PTB Graph Parsing with Tree Approximation

Yoshihide Kato and **Shigeki Matsubara**
Information & Communications, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8601 Japan
yoshihide@icts.nagoya-u.ac.jp

Abstract

The Penn Treebank (PTB) represents syntactic structures as graphs due to nonlocal dependencies. This paper proposes a method that approximates PTB graph-structured representations by trees. By our approximation method, we can reduce nonlocal dependency identification and constituency parsing into single tree-based parsing. An experimental result demonstrates that our approximation method with an off-the-shelf tree-based constituency parser significantly outperforms the previous methods in nonlocal dependency identification.

1 Introduction

In the Penn Treebank (PTB) (Marcus et al., 1993), syntactic structures are represented as graphs due to nonlocal dependencies, which capture syntactic discontinuities. This paper proposes a method that approximates PTB graph-structured representations by trees. By our approximation method, we can reduce nonlocal dependency identification and constituency parsing into single tree-based parsing. The information loss of our approximation method is slight, and we can easily recover original PTB graphs from the output of a parser trained using the approximated ones. An experimental result demonstrates that our approximation method with an off-the-shelf tree-based parser significantly outperforms the previous nonlocal dependency identification methods.

2 Nonlocal Dependency Identification

This section explains nonlocal dependencies in the PTB, and summarizes previous work on nonlocal dependency identification.

2.1 Nonlocal dependency in PTB

In the PTB, a nonlocal dependency is represented as an edge. One node is called an *empty element*, which is a covert element in the syntactic representation. The other is called a *filler*. PTB's syntactic representations are graph-structured, while its constituency structures are represented by trees. Below, a syntactic representation in the PTB is called a *PTB graph*. The left graph in Figure 1 is an example of PTB graph. The empty elements are labelled with `-NONE-`. The terminal symbols such as `0` and ∗`T`∗ designate their types of the empty elements. `0` and ∗`T`∗ represent a zero relative pronoun and a trace of wh-movement, respectively. If a terminal symbol of an empty element is indexed with a number, its corresponding filler exists in the PTB graph, and is indexed with the same number. For example, the empty element of type ∗`T`∗ is indexed with 1 and it has the corresponding filler `WHNP-1`. For more details about PTB nonlocal dependencies, we refer readers to (Bies et al., 1995).

2.2 Previous Work

Most PTB-based parsers deal with the trees obtained by removing nonlocal dependencies and empty elements (we call such trees *PTB trees*). While such parsers are simple, efficient and accurate, they cannot handle nonlocal dependencies. To fill this gap, several methods have been proposed so far. They can be classified into the following two categories: the methods that introduce special operations handling nonlocal dependencies or empty elements into parsing algorithm (Dienes and Dubey, 2003; Schmid, 2006; Cai et al., 2011; Evang and Kallmeyer, 2011; Maier, 2015; Kato and Matsubara, 2016; Hayashi and Nagata, 2016; Kummerfeld and Klein, 2017), and the ones that recover PTB graphs from PTB trees generated by a parser (Johnson, 2002; Campbell, 2004; Levy and Manning, 2004). The former approach is required to design a parsing model that is suitable for the algorithm. In the latter post-processing approach, the pre-processing parser cannot reflect

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 5344–5349
Florence, Italy, July 28 - August 2, 2019.

