# Don't Parse, Choose Spans! Continuous and Discontinuous Constituency Parsing via Autoregressive Span Selection

**Songlin Yang, Kewei Tu**[*]
School of Information Science and Technology, ShanghaiTech University
Shanghai Engineering Research Center of Intelligent Vision and Imaging
`yangsl66@mit.edu, tukw@shanghaitech.edu.cn`

## Abstract

We present a simple and unified approach for both continuous and discontinuous constituency parsing via *autoregressive span selection*. Constituency parsing aims to produce a set of non-crossing spans so that they can form a constituency parse tree. We sort gold spans in a predefined order and train a pointer network to autoregressively select spans by that order. To deal with a discontinuous span, we consecutively select its subspans from left to right, label all but the last subspans with a special discontinuous label, and label the last subspan with the whole discontinuous span's label. We use a simple heuristic to output valid trees from selected spans so that our approach is able to predict all possible continuous and discontinuous constituency trees without sacrificing data coverage and without the need to use expensive chart-based parsing algorithms. Extensive experiments show that our model achieves state-of-the-art or competitive performance on all benchmarks of continuous and discontinuous constituency parsing .[1]

## 1 Introduction

Constituency parsing is a fundamental task in natural language processing, having many applications in downstream tasks such as language modeling (Sartran et al., 2022) and machine translation (Akoury et al., 2019). Continuous constituency parsing is not capable of capturing *discontinuous* language phenomena, such as wh-movements and extraposition, which are deemed *unavoidable* in syntactic analysis (McCawley, 1982; Bunt et al., 1987; Müller, 2004). Discontinuous constituency parsing enables modeling these phenomena by allowing each constituent node governing nonadjacent strings in the yields, resulting in discontinuous parse trees as exemplified in Fig. 1(b). In this work

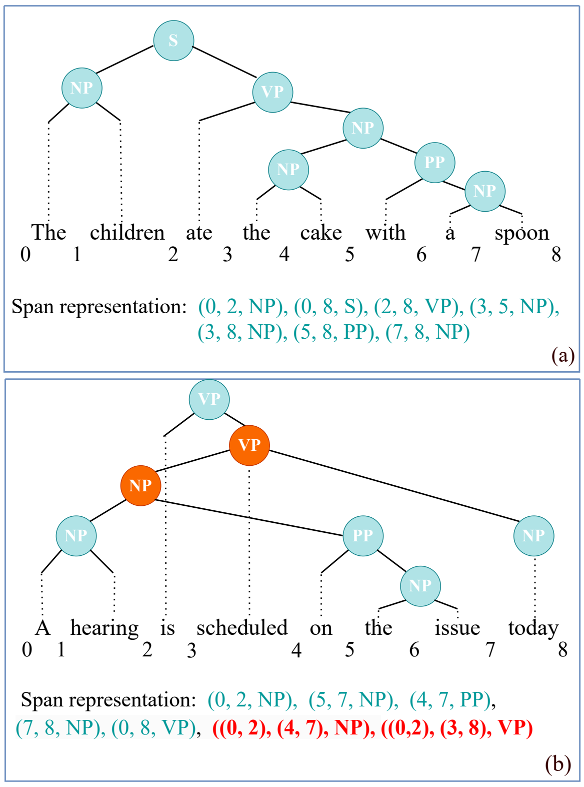
S
NP
VP
NP
NP
PP
NP
The children ate the cake with a spoon
0 1 2 3 4 5 6 7 8
Span representation: (0, 2, NP), (0, 8, S), (2, 8, VP), (3, 5, NP), (3, 8, NP), (5, 8, PP), (7, 8, NP)
(a)
VP
VP
NP
NP
PP
NP
NP
A hearing is scheduled on the issue today
0 1 2 3 4 5 6 7 8
Span representation: (0, 2, NP), (5, 7, NP), (4, 7, PP), (7, 8, NP), (0, 8, VP), ((0, 2), (4, 7), NP), ((0,2), (3, 8), VP)
(b)

**Figure 1:** Examples of (a) continuous and (b) discontinuous constituency parse trees with their span representations. Continuous and discontinuous nodes are colored in blue and orange, respectively. In (b) all discontinuous nodes are of fan-out two.

we study both continuous and discontinuous constituency parsing under a unified framework.

Both continuous and discontinuous parsing can be framed as span prediction problems. In continuous parsing, each span corresponds to a single interval (of the observed sentence), while in discontinuous parsing a discontinuous span could correspond to multiple nonadjacent intervals, the number of which we refer to as *fan-out*. Fig. 1 shows the span representations of both types of parse trees. Based on span representations, span-based methods decompose the score of a parse tree into scores of spans, scoring all possible spans and finding the

[*]Corresponding author
[1]Code is available at `https://github.com/sustcsonglin/disco-pointer`.

best parse with dynamic programming algorithms. They have achieved high accuracy in both continuous parsing (Stern et al., 2017; Kitaev and Klein, 2018; Zhang et al., 2020b) and discontinuous parsing (Corro, 2020; Stanojević and Steedman, 2020). However, they have several issues:

- **Their strong conditional independence assumption is problematic.** Span-based methods score each span independently for global training and decoding, hoping that powerful neural encoders could capture sufficient contextual information for parsing. However, Cui et al. (2022) show that independent local span scoring is suboptimal and they propose auxiliary training objectives to inject non-local features into neural span-based continuous parsing. In discontinuous parsing, span-based parsers impose even stronger conditional independence assumptions when scoring discontinuous spans to alleviate high space and time complexities,[2] which exacerbates the issue of insufficient modeling of non-local contextual information.
- **Data coverage of discontinuous parsing is low.** Current span-based discontinuous parsers can only deal with discontinuous spans of fan-out of at most two due to high parsing time complexity, thereby having limited data coverage. For instance, the most expressive variant of LCFRS-2 (Stanojević and Steedman, 2020) can only cover up to 87% discontinuous spans in the NEGRA treebank, greatly upper-bounding the prediction performance on discontinuous constituents.
- **Global decoding for discontinuous parsing is time consuming.** Continuous parsing needs the cubic-time CKY algorithm for global decoding, which is not problematic when efficient parallel implementations (Zhang et al., 2020b; Rush, 2020) are available. However, discontinuous parsing to LCFRS-2 needs $\mathcal{O}(n^6)$ time, which is computationally prohibitive. While Corro (2020) and Stanojević and Steedman (2020) propose to restrict the search space to speed up parsing,[3] that further damages data coverage.

