Reducing Discontinuous to Continuous Parsing with Pointer Network Reordering

Daniel Fernández-González and **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, 15071 A Coruña, Spain

d.fgonzalez@udc.es, carlos.gomez@udc.es

Abstract

Discontinuous constituent parsers have always lagged behind continuous approaches in terms of accuracy and speed, as the presence of constituents with discontinuous yield introduces extra complexity to the task. However, a discontinuous tree can be converted into a continuous variant by reordering tokens. Based on that, we propose to reduce discontinuous parsing to a continuous problem, which can then be directly solved by any off-the-shelf continuous parser. To that end, we develop a Pointer Network capable of accurately generating the continuous token arrangement for a given input sentence and define a bijective function to recover the original order. Experiments on the main benchmarks with two continuous parsers prove that our approach is on par in accuracy with purely discontinuous state-of-the-art algorithms, but considerably faster.

1 Introduction

Discontinuous phrase-structure trees (with crossing branches like the one in Figure 1(a)) are crucial for fully representing the wide range of syntactic phenomena present in human languages such as long-distance extractions, dislocations or cross-serial dependencies, among others.

Although continuous approaches ignore these linguistic phenomena by, for instance, removing them from the original treebank (a common practice in the Penn Treebank (Marcus et al., 1993)), there exist different algorithms that can handle discontinuous parsing. Currently, we can highlight (1) those based in Linear Context-Free Rewriting Systems (LCFRS) (Vijay-Shanker et al., 1987), which allow exact CKY-style parsing of discontinuous structures at a high computational cost (Gebhardt, 2020; Ruprecht and Mörbitz, 2021); (2) a variant of the former that, while still making use of LCFRS formalisms, increases parsing speed by implementing a span-based scoring algorithm (Stern et al., 2017) and not explicitly defining a set of rules

(Stanojević and Steedman, 2020; Corro, 2020); (3) transition-based parsers that deal with discontinuities by adding a specific transition in charge of changing token order (Versley, 2014; Maier, 2015; Maier and Lichte, 2016; Stanojević and Alhama, 2017; Coavoux and Crabbé, 2017) or by designing new data structures that allow interactions between already-created non-adjacent subtrees (Coavoux et al., 2019; Coavoux and Cohen, 2019); and, finally, (4) several approaches that reduce discontinuous constituent parsing to a simpler problem, converting it, for instance, into a non-projective dependency parsing task (Fernández-González and Martins, 2015; Fernández-González and Gómez-Rodríguez, 2020a) or into a sequence labelling problem (Vilares and Gómez-Rodríguez, 2020). In (4), we can also include the solutions proposed by Boyd (2007) and Versley (2016), which transform discontinuous treebanks into continuous variants where discontinuous constituents are encoded by creating additional constituent nodes and extending the original non-terminal label set (following a pseudo-projective technique (Nivre and Nilsson, 2005)), to then be processed by continuous parsing models and discontinuities recovered in a post-processing step.

It is well known that discontinuities are inherently related to the order of tokens in the sentence, and a discontinuous tree can be transformed into a continuous one by just reordering the words and without including additional structures, an idea that has been exploited in practically all transition-based parsers and other approaches (Vilares and Gómez-Rodríguez, 2020). However, in these models the reordering process is tightly integrated and inseparable from the parsing process.

Likely due to the lack of accurate models to accomplish reordering in isolation, we are not aware of any approach framed in (4) that explicitly reduces discontinuous constituent parsing into a continuous problem, keeping the original set of con-
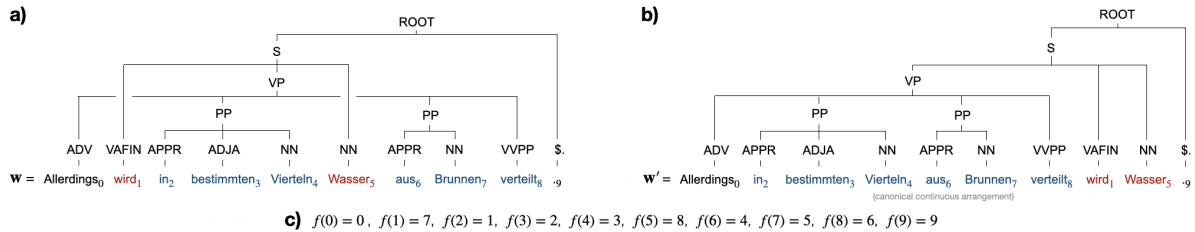
Figure 1: **a)** Discontinuous constituent tree for a sentence in German NEGRA development split, **b)** its canonical continuous arrangement and **c)** conversion of original positions into absolute CCA positions through function f implemented by a Pointer Network.

