Neurocomputing 524 (2023) 43–58

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Discontinuous grammar as a foreign language

Daniel Fernández-González *, Carlos Gómez-Rodríguez

Universidade da Coruña, CITIC, FASTPARSE Lab, LyS Group, Depto. de Ciencias de la Computación y Tecnologías de la Información, Campus de Elviña, s/n, A Coruña, 15071, Spain

ARTICLE INFO

Article history:
Received 10 June 2022
Revised 13 October 2022
Accepted 18 December 2022
Available online 22 December 2022

Keywords:
Natural language processing
Computational linguistics
Parsing
Discontinuous constituent parsing
Neural network
Deep learning
Sequence-to-sequence model

ABSTRACT

In order to achieve deep natural language understanding, syntactic constituent parsing is a vital step, highly demanded by many artificial intelligence systems to process both text and speech. One of the most recent proposals is the use of standard sequence-to-sequence models to perform constituent parsing as a machine translation task, instead of applying task-specific parsers. While they show a competitive performance, these text-to-parse transducers are still lagging behind classic techniques in terms of accuracy, coverage and speed. To close the gap, we here extend the framework of sequence-to-sequence models for constituent parsing, not only by providing a more powerful neural architecture for improving their performance, but also by enlarging their coverage to handle the most complex syntactic phenomena: discontinuous structures. To that end, we design several novel linearizations that can fully produce discontinuities and, for the first time, we test a sequence-to-sequence model on the main discontinuous benchmarks, obtaining competitive results on par with task-specific discontinuous constituent parsers and achieving state-of-the-art scores on the (discontinuous) English Penn Treebank.

1. Introduction

Syntactic parsing is a fundamental problem for Natural Language Processing in its pursuit towards deep understanding and computer-friendly representation of human linguistic input. Parsers are in charge of efficiently and accurately providing syntactic information so that it can be used for downstream artificial intelligence applications such as machine translation [82,78,83], opinion mining [81], relation and event extraction [48], question answering [6], summarization [4], sentiment classification [3], sentence classification [83] or semantic role labeling and named entity recognition [53], among others.

One of the widely-used formalisms for representing the grammatical structure of a given sentence in human languages is *constituent trees*. These structures decompose the sentence into *constituents* (also called *phrases*) and establish hierarchical relations between them and the sentence's words, resulting in a tree structure. While regular (or *continuous*) constituent trees (as the one depicted in Fig. 1(a)) are enough for representing a wide range of syntactic structures, it is necessary to use *discontinuous* constituent trees to fully describe all linguistic phenomena present in human languages [29]. Although producing the latter is considered an especially challenging problem in constituent parsing [11], they are necessary for adequately representing some syntactic phenomena that occur in almost the 20% of the sentences from the most widely-used syntactically-annotated corpus of English, the Penn Treebank [45] such as cross-serial dependencies, dislocations, long-distance extractions and some wh-movements [17], which require constituents with discontinuous spans and result in phrase structure trees with crossing branches. For instance, it can be seen in Fig. 1(b) that the span of the constituent *VP* (composed of the words *Allerdings*, *in*, *bestimmten*, *Vierteln*, *aus*, *Brunnen* and *verteilt*) is a discontinuous string, since it is interrupted by the words *wird* and *Wasser* from constituent *S*. Unlike the continuous constituent tree in Fig. 1(a)), this phenomenon generates a discontinuous phrase structure tree with two crossing branches.

In the last three decades, different techniques have been proposed for performing continuous and discontinuous constituent parsing. One of the most recent approaches, introduced by Vinyals, Kaiser, Koo, Petrov, Sutskever [74], consists in using generic *sequence-to-sequence* models to directly *translate* text into phrase structure trees, mimicking a machine translation task where these models had previously achieved considerable success [64]. This made it possible to perform continuous constituent parsing, which until that moment required specific parsing algorithms, using a task-independent model (without any further adaptation) that, given an input sequence of words, predicts a sequence of tokens that represent a linearization of a parse tree.

* Corresponding author.
E-mail addresses: d.fgonzalez@udc.es (D. Fernández-González), carlos.gomez@udc.es (C. Gómez-Rodríguez).
URLs: https://danifg.github.io (D. Fernández-González), https://www.grupolys.org/cgomezr/ (C. Gómez-Rodríguez).

https://doi.org/10.1016/j.neucom.2022.12.045
0925-2312/

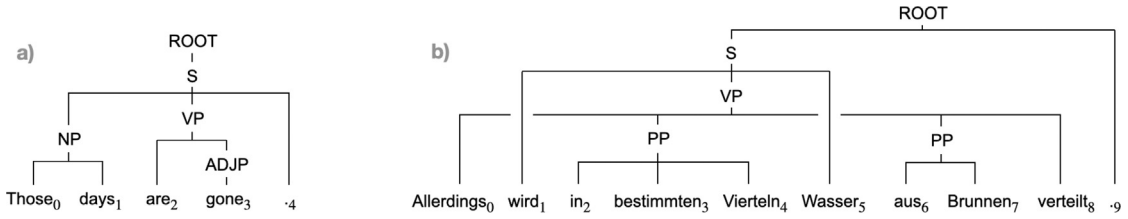


Fig. 1. Continuous (a) and discontinuous (b) constituent trees taken from PTB train and NEGRA dev splits, respectively.

However, while recent efforts on sequence-to-sequence constituent parsing provided promising results, this trend did not reach state-of-the-art results as it did on other natural language processing tasks such as machine translation [40,75] or speech recognition [46,76]. In fact, they lagged behind classic parsers based on explicit tree-structured algorithms and supported by a more extensive research background. The gap between task-specific constituent parsers and sequence-to-sequence models cannot be only quantified in terms of accuracy and speed [21], but also in coverage: to the best of our knowledge, the latter have not been applied to discontinuous constituent parsing to date. While we can find numerous studies where transition-based algorithms [9,8] and chart-based methods [60,11] were successfully designed for producing discontinuous structures, there has been no attempt to address the discontinuous constituent parsing task with a sequence-to-sequence neural architecture.

In order to improve the coverage of sequence-to-sequence constituent parsers, it is necessary to design novel linearization techniques capable of fully encoding discontinuous constituent trees into a sequence of tokens. Taking as starting point previous sequence-to-sequence approaches for continuous constituent parsing [21], we develop several linearization strategies inspired on how transition-based parsers handle discontinuities. Then, we implement a powerful neural architecture based on the cutting-edge sequence-to-sequence model proposed by Fernandez Astudillo, Ballesteros, Naseem, Blodgett and Florian [18] for graph parsing. The resulting system is not only the first sequence-to-sequence model for discontinuous constituent parsing, but also an accurate approach that delivers a high performance on the main benchmarks.

Therefore, our main contributions are:

- *The implementation of a novel sequence-to-sequence constituent parser*, building on the work developed by Fernández-González and Gómez-Rodríguez [21] and Fernandez Astudillo et al. [18]. While the former defines linearizations for continuous parsing that outperform those previously proposed, the latter introduces a deterministic attention technique over a powerful Transformer sequence-to-sequence architecture [50] that significantly increases prediction accuracy. The resulting system outperforms all existing sequence-to-sequence models and is on par with state-of-the-art task-specific constituent parsers.
- *Novel linearizations to model discontinuous structures.* We design different strategies to linearize discontinuous constituent trees and test them with the proposed neural architecture, becoming the first sequence-to-sequence model that, not only can produce discontinuous representations, but achieves a competitive performance on the main discontinuous benchmarks: the discontinuous version of the English Penn Treebank (DPTB) [17], and the German NEGRA [58] and TIGER [5] treebanks.

The remainder of this article is organized as follows: Section 2 firstly presents previous research work that contributed to model and improve constituent parsing as a sequence-to-sequence problem and, secondly, introduces task-specific constituent parsers proposed so far for dealing with discontinuous trees. Section 3 explains how the parsing problem can be cast as a sequence prediction task and other relevant concepts. In Section 4, we detail our approach: we present the novel linearizations developed for generating discontinuities and describe the proposed neural architecture. In Section 5, we extensively evaluate our sequence-to-sequence model on continuous and discontinuous treebanks and include a thorough analysis of their performance. Lastly, Section 6 contains a final discussion.

2. Related work

2.1. Constituent parsing as a sequence-to-sequence task

Since Vinyals, Kaiser, Koo, Petrov, Sutskever [74] presented the first sequence-to-sequence model for constituent parsing, several variants have been proposed seeking improvements in its performance. These efforts have mainly focused on improving either the proposed attentional sequence-to-sequence neural network [2], or the linearization technique necessary for casting constituent parsing as a sequence prediction problem, or both.

With respect to the original architecture by [2] based on recurrent neural networks, several attempts have modified the original design by introducing *deterministic attention* strategies [34,42,37,39,21]. These aim to improve the probabilistic attention mechanism (implemented in sequence-to-sequence models to select relevant context) for two purposes: (1) to obtain accuracy gains by deterministically focusing on those input tokens that are crucial for the parsing task and (2) to speed up the decoding process by avoiding the need to go over the whole input sequence when the attentional probabilities are computed. Lastly, Vaswani et al. [68] propose a novel sequence-to-sequence architecture based on Transformers, which manages to improve both accuracy and speed.

Regarding the linearization strategy, Vinyals, Kaiser, Koo, Petrov, Sutskever [74] opted for encoding the parse tree from top to bottom by grouping constituents and words by means of brackets. While the overwhelming majority of subsequent work assumed this linearization, there are some exceptions that designed alternative representations. In particular, Ma et al. [42] and Liu and Zhang [37] were the first in designing a linearization based on transition-based actions as sequence tokens. The former used actions from the bottom-up transition-based algorithm by Cross and Huang [13] to encode constituent trees and proved that it underperforms the original top-down linearization when they are tested under the same conditions; and the latter based the linearization on the top-down transition system of Dyer, Kuncoro, Ballesteros and Smith [16] and showed that (combined with a specific deterministic attention strategy) this yields some accuracy gains. However, a follow-up study by Fernández-González and Gómez-Rodríguez [21] contradicted this last conclusion: they demonstrated that the original bracketed encoding can be represented as sequences of actions from Dyer et al. [16]'s transition system and, under the same neural network, this representation outperforms the

top-down linearization defined by Liu and Zhang [37]. Additionally, Fernández-González and Gómez-Rodríguez [21] proposed a novel linearization method based on the in-order transition system [38], notably outperforming all existing sequence-to-sequence constituent parsers.

Finally, it is worth mentioning that there also exists a recent trend of casting constituent parsing as sequence labeling [30,71]. While these might be considered a kind of sequence-to-sequence methods (where the input and target sequences are constrained to the same length, as each input word is assigned exactly one label as output), they are not usually framed within sequence-to-sequence constituent parsers since the neural architecture used for sequence labeling is simpler than the setup designed by [2], obtaining worse accuracy due to the lack of attention mechanisms[1] and a larger label dictionary, but being significantly faster due to its simplicity.

2.2. Discontinuous constituent parsing

Discontinuous structures were initially derived by complex and computationally-expensive *chart* parsers based on *Linear Context-Free Rewriting Systems* (LCFRS) [70] or *Multiple Context Free Grammmars* (MCFGs) [57], which use the CYK algorithm for exact decoding [17,12,28]. However, the computational complexity of these approaches makes them impractical for long sentences, to the point that all the mentioned parsers evaluate with a cap on sentence length (typically, 40) due to the infeasibility of processing longer sentences. For this reason, we can also find several variations of these original grammar-based parsers that attempt to reduce their computational cost and make then runnable on long sentences. For instance, [60,11] speed up decoding by not explicitly defining a set of rules and using a span-based scoring algorithm [61]. Additionally, Ruprecht and Mörbitz [52] present the first suppertagging-based parser for LCFRS that notably reduces parsing time.

Alternatively, bottom-up *transition-based* (or *shift-reduce*) parsers, originally restricted to continuous structures [54,85], were extended to generate discontinuities. In particular, some of these parsers incorporate new actions for changing the original token order (allowing to treat discontinuous structures as continuous ones) [43,44,59] or directly processing non-adjacent words [9]; and others opted for designing novel data structures to facilitate building constituents on non-local items [8].

Finally, several efforts focused on reducing discontinuous constituent parsing into a simpler task. For instance, discontinuous phase-structure trees can be encoded as non-projective dependency trees and, then, produced by a dependency parser [32,23]; or represented as a sequence of tags and derived by any tagger [71]. Recently, Fernández-González and Gómez-Rodríguez [22] also reduced discontinuous into continuous constituent parsing by just accurately reordering input words.

Our work is framed within this last category and the closest approach is the sequence labeling strategy introduced by Vilares and Gómez-Rodríguez [71]. However, as mentioned above, they do not use a sequence-to-sequence model to perform the tagging and, since the target sequence shares the same length as the input sentence, they just employ a regular tagger. This constraint requires a complex encoding scheme that results in a remarkably large output dictionary. In contrast, a sequence-to-sequence approach can deal with longer target sequences and, therefore, maintain a smaller dictionary size, outperforming a regular sequence tagger by a wide margin.

