88	85.78	**86.67**	**82.44**		1p-brackets	40.97	25.63	34.62	19.42
	2p-greedy	88.88	85.76	86.51	82.39		2p-greedy	41.34	26.46	**35.08**	20.45
	2p-prop	89.00	85.82	86.52	82.17		2p-prop	44.19	29.03	34.80	**21.29**
Urdu$_{\text{UDTB}}$	rel-PoS	80.98	75.09	81.18	75.26	*Japanese$_{GSD}$*	*rel-PoS*	*76.60*	*75.83*	*74.83*	*73.96*
	1p-brackets	84.22	77.23	**84.28**	77.19		*1p-brackets*	*78.73*	*77.67*	*77.34*	*76.10*
	2p-greedy	84.01	77.16	84.08	77.19		*2p-greedy*	*78.81*	*77.78*	*77.47*	*76.24*
	2p-prop	83.89	77.30	84.26	**77.41**		*2p-prop*	*78.81*	*77.78*	*77.47*	*76.24*
Afrikaans$_{\text{AfriBooms}}$	rel-PoS	79.00	74.58	78.93	74.65	*Galician$_{CTG}$*	*rel-PoS*	*79.72*	*76.40*	*78.36*	*75.05*
	1p-brackets	80.77	75.54	79.52	74.86		*1p-brackets*	*80.82*	*77.35*	*80.02*	*76.33*
	2p-greedy	81.41	76.33	**80.13**	**75.53**		*2p-greedy*	*80.90*	*77.36*	*79.91*	*76.32*
	2p-prop	81.50	76.30	79.96	75.43		*2p-prop*	*80.90*	*77.36*	*79.91*	*76.32*

Table 6: UAS and LAS (%) for the respective encodings on the predicted dev and test set of highly non-projective treebanks and control treebanks.

Table 6 compares the LAS and UAS performance of the 1- and 2-planar, and also of the rel-PoS encoding.[6] 2-planar encodings outperform the existing bracketing encoding in the majority of treebanks. The gains vary between languages but on average 2p-greedy improves UAS by 0.4 and 2p-prop by 0.3, and both improve LAS by 0.4 across highly non-projective treebanks. Comparing both assignment strategies for the 2-planar encoding, the theoretical advantage in coverage provided by 2p-prop over 2p-greedy does not translate into accuracy gains in general, as the actual difference in coverage is small when measured in the treebanks (as was seen in Table 2) and the simpler greedy assignment strategy is likely to be easier to learn by the machine learning setup.

Since the syntactic dependencies are represented by a finite set of labels that have been seen in the training and development sets, as in all parsing as sequence labeling approaches, it is expected that at test time our model may encounter unseen labels. In Appendix B we show the label coverage of all encodings on the test set. In general, it seems that the unseen labels do not have significant impact on the overall performance due to their rare occurrence.

[5]The reported precision and recall for Lithuanian is lower than for the other treebanks. As we show in Appendix A, the Lithuanian treebank contains a small number of sentences and therefore it is hard to draw robust conclusions about its poor performance.

[6]Note that the implementations compared here do not use PoS tags as features. This is sensible for the bracketing encodings, and for the focus of this paper where we are interested in encodings that can be run using raw words as the only input, but differs from the standard setup in the rel-PoS encoding (where using PoS tags comes at no extra significant cost, because they need to be computed for decoding in any case).