In this work, we present a simple yet effective approach to address all aforementioned problems via *autogressive span selection.* [4] **To solve the first issue**, we sort gold spans by a predefined order and train a pointer network (Vinyals et al., 2015) to *autoregressively* choose spans by that order. As such, the independence assumption of span-based parsing is weakened due to the existence of the autoregressive neural decoder, which captures span correlations in a similar way to Nguyen et al. (2021) and Yang and Tu (2022) (but see section 4 for how our method differs from theirs). **To solve the second issue**, we devise a simple strategy of decoding discontinuous spans that consecutively selects all continuous subspans of a discontinuous span from left to right and labels all but the last subspans with a special discontinuous label and the last subspan with the label of the entire discontinuous span. As such, we can *connect* an arbitrary number of subspans to obtain a discontinuous span, so we can decode discontinuous spans of arbitrary fan-out and thus arbitrary discontinuous constituency parse trees, enjoying *perfect* data coverage. **For the third issue**, we find a frustratingly simple and fairly fast strategy that works quite well. Concretely, we use *greedy* decoding to output spans in an unconstrained manner and use a heuristic to resolve possible inconsistency (i.e., two spans crossing each other) afterwards by simply discarding conflicting spans. Since $\mathcal{O}(n)$ spans are needed to decode, our method needs linear steps for parsing, thus is efficient. In continuous parsing, this simple strategy combined with our autoregressive neural decoder works even better than global CKY decoding in our experiments. In discontinuous parsing, it removes the need to use expensive chart-based algorithms for decoding, greatly reducing the computational complexity.

We conduct extensive experiments on benchmarks, achieving state-of-the-art performance on PTB and competitive performance on CTB for continuous parsing; and state-of-the-art performance on three benchmarks: Tiger, NeGra, and DPTB for discontinuous parsing.

[2]Scoring all fan-out-2 spans needs $\mathcal{O}(n^4)$ time and space, which is computationally expensive (mainly in space complexity to store scoring tensors). To solve this issue, Corro (2020) and Stanojević and Steedman (2020) decompose the scoring of discontinuous spans to the scoring of multiple continuous spans (e.g., left and right subspans and the gap in between), making stronger conditional independence assumptions than the continuous parsing case.

[3]For example, Corro (2020) proposes a cubic-time discontinuous parser that can only predict the simplest discontinuous structures, i.e., a discontinuous parent span can only have two continuous subspans. Though their parser has a high overall parsing accuracy, the discontinuous F1 is *much* lower than other comparable methods due to the restriction in search space.

[4]Span-based methods can be viewed as non-autoregressive span selection with independence assumptions, in analogy to autoregressive vs. non-autoregressive machine translation.

## 2 Approach

### 2.1 Constituency parsing as span selection

We formally frame (dis)continuous constituency parsing as a span selection problem. A (dis)continuous constituency parse tree $t$ comprises a set of nodes and for each node $s$ we have $\text{yield}(s) = \{s_1, ..., s_l\}$ which is the set of *sorted* token indices in the yield of $s$ in $t$ with $s_1 < \cdots < s_l$. The bidirectional conversion between spans and trees are shown as follows.

**Tree to spans.** It is somewhat trivial to derive the span representation of a tree node $s$ based on $\text{yield}(s)$ by merging consecutive token indices into continuous intervals represented by left and right boundary indices. We do not allow two resulting intervals sharing any boundaries for eliminating potential ambiguities. If a single interval is obtained, then it is a continuous span; if multiple intervals are obtained, then it is a discontinuous span. Fig. 1 shows examples of span representations converted from trees.

**Spans to tree.** We say a set of spans $S$ is *consistent* (to form a valid tree) iff $\forall s, t \in S$, $\text{yield}(s) \cap \text{yield}(t) = \emptyset$ or $\text{yield}(s) \subset \text{yield}(t)$ or $\text{yield}(t) \subset \text{yield}(s)$.[5] We can build a tree from any consistent $S$ as follows. For each $s$ with $|\text{yield}(s)| < n$, we define $P_s \subset S$ as a set of spans in which $s$ is *properly* contained. We say $t$ is parent node of $s$ if $t = \arg\min_{t' \in P_s} \text{len}(t')$. We can thus determine the parent node of each span and thereby construct the whole tree. We find it convenient to reconstruct the tree if all spans are sorted in post-order tree traversal order , i.e., spans of smaller end position (i.e., $\max \text{yield}(s)$) come first, and for tie breaking, spans of smaller width come first. As such, we only need to sequentially scan the sorted spans to build the tree from the bottom up: for a scanned span we build a node for it by looking up prior decoded nodes to find all its children nodes. Finally, we add a <TOP> node spanning the whole sentence to connect all unconnected nodes for obtaining the final parse tree.

It also gives us a simple heuristic to build a tree from a set of inconsistent spans: sort spans by post-order, scan sorted spans and if a span crosses to any prior decoded spans, simply discard it, otherwise build a node for it likewise the aforementioned procedure. We adopt this strategy in the post-processing stage to build trees.

[5]Here we slightly abuse the notation yield.

**Handling discontinuous spans.** We use a simple strategy to deal with discontinuous spans. For a given discontinuous span of fan-out $f$ $\{(l_1, r_1), ..., (l_f, r_f)\}$, we label all but the last subspans (i.e., $\{(l_1, r_1), ..., (l_{f-1}, r_{f-1})\}$) with a special discontinuous label <dis> and the last subspan (i.e., $(l_f, r_f)$) with the label of the entire discontinuous span. For example, a discontinuous span $((0,2),(4,7),(10,14)),\texttt{NP})$ would be transformed to $(0, 2, \texttt{<dis>})$, $(4, 7, \texttt{<dis>})$, $(10, 14, \texttt{NP})$. As such, we can *reduce* discontinuous parsing to continuous parsing, except that we need to guarantee all continuous subspans of a discontinuous span occur consecutively in the (ordered) decoded span list (and also with correctly predicted labels <dis>), so that we can connect subspans to recover the parent discontinuous span. We achieve it using an autoregressive decoder as described later. The key advantage here is that we can handle discontinuous spans of arbitrary fan-out easily.

