Headed-Span-Based Projective Dependency Parsing

Anonymous ACL submission

Abstract

We propose a new method for projective dependency parsing based on headed spans. In a projective dependency tree, the largest subtree rooted at each word covers a contiguous sequence (i.e., a span) in the surface order. We call such a span marked by a root word *headed span*. A projective dependency tree can be represented as a collection of headed spans. We decompose the score of a dependency tree into the scores of the headed spans and design a novel $O(n^3)$ dynamic programming algorithm to enable global training and exact inference. We evaluate our method on PTB, CTB, and UD and it achieves state-of-the-art or competitive results. We will release our code at github.com.

1 Introduction

Dependency parsing is an important task in natural language processing, which has numerous applications in downstream tasks, such as opinion mining (Zhang et al., 2020a), relation extraction (Jin et al., 2020), named entity recognition (Jie and Lu, 2019), machine translation (Bugliarello and Okazaki, 2020), among others.

There are two main paradigms in dependency parsing: graph-based and transition-based methods. Graph-based methods decompose the score of a tree into the scores of parts. In the simplest first-order graph-based methods (McDonald et al., 2005, *inter alia)*, the parts are single dependency arcs. In higher-order graph-based methods (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010; Ma and Zhao, 2012), the parts are combinations of multiple arcs. Transition-based methods (Nivre and Scholz, 2004; Chen and Manning, 2014, *inter alia)* read the sentence sequentially and conduct a series of local decisions to build the final parse. Recently, transition-based methods with Pointer Networks (Vinyals et al., 2015) have obtained competitive performance to graph-based methods (Ma et al., 2018; Liu et al., 2019; Fernández-González and Gómez-Rodríguez, 2019; Fernández-González and Gómez-Rodríguez, 2021).

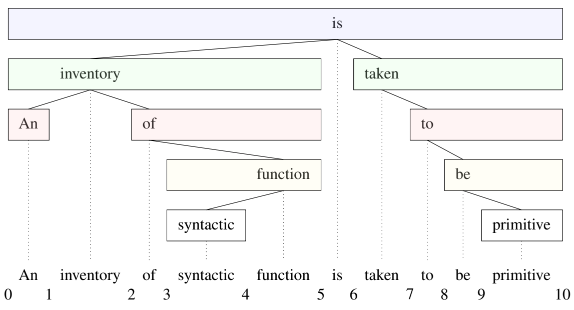

Figure 1: Illustration of a projective dependency parse tree. Each rectangle represents a headed span.

A main limitation of first-order graph-based methods is that they independently score each arc based solely on the two words connected by the arc. Ideally, the appropriateness of an arc should depend on the whole parse tree, particularly the subtrees rooted at the two words connected by the arc. Although subtree information could be implicitly encoded (Falenska and Kuhn, 2019) in powerful neural encoders such as LSTMs (Hochreiter and Schmidhuber, 1997) and Transformers (Vaswani et al., 2017), there is evidence that their encoding of such information is inadequate. For example, higher-order graph-based methods, which capture more subtree information by simultaneously considering multiple arcs, have been found to outperform first-order methods despite using powerful encoders (Fonseca and Martins, 2020; Zhang et al., 2020b). In contrast to the line of work on higher-order parsing, we propose a different way to incorporate more subtree information as discussed later.

Transition-based methods, on the other hand, can easily utilize information from partially built subtrees, but they have their own shortcomings. For instance, most of them cannot perform global optimization during decoding [1] and rely on greedy or beam search to find a locally optimal parse, and their sequential decoding may cause error propagation as past decision mistakes will negatively affect the decisions in the future.

To overcome the aforementioned limitations of first-order graph-based and transition-based methods, we propose a new method for projective dependency parsing based on so-called headed spans. A projective dependency tree has a nice structural property that the **largest** subtree rooted at each word covers a contiguous sequence (i.e., a span) in the surface order. We call such a span marked with its root word a *headed span*. A projective dependency tree can be treated as a collection of headed spans such that each word corresponds to exactly one headed span, as illustrated in Figure 1. For example, $(0, 5, \text{inventory})$ is a headed span, in which span $(0, 5)$ has a head word *inventory*. In this view, projective dependency parsing is similar to constituency parsing as a constituency tree can be treated as a collection of constituent spans. The main difference is that in a binary constituency tree, a constituent span (i, k) is made up by two adjacent spans (i, j) and (j, k), while in a projective dependency tree, a headed span (i, k, x_h) is made up by one or more smaller headed spans and a single word span $(h - 1, h)$. For instance, $(0, 5, \text{inventory})$ is made up by $(0, 1, \text{An})$, $(1, 2)$ and $(2, 5, \text{of})$. There are a few constraints between headed spans to force projectivity (section 3). These structural constraints are the key to designing an efficient dynamic programming algorithm for exact inference.

Because of the similarity between constituency parsing and our head-span-based view of projective dependency parsing, we can draw inspirations from the constituency parsing literature to design our dependency parsing method. Specifically, span-based constituency parsers (Stern et al., 2017; Kitaev and Klein, 2018; Zhang et al., 2020c; Xin et al., 2021) decompose the score of a constituency tree into the scores of its constituent spans and use the CYK algorithm (Cocke, 1969; Younger, 1967; Kasami, 1965) for global training and inference. Built upon powerful neural encoders, they have obtained state-of-the-art performance in constituency parsing. Inspired by them, we propose to decompose the score of a projective dependency tree into the scores of headed spans and design a novel $O(n^3)$ dynamic programming algorithm for global training and exact inference, which is on par with the Eisner algorithm (Eisner, 1996) in time complexity for projective dependency parsing. We make a departure from existing graph-based methods since we do not model dependency arcs directly. Instead, the dependency arcs are *induced* from the collection of headed spans (section 3). Compared with first-order graph-based methods, our method can utilize more subtree information since a headed span contains all children (if any) of the corresponding headword (and all words within the subtree). Compared with most of transition-based methods, our method allows global training and exact inference and does not suffer from error propagation or exposure bias.

