Language-Independent Parsing with Empty Elements

Shu Cai and **David Chiang**
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292
{shucai,chiang}@isi.edu

Yoav Goldberg
Ben Gurion University of the Negev
Department of Computer Science
POB 653 Be'er Sheva, 84105, Israel
yoavg@cs.bgu.ac.il

Abstract

We present a simple, language-independent method for integrating recovery of empty elements into syntactic parsing. This method outperforms the best published method we are aware of on English and a recently published method on Chinese.

1 Introduction

Empty elements in the syntactic analysis of a sentence are markers that show where a word or phrase might otherwise be expected to appear, but does not. They play an important role in understanding the grammatical relations in the sentence. For example, in the tree of Figure 2a, the first empty element (*) marks where *John* would be if *believed* were in the active voice (*someone believed...*), and the second empty element (*T*) marks where *the man* would be if *who* were not fronted (*John was believed to admire who?*).

Empty elements exist in many languages and serve different purposes. In languages such as Chinese and Korean, where subjects and objects can be dropped to avoid duplication, empty elements are particularly important, as they indicate the position of dropped arguments. Figure 1 gives an example of a Chinese parse tree with empty elements. The first empty element (*pro*) marks the subject of the whole sentence, a pronoun inferable from context. The second empty element (*PRO*) marks the subject of the dependent VP (*shíshī fǎlǜ tiáowén*).

The Penn Treebanks (Marcus et al., 1993; Xue et al., 2005) contain detailed annotations of empty elements. Yet most parsing work based on these resources has ignored empty elements, with some notable exceptions. Johnson (2002) studied empty-element recovery in English, followed by several others (Dienes and Dubey, 2003; Campbell, 2004; Gabbard et al., 2006); the best results we are aware of are due to Schmid (2006). Recently, empty-element recovery for Chinese has begun to receive attention: Yang and Xue (2010) treat it as classification problem, while Chung and Gildea (2010) pursue several approaches for both Korean and Chinese, and explore applications to machine translation.

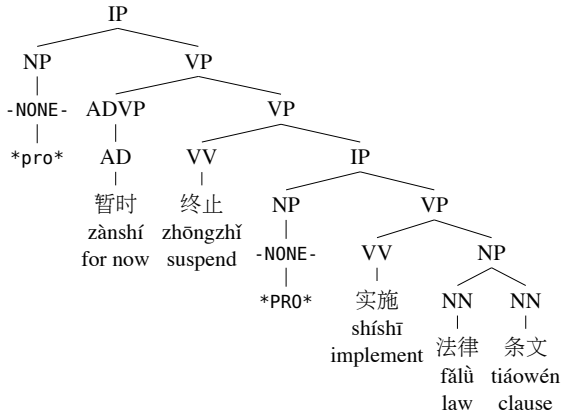


Figure 1: Chinese parse tree with empty elements marked. The meaning of the sentence is, "Implementation of the law is temporarily suspended."

Our intuition motivating this work is that empty elements are an integral part of syntactic structure, and should be constructed jointly with it, not added in afterwards. Moreover, we expect empty-element recovery to improve as the parsing quality improves. Our method makes use of a strong syntactic model, the PCFGs with latent annotation of Petrov et al. (2006), which we extend to predict empty cate-

Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics:shortpapers, pages 212–216,
Portland, Oregon, June 19-24, 2011.

gories by the use of *lattice parsing*. The method is language-independent and performs very well on both languages we tested it on: for English, it outperforms the best published method we are aware of (Schmid, 2006), and for Chinese, it outperforms the method of Yang and Xue (2010).[1]

2 Method

Our method is fairly simple. We take a state-of-the-art parsing model, the Berkeley parser (Petrov et al., 2006), train it on data with explicit empty elements, and test it on word lattices that can nondeterministically insert empty elements anywhere. The idea is that the state-splitting of the parsing model will enable it to learn where to expect empty elements to be inserted into the test sentences.

Tree transformations Prior to training, we alter the annotation of empty elements so that the terminal label is a consistent symbol (ϵ), the preterminal label is the type of the empty element, and `-NONE-` is deleted (see Figure 2b). This simplifies the lattices because there is only one empty symbol, and helps the parsing model to learn dependencies between nonterminal labels and empty-category types because there is no intervening `-NONE-`.

Then, following Schmid (2006), if a constituent contains an empty element that is linked to another node with label X, then we append $/X$ to its label. If there is more than one empty element, we process them bottom-up (see Figure 2b). This helps the parser learn to expect where to find empty elements. In our experiments, we did this only for elements of type `*T*`. Finally, we train the Berkeley parser on the preprocessed training data.