Language	Encoding	sent/s CPU	sent/s GPU	Language	Encoding	sent/s CPU	sent/s GPU	Language	Encoding	sent/s CPU	sent/s GPU
Ancient Greek$_{\text{Perseus}}$	rel-PoS	305*	1012*	Urdu$_{\text{UDTB}}$	rel-PoS	182*	625*	Gothic$_{\text{PROIEL}}$	rel-PoS	442*	1718*
	1p-brackets	303	1011		1p-brackets	186	616		1p-brackets	447	1718
	2p-greedy	288	889		2p-greedy	174	544		2p-greedy	434	1598
	2p-prop	289	886		2p-prop	175	549		2p-prop	435	1544
Basque$_{\text{BDT}}$	rel-PoS	387*	1461*	Afrikaans$_{\text{AfriBooms}}$	rel-PoS	228*	861*	Lithuanian$_{\text{HSE}}$	rel-PoS	239*	828*
	1p-brackets	388	1454		1p-brackets	228	857		1p-brackets	235	769
	2p-greedy	378	1369		2p-greedy	220	805		2p-greedy	228	740
	2p-prop	378	1369		2p-prop	221	805		2p-prop	229	730
Hungarian$_{\text{Szeged}}$	rel-PoS	219*	802*	Korean$_{\text{Kaist}}$	rel-PoS	442*	1718*	*Japanese$_{GSD}$*	*rel-PoS*	*214**	*663**
	1p-brackets	221	797		1p-brackets	447	1718		*1p-brackets*	*214*	*661*
	2p-greedy	212	739		2p-greedy	434	1598		*2p-greedy*	*206*	*611*
	2p-prop	213	750		2p-prop	435	1544		*2p-prop*	*205*	*616*
Portuguese$_{\text{Bosque}}$	rel-PoS	242*	868*	Danish$_{\text{DDT}}$	rel-PoS	442*	1718*	*Galician$_{CTG}$*	*rel-PoS*	*175**	*752**
	1p-brackets	246	872		1p-brackets	447	1718		*1p-brackets*	*177*	*756*
	2p-greedy	236	811		2p-greedy	434	1598		*2p-greedy*	*170*	*673*
	2p-prop	237	814		2p-prop	435	1544		*2p-prop*	*170*	*673*

Table 7: Comparison of parsing speeds (sent/s) on a single core CPU and GPU. The reported speeds are averaged over 5 runs. Times with a * are a reminder that the rel-PoS encoding additionally requires PoS tagging, whose time is not included in these speeds. As an example, the UDPipe tagging speed on the test set (in sent/s) is: Ancient Greek-567, Basque-881, Hungarian-755, Portuguese-210, Urdu-45, Afrikaans-465, Korean-921, Danish-671, Gothic-861, Lithuanian-1418, Japanese-767 and Galician-366.

Finally, we measured speeds for each of the encodings on various treebanks, run on a single core CPU[7] and GPU[8], which we breakdown in Table 7. We can observe that the speed is very similar between 1-planar and 2-planar encodings. This is because the bottleneck of the model is in the BiLSTMs, and computing the softmaxes comes at almost no cost despite the differences in the output vocabularies.

6 Conclusion

We have shown a new bracketing-based linearization of 2-planar trees compatible with parsing as sequence labeling. Our main goal was to introduce a bracketing encoding with the ability to perform unrestricted non-projective dependency parsing, which remained as an open challenge in sequence labeling parsing under the family of bracketing encodings. Together with the proposed plane assignment strategies and a BiLSTM-based network, our 2-planar bracket representations improve the performance over the existing bracketing-based encoding for parsing as sequence labeling, and also outperform the PoS-based encoding in the absence of PoS-tags as input parameters to the model. Thus, it can be a useful alternative where an encoding that depends on PoS tags is not desirable, e.g. domains with low-frequency or low-quality PoS tags, or to decrease even further the latency of sequence labeling parsers.

Finally, it is worth noting that we have proposed plane assignment strategies that minimize the use of the second plane. However, it is a possible avenue for future work to examine other strategies based on different criteria than the one presented in this paper.

Acknowledgements

This work has received funding from the European Research Council (ERC), which has funded this research under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from MINECO (ANSWER-ASAP, TIN2017-85160-C2-1-R), from Xunta de Galicia (ED431C 2020/11), and from Centro de Investigación de Galicia 'CITIC', funded by Xunta de Galicia and the European Union (European Regional Development Fund- Galicia 2014-2020 Program), by grant ED431G 2019/01. DV is supported by a 2020 Leonardo Grant for Researchers and Cultural Creators from the BBVA Foundation.