Our contributions can be summarized as follows:

- We treat a projective dependency tree as a collection of headed spans, providing a new perspective of projective dependency parsing.
- We design a novel $O(n^3)$ dynamic programming algorithm to enable global training and exact inference for our proposed model.
- We have obtained the state-of-the-art or competitive results on PTB, CTB, and UD v2.2, showing the effectiveness of our proposed method.

2 Parsing

We adopt the two-stage parsing strategy, i.e., we first predict an unlabeled tree and then predict the dependency labels. Given a sentence $x_1, ..., x_n$, its unlabeled projective dependency parse tree y can be regarded as a collection of headed spans (l_i, r_i, x_i) where $1 \leq i \leq n$. For each word x_i, we can find exactly one headed span (l_i, r_i, i) (where l_i and r_i are the left and right span boundaries) given parse tree y, so there are totally n headed spans in y as we can see in Figure 1. We can use a simple post-order traversal algorithm to obtain all headed spans in $O(n)$ time. We then define the score of y as:

$$s(y) = \sum_{i=1,\ldots,n} s^{\text{span}}_{l_i, r_i, i}$$

and we show how to compute them using neural networks in the next section.

Our parsing algorithm is based on the following key observations:

- For a given parent word x_k, if it has any children to the left (right), then all headed spans of

[1] We are aware of few transition-based parsers performing global optimization via dynamic programming algorithms, cf. Kuhlmann et al. (2011); Shi et al. (2017); Gómez-Rodríguez et al. (2018).

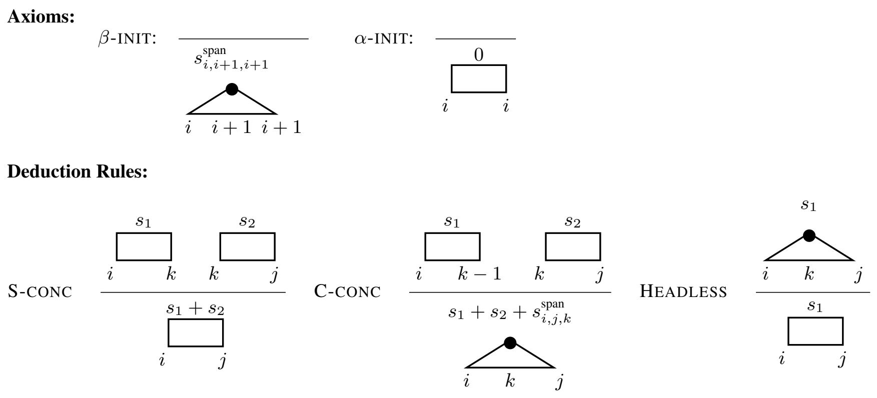

Figure 2: Deduction rules for our proposed parsing algorithm. All deduction items are annotated with their scores.

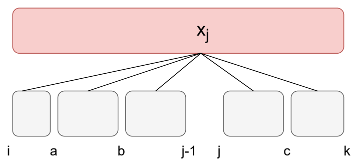

Figure 3: An example subtree.

its children in this direction should be consecutive and form a larger span, which we refer to as the left (right) child span. The left (right) boundary of the headed span of x_k is the left (right) boundary of the leftmost (rightmost) child span, or $k-1$ (k) if x_k has no child to the left (right).

- If a parent word x_k has children in both directions, then its left span and right span are separated by the single word span $(k-1, k)$.

Figure 3 shows an example subtree. The left child span is $(i, j-1)$ and the right child span is (j, k). They are separated by the single word span $(j-1, j)$. The headed span (i, k, j) can be generated by concatenating the left child span, right child span, and the single word span. Note that the left (right) child span can contain one or more headed spans. Based on these observations, we design the following parsing items: (1) $\alpha_{i,j}$: the accumulated score of span (i, j) serving as a left or right child span. (2) $\beta_{i,j,k}$: the accumulated score of the headed span (i, j, k). We use the parsing-as-deduction framework (Pereira and Warren, 1983) to describe our algorithm in Fig. 2. We draw $\alpha_{i,j}$ as rectangles and $\beta_{i,j,k}$ as triangles. The rule S-CONC is used to concatenate two consecutive child spans into a single child span; C-CONC is used to concatenate left and right child span $(i, k-1)$ and (k, j) along with the root word-span $(k-1, k)$ to form a headed span (i, j, k); HEADLESS is used to obtain a headless child span from a headed span. Fig. 2 corresponds to the following recursive formulas:

$$\beta_{i,i+1,i+1} = s^{\text{span}}_{i,i+1,i+1} \tag{1}$$

$$\alpha_{i,i} = 0 \tag{2}$$

$$\beta_{i,j,k} = \alpha_{i,k-1} + \alpha_{k,j} + s^{\text{span}}_{i,j,k} \tag{3}$$

$$\begin{aligned}\alpha_{i,j} = \max(&\max_{i<k<j}(\alpha_{i,k} + \alpha_{k,j}),\\ &\max_{i<h\leq j}(\beta_{i,j,h}))\end{aligned} \tag{4}$$

We set $\alpha_{i,i} = 0$ for the convenience of calculating $\beta_{i,j,k}$ when x_k does not have children on either side. In Eq. 4, we can see that the child span comes from either multiple smaller consecutive child spans (i.e., $\max_{i<k<j}(\alpha(i,k) + \alpha(k,j)))$ or a single headed span (i.e., $\max_{i<h\leq j}(\beta(i,j,h))))$. We also maintain backpointers based on these equations (i.e., maintain all arg max) for parsing.