[1] Please note that recent sequence labeling approaches [72,71] also include variants with attention mechanisms that, while notably increasing their accuracy, significantly penalize their speed.

3. Preliminaries

3.1. Continuous linearizations

In order to properly define the sequence prediction problem of *translating* an input sentence of n words $\mathbf{w} = w_1, \ldots, w_n$ into a constituent tree C, the latter needs to be encoded (or linearized) as a sequence of tokens $\mathbf{y} = y_1, \ldots, y_m$, with $n < m$ in the particular case of sequence-to-sequence constituent parsing.[2] This conversion must be invertible so that the original tree can be recovered from the sequence of tokens. It is also worth mentioning that, unlike in machine translation, this is an unbalanced sequence prediction task, since target sequences are significantly longer than inputs.

A constituent tree C is composed of words $w_1, \ldots w_n$ as leaf nodes and, above them, a number of hierarchically-organized constituents. Each constituent (or phrase) can be represented as a tuple $(X, \mathscr{y})$ where X is the non-terminal label and $\mathscr{y}$ is the set of word positions that constitute its yield or span. Moreover, we define a constituent tree C as *continuous* if the span of every constituent of C is a continuous substring (or, equivalently, a sequence of consecutive word positions). The phase structure tree in Fig. 1(a) is an example of a continuous structure. Conversely, if there are at least one constituent of C with a yield composed of non-successive word positions, it will be classified as *discontinuous*. The constituent tree in Fig. 1(b) is discontinuous since the span of the constituent (VP, {0, 2, 3, 4, 6, 7, 8}) is interrupted by words $wird_1$ and $Wasser_5$ from constituent (S, {0, 1, 2, 3, 4, 5, 6, 7, 8}), resulting in a yield with a sequence of non-consecutive word positions.

Initially, Vinyals, Kaiser, Koo, Petrov, Sutskever [74] introduced a simple top-down bracketed encoding for linearizing continuous constituent trees. This consists of defining phrase spans with opening and closing brackets and parametrizing these brackets with non-terminal labels to fully encode constituent information. Additionally, words were normalized by replacing them with a tag `XX`. An example of this tree linearization is depicted in Fig. 2(a).

As an alternative, we can find linearizations that use transition-based actions as output tokens (instead of brackets and `XX`-tags) and represent a syntactic structure as a sequence of transitions [42,37,21]. There exist different shift-reduce transition systems (mainly focused on transition-based parsing) that can also be applied to encode a constituent tree in different manners: from top to bottom, based on an in-order traversal or following a bottom-up strategy. The most successful shift-reduce linearizations are mainly based on two transition systems: the *top-down* transition system [16] and the *in-order* shift-reduce algorithm [38]. Additionally, in this research work, we will also apply and test a linearization based on the bottom-up transition system defined by [24]. Unlike classic bottom-up approaches [54,85], this transition system does not require to previously binarize the constituent tree and, although it was not used to date as a linearization by any previous work and it was already shown that shift-reduce bottom-up linearizations are not the best option for sequence-to-sequence constituent parsing [42], we will also include it for comparison purposes.

Transition systems are state machines that traverse a sequence of configurations by means of a set of transitions, until they reach a terminal configuration from which the output tree can be recovered. Thus, to understand how these specific transition systems work, we need to formally define their *parser configurations* and available *transitions*. The top-down algorithm has parser configurations of the form $c = \langle \Sigma, B \rangle$, where B is the *buffer* that initially contains all the input words and Σ is a *stack*, which is empty in the

[2] As mentioned before, [30] developed an encoding that, thanks to a large dictionary, is able to linearize a sentence of n words into a sequence of n tokens.

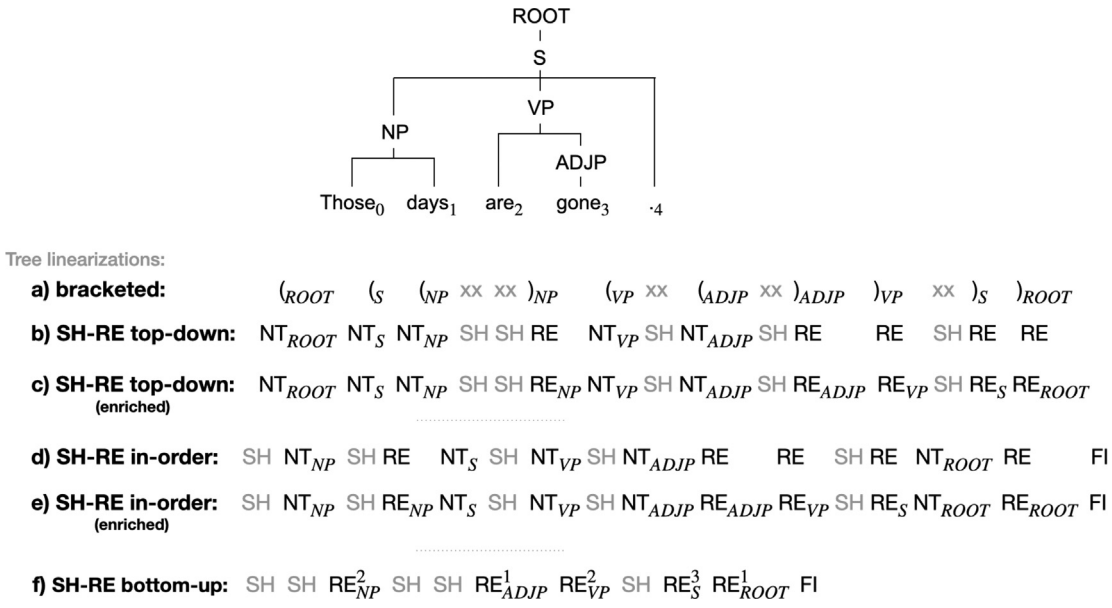


Fig. 2. Bracketed and shift-reduce (SH-RE) tree linearizations for encoding the continuous constituent tree in Fig. 1(a). SH = SHIFT, NT_X = NON-TERMINAL-X, RE = REDUCE, RE_X = REDUCE-X, RE^k_X = REDUCE#K-X and FI = FINISH.

initial configuration and will store constituents and unprocessed words during the parsing process. Additionally, while top-down terminal configurations are those with an empty buffer and a single element on the stack, the in-order and bottom-up approaches need a third component in their parser configurations for marking whether a parser configuration is terminal or not. This is implemented by a boolean variable f (which will be *false* in the initial configuration) and their configurations will be of the form $c = \langle \Sigma, B, f \rangle$.

Knowing the configurations, we can now define the transitions for each parser. Firstly, the top-down algorithm provides three transitions (defined in Fig. 3) that modify the stack and the buffer to generate a valid constituent tree. Concretely:

- a SHIFT transition pushes words from the buffer to the stack,
- a NON-TERMINAL-X action adds a non-terminal node X on the stack,
- and a REDUCE transition pops items from the stack until a non-terminal node is reached and groups all these items as a new constituent on the top of the stack.

Secondly, the in-order algorithm (described in Fig. 4) uses practically the same actions as the top-down variant, but they are applied in a different order and some of them have a different behavior:

- a SHIFT transition is used to move words from the buffer to the stack,
- a NON-TERMINAL-X transition is applied to push a non-terminal node X into the stack, but, unlike in the top-down transition system, it should only be used if the first child node of the future constituent is on top of the stack,
- a REDUCE transition is available to pop all items from the stack until the first non-terminal node is reached, which is also popped together with the preceding item to build a new constituent on top of the stack,
- and, finally, a FINISH transition is used to terminate the parsing process.

Lastly, the non-binary bottom-up transition system provides the following actions (described in Fig. 5):

Discontinuous Grammar as a Foreign Language

$$\text{SHIFT:} \quad \langle \Sigma, w_i|B \rangle \Rightarrow \langle \Sigma|w_i, B \rangle$$
$$\text{NT-X:} \quad \langle \Sigma, B \rangle \Rightarrow \langle \Sigma|X, B \rangle$$
$$\text{REDUCE:} \quad \langle \Sigma|X|s_k|\dots|s_0, B \rangle \Rightarrow \langle \Sigma|X_{s_k \dots s_0}, B \rangle$$

Fig. 3. Transitions of the top-down transition system (NT-X = NON-TERMINAL-X).

$$\text{SHIFT:} \quad \langle \Sigma, w_i|B, false \rangle \Rightarrow \langle \Sigma|w_i, B, false \rangle$$
$$\text{NT-X:} \quad \langle \Sigma, B, false \rangle \Rightarrow \langle \Sigma|s_0|X, B, false \rangle$$
$$\text{REDUCE:} \quad \langle \Sigma|s_k|X|s_{k-1}|\dots|s_0, B, false \rangle \Rightarrow \langle \Sigma|X_{s_k \dots s_0}, B, false \rangle$$
$$\text{FINISH:} \quad \langle \Sigma, B, false \rangle \Rightarrow \langle \Sigma, B, true \rangle$$

Fig. 4. Transitions of the in-order transition system (NT-X = NON-TERMINAL-X).

- a SHIFT transition that pushes words from the buffer to the stack,
- a REDUCE#K-X action parameterized with an integer k and the non-terminal label X that pops k items from the stack and builds a new constituent with all of them on the top of the stack,
- and, finally, a FINISH transition that marks the end of the process.

Given the described transition systems, a constituent tree C that represents the syntactic information of the input sentence $\mathbf{w}$ can be encoded into a sequence of shift-reduce actions $\mathbf{y} = y_1, \dots, y_m$ by following a top-down, in-order or bottom-up transition system, as exemplified by sequences (b), (d) and (f) in Fig. 2, respectively. Depending on the shift-reduce strategy used, the resulting target sequence may have a different length, since each transition system utilizes a different number of transitions to build the same phrase structure.

Additionally, Fernández-González and Gómez-Rodríguez [21] noticed that, if REDUCE transitions are parameterized with the non-terminal label (REDUCE-X), the bracketed [74] and shift-reduce top-down [37] linearizations are equivalent and this *enriched* top-down variant improves prediction accuracy (see an example in Fig. 2(c)). Applying this idea to the in-order strategy, they also pro-

$$\text{SHIFT:} \quad \langle \Sigma, w_i | B, false \rangle \Rightarrow \langle \Sigma | w_i, B, false \rangle$$
$$\text{REDUCE\#K-X:} \quad \langle \Sigma | s_{k-1} | \dots | s_0, B, false \rangle \Rightarrow \langle \Sigma | X_{s_{k-1} \dots s_0}, B, false \rangle$$
$$\text{FINISH:} \quad \langle \Sigma, B, false \rangle \Rightarrow \langle \Sigma, B, true \rangle$$

Fig. 5. Transitions of the non-binary bottom-up transition system.

posed an enriched variant and proved that, while enlarging the output dictionary size, it outperformed all existing tree linearizations tested on the framework designed by Liu and Zhang [37] for sequence-to-sequence constituent parsing (Fig. 2(e) exemplifies this tree linearization).

In spite of using the same transition systems, the main reason why sequence-to-sequence models are so far not obtaining comparable results to task-specific transition-based parsers is the lack of structural constraints explicitly provided by the stack and the buffer, which can help the algorithm to handle and hierarchically organize phrases and words during parsing. Sequence-to-sequence constituent parsers are agnostic to any structural information during decoding and exclusively *translate* an input sentence into a sequence of tokens, which (after a post-processing step) will be converted into a tree structure.

3.2. Sequence-to-sequence neural architecture

[74] propose to address constituent parsing using the attentional sequence-to-sequence neural model defined by [2]. This work introduces an attention mechanism to the original neural architecture defined by [64] for solving sequence-to-sequence problems and applies it to machine translation. The attention mechanism allows the model to focus, at each time step, on the most relevant information from the input in order to accurately predict the output tokens. This is especially important for handling long sequences, since the prediction accuracy deteriorates as the length of the input sequence increases [7].

More in detail, [2] define an *encoder-decoder* neural architecture where the *encoder* reads tokens from the input sequence (words in constituent parsing and machine translation) represented as a sequence of vectors $\mathbf{x} = x_1, \dots, x_n$ (where each x_i can be obtained from pre-trained word embeddings) and encodes them as a sequence of *encoder hidden states* $\mathbf{h} = h_1, \dots, h_n$. The encoder was initially implemented as a recurrent neural network (RNN), in particular a bidirectional LSTM (BiLSTM) [31] that processes the input in both directions.