stituent nodes and solving it with a completely independent continuous parser that does not have to deal with an extended label set. Please note that existing approaches that perform discontinuous-to-continuous conversion, such as (Boyd, 2007) and (Versley, 2016), not only modify the original discontinuous tree by including artificial constituent nodes and enlarging its label scheme (probably penalizing parsing performance), but they are not able to fully recover the original discontinuous tree due to limitations of the proposed encodings.

In this paper, we study the (fully reversible) discontinuous-to-continuous conversion by token reordering and how any off-the-shelf continuous parser can be directly applied without any further adaptation or extended label set. To undertake the independent token reordering, we rely on a Pointer Network architecture (Vinyals et al., 2015) that can accurately relocate those tokens causing discontinuities in the sentence to new positions, generating new sentences that can be directly parsed by any continuous parser. We test our approach [1] with two continuous algorithms (Kitaev et al., 2019; Yang and Deng, 2020) on three widely-used discontinuous treebanks, obtaining remarkable accuracies and outperforming current state-of-the-art discontinuous parsers in terms of speed.

2 Pointer Network Reordering

2.1 Continuous Canonical Arrangement

Let $\mathbf{w} = w_0, \ldots, w_{n-1}$ be an input sentence of n tokens, and t a discontinuous constituent tree for $\mathbf{w}$. We are interested in a permutation (reordering) $\mathbf{w}'$ of $\mathbf{w}$ that turns t into a continuous tree t'. While there can be various permutations that achieve this for a given tree, we will call *continuous canonical arrangement (CCA)* of $\mathbf{w}$ and t the permutation obtained by placing the tokens of $\mathbf{w}$ in the order

given by an in-order traversal of t.

This permutation defines a bijective function, $f : \{0, \ldots, n-1\} \rightarrow \{0, \ldots, n-1\}$, mapping each token at position i in $\mathbf{w}$ to its new *CCA position* j in $\mathbf{w}'$. Then, $\mathbf{w}'$ can be parsed by a continuous parser and, by keeping track of f (i.e., storing original token positions), it is trivial to recover the discontinuous tree by applying its inverse f^{-1}. The challenge is in accurately predicting the CCA positions for a given sentence $\mathbf{w}$ (i.e. learning f) *without* knowing the parse tree t, a complex task that will have a large impact on discontinous parsing performance, as observed by e.g. Vilares and Gómez-Rodríguez (2020), who recently dealt with reordering to extend their sequence-tagging encoding for discontinuous parsing.

In Figure 1, we depict how a discontinuous tree (a) is converted into a continuous variant (b) by applying function f to map each original position to its corresponding CCA position (c).

2.2 Pointer Networks

To implement function f and accurately obtain the CCA positions for each token, we rely on *Pointer Networks* (Vinyals et al., 2015). This neural architecture was developed to, given an input sequence, output a sequence of discrete numbers that correspond to positions from the input. Unlike regular sequence-to-sequence models that use the same dictionary of output labels for the whole training dataset, Pointer Networks employ an attention mechanism (Bahdanau et al., 2014) to select positions from the input, so they can handle as many labels as the length of each sentence instead of having a fixed output dictionary size.

For our purpose, the input sequence will be $\mathbf{w}$ and the output sequence, the absolute CCA positions (i.e., positions j in $\mathbf{w}'$). Additionally, we keep track of already-assigned CCA positions and extend the Pointer Network with the *uniqueness constraint*: once a CCA position is assigned to an

[1] Source code available at https://github.com/danifg/Pointer-Network-Reordering.

input token, it is no longer available for the rest of the sentence. As a consequence, the Pointer Network will just need n-1 steps to relocate each token of the original sentence from left to right, assigning to the last token the remaining CCA position.