A key point of our parsing algorithm is that, during backtracking, we add arcs emanated from the headword of a large headed span to **every** headword of (zero or more) smaller headed spans within the left/right child span, so that we can induce a dependency tree. Finding all smaller headed spans within left and right child spans requires finding the best segmentation, which is similar to the inference procedure of the semi-Markov CRF model (Sarawagi and Cohen, 2004). We provide the pseudocode of our parsing algorithm in Appd. A.

Parsing complexity. From Eq. 1 to 4, we can see that at most three variables (i.e., i, j, k) are required to iterate over and therefore the total parsing time complexity is $O(n^3)$.

Spurious ambiguity. Note that different order of concatenation of child spans can result in the

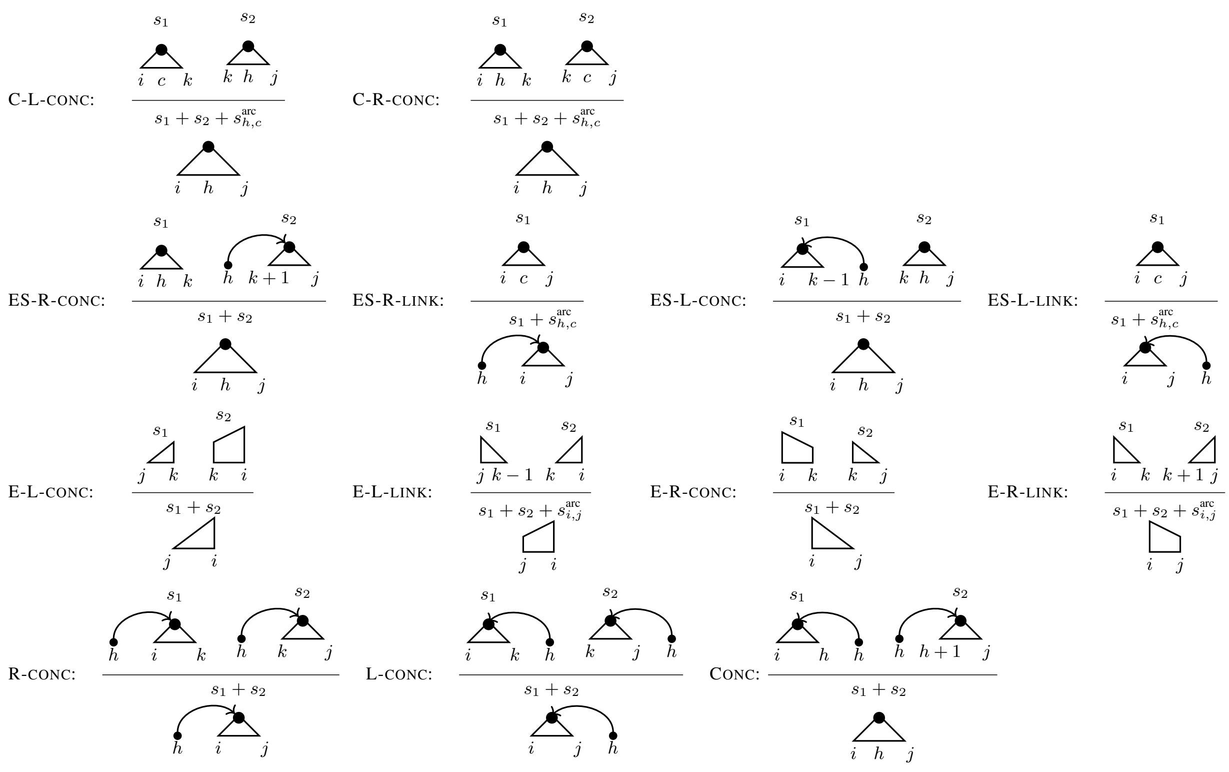

Figure 4: Deductive rules of the parsing algorithms of Collins (1996) (the first line), Eisner and Satta (1999) (the second line), Eisner (1997) (the third line). The last line is the resulting deduction rules after applying head-splitting on ES-L-CONC and ES-R-CONC. All deduction items are annotated with their scores. We only consider the pure dependency versions of these algorithms. We omit axiom items for simplicity.

same parse, although this does not affect finding the optimal parse.

Comparison with previous parsing algorithms. We compare our algorithm with three classical parsing algorithms (Collins, 1996; Eisner and Satta, 1999; Eisner, 1997) in order to help readers better understand our algorithm. We only consider their pure dependency versions[2] for the convenience of discussion. Fig. 2 shows the deductive rules of the three algorithms.

Collins (1996) adapt the CYK algorithm by maintaining head positions for both sides, thereby increasing the parsing complexity from $O(n^3)$ to $O(n^5)$. Their parsing items are identified by two endpoints and a head position, which is similar to our concept of headed spans superficially. However, in their algorithm, there could be multiple spans sharing the same head position within a single parse. For instance, (i, j) and (k, j) share the same head position h in C-L-CONC. In contrast, spans cannot share a head position in a single parse under our definition, because there is exactly one headed span for each word. Besides, the concatenation order of subtrees differs.

Eisner and Satta (1999) note that the linking of heads and the concatenation of subtrees can be separated (e.g., C-R-CONC can be decomposed into two rules, ES-R-CONC and ES-R-LINK) so that the parsing complexity can be reduced to $O(n^4)$. This strategy is also known as the hook trick, which reduces subtrees to headless spans (e.g., (i, c, j) to (i, j) in ES-L-LINK and ES-R-LINK).