Then, at each time step t, a *decoder* is used for predicting the next output token y_t from the current decoder hidden state s_t, which is generated by a function f fed with the context vector c_t and the previous decoder hidden state s_{t-1}:

$$s_t = f(s_{t-1}, c_t)$$

Function f is usually implemented as a unidirectional LSTM [33] and the context vector c_t is computed at time step t as follows:

$$c_t = \sum_{i=1}^{n} \alpha_{ti} h_i, \quad \alpha_{ti} = \frac{exp(\beta_{ti})}{\sum_{k=1}^{n} exp(\beta_{tk})}, \quad \beta_{ti} = g(s_{t-1}, h_i)$$

where g is a scoring function (implemented as a feed-forward neural network jointly trained with the other components) that computes scores between each input token x_i (encoded as h_i) and the previous decoder hidden state s_{t-1}. Then, a probability distribution over the whole input is computed using a softmax function, reflecting in c_t the weight of each input token in the prediction of the current output token y_t.

Especially in constituent parsing, it has been observed that by using *deterministic* attention mechanisms the performance of the original sequence-to-sequence model increases [34,42,39,21]. The most successful variant was developed by Liu and Zhang [37] and is based on the top-down shift-reduce linearization. They propose to use two separate attention models and, therefore, calculate two different context vectors instead of just c_t. For this purpose, the model splits the input into two variable-length segments obtained by dividing the input sequence by index p, which in the initial decoding step points to the first input token, and then is incremented to point to the next word whenever a SHIFT transition is predicted. Then, vector c_t^l is computed over the left segment $w_1, \dots, w_p$ and, c_t^r, over the right segment $w_{p+1}, \dots, w_n$ as follows:

$$c_t^l = \sum_{i=1}^{p} \alpha_{ti} h_i, \quad c_t^r = \sum_{i=p+1}^{n} \alpha_{ti} h_i$$

This approach led to notable accuracy gains [21]: on the one hand, it intuitively models a stack (c_t^l) and a buffer (c_t^r) over the input that are modified when a SHIFT transition is applied. On the other hand, it provides a deterministic alignment between input words and SHIFT tokens that is crucial for choosing the most relevant context information at each time step.

In the last few years, these RNN sequence-to-sequence models were substituted by a Transformer architecture thanks to the remarkable performance presented by Vaswani et al. [68]. The authors not only prove that Transformers notably outperform RNNs on machine translation, but they also apply this new architecture on constituent parsing, obtaining promising results. Both the encoder and decoder are implemented with Transformers, which provide a multi-head self-attention mechanism that is more powerful than the attention techniques developed in RNN approaches.

More in detail, this novel architecture starts by injecting a positional encoding to each input word representation x_i necessary for handling sequences, as, unlike RNNs, Transformers contain no recurrence and do not have any built-in notion of sequential order. After that, the encoder implemented by six Transformers generates the sequence of encoder hidden states $\mathbf{h} = h_1, \dots, h_n$for the input sequence. To do this, each Transformer implements a *multi-head self-attention layer* that is composed of several parallel *attention heads*, which will score the relevance of a specific word with respect to the other words in the sentence. In particular, each head computes attention vectors z_i for each input word representation x_i as a weighted sum of linearly transformed input vectors:

$$z_i = \sum_{j=1}^{n} \alpha_{ij} (x_j W^V), \quad \alpha_{ij} = \frac{exp(\beta_{ij})}{\sum_{k=1}^{n} exp(\beta_{ik})}, \quad \beta_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d}}$$

where W^Q, W^K and W^V are parameter matrices unique per attention head, d is the dimension of the resulting vector z_i and β_{ij} is computed by a compatibility function (implemented as an efficient scaled dot product) that compares two input words. The multi-head self-attention layer is followed by a feed-forward network for finally generating encoder hidden states $\mathbf{h}$. Unlike RNNs, this process can be easily parallelized, speeding up Transformers performance.

Regarding the decoder, it is also implemented by six Transformers, but each one has an additional component. Apart from a *masked multi-head self-attention mechanism* that works practically the same as the encoder (which is used to encode previously-generated output tokens $\mathbf{y} = y_1, \dots, y_{t-1}$ into a sequence $\mathbf{q} = q_0, \dots, q_{t-1}$ at time step t), it implements a posterior *encoder-decoder cross-attention layer* that computes the compatibility between each target token with each input word. More in detail, this cross-attention module is also composed of several attention

heads that, given the sequences of encoder and decoder hidden states **h** and **q**, generate at each time step t an attention vector z_t as follows:

$$z_t = \sum_{i=1}^{n} \alpha_{ti}(h_i W_d^V), \quad \alpha_{ti} = \frac{exp(\beta_{ti})}{\sum_{k=1}^{n} exp(\beta_{tk})}, \quad \beta_{ti} = \frac{(q_{t-1} W_d^Q)(h_i W_d^K)^T}{\sqrt{d}}$$

where W_d^Q, W_d^K and W_d^V are parameter matrices, d is the dimension of the resulting vector z_t and β_{ti} computes the interaction between the last predicted token encoding (q_{t-1}, which represents target token history) with each word from the input sequence (represented by its encoder hidden state h_i). The attention vectors z_t computed by each head will be combined and used by posterior linear and softmax layers to finally generate the output token y_t. Please see in Fig. 6 a sketch of the described neural architecture.

Recently, Fernandez Astudillo et al. [18] implemented the idea introduced by Liu and Zhang [37] into this Transformer sequence-to-sequence architecture for dependency and AMR (Abstract Meaning Representation) parsing. To that end, they followed the current trend of modifying Vaswani et al. [68]'s architecture by implementing dedicated attention heads to focus on one or several tokens from the input and, thus, encode local relevant information [63,77]. In particular, instead of using an index p to delimitate the stack and buffer, they propose to specialize two of the attention heads (from the Transformer decoder's cross-attention layer): one for just attending input words that should be into the stack according to the shift-reduce tokens applied so far, and the other for exclusively considering words that are left in the buffer. With these two dedicated heads for the decoding, they manage to substantially improve over the original setup by Vaswani et al. [68] on these two graph parsing tasks. In this article, we accordingly modify this recent sequence-to-sequence model for handling constituent parsing and accurately produce continuous and discontinuous phrase structure trees.

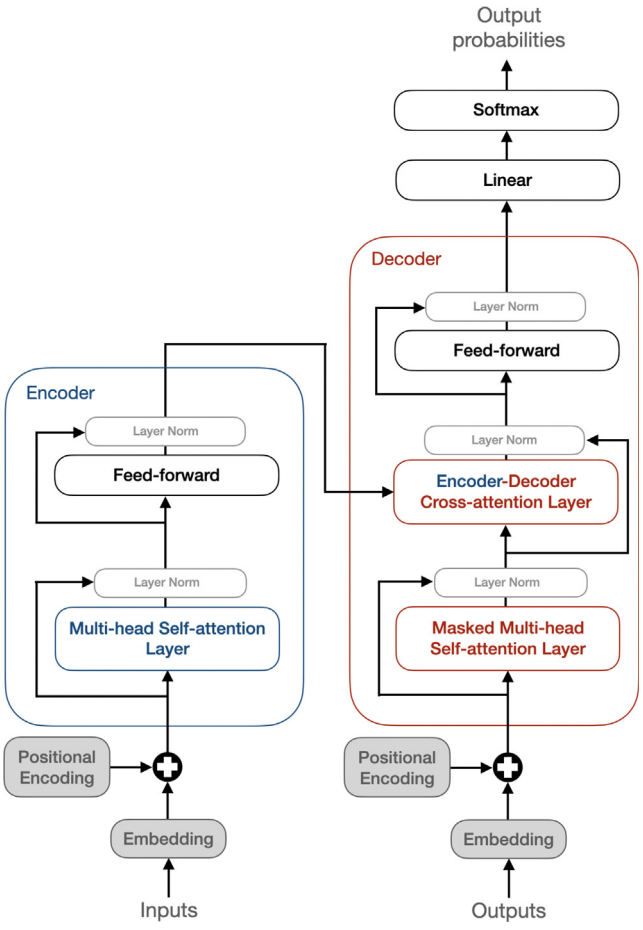


Fig. 6. Neural architecture for the Transformer sequence-to-sequence model introduced by Vaswani et al. [68]. Please note that *Layer Norm* stands for the layer normalization technique introduced by [1].

4. Discontinuous sequence-to-sequence parsing

4.1. Discontinuous linearizations

Since all transition systems (and, therefore, the resulting tree linearizations) explained in Section 3.1 are restricted to continuous structures, we need to extend them to handle discontinuities. For that purpose, novel transition systems must be defined so that they can be used for linearizing discontinuous structures and extending the coverage of sequence-to-sequence models (currently constrained to continuous structures) to any kind of constituent trees.

Please note that, while it can be argued that there already exist transition systems that can produce discontinuous constituent trees [43,44,59,9], all of these follow a binary bottom-up strategy that is not the most adequate for sequence-to-sequence constituent parsing as shown by Ma et al. [42]. However, for completeness of comparison, we also include in our experiments a discontinuous extension of the non-binary bottom-up transition system by [24], which was shown to be superior to the binary variants in continuous transition-based parsing.

Firstly, we define new discontinuous transition systems by adding to the top-down [16], in-order [38] and the mentioned non-binary bottom-up algorithms a SWAP transition. This action (initially proposed for transition-based constituent parsing by [69])[3] is used to reorder the original sentence by moving the second word on top of the stack back to the buffer (as detailed in Fig. 7(a)). Thanks to this online reordering of the input during parsing, any discontinuous structure can be created with the available continuous transitions. This relies on the fact that any discontinuous constituent tree can be transformed into a continuous variant by just changing the order of tokens. For instance, the discontinuous tree in Fig. 1(b) can be converted into a continuous one by moving the word $wird_1$ before the word $Allerdings_0$, and the word $Wasser_5$ after the word $verteilt_8$. We show in Fig. 8 how the in-order transition system extended with the SWAP transition is able to handle the discontinuities of the tree in Fig. 1(b) by means of buffer and stack structures.

The main drawback of adding the SWAP action to continuous transition systems is that it tends to produce considerably long transition sequences and, when used as a linearization technique for discontinuous constituent trees, it will generate such long target sequences that it might harm accuracy of prediction. To address this, we also develop two variants based on two transitions already studied in shift-reduce parsing:

- a SWAP#K transition [43] (detailed in Fig. 7(b)), which is equivalent to applying k SWAPs in a row, with SWAP#1 being equivalent to applying a single SWAP action.
- a SHIFT#K action [44] (described in Fig. 7(c)), which moves the kth word in the buffer to the stack, with SHIFT#0 being equivalent to the regular SHIFT action. In Fig. 9, we present an example of how the SHIFT#K transition works on the in-order transition system in a shift-reduce parsing process with a buffer and a stack.

While adding the SWAP#K transition will have a minor impact in shortening the resulting sequences, extending the original in-order transition system with the SHIFT#K action will lead to a transition

[3] The concept of a SWAP transition was initially introduced for non-projective dependency parsing by [49] and then adapted by [69] to constituent parsing.

a) SWAP: $\langle \Sigma | w_1 | w_0, B, false \rangle \Rightarrow \langle \Sigma | w_0, w_1 | B, false \rangle$

b) SWAP#K: $\langle \Sigma | w_k | \dots | w_1 | w_0, B, false \rangle \Rightarrow \langle \Sigma | w_0, w_k | \dots | w_1 | B, false \rangle$

c) SHIFT#K: $\langle \Sigma, w_0 | \dots | w_{n-1}, false \rangle \Rightarrow \langle \Sigma | w_k, w_0 | \dots | w_{k-1} | w_{k+1} | \dots | w_{n-1}, false \rangle$

Fig. 7. Available transitions for reordering words from the input sentence.

Transition	Stack	Buffer
	[ ]	[Allerdings_0, wird_1, ... , .]
SHIFT	[Allerdings_0]	[wird_1, in_2, ... , .]
NON-TERMINAL-VP	[Allerdings_0, VP]	[wird_1, in_2, ... , .]
SHIFT	[Allerdings_0, VP, wird_1]	[in_2, bestimmten_3, ... , .]
SHIFT	[Allerdings_0, VP, wird_1, in_2]	[bestimmten_3, Vierteln_4, ... , .]
SWAP	[Allerdings_0, VP, in_2]	[wird_1, bestimmten_3, Vierteln_4, ... , .]
NON-TERMINAL-PP	[Allerdings_0, VP, in_2, PP]	[wird_1, bestimmten_3, Vierteln_4, ... , .]
SHIFT	[Allerdings_0, VP, in_2, PP, wird_1]	[bestimmten_3, Vierteln_4, ... , .]
SHIFT	[Allerdings_0, VP, in_2, PP, wird_1, bestimmten_3]	[Vierteln_4, Wasser_5, ... , .]
SWAP	[Allerdings_0, VP, in_2, PP, bestimmten_3]	[wird_1, Vierteln_4, Wasser_5, ... , .]
SHIFT	[Allerdings_0, VP, in_2, PP, bestimmten_3, wird_1]	[Vierteln_4, Wasser_5, ... , .]
SHIFT	[Allerdings_0, VP, in_2, PP, bestimmten_3, wird_1, Vierteln_4]	[Wasser_5, aus_6, ... , .]
SWAP	[Allerdings_0, VP, in_2, PP, bestimmten_3, Vierteln_4]	[wird_1, Wasser_5, ... , .]
REDUCE	[Allerdings_0, VP, $\text{PP}_{(in_2\ bestimmten_3\ Vierteln_4)}$]	[wird_1, Wasser_5, ... , .]
...	...	...

