Combining (Second-Order) Graph-Based and Headed-Span-Based Projective Dependency Parsing

Anonymous ACL submission

Abstract

Graph-based methods are popular in dependency parsing for decades, which decompose the score of a dependency tree into scores of dependency arcs. Recently, Yang and Tu (2021) propose a headed-span-based method that decomposes the score of a dependency tree into scores of headed spans. In this paper, we combine the two types of methods by considering both arc scores and headed-span scores, designing three scoring methods and the corresponding dynamic programming algorithms for joint inference. Experiments show the effectiveness of our proposed methods.

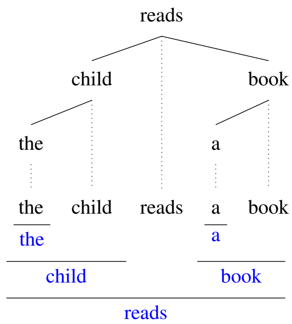


Figure 1: An example projective dependency parse tree with all its headed-spans.

1 Introduction

Dependency parsing is an important task in natural language processing. There are many methods to tackle projective dependency parsing. In this paper, we focus on two kinds of *global methods*: graph-based methods and headed-span-based methods. They both score all parse trees and globally find the highest scoring tree. The difference between the two is how they score dependency trees. The simplest first-order graph-based methods (McDonald et al., 2005) decompose the score of a dependency tree into the scores of dependency arcs. Second-order graph-based methods (McDonald and Pereira, 2006) additionally score adjacent siblings, i.e., pairs of adjacent arcs with a shared head. There are many other higher-order graph-based methods (Carreras, 2007; Koo and Collins, 2010; Ma and Zhao, 2012). In contrast, headed-span-based method (Yang and Tu, 2021) decomposes the score of a dependency tree into the scores of *headed-spans*: in a projective tree, a headed-span is a word-span pair such that the subtree rooted at the word covers the span in the surface order. Figure 1 shows an example projective parse tree and all its headed-spans. Because graph-based methods and the headed-span-based method take very different perspectives (i.e., composition of arcs vs. headed-spans) towards projective dependency parsing, it is theoretically interesting and intuitively beneficial to combine them.

We design three different ways to score a dependency tree with both arc and headed-span decomposition and propose the following dynamic programming algorithms for joint inference. We first design an $O(n^4)$ algorithm adapted from the Eisner-Satta algorithm (Eisner and Satta, 1999) to combine first-order graph-based and headed-span-based methods using the *hook trick*. Then we design two $O(n^3)$ algorithms adapted from the Eisner algorithm (Eisner, 1997) and its second-order extension (McDonald and Pereira, 2006) to combine first/second-order graph-based methods and the headed-span-based method using the *head-splitting trick*: we decompose the score of a headed-span into two terms assuming that the score of the left span boundary is independent to that of the right span boundary for each headword. We conduct extensive experiments on PTB, CTB, and UD, showing the effectiveness of our proposed methods.

2 Model

2.1 Scoring

Given an input sentences $x_1, ..., x_n$, we add <bos> (beginning of sentence) and <eos> (end of sentence) as x_0 and x_{n+1}. We apply mean-pooling at the last layer of BERT (Devlin et al., 2019) (i.e., averaging all subwords embeddings) to obtain the word-level embeddings e_i[1]. Then we feed $e_0, ..., e_{n+1}$ into a three-layer BiLSTM (Hochreiter and Schmidhuber, 1997) network to get $c_0, ..., c_{n+1}$, where $c_i = [f_i; b_i]$, f_i and b_i are the forward and backward hidden states of the last BiLSTM layer at position i respectively. We use $h_k = [f_k, b_{k+1}]$ to represent the kth boundary lying between x_k and x_{k+1}, and use $e_{i,j} = h_j - h_{i-1}$ to represent span (i, j) from position i to j inclusive where $1 \le i \le j \le n$. Then we compute:

- $s^{\text{arc}}_{i,j}$ (for arc $x_i \to x_j$, used in all three models) by feeding c_i, c_j into a deep biaffine function (Dozat and Manning, 2017).
- $s^{\text{span}}_{i,j,k}$ (for headed-span (i, j, k) where x_k is the headword of span (i, j), used in §3.1) by feeding $e_{i,j}, c_k$ to a deep biaffine function.
- $s^{\text{left}}_{k,i}$ and $s^{\text{right}}_{k,j}$ (for headed-span (i, j, k), used in §3.2 and §3.3) by feeding c_k, h_{i-1} and c_k, h_j into two different deep biaffine functions.
- $s^{\text{sib}}_{i,j,k}$ (for adjacent siblings $x_i \to \{x_j, x_k\}$ with $k < j < i$ or $i < j < k$, used in §3.3) by feeding c_i, c_k, c_j into a deep triaffine function (Zhang et al., 2020).