Although the overall performance of the pointer is high enough, we note that the specific accuracy on tokens affected by discontinuities is substantially lower. This was expected due to the complexity of the task and can be explained by the fact that these kind of tokens are less frequent in the training dataset and, in languages such as English, the amount of discontinuous sentences is scarce, not providing enough examples to adequately train the pointer. To increase the pointer performance, we decided to jointly train a labeller in charge of identifying those tokens. More specifically, we consider that a token is involved in a discontinuity if its original position i differs from the CCA position j. This is regardless of whether the token is part of a discontinuous constituent or not, e.g., in Figure 1 it includes both the tokens in blue (that move left) and those in red (that move right). The idea behind this strategy is to prefer those models that better relocate tokens that change its absolute position in the resulting CCA.

While it can be argued that directly handling absolute CCA positions might underperform approaches that use relative positions instead (as reported by Vilares and Gómez-Rodríguez (2020)), we already explored that strategy and found that the use of relative CCA positions yielded worse accuracy in a Pointer Network framework. This can be mainly explained by the fact that we cannot apply the uniqueness constraint when relative positions are used, not reducing the search space while the sentence processing advances. Moreover, in regular sequence-to-sequence approaches, the use of relative positions leads to a lower size of the output dictionary, but this benefit has no impact in Pointer Networks since the size of the dictionary will always be the sentence length.

2.3 Neural Architecture

Following other pointer-network-based models (Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019), we design a specific neural architecture for our problem:

Encoder Each input sentence $\mathbf{w}$ is encoded, token by token, by a BiLSTM-CNN architecture (Ma and Hovy, 2016) into a sequence of *encoder hidden states* $\mathbf{h}_0, \ldots, \mathbf{h}_{n-1}$. To that end, each input token is initially represented as the concatenation of three different vectors obtained from character-level representations, regular pre-trained word embeddings and fixed contextualized word embeddings extracted from the pre-trained language model BERT (Devlin et al., 2019).

Decoder An LSTM is used to model the decoding process. At each time step t, the decoder is fed the encoder hidden state $\mathbf{h}_i$ of the current token w_i to be relocated and generates a *decoder hidden state* $\mathbf{s}_t$ that will be used for computing the probability distribution over all available CCA positions from the input (i.e., $j \in [0, n-1] \setminus A$, with A being the set of already-assigned CCA positions). A biaffine scoring function (Dozat and Manning, 2017) is used for computing this probability distribution that will implement the attention mechanism:

$$\mathbf{v}_{tj} = \mathbf{score}(\mathbf{s}_t, \mathbf{h}_j) = g_1(\mathbf{s}_t)^T \mathbf{W} g_2(\mathbf{h}_j)$$
$$+ \mathbf{U}^T g_1(\mathbf{s}_t) + \mathbf{V}^T g_2(\mathbf{h}_j) + \mathbf{b};$$
$$\mathbf{a}_t = \mathbf{softmax}(\mathbf{v}_t)$$

where $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ are the weights and $g_1(\cdot)$ and $g_2(\cdot)$ are multilayer perceptrons (MLP).

The attention vector $\mathbf{a}_t$ is then used as a pointer that, at time step t, will select the highest-scoring position j as the new CCA position for the token originally located at i.

The Pointer Network is trained by minimizing the total log loss (cross entropy) to choose the correct sequence of CCA positions. Additionally, a binary biaffine classifier (Dozat and Manning, 2017) that identifies relocated tokens is jointly trained by summing the pointer and labeller losses. Since the decoding process requires $n - 1$ steps to assign the CCA position to each token and at each step the attention vector $\mathbf{a}_t$ is computed over the whole input, the proposed neural model can process a sentence in $O(n^2)$ time complexity. Figure 2 depicts the neural architecture and the decoding procedure for reordering the sentence in Figure 1(a).

3 Experiments

3.1 Setup

Data We test our approach on two German discontinuous treebanks, NEGRA (Skut et al., 1997) and TIGER (Brants et al., 2002), and the discontinuous English Penn Treebank (DPTB) (Evang and Kallmeyer, 2011) with standard splits as described in Appendix A.1, discarding PoS tags in all
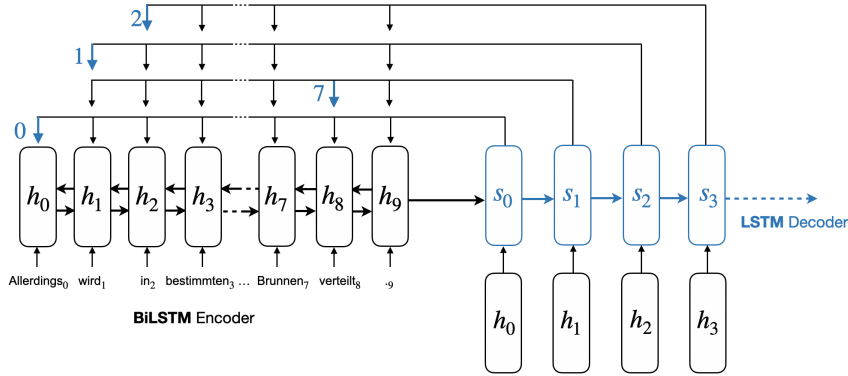
Figure 2: Simplified sketch of the Pointer Network architecture and decoding steps to reorder the sentence in Figure 1(a).