Lattice parsing Unlike the training data, the test data does not mark any empty elements. We allow the parser to produce empty elements by means of *lattice-parsing* (Chappelier et al., 1999), a generalization of CKY parsing allowing it to parse a word-lattice instead of a predetermined list of terminals. Lattice parsing adds a layer of flexibility to existing parsing technology, and allows parsing in situations where the yield of the tree is not known in advance. Lattice parsing originated in the speech processing community (Hall, 2005; Chappelier et al., 1999), and was recently applied to the task of joint clitic-segmentation and syntactic-parsing in Hebrew (Goldberg and Tsarfaty, 2008; Goldberg and Elhadad, 2011) and Arabic (Green and Manning, 2010). Here, we use lattice parsing for empty-element recovery.

We use a modified version of the Berkeley parser which allows handling lattices as input.[2] The modification is fairly straightforward: Each lattice arc correspond to a lexical item. Lexical items are now indexed by their start and end states rather than by their sentence position, and the initialization procedure of the CKY chart is changed to allow lexical items of spans greater than 1. We then make the necessary adjustments to the parsing algorithm to support this change: trying rules involving preterminals even when the span is greater than 1, and not relying on span size for identifying lexical items.

At test time, we first construct a lattice for each test sentence that allows 0, 1, or 2 empty symbols (ϵ) between each pair of words or at the start/end of the sentence. Then we feed these lattices through our lattice parser to produce trees with empty elements. Finally, we reverse the transformations that had been applied to the training data.

3 Evaluation Measures

Evaluation metrics for empty-element recovery are not well established, and previous studies use a variety of metrics. We review several of these here and additionally propose a unified evaluation of parsing and empty-element recovery.[3]

If A and B are multisets, let $A(x)$ be the number of occurrences of x in A, let $|A| = \sum_x A(x)$, and let $A \cap B$ be the multiset such that $(A \cap B)(x) = \min(A(x), B(x))$. If T is the multiset of "items" in the trees being tested and G is the multiset of "items" in the gold-standard trees, then

$$\text{precision} = \frac{|G \cap T|}{|T|} \qquad \text{recall} = \frac{|G \cap T|}{|G|}$$

$$\text{F}_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

[1] Unfortunately, not enough information was available to carry out comparison with the method of Chung and Gildea (2010).

[2] The modified parser is available at `http://www.cs.bgu.ac.il/~yoavg/software/blatt/`

[3] We provide a scoring script which supports all of these evaluation metrics. The code is available at `http://www.isi.edu/~chiang/software/eevalb.py` .

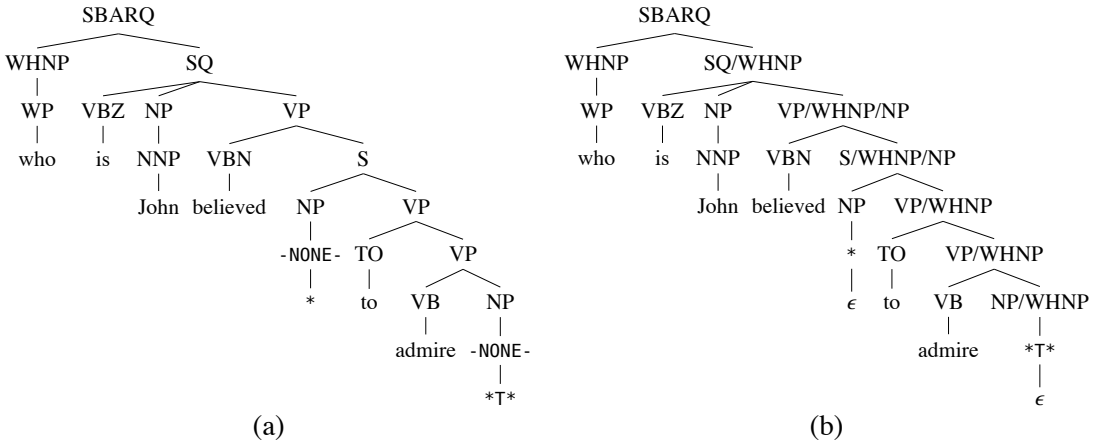


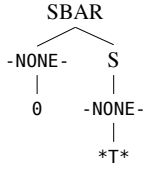
Figure 2: English parse tree with empty elements marked. (a) As annotated in the Penn Treebank. (b) With empty elements reconfigured and slash categories added.

where "items" are defined differently for each metric, as follows. Define a *nonterminal* node, for present purposes, to be a node which is neither a terminal nor preterminal node.