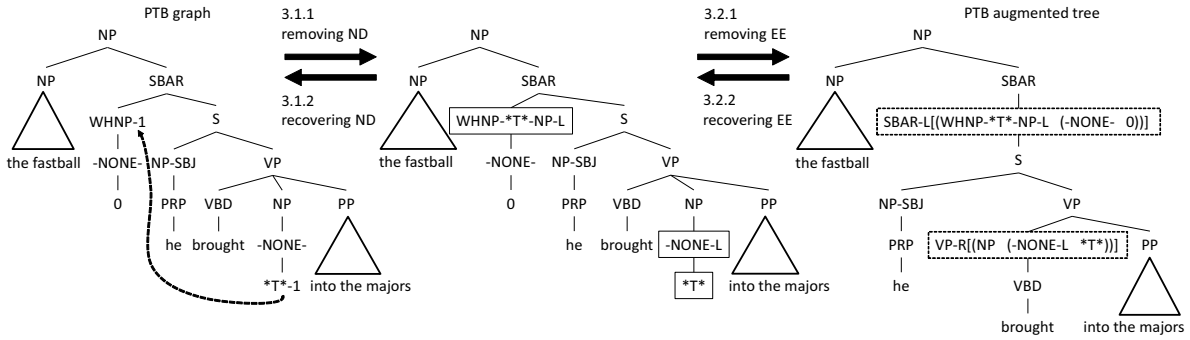

Figure 1: PTB graph and PTB augmented tree.

the information about nonlocal dependencies.

3 Tree Approximation of PTB Graphs

This section proposes a new approach of nonlocal dependency identification. We reduce nonlocal dependency identification and constituency parsing into single tree-based parsing. In our approach, a PTB graph is converted to a tree which approximately represents the PTB graph. The conversion consists of the following two steps:

Removing nonlocal dependency removes the edges between the empty elements and their fillers, and augments the labels of them. Augmented labels are used in order to recover the removed edges.

Removing empty element removes the empty elements and inserts new inner nodes that encode the empty elements.

We call the trees obtained by this conversion *PTB augmented trees*. Figure 1 shows an example of the conversion. Below, we explain each step in detail.

3.1 Removing nonlocal dependency

By removing the nonlocal dependency edges, a PTB graph becomes a tree. In order to approximately represent the edges in the resulting tree, we augment node labels in the annotation scheme identical to that proposed by Kato and Matsubara (2016). In this scheme, the labels of empty elements and their fillers are augmented with special tags. We first describe the annotation scheme, and then how to recover removed edges using augmented labels.

3.1.1 Annotation approximately representing nonlocal dependency

Algorithm 1 is the annotation algorithm of Kato and Matsubara (2016). Here, $\texttt{posi}(x, y)$ is the relative position of x for y and defined as follows:

$$\texttt{posi}(x, y) = \begin{cases} \texttt{A} & (x \text{ is an ancestor of } y) \\ \texttt{L} & (x \text{ occurs to the left of } y) \\ \texttt{R} & (x \text{ occurs to the right of } y) \end{cases}$$

The tag `OBJCTRL` enables us to distinguish between subject and object control.

Algorithm 1 Removing nonlocal dependency

$\texttt{type}(e)$ is the type of an empty element e.
$\texttt{cat}(x)$ is the category of x.
$\texttt{par}(x)$ is the parent of x.
$\texttt{SBJ}(x)$ means the label of x has the tag `SBJ`

Input: an empty element e and e's co-indexed filler f
remove the edge (e, f)
assign $\texttt{posi}(f, e)$ to e
if $\texttt{type}(e) = * \wedge \neg\texttt{SBJ}(f)$ **then**
  assign `OBJCTRL` to e
end if
if $\texttt{type}(e) \in \{\texttt{*EXP*}, \texttt{*ICH*}, \texttt{*RNR*}, \texttt{*T*}\}$ **then**
  assign $\texttt{type}(e)$, $\texttt{cat}(\texttt{par}(e))$ and $\texttt{posi}(f, e)$ to f
end if

For example, the left PTB graph in Figure 1 is converted to the middle tree. The boxes designate the augmented empty element and filler.

3.1.2 nonlocal dependency recovery

This section proposes a method of recovering nonlocal dependencies using the annotation described in the previous section. This method is based on heuristic rules, which are similar to, but simpler than those of Kato and Matsubara (2016).[1]

[1] Kato and Matsubara (2016) defined their recovery rules for intermediate results in parsing. This makes their rules somewhat complex.