**Example.** We take the discontinuous parse tree from Fig. 1(b) for example to show how we convert it to a list of sorted spans. First, we sort all spans in post-order, obtaining $\{(0, 2, \texttt{NP}), (5, 7, \texttt{NP}), (4, 7, \texttt{PP}), ((0, 2), (4, 7), \texttt{NP}), (7, 8, \texttt{NP}), ((0, 2), (3, 8), \texttt{VP})$ $(0, 8, \texttt{VP})\}$. Then we use the introduced simple trick to handle discontinuous spans, resulting in $\{(0, 2, \texttt{NP}), (5, 7, \texttt{NP}), (4, 7, \texttt{PP}), (0, 2, \texttt{<dis>}),$ $(4, 7, \texttt{NP}), (7, 8, \texttt{NP}), (0, 2, \texttt{<dis>}), (3, 8, \texttt{VP}),$ $(0, 8, \texttt{VP})\}$.

### 2.2 Neural architecture

**Input.** Given a sentence $w = w_1, \cdots, w_n$, we add <bos> at $w_0$ and <eos> at $w_{n+1}$, and sort the gold spans (for discontinuous parsing we first do the transformation descried earlier) by post-order to get $S = \{(l_i, r_i, y_i)\}_i$ where $l_i, r_i, y_i$ are the left boundary index, right boundary index, and the label index of the $i$-th span, respectively. Since we adopt an autoregressive selection strategy, we need to inform the model when to stop. We achieve it by appending a special indicator span $(0, 0, \texttt{<end>})$ to $S$, and denote the resulting size of $S$ as $m$.

**Encoder.** We tokenize the input sentence and feed it into pre-trained language models such as BERT (Devlin et al., 2019) and XLNet (Yang et al.,

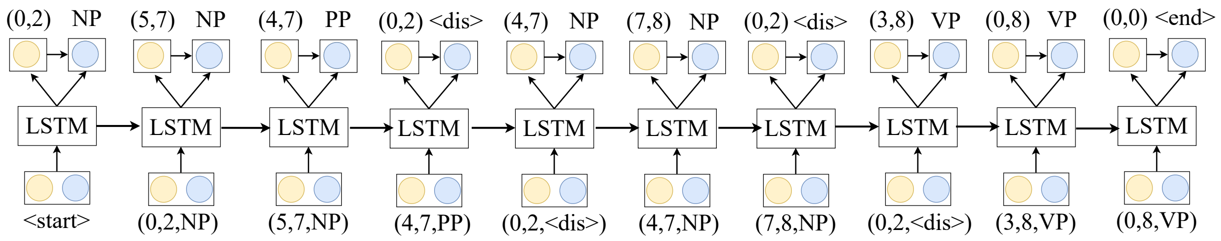

**Figure 2:** The depiction of the decoder architecture and how it outputs spans of the discontinuous parse tree from Fig. 1(b).

2019) to obtain *word-level* [6] contextualized embedding $x = x_1, \cdots, x_n$, then feed $x$ into a multi-layer bidirectional LSTM (BiLSTM):

$$\ldots, (\overrightarrow{f_i}, \overleftarrow{f_i}), \cdots = \text{BiLSTM}([\ldots, x_i, \ldots])$$

We use the fencepost representation (Cross and Huang, 2016; Stern et al., 2017) to represent the $i$-th boundary sitting between $x_i$ and $x_{i+1}$:

$$b_i = [\overrightarrow{f_i}; \overleftarrow{f_{i+1}}]$$

and obtain span embedding $e_{i,j}$ for $(i, j)$ using the concatenation of the LSTM-minus feature (Wang and Chang, 2016; Cross and Huang, 2016) and max-pooling over representations of all words in the span:

$$e_{i,j} = [b_j - b_i; \text{max-pooling}(x_{i+1}, ..., x_j)]$$

**Decoder.** For the decoder we use a unidirectional LSTM network,

$$d_1 = \text{LSTM}(d_0, e^{start})$$
$$d_t = \text{LSTM}(d_{t-1}, [\text{MLP}(e_{l_{t-1}, r_{t-1}}); E_{y_{t-1}}]), t \geq 2$$

where $d_t$ is the hidden state of the LSTM decoder at time step $t$; $d_0, e^{start}$ are randomly initialized trainable vectors; $E$ is the label embedding matrix. Fig. 2 demonstrates the working mechanism of the autoregressive decoder for outputting spans of the discontinuous parse tree from Fig. 1(b).

For each step, the decoder hidden state is used as a *query* to select the target spans and then the decoder hidden state and the selected span are used together to predict the label of the selected span. For span selection, we use deep triaffine attention (Zhang et al., 2020a) to estimate the score $s^t_{i,j}$ of selecting the span $(i, j)$ at time step $t$,

$$\begin{aligned} b_i^{l/r} &= \text{MLP}^{l/r}(b_i) \\ d'_t &= \text{MLP}^{decoder}(d_t) \\ s^t_{i,j} &= \text{TriAff}(b^l_i, b^r_j, d'_t), \end{aligned}$$

[6]For each word we take its last subtoken's representation from the last output layer of `BERT` or `XLNet`

For span labeling, we compute the label score $g^t \in \mathcal{R}^{\mathcal{L}}$ as,

$$\begin{aligned} H^t &= \text{MLP}^{\text{label}}([d_t; e_{l_{t-1}, r_{t-1}}]) \\ g^t &= H^t E^T \end{aligned}$$

where $\mathcal{L}$ is the size of label set, $E$ is the label embedding matrix. We remark that it is crucial to incorporate decoder state embedding into label prediction in discontinuous parsing. A span can be both a standalone continuous span and a continuous subspan of a discontinuous parent span (e.g, $(0, 2)$ and $(4, 7)$ in Fig. 1(b)) simultaneously. If we do not use the decoder state embedding to provide side information, it would be hard to distinguish such difference, thereby confusing the model.