The standard PARSEVAL metric (Black et al., 1991) counts *labeled nonempty brackets*: items are (X, i, j) for each nonempty nonterminal node, where X is its label and i, j are the start and end positions of its span.

Yang and Xue (2010) simply count *unlabeled empty elements*: items are (i, i) for each empty element, where i is its position. If multiple empty elements occur at the same position, they only count the last one.

The metric originally proposed by Johnson (2002) counts *labeled empty brackets*: items are $(X/t, i, i)$ for each empty nonterminal node, where X is its label and t is the type of the empty element it dominates, but also (t, i, i) for each empty element not dominated by an empty nonterminal node.[4] The following structure has an empty nonterminal dominating two empty elements:



Johnson counts this as $(\text{SBAR}, i, i), (\text{S}/\texttt{*T*}, i, i)$; Schmid (2006) counts it as a single $(\text{SBAR-S}/\texttt{*T*}, i, i)$.[5] We tried to follow Schmid in a generic way: we collapse any vertical chain of empty nonterminals into a single nonterminal.

In order to avoid problems associated with cases like this, we suggest a pair of simpler metrics. The first is to count *labeled empty elements*, i.e., items are (t, i, i) for each empty element, and the second, similar in spirit to SParseval (Roark et al., 2006), is to count *all labeled brackets*, i.e., items are (X, i, j) for each nonterminal node (whether nonempty or empty). These two metrics, together with part-of-speech accuracy, cover all possible nodes in the tree.

4 Experiments and Results

English As is standard, we trained the parser on sections 02–21 of the Penn Treebank Wall Street Journal corpus, used section 00 for development, and section 23 for testing. We ran 6 cycles of training; then, because we were unable to complete the 7th split-merge cycle with the default setting of merging 50% of splits, we tried increasing merges to 75% and ran 7 cycles of training. Table 1 presents our results. We chose the parser settings that gave the best *labeled empty elements* F_1 on the dev set, and used these settings for the test set. We outperform the state of the art at recovering empty elements, as well as achieving state of the art accuracy at recovering phrase structure.

[4] This happens in the Penn Treebank for types `*U*` and `0`, but never in the Penn Chinese Treebank except by mistake.

[5] This difference is not small; scores using Schmid's metric are lower by roughly 1%. There are other minor differences in Schmid's metric which we do not detail here.

Section	System	Labeled Empty Brackets P	R	F_1	Labeled Empty Elements P	R	F_1	All Labeled Brackets P	R	F_1
00	Schmid (2006)	88.3	82.9	85.5	89.4	83.8	86.5	87.1	85.6	86.3
	split 5× merge 50%	91.0	79.8	85.0	93.1	81.8	87.1	90.4	88.7	89.5
	split 6× merge 50%	91.9	81.1	86.1	93.6	82.4	87.6	90.4	89.1	89.7
	split 6× merge 75%	92.7	80.7	86.3	94.6	82.0	87.9	90.3	88.5	89.3
	split 7× merge 75%	91.0	80.4	85.4	93.2	82.1	87.3	90.5	88.9	89.7
23	Schmid (2006)	86.1	81.7	83.8	87.9	83.0	85.4	86.8	85.9	86.4
	split 6× merge 75%	90.1	79.5	84.5	92.3	80.9	86.2	90.1	88.5	89.3

Table 1: Results on Penn (English) Treebank, Wall Street Journal, sentences with 100 words or fewer.

Task	System	Unlabeled Empty Elements P	R	F_1	Labeled Empty Elements P	R	F_1	All Labeled Brackets P	R	F_1
Dev	split 5× merge 50%	82.5	58.0	68.1	72.6	51.8	60.5	84.6	80.7	82.6
	split 6× merge 50%	76.4	60.5	67.5	68.2	55.1	60.9	83.2	81.3	82.2
	split 7× merge 50%	74.9	58.7	65.8	65.9	52.5	58.5	82.7	81.1	81.9
Test	Yang and Xue (2010)	80.3	57.9	63.2						
	split 6× merge 50%	74.0	61.3	67.0	66.0	54.5	58.6	82.7	80.8	81.7

Table 2: Results on Penn (Chinese) Treebank.