Fig. 8. Transition sequence for partially producing the discontinuous tree in Fig. 1(b) using the in-order + SWAP transition system in transition-based constituent parsing.

sequence with the same number of items as if the encoded tree were continuous.

All these novel discontinuous transition systems can be used as linearization techniques for casting discontinuous constituent trees as sequences of tokens (which then can be used for training a sequence-to-sequence model). For instance, in Figs. 10(a), (b) and (c), we can see the resulting linearizations of a discontinuous tree by the top-down, in-order and bottom-up algorithms extended with the SWAP transition, respectively. Additionally, Figs. 10(d) and (e) show the discontinuous variant of the in-order linearization with the SWAP#K and SHIFT#K transitions, respectively. Although an analysis in this regard is included in Section 5.3, it can be noticed in this example how the presence of SWAP tokens notably lengthens tree linearizations in comparison to the variant with the SHIFT#K transition, which has the same number of tokens as linearizing a continuous tree. While SWAP#K and SHIFT#K transitions can also be applied to the top-down and bottom-up linearization methods, we only test these variants on the best-performing strategy, which is the one based on the in-order tree linearization (as will be seen in Section 5).

We opted for not using the enriched variants proposed by Fernández-González and Gómez-Rodríguez [21] (which parametrize REDUCE actions in the top-down and in-order transition systems) since, as we will see in Section 5, the accuracy of the regular versions is on par with these enhanced variants on the proposed neural architecture (while requiring a smaller output dictionary).

Finally, please note that the proposed transition systems are exclusively used for linearizing constituent trees and defining the target sequence of tokens for our sequence-to-sequence model. This contrasts with task-specific transition-based parsers, which leverage data structures (two or more stacks) to explicitly build partial constituent trees at each step of the parsing process. In addition, it is also worth mentioning that the presented discontinuous transition-based algorithms were never proposed before as far as we know and, while they might certainly achieve a good performance under a traditional shift-reduce parsing implementation, this is out of the scope of this research work and we will exclusively apply them as tree linearization strategies.

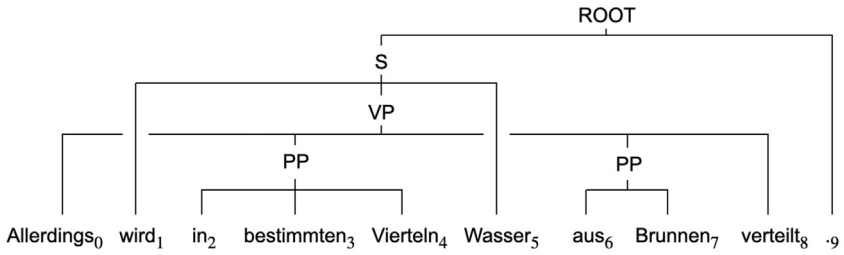
4.2. Neural architecture

Based on the approach proposed by [18] for dependency and AMR parsing, we present a Transformer sequence-to-sequence architecture for unrestricted constituent parsing. Unlike the original work by Vaswani et al. [68] (where all *heads* of each Transformer decoder's cross-attention layer attend to the whole input), one specialized head is exclusively applied over input words that should be into the stack according to the current time step t (following a shift-reduce parsing process), and another dedicated head will focus only on those words that are still left in the buffer in t. In that way, while not explicitly using data structures to process the input sentence (as done by regular transition-based parsers), some structural information is *deterministically* induced to the sequence-to-sequence model, with the purpose of substantially increasing parsing performance as shown on other syntactic formalisms by Fernandez Astudillo et al. [18].

The implementation proposed by Fernandez Astudillo et al. [18] was exclusively focused on dependency and AMR parsing, where the buffer and the stack of a purely transition-based dependency parser can only contain words that belong to partial graph structures. In contrast, transition-based constituent parsers process tree structures and, in addition to nodes corresponding to words, they also have to push non-terminal nodes into the stack. These are nec-

Transition	Stack	Buffer
	[ ]	[Allerdings_0, wird_1, ... , .]
SHIFT#0	[Allerdings_0]	[wird_1, in_2, ... , .]
NON-TERMINAL-VP	[Allerdings_0, VP]	[wird_1, in_2, ... , .]
SHIFT#1	[Allerdings_0, VP, in_2]	[wird_1, bestimmten_3, Vierteln_4, ... , .]
NON-TERMINAL-PP	[Allerdings_0, VP, in_2, PP]	[wird_1, bestimmten_3, Vierteln_4, ... , .]
SHIFT#1	[Allerdings_0, VP, in_2, PP, bestimmten_3]	[wird_1, Vierteln_4, Wasser_5, ... , .]
SHIFT#1	[Allerdings_0, VP, in_2, PP, bestimmten_3, Vierteln_4]	[wird_1, Wasser_5, ... , .]
REDUCE	[Allerdings_0, VP, $\text{PP}_{(in_2\ bestimmten_3\ Vierteln_4)}$]	[wird_1, Wasser_5, ... , .]
...	...	...

Fig. 9. Transition sequence for partially building the discontinuous tree in Fig. 1(b) using the in-order + SHIFT#K transition system in transition-based constituent parsing.



Tree linearizations:

a) top-down + SWAP: NT_{ROOT} NT_{S} NT_{VP} SH NT_{PP} SH SH SW SH SH SW SH SH SW RE NT_{PP} SH SH SH SW SW SH SH SH SW SW RE SH SH SH SW SW RE SH SH RE SH RE

b) in-order + SWAP: SH NT_{VP} SH SH SW NT_{PP} SH SH SW SH SH SW RE SH SH SH SW SW NT_{PP} SH SH SH SW SW RE SH SH SH SW SW RE NT_{S} SH SH RE NT_{ROOT} SH RE FI

c) bottom-up + SWAP: SH SH SH SW SH SH SW SH SH SW RE^{3}_{PP} SH SH SH SW SW SH SH SH SW SW RE^{2}_{PP} SH SH SH SW SW RE^{4}_{VP} SH SH RE^{3}_{S} SH RE^{2}_{ROOT} FI

d) in-order + SWAP#k: SH NT_{VP} SH SH SW^{1} NT_{PP} SH SH SW^{1} SH SH SW^{1} RE SH SH SH SW^{2} NT_{PP} SH SH SH SW^{2} RE SH SH SH SW^{2} RE NT_{S} SH SH RE NT_{ROOT} SH RE FI

e) in-order + SHIFT#k: SH^{1} NT_{VP} SH^{2} NT_{PP} SH^{2} SH^{2} RE SH^{3} NT_{PP} SH^{3} RE SH^{3} RE NT_{S} SH^{1} SH^{1} RE NT_{ROOT} SH^{1} RE FI

Fig. 10. Novel shift-reduce linearizations for encoding the discontinuous constituent tree in Fig. 1(b). SH = SHIFT, SH^{k} = SHIFT#K, SW = SWAP, SW^{k} = SWAP#K, NT_{X} = NON-TERMINAL-X, RE = REDUCE, RE^{k}_{X} = REDUCE#K-X and FI = FINISH.

essary for naming constituents and are especially required for the top-down and in-order transition systems (which make use of the NON-TERMINAL-X transition for that purpose). In addition, REDUCE actions in transition-based constituent parsing build partial subtrees, affecting several words of the input sequence (while REDUCE transitions in dependency parsing only affect one single word). Therefore, all these specifics must be taken into consideration for representing the behavior of the data structures in constituent parsing.

More in detail, to implement these dedicated stack and buffer heads of the Transformer decoder's cross-attention mechanism, two masks m^{stack} and m^{buffer} over the input are defined. Additionally, they must be updated at each time step t based on the output token predicted in time step $t-1$ and accordingly to a shift-reduce constituent parsing standpoint:

- If a SHIFT transition was the previous output token, m^{buffer} will mask out the first masked word and this will be included in m^{stack}. The prediction of a SHIFT#K token will have a similar behavior, but affecting the word in the kth position of m^{buffer}.
- Generating a SWAP token in the previous step will modify m^{stack} by masking out the second-to-last word, and m^{buffer} by adding this word. This modification will be applied k times if a SWAP#K action was the previous output token.
- The NON-TERMINAL-X token will have no effect into either m^{stack} or m^{buffer}, since heads can only attend to input words and non-terminal node X is not a token from the input sequence.
- Predicting a REDUCE token (including REDUCE-X and REDUCE#K-X) will mask out all words from m^{stack} that form the resulting con-

stituent, except the first token that will be kept in m^{stack} as a representation of the reduced constituent.

In Fig. 11, we graphically depict how these masks change each time a shift-reduce token is predicted and how they model the content of the stack and buffer during the decoding process.

Given the mask m_t^{stack} (implemented as a vector of $-\infty$ or 0 values) at time step t, the equation proposed by Vaswani et al. [68] (and introduced in Section 3.2) is modified for computing the attention head z_t^{stack} that exclusively attends words in the stack following a shift-reduce point of view:

$$z_t^{stack} = \sum_{i=1}^{n} \alpha_{ti}(h_i W_d^V), \quad \alpha_{ti} = \frac{exp(\beta_{ti})}{\sum_{k=1}^{n} exp(\beta_{tk})},$$

$$\beta_{ti} = \frac{(q_{t-1} W_d^Q)(h_i W_d^K)^T}{\sqrt{d}} + m_{ti}^{stack}$$

The same computation is done for, given mask m_t^{buffer}, obtaining z_t^{buffer} that attends to input tokens still in the buffer at time step t according to a transition-based parsing process:

$$z_t^{buffer} = \sum_{i=1}^{n} \alpha_{ti}(h_i W_d^V), \quad \alpha_{ti} = \frac{exp(\beta_{ti})}{\sum_{k=1}^{n} exp(\beta_{tk})},$$

$$\beta_{ti} = \frac{(q_{t-1} W_d^Q)(h_i W_d^K)^T}{\sqrt{d}} + m_{ti}^{buffer}$$

All shift-reduce linearizations described in Sections 3.1 and 4.1 can be learnt by this neural architecture without further modifications. Additionally, new shift-reduce linearizations can be included by just adapting the modifications to stack and buffer masks if new transitions are incorporated.

We also want to point out the importance of inducing some structural information with the proposed deterministic attention technique especially in discontinuous parsing, since the lack of explicit structures (crucial for handling the word reordering with the SWAP action) may penalize parsing performance.

Finally, we adopt the same encoder and first component of the decoder (named as *masked multi-head self-attention*) from [68], just applying the mentioned modifications to two of the heads from the *encoder-decoder cross-attention multihead attention* module.

5. Experiments

5.1. Setup

Data For properly testing our approach, we include both continuous and discontinuous constituent treebanks. Concretely, the continuous English Penn Treebank (PTB) [45] and its discontinuous version (DPTB) [17] with standard splits defined as follows: Sections 2 to 21 for training, 22 for development and 23 for testing. We also include in the evaluation German treebanks with a higher degree of discontinuity: NEGRA [58] and TIGER [5] with commonly-used splits defined by Dubey and Keller [15] and Seddah, Tsarfaty, Kübler, Candito, Choi, Farkas, Foster, Goenaga, Gojenola Galletebeitia, Goldberg, Green, Habash, Kuhlmann, Maier, Nivre, Przepiórkowski, Roth, Seeker, Versley, Vincze,Woliński,Wróblewska and Villemonte de la Clergerie [56], respectively. In all cases, we discard Part-of-Speech (PoS) tag information and the number of samples per treebank split are detailed in Table 1.

Implementation Following [18], the proposed neural architecture was developed based on the neural model by Ott, Edunov, Grangier and Auli [51]. The latter implements the Transformer model [68] in Pytorch on the `fairseq-py` toolkit[4] and applies it for sequence-to-sequence machine translation. Since constituent parsing can be cast as a sequence-to-sequence task, their model can be easily adapted to our specific problem with minor modifications. In fact, we do not undertake further parameter optimization to our specific task and directly use hyper-parameters reported by Fernandez Astudillo et al. [18] for cross-entropy training with label smoothing and with the learning rate increasing linearly for 4,000 warm-up updates to $5e^{-4}$ and then being decayed proportionally to the inverse square root of the number of steps. These hyperparameters are summarized in Table 2 and we refer the reader to [51] for further implementation details. Finally, we use average weights of the three best checkpoints on the development splits as final models, and beam 10 for decoding.

Pre-trained Embeddings For initializing word embeddings, we use fixed weights extracted from the RoBERTa-large [41] and GottBERT-base [55] pre-trained language models for English and German, respectively. We apply average weights from wordpieces when required and do not fine-tune word embeddings during training.

Evaluation We follow standard practice for evaluation and report F-scores with the EVALB script[5] for the continuous PTB (discarding punctuation), and DISCODOP[6] [12] for discontinuous treebanks (ignoring punctuation and root symbols). The latter also delivers a Discontinuous F-score (DF1) measured only on discontinuous constituents. For each experiment, we report the average score and standard deviation over three executions with different seeds.