	label.	UAS	Rec.	Prec.	F1	%gold	%pred
TIGER	no	94.16	76.11	**76.20**	76.15	17.65	17.63
	yes	**94.19**	**77.66**	76.14	**76.89**	17.65	18.00
NEGRA	no	94.56	79.44	80.20	79.82	18.73	18.55
	yes	**94.82**	**81.58**	**80.21**	**80.89**	18.73	19.05
DPTB	no	97.69	78.63	77.56	78.09	7.25	7.35
	yes	**97.88**	**82.20**	**78.40**	**80.26**	7.25	7.61

Table 1: Pointer performance with and without the labeller on dev splits. **label.** = using labeller or not. **UAS** = Unlabelled Attachment Score. **Rec.**, **Prec.** and **F1** = Recall, precision and F-score on relocated tokens. **%gold** = % of relocated tokens on the gold set. **%pred** = % of relocated tokens predicted by the pointer.

cases. We apply `discodop`[2] (van Cranenburgh et al., 2016) to transform them into continuous treebanks. This tool follows a depth-first in-order traversal that reorders words to remove crossing branches. For all treebanks, we convert discontinuous trees in `export` format into continuous variants in `discbracket` format, using the resulting word permutation as CCAs for training the pointer and keeping track of the original word order for implementing the inverse function f^{-1}. Additionally, the resulting continuous treebanks in `discbracket` format are also converted by `discodop` into the commonly-used `bracket` format for training continuous parsers.

Pointer settings Word vectors are initialized with a concatenation of pre-trained structured-skipgram embeddings (Ling et al., 2015) and fixed weights extracted from one or several layers of the BASE and LARGE sizes of the pre-trained language model BERT (Devlin et al., 2019). In particular, we follow (Fernández-González and Gómez-Rodríguez, 2020b) and extract weights from the second-to-last layer for the BASE models and, for

the LARGE models, we use a combination of four layers from 17 to 20. We do not try other variations that might probably work better for our specific task. While regular word embeddings are fine-tuned during training, BERT-based embeddings are kept fixed following a less resource-consuming strategy. See Appendix A.2 for further details.

Parsers For parsing the CCAs generated by the pointer, we employ two off-the-shelf continuous constituent parsers that excel in continuous benchmarks: the chart-based parser by Kitaev et al. (2019) and the transition-based model by Yang and Deng (2020). In both cases, we adopt the basic configuration (described in their respective papers) and just vary the encoder initialization with BERT_BASE and BERT_LARGE (Devlin et al., 2019), as well as XLNet (Yang et al., 2019).

Metrics Following standard practice, we ignore punctuation and root symbols for evaluating discontinuous parsing and use `discodop` for reporting F-score and discontinuous F-score (DF1).[3] For jointly evaluating the pointer and labeller performance, we rely on the Labelled Attachment Score[4] (LAS) and choose the model with the highest score on the development set. For reporting speeds, we use sentences per second (sent/s).

3.2 Results

Table 1 highlights how the labeller enhances the pointer's performance. While the overall UAS (or LAS when the labeller is used, since its accuracy is 100% in all cases) is not affected substantially, it can be seen that the percentage of relocated to-

---

[3]F-score measured only on discontinuous constituents.

[4]A standard metric used for dependency parsing that, in our case, measures which tokens have the correct CCA position and also were correctly identified as relocated token.