[7]For CPU experiments, we used a CPU core Intel Core i7-8700 CPU 3.2 GHz.

[8]For GPU experiments, we used an Nvidia TITAN Xp.

References

Mark Anderson and Carlos Gómez-Rodríguez. 2020. On the Frailty of Universal POS Tags for Neural UD Parsers. In *Proceedings of the 24th Conference on Computational Natural Language Learning*. To appear.

Liang-Fang Chao and Edwin Hsing-Mean Sha. 1992. Algorithms for min-cut linear arrangements of outerplanar graphs. In *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems*, volume 4, pages 1851–1854 vol.4.

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*, pages 207–217, Vancouver, Canada, August. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2018. A dynamic oracle for linear-time 2-planar dependency parsing. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 386–392, New Orleans, Louisiana, June. Association for Computational Linguistics.

Ramon Ferrer-i-Cancho, Carlos Gómez-Rodríguez, and Juan Luis Esteban. 2018. Are crossing dependencies really scarce? *Physica A: Statistical Mechanics and its Applications*, 493:311–329.

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*, pages 1492–1501, Uppsala, Sweden, July. Association for Computational Linguistics.

Carlos Gómez-Rodríguez and Joakim Nivre. 2013. Divisible transition systems and multiplanar dependency parsing. *Computational Linguistics*, 39(4):799–845.

Carlos Gómez-Rodríguez and David Vilares. 2018. Constituent parsing as sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1314–1324, Brussels, Belgium, October-November. Association for Computational Linguistics.

Carlos Gómez-Rodríguez. 2016. Restricted non-projectivity: Coverage vs. efficiency. *Computational Linguistics*, 42(4):809–817, December.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia, July. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2019. Tetra-tagging: Word-synchronous parsing with linear-time inference. *CoRR*, abs/1904.09745.

Marco Kuhlmann and Peter Jonsson. 2015. Parsing to noncrossing dependency graphs. *Transactions of the Association for Computational Linguistics*, 3:559–570.

Ophélie Lacroix. 2019. Dependency parsing as sequence labeling with head-based encoding and multi-task learning. In *Proceedings of the Fifth International Conference on Dependency Linguistics (Depling, SyntaxFest 2019)*, pages 136–143, Paris, France, August. Association for Computational Linguistics.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214, Santa Fe, New Mexico, USA, August. Association for Computational Linguistics.

Joakim Nivre et al. 2019. Universal dependencies 2.4. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2013. Finding optimal 1-endpoint-crossing trees. *Transactions of the Association for Computational Linguistics*, 1:13–24.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Aaron Smith, Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2018. An investigation of the interactions between pre-trained word embeddings, character models and POS tags in dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2711–2720, Brussels, Belgium, October-November. Association for Computational Linguistics.

Drahomíra Spoustová and Miroslav Spousta. 2010. Dependency parsing as a sequence labeling task. *The Prague Bulletin of Mathematical Linguistics*, 94(1):7–14.

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, August. Association for Computational Linguistics.

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez. 2019a. Sequence labeling parsing by learning across representations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5350–5357, Florence, Italy, July. Association for Computational Linguistics.

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez. 2019b. Viable dependency parsing as sequence labeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 717–723, Minneapolis, Minnesota, June. Association for Computational Linguistics.

David Vilares, Michalina Strzyz, Anders Søgaard, and Carlos Gómez-Rodríguez. 2020. Parsing as pretraining. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-20)*, New York, NY, USA.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2773–2781, Cambridge, MA, USA. MIT Press.

Jie Yang and Yue Zhang. 2018. Ncrf++: An open-source neural sequence labeling toolkit. In *Proceedings of ACL 2018, System Demonstrations*, pages 74–79.

Anssi Yli-Jyrä and Carlos Gómez-Rodríguez. 2017. Generic axiomatization of families of noncrossing graphs in dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1745–1755, Vancouver, Canada, July. Association for Computational Linguistics.