Hardware Our approach was fully tested on an Intel(R) Core (TM) i9-10920X CPU @ 3.50 GHz with a single 24 GB TESLA P40 GPU.

5.2. Results

Accuracy incontinuous parsing We first test all described linearizations under the proposed neural network on the continuous version of PTB. In Table 3, we report accuracies on dev and test splits and compare them against state-of-the-art approaches, including all existing sequence-to-sequence constituent parsers. While our approach achieves competitive accuracies with any tree linearization, we can observe that no substantial differences can be found between top-down and in-order strategies, and between enriched and regular variants (contrary to the observations by Fernández-González and Gómez-Rodríguez [21] on RNN sequence-to-sequence constituent parsing). Moreover, we confirm that bottom-up linearizations underperform top-down and in-order variants also under this architecture, as in the results by Ma et al. [42], although the difference is smaller than in their case.

With respect to other sequence-to-sequence constituent parsers, our approach outperforms all existing models by a wide margin; and, in comparison with the best task-specific algorithms, top-down and in-order tree linearizations are only surpassed by models that are enhanced with the pre-trained language model XLNet [80], notably larger than BERT [14] and RoBERTa. In fact, our approach is on par, for instance, with Yang and Deng [79] (a purely transition-based parser) and Tian, Song, Xia and Zhang [67] (a chart-based model) when BERT_{Large} is used instead.*Accuracy indiscontinuous parsing* We further evaluate the proposed sequence-to-sequence model and novel tree linearizations on dev (Table 4) and test (Table 5) splits from discontinuous treebanks, additionally including in the latter table the best approaches to date for a comparison with the current state of the art. As observed on the continuous benchmark, top-down and in-order transition systems (augmented with the SWAP transition) achieve similar overall F-

[4] https://github.com/pytorch/fairseq.
[5] https://nlp.cs.nyu.edu/evalb/.
[6] https://github.com/andreasvc/disco-dop.

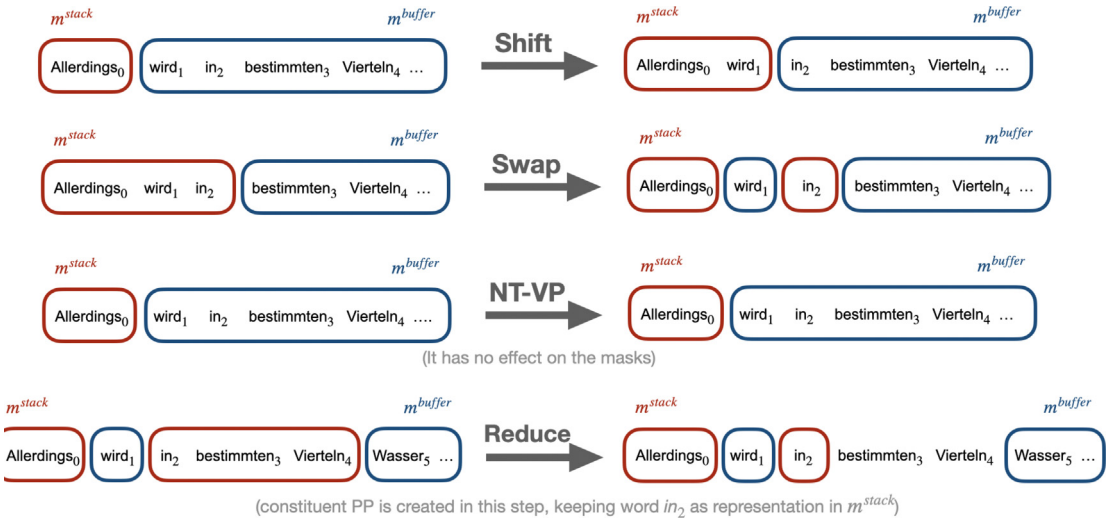


Fig. 11. Modifications on the m^{stack} and m^{buffer} masks for modelling the changes produced by some transitions (of the inorder + SWAP transition system) on the buffer and stack structures during the parsing process described in Fig. 8.

Table 1
Number of samples per treebank split.

Treebank	Training	Dev	Test
PTB/DPTB	39,832	1,700	2,416
NEGRA	18,602	1,000	1,000
TIGER	40,472	5,000	5,000

Table 2
Model hyper-parameters.

Architecture and optimizer hyper-parameters	
Transformer Encoder layers	6
Transformer Encoder size	256
Transformer Decoder layers	6
Transformer Decoder size	256
Heads per self-attention layer	4
RoBERTa embedding dimension	1024
GottBERT embedding dimension	768
Dropout	0.3
Optimizer	Adam [35]
Loss	cross-entropy
β_1	0.9
β_2	0.98
Learning rate	$5e^{-4}$
Learning rate scheduler	Inverse square root
Warm-up initial learning rate	$1e^{-7}$
Warm-up updates	4000
Minimum learning rate	$1e^{-9}$
Label smoothing	0.01
Batch size	3584
Training epochs	80

scores on the English dataset; however, on German treebanks (especially on NEGRA), the in-order tree linearization outperforms the top-down strategy. Regarding the accuracy on discontinuities, the top-down linearization obtains the best F-scores. Again, the bottom-up technique underperforms its counterparts in all datasets by a wide margin.

With respect to the alternatives with SWAP#K and SHIFT#K tokens for shortening output sequences, we notice that the latter provides a poor performance and the former, while achieving a similar overall F-score to tree linearizations with the regular SWAP action, shows a clear loss of accuracy on discontinuities.

Overall, our approach delivers competitive accuracies, outperforming recent task-specific discontinuous parsers (such as Ruprecht and Mörbitz [52] in TIGER and DPTB) and excelling in DPTB (where we achieve the best F-score and Discontinuous F-score to date). It can be also noticed that, the sequence tagging strategy (enhanced with the attention mechanism provided by fully fine-tuning the language model BERT) by Vilares and Gómez-Rodríguez [71], also included in Table 3 as Vilares et al. [72] for the continuous version, is clearly outperformed continuous and discontinuous benchmarks by our sequence-to-sequence model, which uses non-fine-tuned word embeddings.*Parsing speed* We report in Table 6 the speeds provided by each linearization technique during decoding on the test splits. This comparison shows that the continuous and discontinuous bottom-up linearizations and the in-order variants with SWAP#K and SHIFT#K transitions achieve the best speeds, with the latter being the fastest option in discontinuous constituent parsing. This behaviour was expected (and also empirically proved in the following section) since the target sequences generated by these linearizations are shorter than those produced by the other methods, with the in-order linearization with the SHIFT#K token generating the shortest output sequences and being on par with the continuous in-order variant in speed (both 29 sent./s.). Although we leverage contextualized word embeddings in the proposed approach, any linearization of our model is twice as fast as other sequence-to-sequence methods in the continuous constituent benchmark, except for the model that applies a deterministic attention technique to speed up decoding [21]. This latter system (which is based on the in-order linearization) is slightly faster than our more accurate variants (top-down and in-order), but it is surpassed by our model with the bottom-up linearization. Finally, while our implementation was not optimized for speed and the reported results are just intended for comparing the proposed linearization variants, we also include other discontinuous constituent parsers in the comparison, showing that our model is behind all of them[7] and that there is still pending work in speeding up sequence-to-sequence models.

5.3. Analysis

It can be argued that a large output dictionary size harms performance, at least, this is one of the reasons that might explain

[7] Please note that Fernández-González and Gómez-Rodríguez [22] recently obtained such high speeds in discontinuous constituent parsing due to the application of faster continuous parsers after the original sentence was reordered by a pointer network [73].

Table 3
F-score comparison of state-of-the-art constituent parsers on the PTB test split. The second block gathers exclusively sequence-to-sequence models. Parsers that use extra dependency information are marked with *+ dependency*, those that ensemble several trained models with *+ensemble*, those that use a language model for reranking predicted trees with *+LM-rerank*, those that use additional parsed data with *+extra-data*, those that use deterministic attention for increasing parsing speed with *+deterministic-attention*, those that use predicted PoS tags as additional input with *+PoS* and, finally, those that use pre-trained language models BERT_{Large} [14] or XLNet [80] for the encoder initialization are marked with $+\text{BERT}_{Large}$/*+XLNet*. We also include performance on the PTB dev split for all the tested linearizations. We report the average accuracy over 3 executions with different random seeds and standard deviations are indicated with ±.

Parser (no tags or predicted PoS tags)		**PTB**
Liu and Zhang [38]		91.8
Stern, Fried and Klein [62]		92.56
Fernández-González and Gómez-Rodríguez [19]		92.0
Fried and Klein [26]		92.2
Gaddy, Stern and Klein [27]		92.08
Teng and Zhang [66]		92.4
Vilares et al. [72] + BERT_{Large}		93.5
Kitaev, Cao and Klein [36] + BERT_{Large}		95.59
Zhou and Zhao [84] + dependency + BERT_{Large}		95.84
Zhou and Zhao [84] + dependency + XLNet		96.33
Mrini, Dernoncourt, Tran, Bui, Chang and Nakashole [47] + dependency + POS + XLNet		96.38
Yang and Deng [79] + BERT_{Large}		95.79
Yang and Deng [79] + XLNet		96.34
Tian et al. [67] + PoS + BERT_{Large}		95.86
Tian et al. [67] + PoS + XLNet		**96.40**
Fernández-González and Gómez-Rodríguez [25] + dependency + BERT_{Large}		95.23
(sequence-to-sequence models)		
Vinyals et al. [74]		88.3
Vinyals et al. [74] + ensemble		90.5
Vinyals et al. [74] + ensemble + extra-data		92.8
Ma et al. [42] + ensemble		90.6
Kamigaito et al. [34] + ensemble		91.5
Liu et al. [39] + ensemble		92.3
Suzuki, Takase, Kamigaito, Morishita and Nagata [65] + ensemble + LM-rerank		94.32
Liu and Zhang [37]		90.5
Fernández-González and Gómez-Rodríguez [21]		91.6
Fernández-González and Gómez-Rodríguez [21] + deterministic-attention		91.2
Vaswani et al. [68]		91.3
Vaswani et al. [68] + extra-data		92.7
This work:	**(dev)**	**(test)**
SH-RE top-down linearization	95.56±0.09	95.78±0.06
enriched SH-RE top-down/bracketed linearization	**95.63**±0.07	95.70±0.10
SH-RE in-order linearization	95.46±0.03	**95.84**±0.02
enriched SH-RE in-order linearization	95.48±0.03	95.71±0.02
SH-RE bottom-up linearization	95.40±0.15	95.58±0.04

Table 4
F-score on TIGER, NEGRA and DPTB development splits. We report the average accuracy over 3 executions with different random seeds and standard deviations are indicated with ±.

Tree linearization	**TIGER**	**NEGRA**	**DPTB**
SH-RE top-down + SWAP	**92.36**±0.07	90.14±0.03	**95.44**±0.03
SH-RE bottom-up + SWAP	91.48±0.09	88.14±0.02	94.81±0.08
SH-RE in-order + SWAP	92.32±0.02	**90.79**±0.08	95.32±0.09
SH-RE in-order + SWAP#K	92.27±0.06	90.36±0.14	95.25±0.16
SH-RE in-order + SHIFT#K	91.11±0.04	88.81±0.16	94.76±0.12

the difference in accuracy between our approach and sequence labeling techniques, where the target vocabulary is significantly larger to keep synchronicity between the length of the input and output sequences. However, some studies, such as Fernández-González and Gómez-Rodríguez [21], claim that, by properly augmenting (*enriching*) the vocabulary, accuracy gains can be obtained.

In order to better understand why some tree linearizations are underperforming others in terms of accuracy (more notable in discontinuous parsing), we report in Table 7 the target vocabulary size and longest sequences for each tree linearization and treebank. From this information, we can extract that:

- The two best-performing linearizations (regular top-down and in-order methods) present the smallest output dictionaries both in continuous and discontinuous datasets.
- Enriched variants in PTB almost duplicate output dictionary sizes, but obtain scores on par with the regular versions (not providing substantial accuracy gains as reported for RNN sequence-to-sequence models [21]).
- The bottom-up strategy requires a larger vocabulary (due to the parameterized REDUCE#K-X) that might penalize its performance. Note that this would hold even on other bottom-up transition systems that require a previous binarization, since this transformation enlarges the amount of non-terminal labels and, as a consequence, the output vocabulary.
- Dealing with short target sequences does not lead to a better performance, since the use of the token SHIFT#K generates the shortest output sequences in discontinuous treebanks, but underperforms practically all other methods.
- Finally, it can be also observed that, although target sequences in discontinuous datasets are significantly longer than those in PTB, the attention mechanism avoids deteriorating accuracy on long sequences, achieving similar accuracies, for instance, on PTB and DPTB.