2.2 Learning

We decompose the training loss L into $L_{\text{parse}} + L_{\text{label}}$. For L_{parse}, we use the max-margin loss (Taskar et al., 2004):

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

where Δ measures the difference between the incorrect tree and gold tree y. Here we let Δ to be the Hamming distance (i.e., the total number of mismatches of arcs, sibling pairs, and (split) headed-spans depending on the setting). We use the same label loss L_{label} in Dozat and Manning (2017).

3 Parsing

We use the parsing-as-deduction framework (Pereira and Warren, 1983) to describe the parsing algorithms of our proposed models.

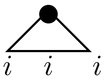
[1]For some datasets requiring the use of gold POS tags, we additionally concatenate the POS tag embedding to obtain e_i

3.1 $O(n^4)$ modified Eisner-Satta algorithm

In this case, we combine first-order graph-based parsing and headed-span-based parsing. The score of a dependency tree y is defined as:

$$s(y) = \sum_{(x_i \to x_j) \in y} s^{\text{arc}}_{i,j} + \sum_{(l_i, r_i, x_i) \in y} s^{\text{span}}_{l_i, r_i, i}$$

We design a dynamic programming algorithm adapted from the Eisner-Satta algorithm, which uses the hook trick to accelerate bilexicalized context-free parsing. The axiom items are



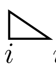
with initial score 0 and the deduction rules are listed in Figure 2. Unlike the original Eisner-Satta algorithm, we distinguish between "finished" spans and "unfinished" spans. An "unfinished" span can absorb a child span to form a larger span, while in a "finished" span, the headword has already generated all its children, so it cannot expand anymore and corresponds to a headed-span for the given headword. By explicitly distinguishing between "unfinished" spans and "finished" spans, we can incorporate headed-span scores s^{span} into parsing via the newly introduced rule `FINISH`. We then modify the rule `L-LINK` and `R-LINK` accordingly as only a "finished" span can be attached.

3.2 $O(n^3)$ modifed Eisner algorithm

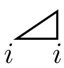
In order to decrease the inference time complexity from $O(n^4)$ to $O(n^3)$, we decompose $s^{\text{span}}_{l,r,i}$ into two terms:

$$s(y) = \sum_{(x_i \to x_j) \in y} s^{\text{arc}}_{i,j} + \sum_{(l_i, r_i, x_i) \in y} (s^{\text{left}}_{i,l_i} + s^{\text{right}}_{i,r_i})$$

and modify the Eisner algorithm accordingly. The axiom items are



and



with initial score 0 and the deduction rules are shown in the first two rows of Figure 3. Similar to the case in the previous subsection, the original Eisner algorithm does not distinguish between "finished" complete spans and "unfinished" complete spans. An "unfinished" complete span can absorb another complete span to form a larger incomplete span, while a "finished" complete span has no more child in the given direction and thus cannot expand anymore. We introduce new rules `L-FINISH` and `R-FINISH` to incorporate the left or right span boundary scores respectively, and adjust other rules accordingly.