## 2.3 Training and decoding

We decompose the training loss $L$ as the *span selection loss* and the *span labeling loss*,

$$L = L_{\text{select}} + L_{\text{labeling}}$$

$$L_{\text{select}} = -\sum_{t=1}^{m} \log \frac{\exp\{s^t_{l_t, r_t}\}}{\sum_{\substack{0 \leq i < j \leq n \\ \text{or} \quad i = j = 0}} \exp\{s^t_{i,j}\}}$$

$$L_{\text{labeling}} = -\sum_{t=1}^{m} \log \frac{\exp\{g^t_{y_t}\}}{\sum_{j=1}^{\mathcal{L}} \exp\{g^t_j\}}$$

For inference, we autoregressively decode spans with no constraint until the special span $(0, 0)$ is selected, and use the simple heuristic (introduced in Sect. 2.1) to build the final parse.

## 3 Experiments

### 3.1 Setting

**Data.** For continuous parsing, we conduct experiments on Penn Treebank (PTB, Marcus et al., 1993) 3.0 with the standard splits of section 02-21 for training, section 22 for development and section 23 for testing; and Chinese Treebank (CTB, Xue et al., 2005) 5.1 with the same split from (Zhang et al., 2020b). For multilingual experiments, we follow Cui et al. (2022) and select six languages from SPMRL 2013-2014 shared task (Seddah et al., 2013), including three rich-resource languages: French (fr), German (de), and Korean (ko), and three low-resource languages: Hungarian (hu), Basque (eu), and Polish (pl). For discontinuous parsing, we conduct experiments on discontinuous version of the English PTB (DPTB, Evang and Kallmeyer, 2011) using the standard split; German Negra treebank (Skut et al., 1997) using the split from Dubey and Keller (2003); and German Tiger treebank (Brants et al., 2001) using the split from SPRML 2014 shared task (Seddah et al., 2014).

**Evaluation.** We use the script `evalb`[7] and `discodop`[8] to evaluate continuous and discontinuous parsing respectively, reporting labeled F1 measures for continuous parsing; labeled overall F1 meansures and labeled discontinuous F1 measures (DF1) for discontinuous parsing. All reported values are averaged over three independent runs with different random seeds.

**Implementation details.** We refer readers to Appendix A for details.

## 3.2 Continuous parsing results

**Main result.** We re-implement the neural TreeCRF parser (Zhang et al., 2020b) — a strong span-based parser — as our baseline, using the same neural architectures (for scoring) and training settings for a fair comparison.

Table 1 shows the results on PTB test set. Our model outperforms the TreeCRF baseline and all other compared models. Interestingly, our model performs similarly to (Yang and Tu, 2022) (both their and our models have similar encoder-decoder neural architectures), suggesting that their bespoke decoding algorithm is not very useful in improving performance. We posit the improvement of their model mainly stems from the autoregressive decoder (sec. 3.4). Table 2 show the results on CTB test set and we can see our model outperforms the TreeCRF baseline and obtains competitive performance compared to other models.

**Multilingual evaluation.** We additionally report the results on multilingual SPMRL datasets following the settings from Cui et al. (2022). For rich-resourced languages our model clearly improves the performance, while for low-resourced languages our approach performs similarly to Cui et al. (2022). We posit that a learnable neural decoder is *data hungry*, and thus when the amount of training data is limited our model might not be advantageous.

[7]`https://nlp.cs.nyu.edu/evalb`

[8]`https://github.com/andreasvc/disco-dop`

| **Model** | **P** | **R** | **F** |
|---|---|---|---|
| w/ BERT$_{\text{large}}$ | | | |
| Kitaev et al. (2019) | 95.46 | 95.73 | 95.59 |
| Zhou and Zhao (2019)$^{\dagger}$ | 95.70 | 95.98 | 95.84 |
| Zhang et al. (2020b) | 95.85 | 95.53 | 95.69 |
| Nguyen et al. (2020) | - | - | 95.48 |
| Wei et al. (2020) | 95.5 | 96.1 | 95.8 |
| Tian et al. (2020) | 96.09 | 95.62 | 95.86 |
| Yang and Deng (2020) | 96.04 | 95.55 | 95.79 |
| Xin et al. (2021) | **96.29** | 95.55 | 95.92 |
| Nguyen et al. (2021) | - | - | 95.7 |
| Cui et al. (2022) | 95.70 | **96.14** | 95.92 |
| Yang and Tu (2022) | 96.19 | 95.83 | 96.01 |
| TreeCRF (re-impl.) | 96.03 | 95.63 | 95.83 |
| Ours | 96.21 | 95.87 | **96.04** |
| w/ XLNet$_{\text{large}}$ | | | |
| Zhou and Zhao (2019)$^{\dagger}$ | 96.21 | 96.46 | 96.33 |
| Mrini et al. (2020)$^{\dagger\star}$ | 96.24 | **96.53** | 96.38 |
| Tian et al. (2020) | **96.64** | 96.07 | 96.36 |
| Tian et al. (2020)$^{\star}$ | 96.61 | 96.19 | 96.40 |
| Yang and Deng (2020) | 96.55 | 96.13 | 96.34 |
| Ours | **96.64** | 96.34 | **96.48** |

**Table 1:** Results on PTB test set. $^{\dagger}$: using additional dependency parse tree information. $^{\star}$: using additional POS tag information.

## 3.3 Discontinuous parsing results

Table 4 shows the main results of discontinuous parsing.[9] Our model outperforms the recent parser Sun et al. (2022), obtaining state-of-the-art performances. Notably, our model's discontinuous F1 measures are significantly higher than all other approaches. This is surprising since our model is conceptually much more simpler, indicating that there is no need to design complex models for discontinuous parsing and warranting rethinking of this research field.

[9]The running speed is measured based on a single NVIDIA A40 GPU and each batch contains five thousands tokens

| Model | P | R | F |
|---|---|---|---|
| Kitaev et al. (2019) | 91.96 | 91.55 | 91.75 |
| Zhou and Zhao (2019)† | 92.03 | 92.33 | 92.18 |
| Zhang et al. (2020b) | 92.51 | 92.04 | 92.27 |
| Wei et al. (2020) | 91.9 | 92.3 | 92.1 |
| Mrini et al. (2020)† | 93.45 | 91.85 | 92.64 |
| Tian et al. (2020) | 92.50 | 91.98 | 92.20 |
| Yang and Deng (2020) | **93.80** | **93.40** | **93.59** |
| Xin et al. (2021) | 92.94 | 92.06 | 92.50 |
| Cui et al. (2022) | 92.45 | 92.17 | 92.31 |
| TreeCRF (re-impl.) | 92.27 | 92.06 | 92.17 |
| Ours | 92.83 | 91.97 | 92.41 |

**Table 2:** Results on CTB 5.1 test set. All models use the base version of BERT. †: using additional dependency parse tree information.