Therefore, the correlation between vocabulary size and performance can be observed, but the target sequence length seems to have no impact in accuracy thanks to the attention mechanism, which is the likely reason why SWAP#K and SHIFT#K seem to produce no discernible advantage over the SWAP transition under our neural architecture.

Moreover, we can clearly see in Table 7 a correlation between target sequence length and decoding speed, regardless of dictionary size. The higher speeds in continuous and discontinuous constituent parsing are respectively delivered by the bottom-up and the in-order + SHIFT#K linearizations (as shown in Table 6), which clearly are the variants that produce the shortest target sequences. Even the fact that the bottom-up alternative has the largest dictionary among continuous linearizations seems to have no effect in decoding speed. In addition, the continuous in-order variant generates output sequences with the same length as those provided by the in-order + SHIFT#K, obtaining the same speed during decoding time. Finally, we can also observe that the top-down and in-order linearizations (with and without the SWAP token augmentation) are the slowest options since they generate the longest target sequences.

We also believe that the logic of each transition system and how the output tokens encode the resulting constituent tree can affect the text-to-parse translation. For instance, from a transition-based standpoint, the use of the SHIFT#K action allows the model to operate over the whole buffer by just predicting the SHIFT transition parameterized with the correct k value, covering a broader context and losing the locality typically found in classic transition-based algorithms (which exclusively modify the first word in the buffer and/or the two words on top of the stack). In sequence-to-sequence models, this means that the model has a broader search space, not only due to dealing with a larger output dictionary, but also due to having several options for the same purpose: in this case, different SHIFT#K tokens can be used for representing each input word, instead of using a single SHIFT transition. This might explain why the linearization with SHIFT#K underperforms the other alternatives that behave like classic shift-reduce algorithms

Table 5
F-score and Discontinuous F-score (DF1) comparison of state-of-the-art discontinuous constituent parsers on TIGER, NEGRA and DPTB test splits. Parsers that use extra dependency information are marked with *+ dep*, and those that use pre-trained language models $BERT_{base}$, $BERT_{Large}$ [14] or XLNet [80] for the encoder initialization are marked with $+BERT_{base}$/$+BERT_{Large}$/*+XLNet* (we use $+BERT_X$ when the model size was not specified). We report the average accuracy over 3 executions with different random seeds and standard deviations are indicated with ±.

	TIGER		NEGRA		DPTB	
Parser(no tags or predicted PoS tags)	**F1**	**DF1**	**F1**	**DF1**	**F1**	**DF1**
Coavoux and Cohen [8]	82.5	55.9	83.2	56.3	90.9	67.3
Coavoux, Crabbé and Cohen [10]	82.7	55.9	83.2	54.6	91.0	71.3
Stanojevic and Steedman [60]	83.4	53.5	83.6	50.7	90.5	67.1
CorroCorro2020SpanbasedDC + $BERT_X$	90.0	62.1	91.6	66.1	94.8	68.9
Vilares and Gómez-Rodríguez [71] + $BERT_{Base}$	84.6	51.1	83.9	45.6	91.9	50.8
Vilares and Gómez-Rodríguez [71] + $BERT_{Large}$	–	–	–	–	92.8	53.9
Fernández-González and Gómez-Rodríguez [20]	85.7	60.4	85.7	58.6	–	–
Ruprecht and Mörbitz [52] + $BERT_{Base}$	88.3	69.0	90.9	72.6	93.3	80.5
Fernández-González and Gómez-Rodríguez [22] + $BERT_{Base}$	88.5	63.0	90.0	65.9	94.0	68.9
Fernández-González and Gómez-Rodríguez [22] + $BERT_{Large}$	**90.5**	68.1	**92.0**	67.9	94.7	72.9
Fernández-González and Gómez-Rodríguez [22] + XLNet	–	–	–	–	**95.1**	**74.1**
Fernández-González and Gómez-Rodríguez [25]+ dep + $BERT_{Base}$	89.8	**71.0**	91.0	**76.6**	–	–
This work:	±0.05	±0.27	±0.09	±1.35	±0.06	±0.46
SH-RE top-down + SWAP **linearization**	88.28	**67.95**	88.59	**67.43**	95.37	**83.85**
	±0.14	±0.61	±0.19	±0.65	±0.04	±0.49
SH-RE bottom-up + SWAP **linearization**	87.02	63.20	85.74	57.76	95.12	82.40
	±0.04	±0.45	±0.02	±0.84	±0.06	±0.25
SH-RE in-order + SWAP **linearization**	**88.53**	67.76	**89.08**	67.06	95.47	83.80
	±0.08	±0.49	±0.10	±0.76	±0.01	±0.38
SH-RE in-order + SWAP#K **linearization**	88.36	65.68	88.93	65.38	**95.48**	82.86
	±0.12	±0.16	±0.13	±0.41	±0.09	±0.66
SH-RE in-order + SHIFT#K **linearization**	87.10	54.27	86.76	46.86	94.96	69.17

Table 6
Speed comparison (sentences/s) of our approach with different linearization techniques on TIGER, NEGRA, DPTB and continuous PTB test splits. We also add in the second block those sequence-to-sequence models whose speed can be found in the literature, including the approach by Fernández-González and Gómez-Rodríguez [21] that speeds up parsing decoding with deterministic attention (marked with *+ deterministic-attention*). In addition, we present in the first block top-performing constituent parsers augmented with pre-trained language models $BERT_{base}$, $BERT_{Large}$ [14] or XLNet [80] ($+BERT_{base}$/$+BERT_{Large}$/*+XLNet*). Please note that the reported speeds from previous work were measured on different hardware setups.

Parser	**TIGER**	**NEGRA**	**DPTB**	**PTB**
Zhou and Zhao [84] + XLNet	–	–	–	65
Mrini et al.[47] + XLNet	–	–	–	59
Yang and Deng (2020) [79] + XLNet	–	–	–	71
Vilares and Gómez-Rodríguez [71] + $BERT_{Base}$	80	80	80	–
Vilares and Gómez-Rodríguez [71] + $BERT_{Large}$	–	–	34	–
Ruprecht and Mörbitz [52] + $BERT_{Base}$	60	68	57	–
Fernández-González and Gómez-Rodríguez [22] + $BERT_{Base}$	**238**	**275**	**231**	–
Fernández-González and Gómez-Rodríguez [22] + $BERT_{Large}$	207	216	193	–
Fernández-González and Gómez-Rodríguez [22] + XLNet	–	–	179	–
(sequence-to-sequence models)				
Liu and Zhang [37]	–	–	–	17
Fernández-González and Gómez-Rodríguez [21]	–	–	–	17
Fernández-González and Gómez-Rodríguez [21] + deterministic-attention	–	–	–	35
This work:				
SH-RE top-down linearization	–	–	–	29
SH-RE bottom-up linearization	–	–	–	**39**
SH-RE in-order linearization	–	–	–	29
SH-RE top-down + SWAP **linearization**	26	21	21	–
SH-RE bottom-up + SWAP **linearization**	29	27	26	–
SH-RE in-order + SWAP **linearization**	26	21	21	–
SH-RE in-order + SWAP#K **linearization**	30	29	24	–
SH-RE in-order + SHIFT#K **linearization**	**47**	**50**	**29**	–

and have a single SHIFT token to read words from the input. Moreover, this same reasoning can be used for explaining how the availability of different SWAP#K tokens (instead of applying a single SWAP) penalizes accuracy prediction and leads to a worse performance on discontinuities. On the other hand, transition systems that mark the beginning and the end of each constituent with NON-TERMINAL-X and REDUCE tokens (as can be seen in sequences generated by the top-down and in-order algorithms) tend to work better as tree linearizations than the bottom-up strategy (which denotes the span of each constituent with a single REDUCE#K-X at the end). This might be the main explanation why bottom-up approaches are less adequate for encoding phrase structure trees.

Additionally, to provide more evidence that can help us understand the differences in performance between different linearizations, we undertake an error analysis relative to structural factors and sentence lengths on a concatenation of the dev splits from the three discontinuous treebanks. In particular, Fig. 12(a) shows the F-score on span identification for different lengths, Fig. 12(b) presents the performance on different sentence length cutoffs and Fig. 12(c) plots the accuracy when assigning the most frequent

Table 7
Output dictionary size (Size) and length of the longest target sequence (Length) in TIGER, NEGRA, DPTB and continuous PTB training datasets for each proposed linearization technique.

	TIGER		NEGRA		DPTB		PTB	
Tree linearization	**Size**	**Length**	**Size**	**Length**	**Size**	**Length**	**Size**	**Length**
SH-RE top-down	–	–	–	–	–	–	29	367
enriched SH-RE top-down/bracketed	–	–	–	–	–	–	55	367
SH-RE in-order	–	–	–	–	–	–	30	368
enriched SH-RE in-order	–	–	–	–	–	–	56	368
SH-RE bottom-up	–	–	–	–	–	–	202	255
SH-RE top-down + SWAP	28	2067	30	2061	31	1497	–	–
SH-RE bottom-up + SWAP	191	2039	205	2021	204	1444	–	–
SH-RE in-order + SWAP	29	2068	31	2062	32	1498	–	–
SH-RE in-order + SWAP#K	66	1150	63	1202	57	873	–	–
SH-RE in-order + SHIFT#K	66	257	63	208	57	368	–	–

non-terminal labels (including the average span length in brackets) by each proposed discontinuous tree linearization. From that information, we can claim that:

- Error propagation, often observed in purely transition-based parsers, can be also seen in Fig. 12(a) and (b) for sequence-to-sequence models. As expected, sequential prediction can cause earlier mistakes to affect future decisions, resulting in more errors in the encoded constituent tree. Its impact can be seen on long spans and long sentences, where the accuracy of all linearizations decreases.
- While the in-order + SWAP linearization shows the best performance on phrases with longer spans; using SWAP#K instead harms accuracy, especially when the span length increases (probably due to the fact that a larger amount of SWAP#K are required for reordering a longer sequence of input words, and more mistakes can be made during that process).
- The in-order variant with the SHIFT#K transition suffers notable accuracy loses on producing longer constituents, probably because it is more likely to make a mistake and predict the wrong SHIFT#K token in constituents that cover more input words (represented by SHIFT#K tokens in the tree linearization) and, therefore, the impact of error propagation is higher. A similar trend can be seen for the bottom-up strategy as, while REDUCE#K-X tokens with a lower k value are more frequent and easier to learn, wrong predictions are more likely on longer constituents.
- Based on the performance on longer sentences, we can also note that error propagation has a higher impact on bottom-up and in-order + SHIFT#K linearizations than on the other alternatives.
- No significant differences in performance between the top-down + SWAP and in-order + SWAP strategies can be found in Fig. 12(a) and (b); however, in Fig. 12(c), we can see that the top-down approach is substantially outperformed by the in-order strategy on building constituents of type AP (Adjective

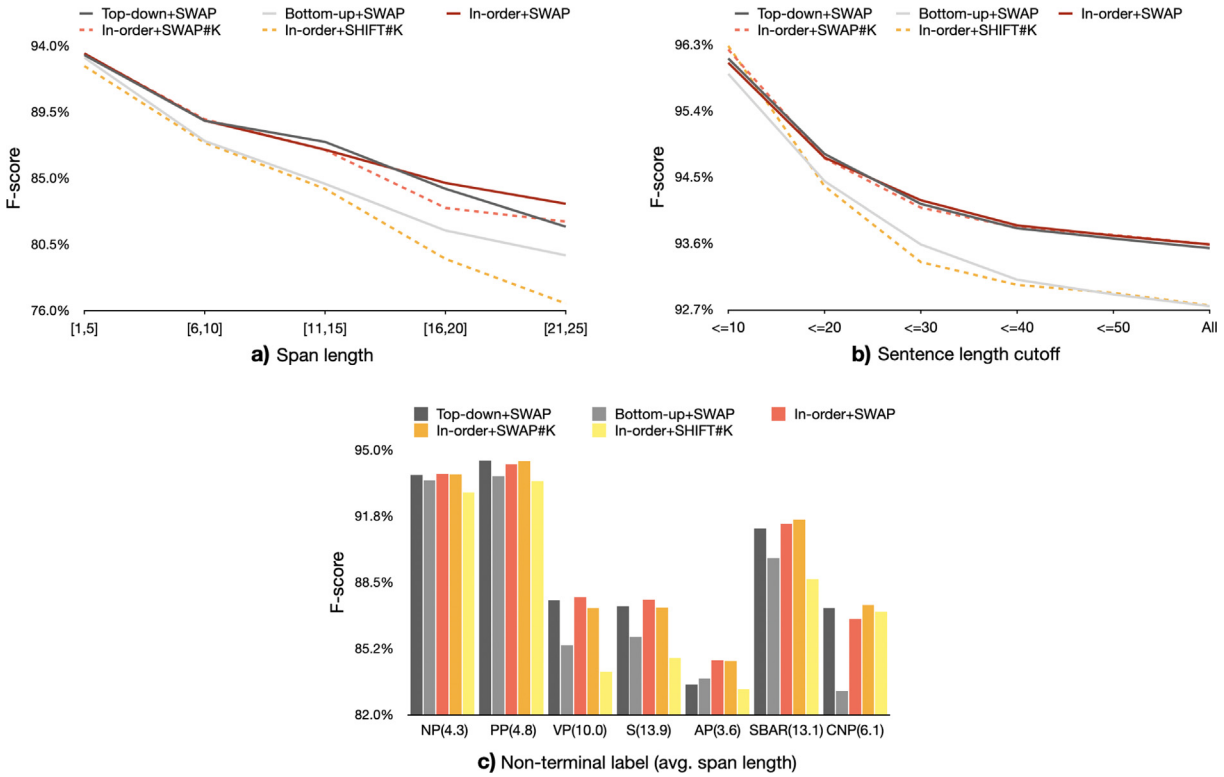


Fig. 12. Accuracy of discontinuous linearization methods relative to structural factors and sentence length.