Parser (no tags or predicted PoS tags)	TIGER			NEGRA			DPTB		
	F1	DF1	sent/s	F1	DF1	sent/s	F1	DF1	sent/s
Versley (2016)	79.5	-	-	-	-	-	-	-	-
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
Corro (2020) + BERT$_X$	90.0	62.1	-	91.6	66.1	-	94.8	68.9	-
Ruprecht and Mörbitz (2021)	82.5	55.9	101	82.7	49.0	136	90.1	72.9	95
Ruprecht and Mörbitz (2021) + BERT$_{BASE}$	88.3	69.0	60	90.9	72.6	68	93.3	**80.5**	57
Vilares and Gómez-Rodríguez (2020)	77.5	39.5	**568**	75.6	34.6	**715**	88.8	45.8	**611**
Vilares and Gómez-Rodríguez (2020) + BERT$_{BASE}$	84.6	51.1	80	83.9	45.6	80	91.9	50.8	80
Vilares and Gómez-Rodríguez (2020) + BERT$_{LARGE}$	-	-	-	-	-	-	92.8	53.9	34
Fernández-González and Gómez-Rodríguez (2020a)	85.7	60.4	-	85.7	58.6	-	-	-	-
Fernández-González and Gómez-Rodríguez (2020b)*	86.6	62.6	-	86.8	69.5	-	-	-	-
Fernández-González and Gómez-Rodríguez (2020b)+BERT$_{BASE}^*$	89.8	**71.0**	-	91.0	**76.6**	-	-	-	-
Pointer + Kitaev et al. (2019) + BERT$_{BASE}$	88.5	63.0	238	90.0	65.9	275	94.0	68.9	231
Pointer + Kitaev et al. (2019) + BERT$_{LARGE}$	**90.5**	68.1	207	**92.0**	67.9	216	94.7	72.9	193
Pointer + Kitaev et al. (2019) + XLNet	-	-	-	-	-	-	95.1	74.1	179
Pointer + Yang and Deng (2020) + BERT$_{BASE}$	88.5	62.7	157	90.4	66.5	188	94.1	67.2	152
Pointer + Yang and Deng (2020) + BERT$_{LARGE}$	**90.5**	68.8	129	91.7	67.9	158	94.8	71.3	135
Pointer + Yang and Deng (2020) + XLNet	-	-	-	-	-	-	**95.5**	73.4	133

Table 2: Comparison of our approach against discontinuous constituent parsers on the test split. BERT$_X$ denotes that the size was not specified in the original paper. In those cases where the parser uses XLNet, the pointer is initialized with BERT$_{LARGE}$. * uses extra dependency information. All reported speeds were measured on GPU.

kens predicted by the pointer is higher (increased recall without harming precision), also leading to an improvement in F-score.

In Table 2, we show how our novel neural architecture (combined with two continuous parsers) achieves competitive accuracies in all datasets, outperforming all existing parsers when the largest pre-trained models are employed. It is important also to remark that F-scores on discontinuities produced by our setup (and where the pointer has an important role) are on par with purely discontinuous parsers.[5] Regarding efficiency, the proposed Pointer Network provides high speeds even with BERT$_{LARGE}$: on the test splits, 553.7 sent/s on TIGER, 613.5 sent/s on NEGRA and 694.3 sent/s on DPTB. As a result, continuous parsers' efficiency is not penalized, and the pointer+parser combinations are faster than all existing approaches that use pre-trained language models (including the fastest parser to date by Vilares and Gómez-Rodríguez (2020), which is also outperformed by a wide margin in terms of accuracy). Finally, as also observed on continuous treebanks, no meaningful differences can be seen between both continuous parsers' performance.

4 Conclusions and Future work

We show that, by accurately removing crossing branches from discontinuous trees, continuous parsers can perform discontinuous parsing more efficiently, achieving accuracies on par with more expensive discontinuous approaches. In addition, the proposed Pointer Network can be easily combined with any off-the-self continuous parser and, while barely affecting its efficiency, it can extend its coverage to fully model discontinuous phenomena.

We will investigate alternatives to the in-order reordering (e.g., pre- and post-order traversal or language-specific rules to generate more continuous-friendly structures). While we think that using a different CCA would have no substantial impact on Pointer Network reordering, it might affect continuous parsing performance (as it may be easier for the parser to process reordered constituent trees with a syntax closer to original continuous structures, and factors like the degree of left vs. right branching may also have an influence).

Acknowledgments

---

[5]We believe that the remarkable performance obtained on discontinuities by Fernández-González and Gómez-Rodríguez (2020b) probably owes to the leverage of additional non-projective dependency information.