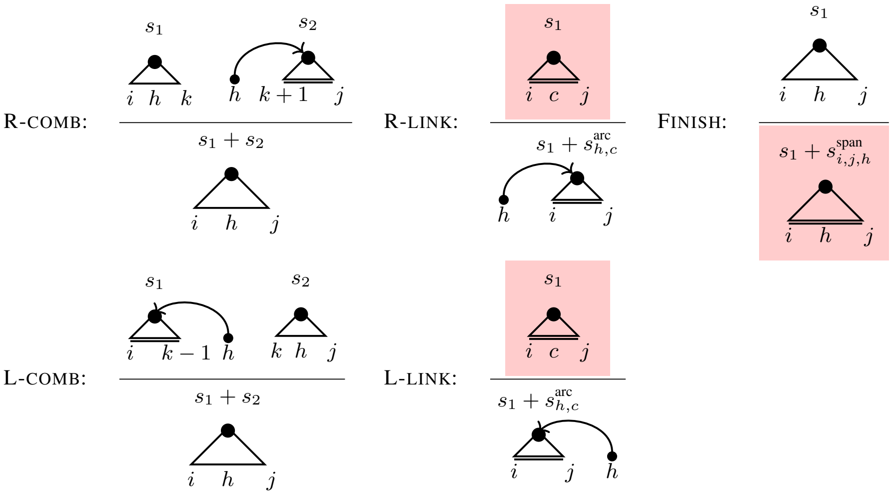


Figure 2: Deduction rules for our modified Eisner-Satta algorithm (Eisner and Satta, 1999). Our modifications are highlighted in red. All deduction items are annotated with their scores.

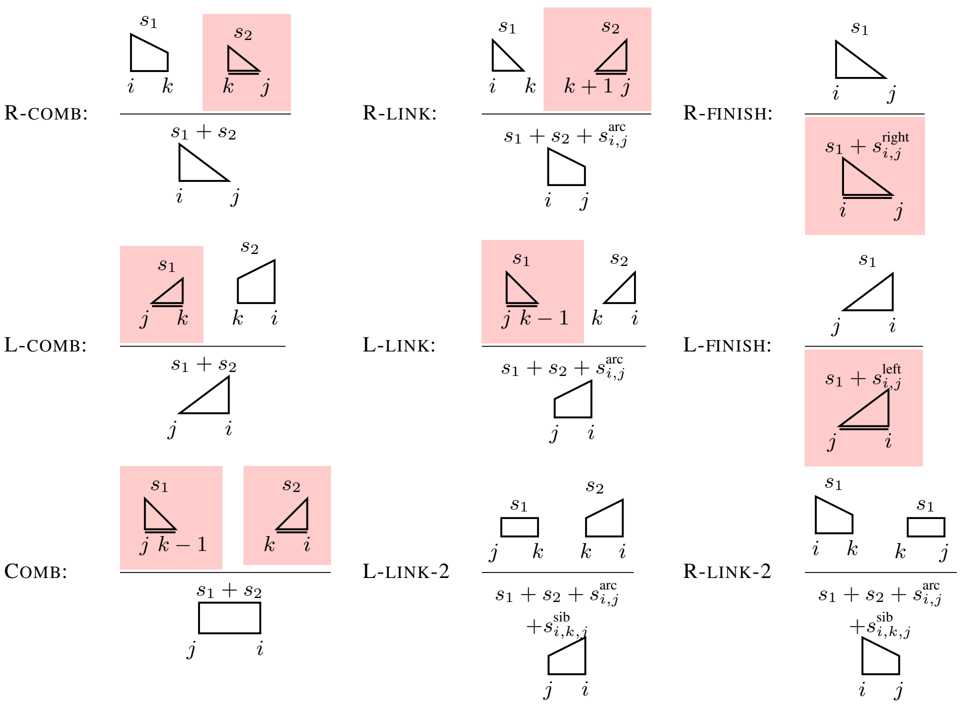


Figure 3: Deduction rules for our modified Eisner algorithm (Eisner, 1997) (first two rows) and its second-order extension (McDonald and Pereira, 2006) (all rows). Our modifications are highlighted in red. All deduction items are annotated with their scores.

3.3 $O(n^3)$ modified second-order Eisner algorithm

We further enhance the model with adjacent sibling information:

$$s(y) = \sum_{(x_i \to x_j) \in y} s^{\text{arc}}_{i,j} + \sum_{(x_i \to \{x_j, x_k\}) \in y} s^{\text{sib}}_{i,j,k} + \sum_{(l_i, r_i, x_i) \in y} (s^{\text{left}}_{i,l_i} + s^{\text{right}}_{i,r_i})$$

where for each adjacent sibling part $x_i \to \{x_j, x_k\}$, x_j and x_k are two adjacent dependents of x_i.