Anssi Mikael Yli-Jyrä. 2003. Multiplanarity – a model for dependency structures in treebanks. In Joakim Nivre and Erhard Hinrichs, editors, *TLT 2003. Proceedings of the Second Workshop on Treebanks and Linguistic Theories*, volume 9 of *Mathematical Modelling in Physics, Engineering and Cognitive Sciences*, pages 189–200, Växjö, Sweden. Växjö University Press.

A Treebank sizes

We provide some statistics about the chosen treebanks. In Table 8, we report the total number of sentences for each dataset split with their respective non-projectivity percentage.

Language	Train	Dev	Test	Language	Train	Dev	Test
Ancient Greek$_{\text{Perseus}}$	11476 (*62.77%*)	1137 (*74.41%*)	1306 (*64.40%*)	Afrikaans$_{\text{AfriBooms}}$	1315 (*22.21%*)	194 (*20.10%*)	425 (*23.76%*)
Basque$_{\text{BDT}}$	5396 (*33.52%*)	1798 (*33.48%*)	1799 (*31.80%*)	Korean$_{\text{Kaist}}$	23010 (*21.92%*)	2066 (*22.12%*)	2287 (*19.15%*)
Hungarian$_{\text{Szeged}}$	910 (*25.71%*)	441 (*33.56%*)	449 (*23.61%*)	Danish$_{\text{DDT}}$	4383 (*21.83%*)	564 (*21.81%*)	565 (*18.58%*)
Portuguese$_{\text{Bosque}}$	8328 (*23.60%*)	560 (*19.46%*)	477 (*22.85%*)	Gothic$_{\text{PROIEL}}$	3387 (*16.77%*)	985 (*19.09%*)	1029 (*18.76%*)
Urdu$_{\text{UDTB}}$	4043 (*23.00%*)	552 (*23.01%*)	535 (*18.88%*)	Lithuanian$_{\text{HSE}}$	153 (*16.34%*)	55 (*16.36%*)	55 (*21.82%*)

Table 8: Total number of sentences per each dataset split (% non-projective sentences).

B Label coverage

At test time, our model assigns a label for each task by choosing one from a finite set learned during training. As a result, it is expected that the model may not be able to predict some of the labels occurring in the test set. Table 9 reports the number of labels that have not been seen in the training and dev set and the total number of unique labels found in the test set. In addition, we include data about the percentage of occurrences of unseen labels with respect to the the occurrences of all labels in the test set.