Eisner (1997) uses the head-splitting trick to decrease parsing complexity to $O(n^3)$. The key idea is to split each subtree into a left and a right fragment, so that the head is always placed at one of the two boundaries of a fragment instead of an internal position, thereby eliminating the need of maintaining the head positions.

Our algorithm adopts a combination of the hook trick and the head-splitting trick. Starting from the rules of Eisner and Satta (1999) that apply the hook trick, we can rewrite ES-L-CONC, ES-R-CONC as L-CONC, R-CONC and COMB. It is easy to verify

[2]The parsing algorithms of Collins (1996) and Eisner and Satta (1999) are defined with (lexicalized) context-free grammars. Gómez-Rodríguez et al. (2008, 2011) provide their pure dependency versions, which amounts to considering arc scores only.

the equivalence of the rules before and after the rewrite[3]. The key difference is in the concatenation order of subtrees. We concatenate all subtrees to the left/right of the new head first, which can be viewed as adopting the head-splitting trick. Then, note that the position of the head is uniquely determined by the two concatenations of subtrees, and that our model does not consider s^{arc}. Consequently, we have no need to maintain head position h in L-CONC and R-CONC and can merge these two rules to S-CONC of fig. 2. Accordingly, CONC can be modified to C-CONC of fig. 2. Eliminating bookkeeping of h is how we can obtain better parsing complexity than Eisner and Satta (1999). Finally, we can incorporate span score $s^{\text{span}}_{i,j,h}$ into C-CONC.

3 Model

3.1 Neural encoding and scoring

We add <bos> (beginning of sentence) at x_0 and <eos> (end of sentence) at x_{n+1}. In the embedding layer, we apply mean-pooling to the last layer of BERT (Devlin et al., 2019) (i.e., taking the mean value of all subword embeddings) to generate dense word-level representation e_i for each token x_i [4]. Then we feed $e_0, ..., e_{n+1}$ into a 3-layer bidirectional LSTM (BiLSTM) to get $c_0, ..., c_{n+1}$, where $c_i = [f_i; b_i]$ and f_i and b_i are the forward and backward hidden states of the last BiLSTM layer at position i respectively. We then use $e_{i,j}$ to represent span (i, j):

$$\begin{aligned} h_k &= [f_k, b_{k+1}] \\ e_{i,j} &= h_j - h_i \end{aligned}$$

After obtaining the word and span representations, we use deep biaffine function (Dozat and Manning, 2017) to score headed spans:

$$\begin{aligned} c'_k &= \text{MLP}_{\text{word}}(c_k) \\ e'_{i,j} &= \text{MLP}_{\text{span}}(e_{i,j}) \\ s^{\text{span}}_{i,j,k} &= \left[c'_k; 1\right]^\top W^{\text{span}} \left[e'_{i,j}; 1\right] \end{aligned}$$

where MLP_{word} and MLP_{span} are multi-layer perceptrons (MLPs) that project word and span representations into d-dimensional spaces respectively; $W^{\text{span}} \in \mathcal{R}^{(d+1)\times(d+1)}$.

[3]Note that this only holds for the pure dependency version, since otherwise we cannot track some intermediate constituent spans after changing the concatenation order of subtrees.

[4]For some datasets (e.g., Chinese Treebank), we concatenate the POS tag embedding with the BERT embedding as e_i.

Similarly, we use deep biaffine functions to score the labels of dependency arcs for a given gold or predicted tree [5]:

$$\begin{aligned} c'_i &= \text{MLP}_{\text{parent}}(c_i) \\ c'_j &= \text{MLP}_{\text{child}}(c_j) \\ s^{\text{label}}_{i,j,r} &= \left[c'_i; 1\right]^\top W^{\text{label}}_r \left[c'_j; 1\right] \end{aligned}$$

where $\text{MLP}_{\text{parent}}$ and $\text{MLP}_{\text{child}}$ are MLPs that map word representations into d'-dimensional spaces; $W^{\text{label}}_r \in \mathcal{R}^{(d'+1)\times(d'+1)}$ for each relation type $r \in R$ in which R is the set of all relation types.

3.2 Training loss

Following previous work, we decompose the training loss into the unlabeled parse loss and arc label loss:

$$L = L_{\text{parse}} + L_{\text{label}}$$

For L_{parse}, we can either design a local span-selection loss:

$$L^{\text{local}}_{\text{parse}} = \sum_{(i,j,k)\in y} -\log \frac{\exp(s^{\text{span}}_{i,j,k})}{\sum_{0\le p\le k<q\le n} \exp(s^{\text{span}}_{p,q,k})}$$

which is akin to the head-selection loss (Dozat and Manning, 2017), or use global structural loss. Experimentally, we find that the max-margin loss (Taskar et al., 2004) performs best. The max-margin loss aims to maximize the margin between the score of the gold tree y and the incorrect tree y' of the highest score:

$$L_{\text{parse}} = \max(0, \max_{y' \neq y}(s(y') + \Delta(y', y) - s(y)) \tag{5}$$

where Δ measures the difference between the incorrect tree and gold tree. Here we let Δ to be the Hamming distance (i.e., the total number of mismatches of headed spans). We can perform loss-augmented inference (Taskar et al., 2005) to calculate Eq. 5.