### 3.4 Ablation study

We conduct ablation studies on PTB (with BERT$_{\text{large}}$) and show the results on Table 5.

**The influence of the decoder.** As previously mentioned, span-based parsing amounts to non-autoregreesive span selection with independent assumptions. To show the importance of incorporating an autoregressive neural decoder, we train a span-based parser with local span binary classification loss (Teng and Zhang, 2018) and perform greedy decoding as used in our approach. We find it hurts the performance significantly, verifying the importance of neural decoder when adopting greedy decoding. We further use the CKY algorithm for decoding with the implicit binarization (Stern et al., 2017; Kitaev and Klein, 2018) strategy[10] instead of using greedy decoding and observe quite marginal improvement, which aligns with the experimental results of (Fig. 1, Zhang et al., 2019) in dependency parsing, questioning the utility of global decoding, while we indeed observe that global TreeCRF training (Zhang et al., 2020b) improves the performance compared to local training (but still underperforms our model). We believe this is because TreeCRF modeling weakens the strong independence assumption by introducing a global factor connecting all possible spans to enforce hard tree constraint (Smith and Eisner, 2008). However we are not aware of any greedy (or local) decoding strategies for globally-trained TreeCRF parser, thus cannot investigate the influence of global decoding here.

**The effect of span selection order.** It is also possible to use other (span) sorting orders e.g. the pre-order tree traversal order as used in (Nguyen et al., 2021). We find using post-order indeed performs better than pre-order, which might explain the performance gap between (Nguyen et al., 2021) and (Yang and Tu, 2022) on PTB. We propose that this phenomenon can be attributed to the order in which spans are generated. In general, shorter spans are more easily predictable. Post-order generation prioritizes the generation of child spans before parent spans, whereas pre-order generation generates parent spans before child spans. As a result, pre-order generation is more prone to the issue of error propagation.

## 4 Related work

**Continuous parsing.** Continuous parsing is well-studied in the literature. Span-based parsing is the most popular paradigm and achieves great success. The main issue of span-based parsing stems from the unreasonable conditional independence assumptions imposed in local span scoring. To resolve this issue, one line of work aims to enhance span representation learning. Mrini et al. (2020) propose a *Label Attention Layer* to bring label information into encoding; Tian et al. (2020) incorporate n-gram features and enhance self-attention mechanism to learn better span representation; and Cui et al. (2022) introduce auxiliary tasks to help the model learn non-local patterns. Another line of work focuses on improving output modeling (Zhang et al., 2019) by wearkening independence assumptions. Zhang et al. (2020b) propose a span-based TreeCRF parser, which can be viewed as introducing a global (hard tree constraint) factor (Smith and Eisner, 2008) to connect all spans and thereby reduce independence assumptions. Xin et al. (2021) weaken the independence assumption by taking into accounts the correlations of sibling spans under a common parent span, designing an $\mathcal{O}(n^4)$ dynamic programming algorithm to incorporate the interaction scores between sibling spans into global decoding. Our model weakens the independence assumption by using a neural autoregressive decoder so that spans are decoded conditioned on all previously decoded spans.

**Discontinuous parsing.** Discontinuous parsing is a much more complex problem than continuous

[10]It is also possible to use the algorithm from (Section 4.2, Corro, 2022) for global non-binary parsing

| Model | Rich resource | | | | Low Resource | | | | Avg |
|---|---|---|---|---|---|---|---|---|---|
| | fr | de | ko | Avg | hu | eu | pl | Avg | |
| Kitaev and Klein (2018) | 87.42 | 90.20 | 88.80 | 88.81 | 94.90 | 91.63 | **96.36** | 94.30 | 91.55 |
| Nguyen et al. (2020) | 86.69 | 90.28 | 88.71 | 88.56 | 94.24 | **92.02** | 96.14 | 94.13 | 91.34 |
| Cui et al. (2022) | 87.51 | 90.43 | 89.07 | 89.00 | 94.95 | 91.73 | 96.33 | **94.34** | 91.67 |
| Ours | **87.89** | **91.07** | **89.31** | **89.41** | **95.06** | 91.72 | 96.18 | 94.32 | **91.87** |

**Table 3:** Results on SPMRL test sets. We use ISO 639-1 codes to represent languages. All models use the base version of multilingual BERT as the encoders.