Chinese We also experimented on a subset of the Penn Chinese Treebank 6.0. For comparability with previous work (Yang and Xue, 2010), we trained the parser on sections 0081–0900, used sections 0041–0080 for development, and sections 0001–0040 and 0901–0931 for testing. The results are shown in Table 2. We selected the 6th split-merge cycle based on the *labeled empty elements* F_1 measure. The *unlabeled empty elements* column shows that our system outperforms the baseline system of Yang and Xue (2010). We also analyzed the empty-element recall by type (Table 3). Our system outperformed that of Yang and Xue (2010) especially on `*pro*`, used for dropped arguments, and `*T*`, used for relative clauses and topicalization.

Type	Total Gold	Correct YX	Correct Ours	Recall YX	Recall Ours
`*pro*`	290	125	159	43.1	54.8
`*PRO*`	299	196	199	65.6	66.6
`*T*`	578	338	388	58.5	67.1
`*RNR*`	32	20	15	62.5	46.9
`*OP*`	134	20	65	14.9	48.5
*	19	5	3	26.3	15.8

Table 3: Recall on different types of empty categories. YX = (Yang and Xue, 2010), Ours = split 6×.

5 Discussion and Future Work

The empty-element recovery method we have presented is simple, highly effective, and fully integrated with state of the art parsing. We hope to exploit cross-lingual information about empty elements in machine translation. Chung and Gildea (2010) have shown that such information indeed helps translation, and we plan to extend this work by handling more empty categories (rather than just `*pro*` and `*PRO*`), and to incorporate them into a syntax-based translation model instead of a phrase-based model.

We also plan to extend our work here to recover coindexation information (links between a moved element and the trace which marks the position it was moved from). As a step towards shallow semantic analysis, this may further benefit other natural language processing tasks such as machine translation and summary generation.

Acknowledgements

We would like to thank Slav Petrov for his help in running the Berkeley parser, and Yaqin Yang, Bert

Xue, Tagyoung Chung, and Dan Gildea for their answering our many questions. We would also like to thank our colleagues in the Natural Language Group at ISI for meaningful discussions and the anonymous reviewers for their thoughtful suggestions. This work was supported in part by DARPA under contracts HR0011-06-C-0022 (subcontract to BBN Technologies) and DOI-NBC N10AP20031, and by NSF under contract IIS-0908532.

References

E. Black, S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proc. DARPA Speech and Natural Language Workshop*.

Richard Campbell. 2004. Using linguistic principles to recover empty categories. In *Proc. ACL*.

J.-C. Chappelier, M. Rajman, R. Aragües, and A. Rozenknop. 1999. Lattice parsing for speech recognition. In *Proc. Traitement Automatique du Langage Naturel (TALN)*.

Tagyoung Chung and Daniel Gildea. 2010. Effects of empty categories on machine translation. In *Proc. EMNLP*.

Péter Dienes and Amit Dubey. 2003. Antecedent recovery: Experiments with a trace tagger. In *Proc. EMNLP*.

Ryan Gabbard, Seth Kulick, and Mitchell Marcus. 2006. Fully parsing the Penn Treebank. In *Proc. NAACL HLT*.

Yoav Goldberg and Michael Elhadad. 2011. Joint Hebrew segmentation and parsing using a PCFG-LA lattice parser. In *Proc. of ACL*.

Yoav Goldberg and Reut Tsarfaty. 2008. A single generative model for joint morphological segmentation and syntactic parsing. In *Proc. of ACL*.

Spence Green and Christopher D. Manning. 2010. Better Arabic parsing: Baselines, evaluations, and analysis. In *Proc of COLING-2010*.

Keith B. Hall. 2005. *Best-first word-lattice parsing: techniques for integrated syntactic language modeling*. Ph.D. thesis, Brown University, Providence, RI, USA.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proc. ACL*.

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

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proc. COLING-ACL*.

Brian Roark, Mary Harper, Eugene Charniak, Bonnie Dorr, Mark Johnson, Jeremy G. Kahn, Yang Liu, Mari Ostendorf, John Hale, Anna Krasnyanskaya, Matthew Lease, Izhak Shafran, Matthew Snover, Robin Stewart, and Lisa Yung. 2006. SParseval: Evaluation metrics for parsing speech. In *Proc. LREC*.

Helmut Schmid. 2006. Trace prediction and recovery with unlexicalized PCFGs and slash features. In *Proc. COLING-ACL*.

Nianwen Xue, Fei Xia, Fu-dong Chiou, and Martha Palmer. 2005. The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Yaqin Yang and Nianwen Xue. 2010. Chasing the ghost: recovering empty categories in the Chinese Treebank. In *Proc. COLING*.