node pattern	constraint imposed on the corresponding node x
$e =$ (-NONE-L $*$)	$\texttt{posi}(x, e) = \texttt{L} \wedge \texttt{c-cmd}(x, e) \wedge \texttt{SBJ}(x)$
$e =$ (-NONE-R $*$)	$\texttt{posi}(x, e) = \texttt{R} \wedge \texttt{c-cmd}(x, e) \wedge \texttt{SBJ}(x)$
$e =$ (-NONE-L-OBJCTRL $*$)	$\texttt{posi}(x, e) = \texttt{L} \wedge \texttt{c-cmd}(x, e) \wedge \texttt{cat}(x) \in \{\texttt{NP}, \texttt{PP}\} \wedge \texttt{cat}\big(\texttt{par}(x)\big) = \texttt{VP}$
$e =$ (-NONE-L $*$T$*$)	$\texttt{posi}(x, e) = \texttt{L} \wedge \texttt{c-cmd}(x, e) \wedge \texttt{match}(x, e)$
$e =$ (-NONE-A $*$T$*$)	$\exists y\big(\texttt{posi}(x, y) = \texttt{A} \wedge \texttt{posi}(y, e) = \texttt{A} \wedge \texttt{cat}(y) = \texttt{PRN}\big) \wedge \texttt{cat}\big(\texttt{par}(e)\big) = \texttt{cat}(x)$
$e =$ (-NONE-R $*$RNR$*$)	$\texttt{posi}(x, e) = \texttt{R} \wedge \texttt{c-cmd}(x, e) \wedge \texttt{match}(x, e)$
$e =$ (-NONE-L $*$ICH$*$)	$\texttt{posi}(x, e) = \texttt{L} \wedge \texttt{match}(x, e)$
$e =$ (-NONE-R $*$ICH$*$)	$\texttt{posi}(x, e) = \texttt{R} \wedge \texttt{match}(x, e)$
$f =$ (X-$*$EXP$*$-R $\cdots$)	$\texttt{posi}(f, x) = \texttt{R} \wedge \texttt{c-cmd}(x, f) \wedge x = (\texttt{NP}\ (\texttt{PRP it}))$

$\texttt{match}(f, e)$ means the type, the category and the position tag of a filler f are identical to those of an empty element e.

Table 1: The rules for nonlocal dependency recovery

A rule consists of a *node pattern* and a *constraint*. When there is a node that matches the pattern, we select the nearest node satisfying the constraint as its co-indexed node. Table 1 summarizes the rules.[2] Here, $\texttt{c-cmd}(x, y)$ is the syntactic relation called *c-command*[3] and holds iff the following condition (1) is satisfied:

$$\exists z.\big((z \text{ is a sibling of } x) \wedge \texttt{posi}(z, y) = \texttt{A}\big) \quad (1)$$

For example, the nonlocal dependency in Figure 1 can be recovered by the fourth rule in Table 1.

3.2 Removing empty elements

While the first step in the conversion can remove nonlocal dependency edges, the empty elements still remain. The second step removes empty elements and encodes them as inner nodes. By this conversion, parsing algorithm require no special operations handling empty elements.

3.2.1 Encoding empty elements

Algorithm 2 removes and encodes empty elements. For example, the middle tree in Figure 1 is converted to the right one. The dotted boxes designate the inner nodes encoding the empty elements. Here, note that [(NP (-NONE-L $*$T$*$))] is no more than a part of the label in the PTB augmented tree.

Kummerfeld and Klein (2017) represent empty elements in a similar way, but important difference exists. Our method keeps empty element positions (L and R) and no nonlocal dependencies, while they do not keep empty element positions and reserves nonlocal dependencies. Furthermore, while they require a specially-designed head rule to avoid constructing cyclic graphs in parsing, our method does not need head rules in the first place.

Algorithm 2 Encoding Empty element

$\texttt{null}(x)$ means all the leaves of x are empty elements.
$\texttt{node}(l, C)$ creates a node with a label l and children C.
$\texttt{encode}(x)$ converts the subtree rooted at x to a string.
$\texttt{label}(x)$ is the label of x.