Phrase), even being slightly surpassed by the bottom-up strategy on that task. While no frequent patterns were observed in the data to explain that behaviour, a lower precision on this kind of structures (i.e., tagging as AP constituents of a different type) with respect to the other linearizations is the reason of these differences in F-score.

- Finally, the bottom-up linearization method has important drops in accuracy on the creation of constituents of type VP, S, SBAR and, especially, CNP. While the low performance on constituents VP, S and SBAR can be explained by the fact that they have a high span length and this linearization is more prone to suffer from error propagation; the poor accuracy on constituents of type CNP (Coordinated Noun Phrases) is caused by failing to correctly identify the boundaries of that kind of phrases when long enumerations of Noun Phrases (separated by commas) have to be processed. The variant with the SHIFT#K transition also has a poor performance on large constituents VP, S and SBAR; but surprisingly outperforms the in-order + SWAP (one of the best-performing linearizations) on building CNP constituents.

6. Conclusions

In this article, we present the first sequence-to-sequence constituent parser that can produce discontinuous phrase structure trees. To achieve that, we define novel transition systems for linearizing discontinuous structures and present a more powerful neural architecture to implement a state-of-the-art sequence-to-sequence model. The resulting system not only accurately produces discontinuous constituent trees, but also achieves the best accuracy to date among sequence-to-sequence constituent parsers on the main benchmarks, and advances the state of the art in accuracy on DPTB.

Finally, it is worth mentioning that, to the best of our knowledge, neither of the novel transition systems defined in this work have been studied in a purely transition-based framework and, since the in-order algorithm [38] achieves the best accuracy to date for a transition-based parser on continuous treebanks, the variant enhanced with the SWAP action might outperform current state-of-the-art models on discontinuous benchmarks.

CRediT authorship contribution statement

Daniel Fernández-González: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing. **Carlos Gómez-Rodríguez:** Validation, Formal analysis, Writing - review & editing, Supervision, Project administration, Funding acquisition.

Data availability

Source code available at https://github.com/danifg/Disco-Seq2seq-Parser.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We acknowledge 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) and the Horizon Europe research and innovation programme (SALSA, grant agreement No 101100615), ERDF/MICINN-AEI (SCANNER-UDC, PID2020-113230RB-C21), Xunta de Galicia (ED431C 2020/11), and Centro de Investigación de Galicia "CITIC", funded by Xunta de Galicia and the European Union (ERDF - Galicia 2014–2020 Program), by grant ED431G 2019/01. Funding for open access charge: Universidade da Coruña/CISUG.

References

[1] Ba, J.L., Kiros, J.R., Hinton, G.E., 2016. Layer normalization. URL: https://arxiv.org/abs/1607.06450, 10.48550/ARXIV.1607.06450.
[2] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, CoRR abs/1409.0473 (2014).
[3] Bai, J., Wang, Y., Chen, Y., Yang, Y., Bai, J., Yu, J., Tong, Y., 2021. Syntax-BERT: Improving pre-trained transformers with syntax trees, in: Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, Association for Computational Linguistics, Online. pp. 3011–3020. URL: https://www.aclweb.org/anthology/2021.eacl-main.262.
[4] Balachandran, V., Pagnoni, A., Lee, J.Y., Rajagopal, D., Carbonell, J., Tsvetkov, Y., 2021. StructSum: Summarization via structured representations, in: Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, Association for Computational Linguistics, Online. pp. 2575–2585. URL: https://www.aclweb.org/anthology/2021.eacl-main.220.
[5] Brants, S., Dipper, S., Hansen, S., Lezius, W., Smith, G., 2002. TIGER treebank, in: Proceedings of the 1st Workshop on Treebanks and Linguistic Theories (TLT), pp. 24–42.
[6] Q. Cao, X. Liang, B. Li, L. Lin, Interpretable visual question answering by reasoning on dependency trees, IEEE Trans. Pattern Anal. Mach. Intell. 43 (2021) 887–901, https://doi.org/10.1109/tpami.2019.2943456.
[7] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar. pp. 1724–1734. URL: https://www.aclweb.org/anthology/D14-1179, DOI: 10.3115/v1/D14-1179.
[8] Coavoux, M., Cohen, S.B., 2019. Discontinuous constituency parsing with a stack-free transition system and a dynamic oracle, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, Minneapolis, Minnesota. pp. 204–217.
[9] Coavoux, M., Crabbé, B., 2017. Incremental discontinuous phrase structure parsing with the GAP transition, in: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, Valencia, Spain. pp. 1259–1270.
[10] M. Coavoux, B. Crabbé, S.B. Cohen, Unlexicalized transition-based discontinuous constituency parsing, Trans. Assoc. Comput. Linguist. 7 (2019) 73–89.
[11] Corro, C., 2020. Span-based discontinuous constituency parsing: a family of exact chart-based algorithms with time complexities from O(n6) down to O (n3), in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Online. pp. 2753–2764. URL: https://www.aclweb.org/anthology/2020.emnlp-main.219, 10.18653/v1/2020.emnlp-main.219.
[12] A. van Cranenburgh, R. Scha, R. Bod, Data-oriented parsing with discontinuous constituents and function tags, J. Language Modell. 4 (2016) 57–111.
[13] Cross, J., Huang, L., 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles, in: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Austin, Texas. pp. 1–11. URL: https://www.aclweb.org/anthology/D16-1001, 10.18653/v1/D16-1001.
[14] Devlin, J., Chang, M.W., Lee, K., Toutanova, K., 2019. BERT: Pre-training of deep bidirectional transformers for language understanding, 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), Association for Computational Linguistics, Minneapolis, Minnesota. pp. 4171–4186. URL: https://www.aclweb.org/anthology/N19-1423, 10.18653/v1/N19-1423.
[15] Dubey, A., Keller, F., 2003. Probabilistic parsing for German using sister-head dependencies, in: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, Sapporo, Japan. pp. 96–103.
[16] C. Dyer, A. Kuncoro, M. Ballesteros, N.A. Smith, Recurrent neural network grammars, HLT-NAACL, Assoc. Comput. Linguist. (2016) 199–209.
[17] Evang, K., Kallmeyer, L., 2011. PLCFRS parsing of English discontinuous constituents, in: Proceedings of the 12th International Conference on Parsing Technologies, Association for Computational Linguistics, Dublin, Ireland. pp. 104–116. URL: https://www.aclweb.org/anthology/W11-2913.
[18] Fernandez Astudillo, R., Ballesteros, M., Naseem, T., Blodgett, A., Florian, R., 2020. Transition-based parsing with stack-transformers, in: Findings of the Association for Computational Linguistics: EMNLP 2020, Association for Computational Linguistics, Online. pp. 1001–1007. URL: https://www.

aclweb.org/anthology/2020.findings-emnlp.89, 10.18653/v1/2020.findings-emnlp.89.
[19] Fernández-González, D., Gómez-Rodríguez, C., 2018. Dynamic oracles for top-down and in-order shift-reduce constituent parsing, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Brussels, Belgium. pp. 1303–1313. URL: https://www.aclweb.org/anthology/D18-1161, 10.18653/v1/D18-1161.
[20] Fernández-González, D., Gómez-Rodríguez, C., 2020a. Discontinuous constituent parsing with pointer networks, in: Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7–12, 2020, AAAI Press. pp. 7724–7731.
[21] Fernández-González, D., Gómez-Rodríguez, C., 2020b. Enriched in-order linearization for faster sequence-to-sequence constituent parsing, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online. pp. 4092–4099. URL: https://www.aclweb.org/anthology/2020.acl-main.376, 10.18653/v1/2020.acl-main.376.
[22] Fernández-González, D., Gómez-Rodríguez, C., 2021. Reducing discontinuous to continuous parsing with pointer network reordering, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Online and Punta Cana, Dominican Republic. pp. 10570–10578. URL: https://aclanthology.org/2021.emnlp-main.825.
[23] Fernández-González, D., Martins, A.F.T., 2015. Parsing as reduction, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, Beijing, China. pp. 1523–1533. URL: https://www.aclweb.org/anthology/P15-1147, DOI: 10.3115/v1/P15-1147.
[24] D. Fernández-González, C. Gómez-Rodríguez, Faster shift-reduce constituent parsing with a non-binary, bottom-up strategy, Artif. Intell. 275 (2019) 559–574, https://doi.org/10.1016/j.artint.2019.07.006, URL: http://www.sciencedirect.com/science/article/pii/S000437021830540X.
[25] D. Fernández-González, C. Gómez-Rodríguez, Multitask pointer network for multi-representational parsing, Knowl.-Based Syst. 236 (2022), https://doi.org/10.1016/j.knosys.2021.107760, URL: https://www.sciencedirect.com/science/article/pii/S0950705121009849.
[26] Fried, D., Klein, D., 2018. Policy gradient as a proxy for dynamic oracles in constituency parsing, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Association for Computational Linguistics, Melbourne, Australia. pp. 469–476. URL: https://www.aclweb.org/anthology/P18-2075, 10.18653/v1/P18-2075.
[27] Gaddy, D., Stern, M., Klein, D., 2018. What's going on in neural constituency parsers? an analysis, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), Association for Computational Linguistics, New Orleans, Louisiana. pp. 999–1010. URL: https://www.aclweb.org/anthology/N18-1091, 10.18653/v1/N18-1091.
[28] Gebhardt, K., 2020. Advances in using grammars with latent annotations for discontinuous parsing, in: Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies, Association for Computational Linguistics, Online. pp. 91–97. URL: https://www.aclweb.org/anthology/2020.iwpt-1.9, 10.18653/v1/2020.iwpt-1.9.
[29] K. Gebhardt, M.J. Nederhof, H. Vogler, Hybrid grammars for parsing of discontinuous phrase structures and non-projective dependency structures, Comput. Linguist. 43 (2017) 465–520, https://doi.org/10.1162/COLI_a_00291, URL: https://www.aclweb.org/anthology/J17-3001.
[30] Gómez-Rodríguez, C., Vilares, D., 2018. Constituent parsing as sequence labeling, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Brussels, Belgium. pp. 1314–1324. URL: https://www.aclweb.org/anthology/D18-1162, 10.18653/v1/D18-1162.
[31] A. Graves, J. Schmidhuber, Framewise phoneme classification with bidirectional lstm and other neural network architectures, Neural Networks (2005) 5–6.
[32] Hall, J., Nivre, J., 2008. A dependency-driven parser for German dependency and constituency representations, in: Proceedings of the Workshop on Parsing German, Association for Computational Linguistics, Columbus, Ohio. pp. 47–54. URL: https://www.aclweb.org/anthology/W08-1007.
[33] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735.
[34] Kamigaito, H., Hayashi, K., Hirao, T., Takamura, H., Okumura, M., Nagata, M., 2017. Supervised attention for sequence-to-sequence constituency parsing, in: Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers), Asian Federation of Natural Language Processing, Taipei, Taiwan. pp. 7–12. URL: https://www.aclweb.org/anthology/I17-2002.
[35] Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
[36] Kitaev, N., Cao, S., Klein, D., 2019. Multilingual constituency parsing with self-attention and pre-training, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy. pp. 3499–3505. URL: https://www.aclweb.org/anthology/P19-1340, 10.18653/v1/P19-1340.
[37] Liu, J., Zhang, Y., 2017a. Encoder-decoder shift-reduce syntactic parsing, in: Proceedings of the 15th International Conference on Parsing Technologies, IWPT 2017, Pisa, Italy, September 20–22, 2017, pp. 105–114. URL: https://aclanthology.info/papers/W17-6315/w17-6315.
[38] J. Liu, Y. Zhang, In-order transition-based constituent parsing, Trans. Assoc. Comput. Linguist. 5 (2017) 413–424, URL: https://www.transacl.org/ojs/index.php/tacl/article/view/1199.
[39] Liu, L., Zhu, M., Shi, S., 2018. Improving sequence-to-sequence constituency parsing, in: AAAI Conference on Artificial Intelligence. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16347.
[40] Y. Liu, J. Gu, N. Goyal, X. Li, S. Edunov, M. Ghazvininejad, M. Lewis, L. Zettlemoyer, Multilingual denoising pre-training for neural machine translation, Trans. Assoc. Comput. Linguist. 8 (2020) 726–742, https://doi.org/10.1162/tacl_a_00343, url: https://www.aclweb.org/anthology/2020.tacl-1.47.
[41] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.
[42] Ma, C., Liu, L., Tamura, A., Zhao, T., Sumita, E., 2017. Deterministic attention for sequence-to-sequence constituent parsing, in: AAAI Conference on Artificial Intelligence. URL: https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14317.
[43] Maier, W., 2015. Discontinuous incremental shift-reduce parsing, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, Beijing, China. pp. 1202–1212. URL: https://www.aclweb.org/anthology/P15-1116, DOI: 10.3115/v1/P15-1116.
[44] Maier, W., Lichte, T., 2016. Discontinuous parsing with continuous trees, in: Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing, Association for Computational Linguistics, San Diego, California. pp. 47–57.
[45] M.P. Marcus, B. Santorini, M.A. Marcinkiewicz, Building a large annotated corpus of English: The Penn Treebank, Comput. Linguist. 19 (1993) 313–330.
[46] Mohamed, A., Okhonko, D., Zettlemoyer, L., 2019. Transformers with convolutional context for ASR. CoRR abs/1904.11660. URL: http://arxiv.org/abs/1904.11660, arXiv:1904.11660.
[47] Mrini, K., Dernoncourt, F., Tran, Q.H., Bui, T., Chang, W., Nakashole, N., 2020. Rethinking self-attention: Towards interpretability in neural parsing, in: Findings of the Association for Computational Linguistics: EMNLP 2020, Association for Computational Linguistics, Online. pp. 731–742. URL: https://www.aclweb.org/anthology/2020.findings-emnlp.65, 10.18653/v1/2020.findings-emnlp.65.
[48] D.Q. Nguyen, K. Verspoor, From pos tagging to dependency parsing for biomedical event extraction, BMC Bioinform. 20 (2019) 72, https://doi.org/10.1186/s12859-019-2604-0.
[49] Nivre, J., 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, Association for Computational Linguistics, Suntec, Singapore. pp. 351–359. URL: https://www.aclweb.org/anthology/P09-1040.
[50] Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., Auli, M., 2019. fairseq: A fast, extensible toolkit for sequence modeling, in: Proceedings of NAACL-HLT 2019: Demonstrations.
[51] Ott, M., Edunov, S., Grangier, D., Auli, M., 2018. Scaling neural machine translation, in: Proceedings of the Third Conference on Machine Translation: Research Papers, Association for Computational Linguistics, Belgium, Brussels. pp. 1–9. URL: https://www.aclweb.org/anthology/W18-6301, 10.18653/v1/W18-6301.
[52] Ruprecht, T., Mörbitz, R., 2021. Supertagging-based parsing with linear context-free rewriting systems, in: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Online. pp. 2923–2935. URL: https://www.aclweb.org/anthology/2021.naacl-main.232, 10.18653/v1/2021.naacl-main.232.
[53] Sachan, D., Zhang, Y., Qi, P., Hamilton, W.L., 2021. Do syntax trees help pre-trained transformers extract information?, in: Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, Association for Computational Linguistics, Online. pp. 2647–2661. URL: https://www.aclweb.org/anthology/2021.eacl-main.228.
[54] Sagae, K., Lavie, A., 2005. A classifier-based parser with linear run-time complexity, in: Proceedings of the 9th International Workshop on Parsing Technologies (IWPT), pp. 125–132.
[55] Scheible, R., Thomczyk, F., Tippmann, P., Jaravine, V., Boeker, M., 2020. Gottbert: a pure german language model. arXiv:2012.02110.
[56] Seddah, D., Tsarfaty, R., Kübler, S., Candito, M., Choi, J.D., Farkas, R., Foster, J., Goenaga, I., Gojenola Galletebeitia, K., Goldberg, Y., Green, S., Habash, N., Kuhlmann, M., Maier, W., Nivre, J., Przepiórkowski, A., Roth, R., Seeker, W., Versley, Y., Vincze, V., Woliński, M., Wróblewska, A., Villemonte de la Clergerie, E., 2013. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages, in: Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages, Association for Computational Linguistics, Seattle, Washington, USA. pp. 146–182. URL: https://www.aclweb.org/anthology/W13-4917.