| Parser (no tags or predicted PoS tags) | TIGER | | | NEGRA | | | DPTB | | |
|---|---|---|---|---|---|---|---|---|---|
| | F1 | DF1 | sent/s | F1 | DF1 | sent/s | F1 | DF1 | sent/s |
| w/o pre-trained language models | | | | | | | | | |
| Coavoux and Cohen (2019) | 82.5 | 55.9 | 64 | 83.2 | 56.3 | - | 90.9 | 67.3 | 38 |
| Coavoux et al. (2019) | 82.7 | 55.9 | 126 | 83.2 | 54.6 | - | 91.0 | 71.3 | 80 |
| Stanojević and Steedman (2020) | 83.4 | 53.5 | - | 83.6 | 50.7 | - | 90.5 | 67.1 | - |
| Corro (2020) | 85.2 | 51.2 | 474 | 86.3 | 56.1 | 478 | 92.9 | 64.9 | 355 |
| Ruprecht and Mörbitz (2021) | 82.5 | 55.9 | 101 | 82.7 | 49.0 | 136 | 90.1 | 72.9 | 95 |
| Vilares and Gómez-Rodríguez (2020) | 77.5 | 39.5 | **568** | 75.6 | 34.6 | **715** | 88.8 | 45.8 | **611** |
| w/ pre-trained language models | | | | | | | | | |
| Corro (2020) + BERT$_{\text{base}}$ | 90.0 | 62.1 | - | 91.6 | 66.1 | - | 94.8 | 68.9 | - |
| RM21 + BERT$_{\text{base}}$ | 88.3 | 69.0 | 60 | 90.9 | 72.6 | 68 | 93.3 | 80.5 | 57 |
| Coavoux (2021) + BERT$_{\text{base}}$ | 90.2 | 72.9 | - | 91.7 | 73.3 | - | 95.0 | 82.5 | - |
| VG20 + BERT$_{\text{base}}$ | 84.6 | 51.1 | 80 | 83.9 | 45.6 | 80 | 91.9 | 50.8 | 80 |
| VG20 + BERT$_{\text{large}}$ | - | - | - | - | - | - | 92.8 | 53.9 | 34 |
| FG21 + BERT$_{\text{base}}$ | 88.5 | 63.0 | 238 | 90.0 | 65.9 | 275 | 94.0 | 68.9 | 231 |
| FG21 + BERT$_{\text{large}}$ | 90.5 | 68.1 | 207 | 92.0 | 67.9 | 216 | 94.7 | 72.9 | 193 |
| FG21 + XLNet$_{\text{large}}$ | - | - | - | - | - | - | 95.1 | 74.1 | 179 |
| FG22 + BERT$_{\text{base}}$ | 89.8 | 71.0 | - | 91.0 | 76.6 | - | - | - | - |
| FG23 + GottBERT$_{\text{base}}$/RoBERTa$_{\text{large}}$ | 88.53 | 67.76 | - | 89.08 | 67.06 | - | 95.47 | 83.80 | - |
| Sun et al. (2022) + BERT$_{\text{large}}$ | 91.86 | 74.24 | 185 | 93.67 | 77.18 | 193 | 95.76 | 79.83 | 153 |
| Sun et al. (2022) + XLNet$_{\text{large}}$ | - | - | - | - | - | - | 95.82 | 81.72 | 101 |
| Ours + BERT$_{\text{large}}$ | **92.26** | **76.91** | 192 | **93.79** | **78.81** | 200 | 95.71 | 86.65 | 161 |
| Ours + XLNet$_{\text{large}}$ | - | - | - | - | - | - | **96.10** | **87.05** | 110 |

**Table 4:** Results on test sets of discontinuous constituency parsing benchmarks. VG20: Vilares and Gómez-Rodríguez (2020). FG21: Fernández-González and Gómez-Rodríguez (2021). FG22: Fernández-González and Gómez-Rodríguez (2022) FG23: Fernández-González and Gómez-Rodríguez (2023). RM21: Ruprecht and Mörbitz (2021).

| Model | F1 |
|---|---|
| Ours | 96.05 |
| TreeCRF (reimp.) | 95.83 |
| w/o neural decoder w/ greedy | 95.58 |
| w/o neural decoder w/ CKY | 95.61 |
| w/o post-order w/ pre-order | 95.87 |

**Table 5:** Ablation studies on PTB test set.

parsing due to the much larger search space of discontinuous parse trees. Reducing discontinuous parsing to simpler problems such as nonprojective dependency parsing (Fernández-González and Martins, 2015; Corro et al., 2017; Fernández-González and Gómez-Rodríguez, 2020) and continuous constituency parsing (Versley, 2016; Fernández-González and Gómez-Rodríguez, 2021; Sun et al., 2022) is a viable way to decrease the complexity of discontinuous parsing. This work is closely related to the line of work of reduction to continuous constituency parsing. Boyd (2007) and Versley

(2016) convert the original discontinuous treebanks into continuous ones, using special nonterminal nodes to encode discontinuous constituents similar to pseudo-projective parsing (Nivre and Nilsson, 2005). After treebank transformation, any continuous constituency parsers can be used for training and decoding, and post-processing is needed to recover discontinuous trees. In another line of work that does not involve treebank transformation, researchers reorder words in the sentence so that a discontinuous tree becomes continuous, and then they train off-the-shelf constituency parsers on the resulting reordered sentences. This strategy is referred to as *reorder-then-parse*. Research in this line mainly focuses on the design of the reordering module. Fernández-González and Gómez-Rodríguez (2021) use a pointer network to do the reordering, while Sun et al. (2022) use a dependency graph parsing strategy to improve the reordering accuracy, which they show is crucial to the final parsing performance. In this work, we view a discontinuous span as a combination of multiple continuous subspans and use a simple labeling mechanism together with the autoregressive decoder to *connect* all (nonadjacent) subspans. Our approach is conceptually simpler than previous models, not involving any treebank transformation or reordering and eliminating the need for off-the-shelf continuous constituency parsers.

**Constituency parsing with pointer nets.** Our model has a very similar neural architecture compared to prior pointer network-based constituency parsers: Nguyen et al. (2021) and Yang and Tu (2022). The major difference is that our model directly points to *spans* using deep triaffine attentions, while their models point to span *boundaries* using deep biaffine attentions so that spans are *implicitly* built. Therefore, their models are *highly coupled* with bespoke decoding algorithms: Nguyen et al. (2021) decode spans with pre-order tree traversal by splitting each input parent span into two subspans based on the selected splitting point (i.e., boundary); Yang and Tu (2022) decode spans with post-order tree traversal, and since every two consecutively visited spans share a boundary, their method selects the unshared boundary for each new span. We believe bespoke decoding algorithms are not the keys for obtaining high parsing accuracy and show a much simpler greedy decoding strategy instead. Moreover, both of their models are hard to be extended to discontinuous parsing. [11] This is because their bespoke decoding algorithms leverage *locality properties* of continuous parse trees, which does not hold for discontinuous parse trees due to the cross dependencies of discontinuous spans. In contrast, our model is very flexible in decoding spans and can be easily extended to discontinuous parsing.

**Seq2seq constituency parsing.** Our work is also closely connected to the recent work of (Fernández-González and Gómez-Rodríguez, 2023), who propose a unified seq2seq method for both continuous and discontinuous parsing like ours. The main difference is that they linearize (discontinuous) constituency parse trees to *transition action sequences* (e.g., SHIFT, REDUCE). As such, they rely on transition systems to convert a decoded action sequence to a parse tree. In contrast, we directly linearize trees to span sequences and leverage pointer networks to decode a span for each step .