Input: a node x
$\langle c_1, \ldots, c_n \rangle \leftarrow \texttt{children}(x)$
$i \leftarrow$ the leftmost position such that $\neg\texttt{null}(c_i)$
$C \leftarrow \langle c_i \rangle$
for j **from** $i + 1$ **to** n **do**
  if $\neg\texttt{null}(c_j)$ **then**
    $C \leftarrow C \cdot \langle c_j \rangle$
  else
    $C \leftarrow \langle \texttt{node}(\texttt{cat}(x) + \text{"R"} + \texttt{encode}(c_j), C) \rangle$
  end if
end for
for j **from** $i - 1$ **down to** 1 **do**
  $C \leftarrow \langle \texttt{node}(\texttt{cat}(x) + \text{"L"} + \texttt{encode}(c_j), C) \rangle$
end for
return $\texttt{node}(\texttt{label}(x), C)$

3.2.2 Recovering empty elements

Algorithm 2 is lossless and Algorithm 3 can recover the empty elements from the inner nodes inserted in Algorithm 2.

4 Experiment

To evaluate the performance of our proposed method, we conducted an experiment using the PTB. We used the Kitaev and Klein (henceforth K&K) parser (Kitaev and Klein, 2018a)[4]. The K&K parser is a state-of-the-art tree-based parser, which can use ELMo (Peters et al., 2018) or BERT[5] (Devlin et al., 2018) as external data. PTB graphs in the training (sections 02–21) and development (section 22) data were converted into PTB augmented trees by our tree approximation

[2] In the third rule, if $\texttt{cat}(x) = \texttt{PP}$, e is co-indexed with not x but x's child NP.

[3] Kato and Matsubara (2016) follow Chomsky's GB-theory (Chomsky, 1981) to use this relation, because it holds between co-indexed nodes in most cases. We also use this relation.

[4] https://github.com/nikitakit/self-attentive-parser

[5] The experiment using BERT is reported in (Kitaev and Klein, 2018b).

	Empty element detection (Fillers are ignored.)			Nonlocal dependency identification			Nonlocal dependency identification (Unindexed empty elements are excluded.)		
	pre.	rec.	F1	pre.	rec.	F1	pre.	rec.	F1
(Johnson, 2002)	85	74	79	73	63	68	–	–	–
(Dienes and Dubey, 2003)	–	–	–	81.5	68.7	74.6	–	–	–
(Campbell, 2004)	85.2	81.7	83.4	78.3	75.1	76.7	–	–	–
(Schmid, 2006)	86.0	82.3	84.1	–	–	–	81.7	73.5	77.4
(Cai et al., 2011)	90.1	79.5	84.5	–	–	–	–	–	–
(Hayashi and Nagata, 2016)	90.3	81.7	85.8	–	–	–	–	–	–
(Kato and Matsubara, 2016)	88.5	82.1	85.2	81.4	75.5	78.4	79.8	73.8	76.7
(Kummerfeld and Klein, 2017)	89.5	81.6	85.4	74.3	67.3	70.6	–	–	–
post-processing (using gold PTB trees)									
(Johnson, 2002)	93	83	88	80	70	75	–	–	–
(Campbell, 2004)	94.9	91.1	93.0	90.1	86.6	88.4	–	–	–
ours	92.6	87.7	90.1	88.1	83.4	85.7	88.4	81.1	84.6
ours (with ELMo)	94.2	90.3	92.3	89.9	86.2	88.0	90.4	84.1	87.2
ours (with BERT)	94.9	91.4	93.1	90.8	87.4	89.0	91.6	84.9	88.1

Table 2: Comparison for nonlocal dependency identification on the test data.

Algorithm 3 Recovering empty element

$\texttt{decode}(x)$ creates a tree by decoding a string assigned by `encode` and returns its root.

Input: a node x
$C \leftarrow \texttt{children}(x)$
$C' \leftarrow \langle\rangle$
while $C \neq \langle\rangle$ **do**
  pop the first element c from C
  if c is an inserted node and c has the tag `L` **then**
    $C \leftarrow \langle\texttt{decode}(c)\rangle \cdot \texttt{children}(c) \cdot C$
  else if c is an inserted node and has the tag `R` **then**
    $C \leftarrow \texttt{children}(c) \cdot \langle\texttt{decode}(c)\rangle \cdot C$
  else
    $C' \leftarrow C' \cdot \langle c\rangle$
  end if
end while
return $\texttt{node}(\texttt{label}(x), C')$

method[6], and a parsing model was trained using the PTB augmented trees. The hyperparameters for training were identical to those of Kitaev and Klein (2018a). We selected the model that maximizes the F1 score on the development data, where we treated the node labels of PTB augmented trees as constituent labels. For the test data (section 23), PTB graphs were recovered from the PTB augmented trees generated by the parser. The accuracy of the nonlocal dependency identification was evaluated by the metric proposed by Johnson (2002).