Finally, we use cross entropy for L_{label}:

$$L_{\text{label}} = \sum_{(x_i \to x_j, r)\in y} -\log \frac{\exp(s^{\text{label}}_{i,j,r})}{\sum_{r'\in R} \exp(s^{\text{label}}_{i,j,r'})}$$

[5]In our preliminary experiments, we find that directly calculating the scores based on parent-child word representations leads to a slightly better result (< 0.1 LAS) than those based on span representations. A possible reason is that, since LAS is arc-factorized, even if we predict a correct parent-child pair, we can predict the wrong headed spans for the parent or child or both, thereby negatively affecting the labeling scores and resulting in worse LAS. Therefore, in our work we use arc-based label scores to suit the LAS metric.

where $(x_i \rightarrow x_j, r) \in y$ denotes every dependency arc from x_i to x_j with label r in y.

4 Experiments

4.1 Data and setting

Following Wang and Tu (2020), we evaluate our proposed method on Penn Treebank (PTB) 3.0 (Marcus et al., 1993), Chinese Treebank (CTB) 5.1 (Xue et al., 2005) and 12 languages on Universal Dependencies (UD) 2.2: BG-btb, CA-ancora, CS-pdt, DE-gsd, EN-ewt, ES-ancora, FR-gsd, IT-isdt, NL-alpino, NO-rrt, RO-rrt, RU-syntagrus [6]. For PTB, we use the Stanford Dependencies conversion software of version 3.3 to obtain dependency trees. For CTB, we use head-rules from Zhang and Clark (2008) and Penn2Malt[7] to obtain dependency trees. Following Wang and Tu (2020), we use gold POS tags for CTB and UD. We do not use POS tags in PTB. For PTB/CTB, we drop all non-projective trees during training. For UD, we use MaltParser v1.9.2 [8] to adopt the pseudo-projective transformation (Nivre and Nilsson, 2005) to convert nonprojective trees into projective trees when training, and convert back when evaluating, for both our model and reimplemented baseline model. See Appd. B for implementation details.

4.2 Evaluation methods

We report the unlabeled attachment score (UAS) and labeled attachment score (LAS) averaged from three runs with different random seeds. In each run, we select the model based on the performance on the development set. Following Wang and Tu (2020), we ignore all punctuation marks during evaluation.

4.3 Main result

Table 1 shows the results on PTB and CTB. Note that Biaffine+MM is our reimplementation of the Biaffine Parser that uses the same setting as in our method, including the use of the max-margin loss instead of the local head-selection loss. Interestingly, we find that Biaffine+MM has already surpassed many strong baselines, and this may be due to the proper choices of hyperparameters and the use of the max-margin loss (we observe that using the max-margin loss leads to a better performance compared with the original head-selection loss), so Biaffine+MM is a very strong baseline. It also has the same number of parameters as our methods. Our method surpasses Biaffine+MM on both datasets, showing the competitiveness of our headed-span-based method in a fair comparison with first-order graph-based parsing.

[6]We do not concatenate all datasets during training. We train on each dataset separately.

[7]https://cl.lingfil.uu.se/~nivre/research/Penn2Malt.html

[8]http://www.maltparser.org/download.html

	PTB		CTB	
	UAS	LAS	UAS	LAS
MFVI2O	95.98	94.34	90.81	89.57
TreeCRF2O	96.14	94.49	-	-
HierPtr	96.18	94.59	90.76	89.67
	+BERT$_{\text{base}}$		+BERT$_{\text{base}}$	
RNGTr	96.66	95.01	92.98	91.18
	+BERT$_{\text{large}}$		+BERT$_{\text{base}}$	
MFVI2O	96.91	95.34	92.55	91.69
HierPtr	97.01	95.48	92.65	91.47
Biaffine+MM†	97.22	95.71	93.18	92.10
Ours	**97.24**	**95.73**	**93.33**	**92.30**
	For reference			
	+XLNet$_{\text{large}}$		+BERT$_{\text{base}}$	
HPSG$^\flat$	97.20	95.72	-	-
HPSG+LAL$^\flat$	97.42	96.26	94.56	89.28

Table 1: Results for different model on PTB and CTB. $^\flat$ indicate that they use additional annotated constituency trees in training. † means our reimplementation. *Biaffine*: Dozat and Manning (2017). *MFVI2O*: Wang and Tu (2020). *TreeCRF2O*: Zhang et al. (2020b). *RNGTr*: Mohammadshahi and Henderson (2021). *HierPtr*: Fernández-González and Gómez-Rodríguez (2021). *HPSG*: Zhou and Zhao (2019). *HPSG+LAL*: Mrini et al. (2020).

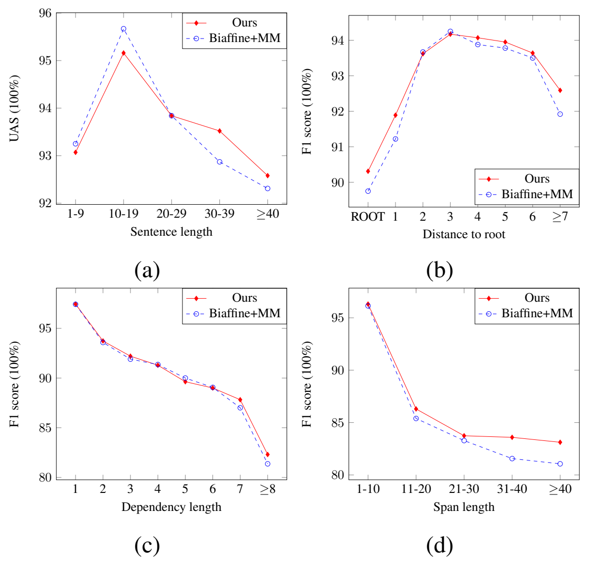

Figure 5: Error analysis on the CTB test set.