Language	Encoding	Task 1 (1st plane)			Task 2 (2nd plain)			Task 3 (deprel)		
		Unseen	Total	% occ.	Unseen	Total	% occ.	Unseen	Total	% occ.
Ancient Greek$_{\text{Perseus}}$	rel-PoS	0 (*0%*)	75	0	–	–	–	0 (*0%*)	25	0
	1p-brackets	4 (*3.39%*)	118	0.02	–	–	–	0 (*0%*)	25	0
	2p-greedy	2 (*3.08%*)	65	0.01	1 (*5.26%*)	19	0.01	0 (*0%*)	25	0
	2p-prop	1 (*1.49%*)	67	0	0 (*0%*)	22	0	0 (*0%*)	25	0
Basque$_{\text{BDT}}$	rel-PoS	4 (*4.3%*)	93	0.02	–	–	–	0 (*0%*)	30	0
	1p-brackets	2 (*2.15%*)	93	0.01	–	–	–	0 (*0%*)	30	0
	2p-greedy	3 (*4.35%*)	69	0.02	2 (*10.53%*)	19	0.01	0 (*0%*)	30	0
	2p-prop	3 (*4.35%*)	69	0.02	2 (*10.0%*)	20	0.01	0 (*0%*)	30	0
Hunagrian$_{\text{Szeged}}$	rel-PoS	6 (*7.5%*)	80	0.06	–	–	–	0 (*0%*)	47	0
	1p-brackets	5 (*6.1%*)	82	0.05	–	–	–	0 (*0%*)	47	0
	2p-greedy	1 (*1.64%*)	61	0.01	1 (*8.33%*)	12	0.01	0 (*0%*)	47	0
	2p-prop	1 (*1.64%*)	61	0.01	1 (*6.67%*)	15	0.01	0 (*0%*)	47	0
Portuguese$_{\text{Bosque}}$	rel-PoS	2 (*2.11%*)	95	0.03	–	–	–	0 (*0%*)	38	0
	1p-brackets	0 (*0%*)	60	0	–	–	–	0 (*0%*)	38	0
	2p-greedy	0 (*0%*)	54	0	0 (*0%*)	14	0	0 (*0%*)	38	0
	2p-prop	0 (*0%*)	54	0	0 (*0%*)	14	0	0 (*0%*)	38	0
Urdu$_{\text{UDTB}}$	rel-PoS	8 (*7.69%*)	104	0.07	–	–	–	0 (*0%*)	24	0
	1p-brackets	4 (*6.06%*)	66	0.03	–	–	–	0 (*0%*)	24	0
	2p-greedy	3 (*5.36%*)	56	0.02	0 (*0%*)	12	0	0 (*0%*)	24	0
	2p-prop	3 (*5.36%*)	56	0.02	0 (*0%*)	14	0	0 (*0%*)	24	0
Afrikaans$_{\text{AfriBooms}}$	rel-PoS	2 (*2.82%*)	71	0.02	–	–	–	0 (*0%*)	26	0
	1p-brackets	4 (*6.06%*)	66	0.04	–	–	–	0 (*0%*)	26	0
	2p-greedy	1 (*1.92%*)	52	0.01	1 (*10.0%*)	10	0.01	0 (*0%*)	26	0
	2p-prop	1 (*1.89%*)	53	0.01	1 (*9.09%*)	11	0.01	0 (*0%*)	26	0
Korean$_{\text{Kaist}}$	rel-PoS	1 (*1.11%*)	90	0	–	–	–	1 (*3.23%*)	31	0
	1p-brackets	1 (*1.59%*)	63	0	–	–	–	1 (*3.23%*)	31	0
	2p-greedy	0 (*0%*)	56	0	0 (*0%*)	8	0	1 (*3.23%*)	31	0
	2p-prop	0 (*0%*)	56	0	0 (*0%*)	8	0	1 (*3.23%*)	31	0
Danish$_{\text{DDT}}$	rel-PoS	2 (*2.25%*)	89	0.02	–	–	–	0 (*0%*)	34	0
	1p-brackets	0 (*0%*)	72	0	–	–	–	0 (*0%*)	34	0
	2p-greedy	0 (*0%*)	63	0	0 (*0%*)	12	0	0 (*0%*)	34	0
	2p-prop	0 (*0%*)	63	0	0 (*0%*)	12	0	0 (*0%*)	34	0
Gothic$_{\text{PROIEL}}$	rel-PoS	3 (*4.11%*)	73	0.03	–	–	–	0 (*0%*)	31	0
	1p-brackets	2 (*2.78%*)	72	0.02	–	–	–	0 (*0%*)	31	0
	2p-greedy	1 (*1.79%*)	56	0.01	2 (*13.33%*)	15	0.02	0 (*0%*)	31	0
	2p-prop	1 (*1.79%*)	56	0.01	2 (*13.33%*)	15	0.02	0 (*0%*)	31	0
Lithuanian$_{\text{HSE}}$	rel-PoS	2 (*4.17%*)	48	0.19	–	–	–	1 (*3.12%*)	32	0.09
	1p-brackets	7 (*15.91%*)	44	0.66	–	–	–	1 (*3.12%*)	32	0.09
	2p-greedy	3 (*8.11%*)	37	0.38	1 (*16.67%*)	6	0.09	1 (*3.12%*)	32	0.09
	2p-prop	3 (*8.11%*)	37	0.38	1 (*16.67%*)	6	0.09	1 (*3.12%*)	32	0.09

Table 9: Label coverage in each task at test time.