First, we evaluated the performance of our approximation method. We recovered PTB graphs from not the parser output but the gold PTB augmented trees in the development data. We obtained 99.5 F1 score in nonlocal dependency identification where unindexed empty elements were excluded. This result means that the information loss is slight in our approximation method.

Table 2 summarizes the performances of our system and previous ones. These results demonstrate that our system significantly outperforms the previous methods in nonlocal dependency identification. Although the main reason for this is because of the performance of the K&K parser, the important point is that our proposed approximation method enables us to use the K&K parser for the nonlocal dependency identification task. The previous methods that introduce additional operations cannot adopt such parser directly. On the other hand, although post-processing approach can use any parser in pre-processing, our approach outperforms the post-processing approach, even if the pre-processing parser is assumed to always generate gold PTB trees.

We converted PTB graphs into PTB trees to evaluate constituency parsing performance. Table 3 shows the F1 scores of our and the K&K parser. These results demonstrate that our tree approximation has little negative impact on the constituency parsing performance.

5 Conclusion

This paper proposes a conversion of PTB graphs into PTB augmented trees, which enables us to reduce nonlocal dependency identification and constituency parsing into single parsing. Our proposed conversion method can be easily combined

[6]The conversion code is available at https://github.com/yosihide/ptb2cf.

	pre.	rec.	F1
K&K	93.90	93.20	93.55
K&K (ELMo)	95.40	94.85	95.13
K&K (BERT)	96.03	95.51	95.77
Ours	93.84	92.78	93.31
Ours (ELMo)	95.27	94.70	94.99
Ours (BERT)	96.04	95.36	95.70

Table 3: Comparison for constituency parsing performance on the test data.

with other tree-based parsers. We can expect that the evolution of tree-based parsing technology makes our approach improve the accuracy of nonlocal dependency identification.

Acknowledgements

This research was partially supported by the Grant-in-Aid for Scientific Research (C) (17K00303) of JSPS.

References

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. *Bracketing guidelines for Treebank II style Penn Treebank project*. University of Pennsylvania.

Shu Cai, David Chiang, and Yoav Goldberg. 2011. Language-independent parsing with empty elements. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 212–216, Portland, Oregon, USA.

Richard Campbell. 2004. Using linguistic principles to recover empty categories. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 645–652, Barcelona, Spain.

Noam Chomsky. 1981. *Lectures on government and binding: The Pisa lectures*. Walter de Gruyter.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Péter Dienes and Amit Dubey. 2003. Antecedent recovery: Experiments with a trace tagger. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 33–40.

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.

Katsuhiko Hayashi and Masaaki Nagata. 2016. Empty element recovery by spinal parser operations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 95–100, Berlin, Germany.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 136–143, Philadelphia, Pennsylvania, USA.

Yoshihide Kato and Shigeki Matsubara. 2016. Transition-based left-corner parsing for identifying PTB-style nonlocal dependencies. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 930–940, Berlin, Germany.

Nikita Kitaev and Dan Klein. 2018a. 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.

Nikita Kitaev and Dan Klein. 2018b. Multilingual constituency parsing with self-attention and pre-training. *CoRR*, abs/1812.11760.

Jonathan K. Kummerfeld and Dan Klein. 2017. Parsing with traces: An $O(n^4)$ algorithm and a structural representation. *Transactions of the Association for Computational Linguistics*, 5:441–454.

Roger Levy and Christopher Manning. 2004. Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 327–334, Barcelona, Spain.

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.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):310–330.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237.

Helmut Schmid. 2006. Trace prediction and recovery with unlexicalized PCFGs and slash features. In *Proceedings of the 21st International Conference on*

Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, pages 177–184, Sydney, Australia.