	bg	ca	cs	de	en	es	fr	it	nl	no	ro	ru	Avg
TreeCRF2O	90.77	91.29	91.54	80.46	87.32	90.86	87.96	91.91	88.62	91.02	86.90	93.33	89.33
MFVI2O	90.53	92.83	92.12	81.73	89.72	92.07	88.53	92.78	90.19	91.88	85.88	92.67	90.07
					+BERT$_{\text{multilingual}}$								
MFVI2O	**91.30**	93.60	92.09	82.00	90.75	92.62	89.32	93.66	91.21	91.74	86.40	92.61	90.61
Biaffine+MM[†]	90.30	**94.49**	**92.65**	**85.98**	91.13	93.78	**91.77**	94.72	91.04	94.21	87.24	**94.53**	91.82
Ours	91.10	94.46	92.57	85.87	**91.32**	**93.84**	91.69	**94.78**	**91.65**	**94.28**	**87.48**	94.45	**91.96**

Table 2: Labeled Attachment Score (LAS) on twelve languages in UD 2.2. We use ISO 639-1 codes to represent languages. † means our implementation.

	PTB		**CTB**	
	UAS	LAS	UAS	LAS
max-margin loss	97.24	95.73	93.33	92.30
span-selection loss	97.07	95.50	93.28	92.20

Table 3: The influence of training loss function on PTB and CTB.

Our method also obtains the state-of-the-art result among methods that only use dependency training data (HPSG+LAL uses additional constituency trees as training data, so it is not directly comparable with the other systems.).

Table 2 shows the results on UD. We can see that our reimplemented Biaffine+MM has already surpassed MFVI2O, which utilizes higher-order information. Our method outperforms Biaffine+MM by 0.14 LAS on average, validating the effectiveness of our proposed method in the multilingual scenarios.

5 Analysis

5.1 Influence of training loss function

Table 3 shows the influence of the training loss function. We find that the max-margin loss performs better on both datasets: 0.17 UAS improvement on PTB and 0.05 UAS improvement on CTB comparing to the local span-selection loss, which shows the effectiveness of using global loss.

5.2 Error analysis

As previously argued, first-order graph-based methods are insufficient to model complex subtrees, so they may have difficulties in parsing long sentences and handling long-range dependencies. To verify this, we follow (McDonald and Nivre, 2011) to plot UAS as a function of the sentence length and plot F1 scores as functions of the distance to root and dependency length on the CTB test set. We additionally plot the F1 score of the predicted headed spans against the gold headed spans with different span lengths.

From Figure 5a, we can see that Biaffine+MM has a better UAS score on short sentences (of length <=20), while for long sentences (of length >=30), our headed span-based method has a higher performance, which validates our conjecture.

Figure 5b shows the F1 score for arcs of varying distances to root. Our model is better at predicting arcs of almost all distances to root in the dependency tree, which reveals our model's superior ability to predict complex subtrees.

Figure 5c shows the F1 score for arcs of varying lengths. Both Biaffine+MM and our model have a very similar performance in predicting arcs of distance < 7, while our model is better at predicting arcs of distance >= 7, which validates the ability of our model at capturing long-range dependencies.

Figure 5d shows the F1 score for headed spans of varying lengths. We can see that when the span length is small (<=10), Biaffine+MM and our model have a very similar performance. However, our model is much better in predicting longer spans (especially for spans of length >30).

5.3 Parsing speed

Inspired by Zhang et al. (2020b) and Rush (2020) who independently propose to batchify the Eisner algorithm using `Pytorch`, we batchify our proposed method so that $O(n^2)$ out of $O(n^3)$ can be computed in parallel, which greatly accelerates parsing. We achieve a similar parsing speed of our method to the fast implementation of the Eisner algorithm by Zhang et al. (2020b): it parses 273 sentences per second, using BERT as the encoder under a single TITAN RTX GPU.

6 Related work

Dependency parsing with more complex subtree information. There has always been an interest to incorporate more complex subtree information into graph-based and transition-based methods since their invention. Before the deep learning era, it was difficult to incorporate sufficient contextual information in first-order graph-based parsers. To mitigate this, researchers develop higher-order dependency parsers to capture more contextual information (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010; Ma and Zhao, 2012). However, incorporating more complex factors worsens inference time complexity. For example, exact inference for third-order projective dependency parsing has a $O(n^4)$ time complexity and exact inference for higher-order non-projective dependency parsing is NP-hard (McDonald and Pereira, 2006). To decrease inference complexity, researchers use approximate parsing methods. Smith and Eisner (2008) use belief propagation (BP) framework for approximate inference to trade accuracy for efficiency. They show that third-order parsing can be done in $O(n^3)$ time using BP. Gormley et al. (2015) unfold the BP process and use gradient descent to train their parser in an end-to-end manner. Wang and Tu (2020) extend their work by using neural scoring functions to score factors. For higher-order non-projective parsing, researchers resort to dual decomposition algorithm (e.g., AD^3) for decoding (Martins et al., 2011, 2013). They observe that the approximate decoding algorithm often obtains exact solutions. Fonseca and Martins (2020) combine neural scoring functions and their decoding algorithms for non-projective higher-order parsing. Zheng (2017) proposes a incremental graph-based method to utilize higher-order information without hurting the advantage of global inference. Ji et al. (2019) use a graph attention network to incorporate higher-order information into the Biaffine Parser. Zhang et al. (2020b) enhance the Biaffine Parser by using a deep triaffine function to score sibling factors. Mohammadshahi and Henderson (2021) propose an iterative refinement network that injects the predicted soft trees from the previous iteration to the self-attention layers to predict the soft trees of the next iteration, so that information of the whole tree is considered in parsing. As for transition-based methods, Ma et al. (2018); Liu et al. (2019); Fernández-González and Gómez-Rodríguez (2021) incorporate sibling and grandparent information into transition-based parsing with Pointer Networks.