[57] H. Seki, T. Matsumura, M. Fujii, T. Kasami, On multiple context-free grammars, Theoret. Comput. Sci. 88 (1991) 191–229, https://doi.org/10.1016/0304-3975(91)90374-B, URL: https://www.sciencedirect.com/science/article/pii/030439759190374B.
[58] Skut, W., Krenn, B., Brants, T., Uszkoreit, H., 1997. An annotation scheme for free word order languages, in: Proceedings of the Fifth Conference on Applied Natural Language Processing, Association for Computational Linguistics, Stroudsburg, PA, USA. pp. 88–95.
[59] Stanojević, M., Alhama, R.G., 2017. Neural discontinuous constituency parsing, in: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Copenhagen, Denmark. pp. 1666–1676. URL: https://www.aclweb.org/anthology/D17-1174, 10.18653/v1/D17-1174.
[60] Stanojević, M., Steedman, M., 2020. Span-based LCFRS-2 parsing, in: Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies, Association for Computational Linguistics, Online. pp. 111–121. URL: https://www.aclweb.org/anthology/2020.iwpt-1.12, 10.18653/v1/2020.iwpt-1.12.
[61] Stern, M., Andreas, J., Klein, D., 2017a. A minimal span-based neural constituency parser, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Vancouver, Canada. pp. 818–827. URL: https://www.aclweb.org/anthology/P17-1076, 10.18653/v1/P17-1076.
[62] Stern, M., Fried, D., Klein, D., 2017b. Effective inference for generative neural parsing, in: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Copenhagen, Denmark. pp. 1695–1700. URL: https://www.aclweb.org/anthology/D17-1178, 10.18653/v1/D17-1178.
[63] Strubell, E., Verga, P., Andor, D., Weiss, D., McCallum, A., 2018. Linguistically-informed self-attention for semantic role labeling, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Brussels, Belgium. pp. 5027–5038. URL: https://www.aclweb.org/anthology/D18-1548, 10.18653/v1/D18-1548.
[64] Sutskever, I., Vinyals, O., Le, Q.V., 2014. Sequence to sequence learning with neural networks, in: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, MIT Press, Cambridge, MA, USA. p. 3104–3112.
[65] Suzuki, J., Takase, S., Kamigaito, H., Morishita, M., Nagata, M., 2018. An empirical study of building a strong baseline for constituency parsing, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Association for Computational Linguistics, Melbourne, Australia. pp. 612–618. URL: https://www.aclweb.org/anthology/P18-2097.
[66] Teng, Z., Zhang, Y., 2018. Two local models for neural constituent parsing, in: Proceedings of the 27th International Conference on Computational Linguistics, Association for Computational Linguistics, Santa Fe, New Mexico, USA. pp. 119–132. URL: https://www.aclweb.org/anthology/C18-1011.
[67] Tian, Y., Song, Y., Xia, F., Zhang, T., 2020. Improving constituency parsing with span attention, in: Findings of the Association for Computational Linguistics: EMNLP 2020, Association for Computational Linguistics, Online. pp. 1691–1703. URL: https://aclanthology.org/2020.findings-emnlp.153, 10.18653/v1/2020.findings-emnlp.153.
[68] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, u., Polosukhin, I., 2017. Attention is all you need, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA. p. 6000–6010.
[69] Versley, Y., 2014. Experiments with easy-first nonprojective constituent parsing, in: Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages, Dublin City University, Dublin, Ireland. pp. 39–53. URL: https://www.aclweb.org/anthology/W14-6104.
[70] Vijay-Shanker, K., Weir, D.J., Joshi, A.K., 1987. Characterizing structural descriptions produced by various grammatical formalisms, in: Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics (ACL'87), Association for Computational Linguistics, Morristown, NJ, USA. pp. 104–111.
[71] Vilares, D., Gómez-Rodríguez, C., 2020. Discontinuous constituent parsing as sequence labeling, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Online. pp. 2771–2785. URL: https://www.aclweb.org/anthology/2020.emnlp-main.221, 10.18653/v1/2020.emnlp-main.221.
[72] Vilares, D., Strzyz, M., Søgaard, A., Gómez-Rodríguez, C., 2020. Parsing as pretraining. Proceedings of the AAAI Conference on Artificial Intelligence 34, 9114–9121. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6446, DOI: 10.1609/aaai.v34i05.6446.
[73] Vinyals, O., Fortunato, M., Jaitly, N., 2015a. Pointer networks, in: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 28. Curran Associates Inc, pp. 2692–2700.
[74] Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., Hinton, G., 2015b. Grammar as a foreign language, in: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, MIT Press, Cambridge, MA, USA. pp. 2773–2781. URL: http://dl.acm.org/citation.cfm?id=2969442.2969550.
[75] Wang, C., Cho, K., Gu, J., 2020. Neural machine translation with byte-level subwords. Proceedings of the AAAI Conference on Artificial Intelligence 34, 9154–9160. DOI: 10.1609/aaai.v34i05.6451.
[76] Wang, C., Pino, J., Gu, J., 2020. Improving Cross-Lingual Transfer Learning for End-to-End Speech Recognition with Speech Translation. arXiv e-prints, arXiv:2006.05474 arXiv:2006.05474.
[77] Xu, M., Wong, D.F., Yang, B., Zhang, Y., Chao, L.S., 2019. Leveraging local and global patterns for self-attention networks, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy. pp. 3069–3075. URL: https://www.aclweb.org/anthology/P19-1295, 10.18653/v1/P19-1295.
[78] B. Yang, D.F. Wong, L.S. Chao, M. Zhang, Improving tree-based neural machine translation with dynamic lexicalized dependency encoding, Knowl.-Based Syst. 188 (2020), https://doi.org/10.1016/j.knosys.2019.105042, URL: https://www.sciencedirect.com/science/article/pii/S095070511930440X.
[79] Yang, K., Deng, J., 2020. Strongly incremental constituency parsing with graph neural networks, in: Neural Information Processing Systems.
[80] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R.R., Le, Q.V., 2019. Xlnet: Generalized autoregressive pretraining for language understanding, in: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf.
[81] Zhang, B., Zhang, Y., Wang, R., Li, Z., Zhang, M., 2020. Syntax-aware opinion role labeling with dependency graph convolutional networks, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online. pp. 3249–3258. URL: https://www.aclweb.org/anthology/2020.acl-main.297, 10.18653/v1/2020.acl-main.297.
[82] Zhang, M., Li, Z., Fu, G., Zhang, M., 2019. Syntax-enhanced neural machine translation with syntax-aware word representations, 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), Association for Computational Linguistics, Minneapolis, Minnesota. pp. 1151–1161. URL: https://www.aclweb.org/anthology/N19-1118, 10.18653/v1/N19-1118.
[83] M. Zhang, Z. Li, G. Fu, M. Zhang, Dependency-based syntax-aware word representations, Artif. Intell. 292 (2021), https://doi.org/10.1016/j.artint.2020.103427, URL: https://www.sciencedirect.com/science/article/pii/S0004370220301764.
[84] Zhou, J., Zhao, H., 2019. Head-driven phrase structure grammar parsing on Penn treebank, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy. pp. 2396–2408. URL:https://www.aclweb.org/anthology/P19-1230, 10.18653/v1/P19-1230.
[85] Zhu, M., Zhang, Y., Chen, W., Zhang, M., Zhu, J., 2013. Fast and accurate shift-reduce constituent parsing, in: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Sofia, Bulgaria. pp. 434–443. URL: https://www.aclweb.org/anthology/P13-1043.

Daniel Fernández-González received M.Sc. and Ph.D. degrees in Computer Science from the University of Vigo in 2010 and the University of A Coruña in 2015, respectively. As a postgraduate student, he was also a visiting scholar at the Uppsala University (Sweden) and Instituto de Telecomunicações, Instituto Superior Técnico (Portugal). He is currently a Postdoctoral Researcher at the University of A Coruña working on the FASTPARSE project led by Prof. Carlos Gómez-Rodríguez. His main research interests lie in natural language parsing and he has authored over 20 peer-reviewed papers that include both theoretical and empirical work on data-driven constituency and dependency-based parsing algorithms.

Carlos Gómez-Rodríguez received his M.Sc. and Ph.D. degrees in computer science from the University of A Coruña in 2005 and 2009, respectively, and is currently Full Professor in the same institution. His research centers on Computational Linguistics, with special focus on natural language parsing, and he has authored a monograph and over 120 peer-reviewed papers in this field. His contributions include both theoretical and empirical work on parsing algorithms, as well as research on applications of parsing and on cognitive aspects of syntax. He is Principal Investigator of several research projects, including an ERC Proof-of-Concept Grant. He has recently received the National Young Researcher Award "María Andresa Casamayor" for his research in natural language processing.