10574

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

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. 2002. TIGER treebank. In *Proceedings of the 1st Workshop on Treebanks and Linguistic Theories (TLT)*, pages 24–42.

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*, pages 204–217, Minneapolis, Minnesota. Association for Computational Linguistics.

Maximin Coavoux and Benoît Crabbé. 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*, pages 1259–1270, Valencia, Spain. 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.

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.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *ICLR*. OpenReview.net.

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.

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. 2019. Left-to-right dependency parsing with pointer networks. 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 710–716, Minneapolis, Minnesota. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 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*, pages 7724–7731. AAAI Press.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020b. Multitask pointer network for multi-representational parsing.

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.

Kilian Gebhardt. 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*, pages 91–97, Online. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

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.

Wang Ling, Chris Dyer, Alan W. Black, and Isabel Trancoso. 2015. Two/too simple adaptations of Word2Vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1299–1304, Denver, Colorado.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 1064–1074. Association for Computational Linguistics.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard H. Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Melbourne, Australia, July 15-20, 2018*, pages 1403–1414.

Wolfgang Maier. 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)*, pages 1202–1212, Beijing, China. Association for Computational Linguistics.

Wolfgang Maier and Timm Lichte. 2016. Discontinuous parsing with continuous trees. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 47–57, San Diego, California. 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*, 19:313–330.

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.

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 *Proceedings of the Fifth Conference on Applied Natural Language Processing*, ANLC '97, pages 88–95, Stroudsburg, PA, USA. Association for Computational Linguistics.

Miloš Stanojević and Raquel G. Alhama. 2017. Neural discontinuous constituency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1666–1676, Copenhagen, Denmark. 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.

Andreas van Cranenburgh, Remko Scha, and Rens Bod. 2016. Data-oriented parsing with discontinuous constituents and function tags. *J. Language Modelling*, 4:57–111.

Yannick Versley. 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*, pages 39–53, Dublin, Ireland. Dublin City University.

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.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics (ACL'87)*, pages 104–111, Morristown, NJ, USA. 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 C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc.

Kaiyu Yang and Jia Deng. 2020. Strongly incremental constituency parsing with graph neural networks. In *Neural Information Processing Systems*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

A Appendices

A.1 Treebank splits

Standard splits for discontinuous German NEGRA (Skut et al., 1997) and TIGER (Brants et al., 2002) treebanks are defined by (Dubey and Keller, 2003) and (Seddah et al., 2013), respectively. For the discontinuous version of the English Penn Treebank (DPTB) (Evang and Kallmeyer, 2011), commonly-used splits are defined as follows: Sections 2 to 21 for training, 22 for development and 23 for testing. In Table 3, we report the number of samples per treebank split.

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

Table 3: Treebank statistics.

A.2 Pointer Settings

Pre-trained word embeddings The specific pre-trained BERT models used in this work were: for English, `bert-base-cased` and `bert-large-cased`; and, for German, `bert-base-german-cased` and `deepset/gbert-large`. The same pre-trained models were also used for both continuous parsers, including also `xlnet-large-cased` in those experiments where XLNet is employed for initializing the parser's encoder.

Hyper-parameters We use the Adam optimizer (Kingma and Ba, 2014) and the same hyper-parameter selection as the dependency parser by (Ma et al., 2018). No further adaptation to our specific task was performed. These are detailed in Table 4.

Hardware Our approach (combined with the two specified continuous parsers) was fully tested on an Intel(R) Core(TM) i9-10920X CPU @ 3.50GHz with a single GeForce RTX 3090 GPU. All speeds are measured considering the time taken by the whole process, i.e., the reordering, continuous parsing and reversal of the reordering.

Architecture hyper-parameters	
CNN window size	3
CNN number of filters	50
BiLSTM encoder layers	3
BiLSTM encoder size	512
LSTM decoder layers	1
LSTM decoder size	512
LSTM layers dropout	0.33
Word/Char. embedding dimension	100
$BERT_{BASE}$ embedding dimension	768
$BERT_{LARGE}$ embedding dimension	1024
Embeddings dropout	0.33
MLP layers	1
MLP activation function	ELU
CCA Position MLP size	512
Label MLP size	128
UNK replacement probability	0.5
Beam size	10
Adam optimizer hyper-parameters	
Initial learning rate	0.001
β_1, β_2	0.9
Batch size	32
Decay rate	0.75
Gradient clipping	5.0

Table 4: Model hyper-parameters.