The hook trick and the head-splitting trick. These two tricks have been used in the parsing literature to accelerate parsing. Eisner and Satta (1999, 2000) use the hook trick to decrease the parsing complexity of lexicalized PCFGs and Tree Adjoining Grammars. Huang et al. (2005, 2009) adapt the hook trick to accelerate machine translation decoding. The parsing algorithms of Corro (2020) and Xin et al. (2021) can be viewed as adapting the hook trick to accelerate discontinuous and continuous constituency parsing. Eisner (1997); Satta and Kuhlmann (2013) use the head-splitting trick to accelerate projective and nonprojective dependency parsing.

Span-based constituency parsing. Span-based parsing is originally proposed in continuous constituency parsing (Stern et al., 2017; Kitaev and Klein, 2018; Zhang et al., 2020c; Xin et al., 2021). Span-based constituency parsers decompose the score of a constituency tree into the scores of its constituents. Recovering the highest-scoring tree can be done via the exact CYK algorithm or greedy top-down approximate inference algorithm (Stern et al., 2017). Kitaev and Klein (2018) propose a self-attentive network to improve the parsing accuracy. They separate content and positional attentions and show the improvement. Zhang et al. (2020c) use a two-stage bracketing-then-labeling framework and replace the max-margin loss with the TreeCRF loss (Finkel et al., 2008). Xin et al. (2021) recently propose a recursive semi-Markov model, incorporating sibling factor scores into the score of a tree to explicitly model n-ary branching structures. Corro (2020) adapts span-based parsing to discontinuous constituency parsing and obtains the state-of-the-art result.

7 Conclusion

In this work, we have presented a headed-span-based method for projective dependency parsing. Our proposed method can utilize more subtree information and meanwhile enjoy global training and exact inference. Experiments show the competitive performance of our method in multiple datasets. In addition to its empirical competitiveness, we believe our work provides a novel perspective of projective dependency parsing and could lay the foundation for further algorithmic advancements in the future.

References

Emanuele Bugliarello and Naoaki Okazaki. 2020. Enhancing machine translation with dependency-aware self-attention. In *Proceedings of the 58th Annual*

Meeting of the Association for Computational Linguistics, pages 1618–1627, Online. Association for Computational Linguistics.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 957–961, Prague, Czech Republic. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

J. Cocke. 1969. Programming languages and their compilers: Preliminary notes.

Michael John Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 184–191, Santa Cruz, California, USA. Association for Computational Linguistics.

Caio Corro. 2020. Span-based discontinuous constituency parsing: a family of exact chart-based algorithms with time complexities from O(nˆ6) down to O(nˆ3). 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 *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Jason Eisner. 1997. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pages 54–65, Boston/Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 457–464, College Park, Maryland, USA. Association for Computational Linguistics.

Jason Eisner and Giorgio Satta. 2000. A faster parsing algorithm for Lexicalized Tree-Adjoining Grammars. In *Proceedings of the Fifth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+5)*, pages 79–84, Université Paris 7.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*.

Agnieszka Falenska and Jonas Kuhn. 2019. The (non-)utility of structural features in BiLSTM-based dependency parsers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 117–128, Florence, Italy. 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. 2021. Dependency parsing with bottom-up hierarchical pointer networks. *CoRR*, abs/2105.09611.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967, Columbus, Ohio. Association for Computational Linguistics.

Erick Fonseca and André F. T. Martins. 2020. Revisiting higher-order dependency parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8795–8800, Online. Association for Computational Linguistics.

Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2008. A deductive approach to dependency parsing. In *Proceedings of ACL-08: HLT*, pages 968–976, Columbus, Ohio. Association for Computational Linguistics.

Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2011. Dependency parsing schemata and mildly non-projective dependency parsing. *Computational Linguistics*, 37(3):541–586.

Carlos Gómez-Rodríguez, Tianze Shi, and Lillian Lee. 2018. Global transition-based non-projective dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2664–2675, Melbourne, Australia. Association for Computational Linguistics.

Matthew R. Gormley, Mark Dredze, and Jason Eisner. 2015. Approximation-aware dependency parsing by belief propagation. *Transactions of the Association for Computational Linguistics*, 3:489–501.

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

Liang Huang, Hao Zhang, and Daniel Gildea. 2005. Machine translation as lexicalized parsing with hooks. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 65–73, Vancouver, British Columbia. Association for Computational Linguistics.

Liang Huang, Hao Zhang, Daniel Gildea, and Kevin Knight. 2009. Binarization of synchronous context-free grammars. *Computational Linguistics*, 35(4):559–595.

Tao Ji, Yuanbin Wu, and Man Lan. 2019. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2475–2485, Florence, Italy. Association for Computational Linguistics.

Zhanming Jie and Wei Lu. 2019. Dependency-guided LSTM-CRF for named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3862–3872, Hong Kong, China. Association for Computational Linguistics.

Lifeng Jin, Linfeng Song, Yue Zhang, Kun Xu, Wei-Yun Ma, and Dong Yu. 2020. Relation extraction exploiting full dependency forests. 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 8034–8041. AAAI Press.

T. Kasami. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

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.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden. Association for Computational Linguistics.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 673–682, Portland, Oregon, USA. Association for Computational Linguistics.