Similarly, we modify the second-order extension of the Eisner algorithm (McDonald and Pereira, 2006) by distinguishing between "unfinished" and "finished" complete spans. The additional deductive rules for second-order parsing are shown in the last row of Figure 3 and the length of the "unfinished" complete span is forced to be 1 in the rule `L-LINK` and `R-LINK`.

	bg	ca	cs	de	en	es	fr	it	nl	no	ro	ru	Avg
						+BERT$_{\text{multilingual}}$							
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
Span	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
1O+Span	91.44	94.54	92.68	85.75	91.23	93.84	91.67	**94.97**	**91.81**	94.35	87.17	94.49	91.99
1O+Span+Headsplit	91.46	94.53	92.63	85.78	91.25	93.77	**91.91**	94.88	91.59	94.18	87.45	94.47	91.99
Biaffine+2O+MM	91.58	94.48	**92.69**	85.72	91.28	93.80	91.89	94.88	91.30	94.23	87.55	**94.55**	92.00
2O+Span+Headsplit	**91.82**	**94.58**	92.59	85.65	91.28	**93.86**	91.80	94.75	91.50	**94.40**	**87.71**	94.51	**92.04**
						For reference							
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

Table 1: Labeled Attachment Score (LAS) on twelves languages in UD 2.2. We use ISO 639-1 codes to represent languages. † means reported by Yang and Tu (2021). MFVI2O: Wang and Tu (2020). Span: Yang and Tu (2021).

	PTB		**CTB**	
	UAS	LAS	UAS	LAS
	+BERT$_{\text{large}}$		+BERT$_{\text{base}}$	
Biaffine+MM†	97.22	95.71	93.18	92.10
Span	97.24	95.73	93.33	92.30
1O+Span	97.26	95.68	93.56	**92.49**
1O+Span+HeadSplit	**97.30**	**95.77**	93.46	92.42
Biaffine+2O+MM	97.28	95.73	93.42	92.34
2O+Span+HeadSplit	97.23	95.69	**93.57**	92.47
	For reference			
MFVI2O	96.91	95.34	92.55	91.69
HierPtr	97.01	95.48	92.65	91.47
	+XLNet$_{\text{large}}$		+BERT$_{\text{base}}$	
HPSG♭	97.20	95.72	-	-
HPSG+LAL♭	97.42	96.26	94.56	89.28

Table 2: Results on PTB and CTB. ♭ denotes use of additional constituency tree data and thus not comparable to our work. † denotes results reported by Yang and Tu (2021). HPSG: Zhou and Zhao (2019); HPSG+LAL: Mrini et al. (2020); HierPtr: Fernández-González and Gómez-Rodríguez (2021).

4 Experiments

4.1 Setup

We conduct experiments on in 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. We use the same data processing, evaluation methods, and hyperparameters as Yang and Tu (2021) for fair comparison and we refer readers to their paper for details due to the limit of space. We set the hidden size of the Triaffine function to 300 additionally. The reported results are averaged over three runs with different random seeds.

4.2 Main result

Table 1 and 2 show the results on UD, PTB and CTB respectively. We additionally reimplement `Biaffine+2O+MM` by replacing the TreeCRF loss of Zhang et al. (2020) with the max-margin loss for fair comparison. We refer to our proposed models as `1O+Span` (§3.1), `1O+Span+Headsplit` (§3.2), and `2O+Span+Headsplit` (§3.3) respectively.

We draw the following observations:

- Second-order information is still helpful even with powerful encoders (i.e., BERT). `Biaffine+2O+MM` outperforms `Biaffine+MM` in almost all cases.
- Combining first-order graph-based and headed-span-based methods is effective. Both `1O+Span` and `1O+Span+Headsplit` beat `Biaffine+MM` and `Span` in almost all cases. However, when incorporating span information into the second-order model, the improvement is slight: only +0.04 average LAS on UD, +0.13 LAS on CTB, and worse performance (-0.04 LAS) on PTB. We speculate that the utility of adjacent sibling information and span information is overlapping.
- Decomposing the headed-span scores is effective. `1O+Span+Headsplit` has a comparable performance to `1O+Span` while manages to decrease the time complexity from $O(n^4)$ to $O(n^3)$. We speculate that powerful encoders mitigate the issue of independent scoring.

5 Conclusion

In this paper we have presented three scoring and decoding methods to combine graph-based and headed-span-based methods for projective dependency parsing. Experiments show the effectiveness of our proposed methods.

References

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.

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.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2021. Dependency parsing with bottom-up hierarchical pointer networks. *CoRR*, abs/2105.09611.

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

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.

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.

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 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.

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.

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.

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.

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.

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.

Songlin Yang and Kewei Tu. 2021. Headed span-based projective dependency parsing.

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

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.