## 5 Conclusion

In this work, we have presented a simple yet effective method for both continuous and discontinuous constituency parsing. We showed that an autoregressive decoder is more desirable than global CKY decoding in span-based continuous parsing, and with a simple labeling mechanism, we obtained state-of-the-art performance in discontinuous parsing.

## Limitations

Though autoregressive span selection effectively weakens the conditional independence assumptions imposed by current span-based parsing methods, this strategy imposes another arguably unreasonable inductive bias of forcing a predefined span selection order. We find using post order performs fairly well but this does not necessarily means it is the best order for span selection, and this work might leave other potentially better order unexplored. Future works might consider using *set* prediction methods (Sui et al., 2020; Tan et al., 2021)

[11]For instance, if we want to extend the model of Nguyen et al. (2021) to discontinuous parsing (assuming binarization), the number of splitting points is nondeterministic which can be difficult to handle: a continuous parent span could be split into two continuous subspans with one splitting point, or one continuous subspan and one discontinuous subspan with two splitting points, or two discontinuous subspans with at least three splitting points (depending on the fan-out of subspans). The cases of discontinuous parent spans are even more complicated.

to eliminate such implausible inductive bias (of forcing orders) while still benefiting from explicit span correlation modeling.

Another issue is regarding time complexity. Though our model needs only linear steps (in sentence length) for parsing, each step takes $\mathcal{O}(n^2)$ time to select a single span over $\mathcal{O}(n^2)$ total spans, making the overall time complexity cubic. We remark that for each step the $\mathcal{O}(n^2)$ operation is parallelizable and—with full GPU parallelization—fairly fast in practice, but it would still be problematic when the sentence is extremely long.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China (61976139).

## References

Nader Akoury, Kalpesh Krishna, and Mohit Iyyer. 2019. Syntactically supervised transformers for faster neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1269–1281, Florence, Italy. Association for Computational Linguistics.

Adriane Boyd. 2007. Discontinuity revisited: An improved conversion to context-free representations. In *Proceedings of the Linguistic Annotation Workshop*, pages 41–44, Prague, Czech Republic. Association for Computational Linguistics.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2001. The tiger treebank.

Harry Bunt, Jan Thesingh, and Ko van der Sloot. 1987. Discontinuous constituents in trees, rules, and parsing. In *Third Conference of the European Chapter of the Association for Computational Linguistics*, Copenhagen, Denmark. Association for Computational Linguistics.

Maximin Coavoux. 2021. BERT-proof syntactic structures: Investigating errors in discontinuous constituency parsing. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3259–3272, Online. Association for Computational Linguistics.

Maximin Coavoux and Shay B. Cohen. 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: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 204–217, Minneapolis, Minnesota. Association for Computational Linguistics.

Maximin Coavoux, Benoît Crabbé, and Shay B. Cohen. 2019. Unlexicalized transition-based discontinuous constituency parsing. *Transactions of the Association for Computational Linguistics*, 7:73–89.

Caio Corro. 2020. Span-based discontinuous constituency parsing: a family of exact chart-based algorithms with time complexities from O(n^6) down to O(n^3). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2753–2764, Online. Association for Computational Linguistics.

Caio Corro. 2022. A dynamic programming algorithm for span-based nested named-entity recognition in $o(n^2)$. *ArXiv*, abs/2210.04738.

Caio Corro, Joseph Le Roux, and Mathieu Lacroix. 2017. Efficient discontinuous phrase-structure parsing via the generalized maximum spanning arborescence. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1644–1654, Copenhagen, Denmark. Association for Computational Linguistics.

James Cross and Liang Huang. 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*, pages 1–11, Austin, Texas. Association for Computational Linguistics.

Leyang Cui, Sen Yang, and Yue Zhang. 2022. Investigating non-local features for neural constituency parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2065–2075, Dublin, Ireland. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Amit Dubey and Frank Keller. 2003. Probabilistic parsing for German using sister-head dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 96–103, Sapporo, Japan. Association for Computational Linguistics.

Kilian Evang and Laura Kallmeyer. 2011. PLCFRS parsing of English discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 104–116, Dublin, Ireland. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Discontinuous constituent parsing with pointer networks. In *The Thirty-Fourth AAAI*

*Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7724–7731. AAAI Press.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2021. Reducing discontinuous to continuous parsing with pointer network reordering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10570–10578, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2022. Multitask pointer network for multi-representational parsing. *Knowledge-Based Systems*, 236:107760.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2023. Discontinuous grammar as a foreign language. *Neurocomputing*, 524:43–58.

Daniel Fernández-González and André F. T. Martins. 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)*, pages 1523–1533, Beijing, China. Association for Computational Linguistics.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

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

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

James D. McCawley. 1982. Parentheticals and discontinuous constituent structure.

Khalil Mrini, Franck Dernoncourt, Quan Hung Tran, Trung Bui, Walter Chang, and Ndapa Nakashole. 2020. Rethinking self-attention: Towards interpretability in neural parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 731–742, Online. Association for Computational Linguistics.

Stefan Müller. 2004. Continuous or discontinuous constituents? a comparison between syntactic analyses for constituent order and their processing systems. *Research on Language and Computation*, 2:209–257.

Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, and Xiaoli Li. 2020. Efficient constituency parsing by pointing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3284–3294, Online. Association for Computational Linguistics.

Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, and Xiaoli Li. 2021. A conditional splitting framework for efficient constituency parsing. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5795–5807, Online. Association for Computational Linguistics.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan. Association for Computational Linguistics.

Thomas Ruprecht and Richard Mörbitz. 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*, pages 2923–2935, Online. Association for Computational Linguistics.

Alexander Rush. 2020. Torch-struct: Deep structured prediction library. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 335–342, Online. Association for Computational Linguistics.

Laurent Sartran, Samuel Barrett, Adhiguna Kuncoro, Milos Stanojevic, Phil Blunsom, and Chris Dyer. 2022. Transformer grammars: Augmenting transformer language models with syntactic inductive biases at scale. *CoRR*, abs/2203.00633.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin, Ireland. Dublin City University.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. 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*, pages 146–182, Seattle, Washington, USA. Association for Computational Linguistics.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Fifth Conference on Applied Natural Language Processing*, pages 88–95, Washington, DC, USA. Association for Computational Linguistics.

David Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 145–156, Honolulu, Hawaii. Association for Computational Linguistics.

Miloš Stanojević and Mark Steedman. 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*, pages 111–121, Online. Association for Computational Linguistics.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada. Association for Computational Linguistics.

Dianbo Sui, Yubo Chen, Kang Liu, Jun Zhao, Xiangrong Zeng, and Shengping Liu. 2020. Joint entity and relation extraction with set prediction networks. *ArXiv*, abs/2011.01675.

Kailai Sun, Zuchao Li, and Hai Zhao. 2022. Reorder and then parse, fast and accurate discontinuous constituency parsing. *EMNLP*.

Zeqi Tan, Yongliang Shen, Shuai Zhang, Weiming Lu, and Yueting Zhuang. 2021. A sequence-to-set network for nested named entity recognition. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 3936–3942. ijcai.org.

Zhiyang Teng and Yue Zhang. 2018. Two local models for neural constituent parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 119–132, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Yuanhe Tian, Yan Song, Fei Xia, and Tong Zhang. 2020. Improving constituency parsing with span attention. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1691–1703, Online. Association for Computational Linguistics.

Yannick Versley. 2016. Discontinuity (re)$^2$-visited: A minimalist approach to pseudoprojective constituent parsing. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 58–69, San Diego, California. Association for Computational Linguistics.

David Vilares and Carlos Gómez-Rodríguez. 2020. Discontinuous constituent parsing as sequence labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2771–2785, Online. Association for Computational Linguistics.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2306–2315, Berlin, Germany. Association for Computational Linguistics.

Yang Wei, Yuanbin Wu, and Man Lan. 2020. A span-based linearization for constituent trees. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3267–3277, Online. Association for Computational Linguistics.

Xin Xin, Jinlong Li, and Zeqi Tan. 2021. N-ary constituent tree parsing with recursive semi-Markov model. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2631–2642, Online. Association for Computational Linguistics.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Martha Palmer. 2005. The penn chinese treebank: Phrase structure annotation of a large corpus. *Nat. Lang. Eng.*, 11(2):207–238.

Kaiyu Yang and Jia Deng. 2020. Strongly incremental constituency parsing with graph neural networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Songlin Yang and Kewei Tu. 2022. Bottom-up constituency parsing and nested named entity recognition with pointer networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2403–2416, Dublin, Ireland. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020a. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3295–3305, Online. Association for Computational Linguistics.

Yu Zhang, Houquan Zhou, and Zhenghua Li. 2020b. Fast and accurate neural CRF constituency parsing. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4046–4053. ijcai.org.

Zhisong Zhang, Xuezhe Ma, and Eduard Hovy. 2019. An empirical investigation of structured output modeling for graph-based neural dependency parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5592–5598, Florence, Italy. Association for Computational Linguistics.

Junru Zhou and Hai Zhao. 2019. Head-Driven Phrase Structure Grammar parsing on Penn Treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408, Florence, Italy. Association for Computational Linguistics.

## A Implementation details

**BERT versions** We use `bert-large-cased` and `xlnet-large-cased` for PTB and DPTB; `bert-base-chinese` for CTB; `bert-base-multilingual-cased` for SPMRL; `deepset/gbert-large` for NeGra and Tiger.

We tune the number of warmup and total training epochs—which influence performance the most—separately for each dataset, leaving all other hyperparameters (almost) the same. Candidate epoch numbers are $\{10, 15, 20, 30, 40, 50, 100\}$ and warmup rates (i.e., the ratio of number of warmup epochs against total epochs) are $\{0.1, 0.3, 0.5\}$ and we use random search to select hyperparameters. We use $(10, 0.1)$ for PTB; $(15, 0.1)$ for CTB, Tiger, and rich-sourced languages in SPMRL; $(20, 0.3)$ for DPTB; $(40, 0.5)$ for Negra; $(100, 0.5)$ for low-resourced languages in SPMRL.

The hyperparameters are summarized in Table 6. Additionally, for the experiments on PTB (XLNet) and SPRML, we do not use max-pooling as part of span representations, using only the LSTM-minus feature, which is slightly better. For CTB the hidden size of Triaffine layer is set to 300 instead of 500, which stabilizes training.

| **Neural architecture** | |
|---|---|
| Embeddings dropout | 0.33 |
| BiLSTM encoder layers | 3 |
| BiLSTM encoder size | 1000 |
| BiLSTM layers dropout | 0.33 |
| MLP layers | 1 |
| MLP activation function | LeakyReLU |
| MLP layers dropout | 0.33 |
| MLP size (span) | 2000 |
| MLP size (label) | 500 |
| MLP size (triaffine) | 500 |
| **Training** | |
| Learning rate for BERT | 5e-5 |
| Learning rate for others | 2.5e-3 |
| Optimizer | AdamW |
| Scheduler | linear warmup |
| Gradient clipping | 5.0 |
| Tokens per batch | 3000 |
| Maximum training sentence length | 200 |

**Table 6:** Summary of hyper-parameters.

## ACL 2023 Responsible NLP Checklist

**A For every submission:**

☑ A1. Did you describe the limitations of your work?
*after conclusion selection, page 9*

☒ A2. Did you discuss any potential risks of your work?
*we do not see any ethical implications or risks of our work*

☑ A3. Do the abstract and introduction summarize the paper's main claims?
*abstract and section 1*

☒ A4. Have you used AI writing assistants when working on this paper?
*Left blank.*

**B ☑ Did you use or create scientific artifacts?**

*section 3*

☑ B1. Did you cite the creators of artifacts you used?
*section 3.1*

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
*Not applicable. Left blank.*

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
*Not applicable. Left blank.*

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
*Not applicable. Left blank.*

☒ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
*Left blank.*

☒ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
*Left blank.*

**C ☑ Did you run computational experiments?**

*section 3*

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
*section 3.2 shows the discontinuous parsing speed*

*The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.*

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
*section3 and appendix A*

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
*section 3*

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
*section 3.1*

**D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?**
*Left blank.*

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
*Not applicable. Left blank.*

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
*Not applicable. Left blank.*

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
*Not applicable. Left blank.*

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
*Not applicable. Left blank.*

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
*Not applicable. Left blank.*