Linlin Liu, Xiang Lin, Shafiq Joty, Simeng Han, and Lidong Bing. 2019. Hierarchical pointer net parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1007–1017, Hong Kong, China. Association for Computational Linguistics.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

Xuezhe Ma and Hai Zhao. 2012. Fourth-order dependency parsing. In *Proceedings of COLING 2012: Posters*, pages 785–796, Mumbai, India. The COLING 2012 Organizing Committee.

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.

André Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria. Association for Computational Linguistics.

André Martins, Noah Smith, Mário Figueiredo, and Pedro Aguiar. 2011. Dual decomposition with many overlapping components. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 238–249, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan. Association for Computational Linguistics.

Ryan McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, Trento, Italy. Association for Computational Linguistics.

Alireza Mohammadshahi and James Henderson. 2021. Recursive non-autoregressive graph-to-graph transformer for dependency parsing with iterative refinement. *Trans. Assoc. Comput. Linguistics*, 9:120–138.

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.

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.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 64–70, Geneva, Switzerland. COLING.

Fernando C. N. Pereira and David H. D. Warren. 1983. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, Cambridge, Massachusetts, USA. 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.

Sunita Sarawagi and William W. Cohen. 2004. Semi-markov conditional random fields for information extraction. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pages 1185–1192.

Giorgio Satta and Marco Kuhlmann. 2013. Efficient parsing for head-split dependency trees. *Transactions of the Association for Computational Linguistics*, 1:267–278.

Tianze Shi, Liang Huang, and Lillian Lee. 2017. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 12–23, Copenhagen, Denmark. 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.

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.

Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 1–8, Barcelona, Spain. Association for Computational Linguistics.

Benjamin Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. 2005. Learning structured prediction models: a large margin approach. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 896–903. ACM.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

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.

Xinyu Wang and Kewei Tu. 2020. Second-order neural dependency parsing with message passing and end-to-end training. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 93–99, Suzhou, China. 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.

D. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Inf. Control.*, 10 : 189 − −208.

Bo Zhang, Yue Zhang, Rui Wang, Zhenghua Li, and Min Zhang. 2020a. Syntax-aware opinion role labeling with dependency graph convolutional networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3249–3258, Online. Association for Computational Linguistics.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020b. 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. 2020c. 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.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii. Association for Computational Linguistics.

Xiaoqing Zheng. 2017. Incremental graph-based neural dependency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1655–1665, Copenhagen, Denmark. 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 Parsing algorithm

The parsing algorithm first computes all the chart items defined above and then recovers the parse tree from top down. For a given headed span, it finds the best segmentation of left child spans and right child spans, and then adds dependency arcs from the headword of the given headed span and the headword of each child span. Finding the best segmentation is similar to the inference procedure of the semi-Markov CRF model (Sarawagi and Cohen, 2004). Then we apply the same procedure to each child headed span (within the best segmentation) recursively. We also maintain the following backtrack points in order to recover the predicted projective tree:

$$B_{i,j} = \begin{cases} 1, & \alpha_{i,j} = \max\limits_{i<h\leq j} (\beta_{i,j,h}) \\ 0, & \alpha_{i,j} = \max\limits_{i<k<j} (\alpha_{i,k} + \alpha_{k,j}) \end{cases}$$

$$C_{i,j} = \arg\max_{i<k<j}(\alpha_{i,k} + \alpha_{k,j})$$

$$H_{i,j} = \arg\max_{i<h\leq j}(\beta_{i,j,h})$$

The parsing algorithm is formalized in Alg.1.

Algorithm 1 Inference algorithm for headed span-based projective dependency parsing

```
Require: Input sentence of length $n$
  Calculate all $\alpha, \beta, B, C, H$.
  arcs $\leftarrow$ {(ROOT $\rightarrow H_{0,n}$)}
  function FINDARC($i, j$)
    if $i + 1 = j$ then
      return {j}
    else if $B_{i,j} = 1$ then
      $h \leftarrow H_{i,j}$
      if i+1 < h < j then
        $L \leftarrow$ FINDARC($i, h - 1$)
        $R \leftarrow$ FINDARC($h, j$)
        Children $\leftarrow L \cup R$
      else if h = j then
        Children $\leftarrow$ FINDARC($i, j - 1$)
      else
        Children $\leftarrow$ FINDARC($i + 1, j$)
      end if
      for c in Children do
        arcs $\leftarrow$ arcs $\cup (h \rightarrow c)$
      end for
      return {h}
    else
      $c \leftarrow C_{i,j}$
      $L \leftarrow$ FINDARC($i, c$)
      $R \leftarrow$ FINDARC($c, j$)
      return $L \cup R$
    end if
  end function
  FINDARC($0, n$)
  return arcs
```

B Implementation details

We use "bert-large-cased" for PTB, "bert-base-chinese" for CTB, and "bert-multilingual-cased" for UD, so the dimension of the input BERT embedding is 1024, 768, and 768 respectively. The dimension of POS tag embedding is set to 100 for CTB and UD. The hidden size of BiLSTM is set to 1000. The hidden size of biaffine functions is set to 600 for scoring spans and arcs (used in our reimplemented Biaffine Parser), 300 for scoring labels. We add a dropout layer after the embedding layer, LSTM layers, and MLP layers. The dropout rate is set to 0.33. We use Adam (Kingma and Ba, 2015) as the optimizer with $\beta_1 = 0.9, \beta_2 = 0.9$ to train our model for 10 epochs. The maximal learning rate is $lr = 5e-5$ for BERT and $lr = 25e-5$ for other components. We linearly warmup the learning rate to the maximal value for the first epoch and gradually decay it to zero for the rest of the epochs. The value of gradient clipping is set to 5. We batch sentences of similar lengths to better utilize GPUs. The token number is 4000 for each batch, i.e., the sum of lengths of sentences is 4000.