Ckylark: A More Robust PCFG-LA Parser

Yusuke Oda **Graham Neubig** **Sakriani Sakti** **Tomoki Toda** **Satoshi Nakamura**
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
{oda.yusuke.on9, neubig, ssakti, tomoki, s-nakamura}@is.naist.jp

Abstract

This paper describes Ckylark, a PCFG-LA style phrase structure parser that is more robust than other parsers in the genre. PCFG-LA parsers are known to achieve highly competitive performance, but sometimes the parsing process fails completely, and no parses can be generated. Ckylark introduces three new techniques that prevent possible causes for parsing failure: outputting intermediate results when coarse-to-fine analysis fails, smoothing lexicon probabilities, and scaling probabilities to avoid underflow. An experiment shows that this allows millions of sentences can be parsed without any failures, in contrast to other publicly available PCFG-LA parsers. Ckylark is implemented in C++, and is available open-source under the LGPL license.[1]

1 Introduction

Parsing accuracy is important. Parsing accuracy has been shown to have a significant effect on downstream applications such as textual entailment (Yuret et al., 2010) and machine translation (Neubig and Duh, 2014), and most work on parsing evaluates accuracy to some extent. However, one element that is equally, or perhaps even more, important from the view of downstream applications is parser *robustness*, or the ability to return at least some parse regardless of the input. Every failed parse is a sentence for which downstream applications have no chance of even performing processing in the normal way, and application developers must perform special checks that detect these sentences and either give up entirely, or fall back to some alternative processing scheme.

Among the various methods for phrase-structure parsing, the probabilistic context free grammar with latent annotations (PCFG-LA, (Matsuzaki et al., 2005; Petrov et al., 2006)) framework is among the most popular for several reasons. The first is that it boasts competitive accuracy, both in intrisinic measures such as F1-score on the Penn Treebank (Marcus et al., 1993), and extrinsic measures (it achieved the highest textual entailment and machine translation accuracy in the papers cited above). The second is the availablity of easy-to-use tools, most notably the Berkeley Parser,[2] but also including Egret,[3] and BUBS Parser.[4]

However, from the point of view of robustness, existing tools for PCFG-LA parsing leave something to be desired; to our knowledge, all existing tools produce a certain number of failed parses when run on large data sets. In this paper, we introduce Ckylark, a new PCFG-LA parser specifically designed for robustness. Specifically, Ckylark makes the following contributions:

- Based on our analysis of three reasons why conventional PCFG-LA parsing models fail (Section 2), Ckylark implements three improvements over the conventional PCFG-LA parsing method to remedy these problems (Section 3).

[1] http://github.com/odashi/ckylark

[2] https://code.google.com/p/berkeleyparser/

[3] https://code.google.com/p/egret-parser/

[4] https://code.google.com/p/bubs-parser/

- An experimental evaluation (Section 4) shows that Ckylark achieves competitive accuracy with other PCFG-LA parsers, and can robustly parse large datasets where other parsers fail.
- Ckylark is implemented in C++, and released under the LGPL license, allowing for free research or commercial use. It is also available in library format, which means that it can be incorporated directly into other programs.

2 Failure of PCFG-LA Parsing

The basic idea behind PCFG-LA parsing is that traditional tags in the Penn Treebank are too coarse, and more accurate grammars can be achieved by automatically splitting tags into finer latent classes. For example, the English words "a" and "the" are classified as determiners (DT), but these words are used in different contexts, so can be assigned different latent classes. The most widely used method to discover these classes uses the EM algorithm to estimate latent classes in stages (Petrov et al., 2006). This method generates hierarchical grammars including relationships between each latent class in a tree structure, and the number of latent classes increases exponentially for each level of the grammar. The standard search method for PCFG grammars is based on the CKY algorithm. However, simply applying CKY directly to the "finest" grammar is not realistic, as the complexity of CKY is propotional to the polynomial of the number of latent classes. To avoid this problem, Petrov et al. (2006) start the analysis with the "coarse" grammar and apply pruning to reduce the amount of computation. This method is called coarse-to-fine analysis. However, this method is not guaranteed to successfully return a parse tree. We describe three reasons why PCFG-LA parsing fails below.

Failure of pruning in coarse-to-fine analysis Coarse-to-fine analysis prunes away candidate paths in the parse graph when their probability is less than a specific threshold ϵ. This pruning can cause problems in the case that all possible paths are pruned and the parser cannot generate any parse tree at the next step.

Inconsistency between model and target If we parse sentences with syntax that diverges from the training data, the parser may fail because the parser needs rules which are not included in the grammar. For example, symbols "(" and ")" become a part of phrase "PRN" only if both of them and some phrase "X" exist with the order "(X)." One approach for this problem is to use smoothed grammars (Petrov et al., 2006), but this increases the size of the probability table needed to save such a grammar.

Underflow of probabilities Parsers calculate joint probabilities of each parse tree, and this value decreases exponentially according to the length of the input sequence. As a result, numerical underflow sometimes occurs if the parser tries to parse longer sentences. Using calculations in logarithmic space is one approach to avoid underflow. However, this approach requires log and exponent operations, which are more computationally expensive than sums or products.

The failure of pruning is a unique problem for PCFG-LA, and the others are general problems of parsing methods based on PCFG. In the next section, we describe three improvements over the basic PCFG-LA method that Ckylark uses to avoid these problems.

3 Improvements of the Parsing Method

3.1 Early Stopping in Coarse-to-fine Analysis

While coarse-to-fine analysis generally uses the parsing result of the finest grammar as output, intermediate grammars also can generate parse trees. Thus, we can use these intermediate results instead of the finest result when parsing fails at later stages. Algorithm 1 shows this "stopping" approach. This approach can avoid all errors due to coarse-to-fine pruning, except in the case of failure during the parsing with the first grammar due to problems of the model itself.

3.2 Lexicon Smoothing

Next, we introduce lexicon smoothing using the probabilities of unknown words at parsing time. This approach not only reduces the size of the grammar, but also allows for treatment of any word as

Algorithm 1 Stopping coarse-to-fine analysis

Require: $\boldsymbol{w}$: input sentence
Require: $G_0, \cdots, G_L$: coarse-to-fine grammars
  $T_{-1} \leftarrow$ nil
  $P_0 \leftarrow \{\}$ ▷ pruned pathes
  for $l \leftarrow 0 \,..\, L$ **do**
    $T_l, P_{l+1} \leftarrow \text{parse\_and\_prune}(\boldsymbol{w}; G_l, P_l)$
    if $T_l = \text{nil}$ **then** ▷ parsing failed
      return T_{l-1} ▷ return intermediate result
    end if
  end for
  return T_L ▷ parsing succeeded

"unknown" if the word appears in an unknown syntactic content. Equation (1) shows the smoothed lexicon probability:

$$P'(X \to w) \equiv (1-\lambda)P(X \to w) + \lambda P(X \to w_{unk}), \quad (1)$$

where X is any pre-terminal (part-of-speech) symbol in the grammar, w is any word, and w_{unk} is the unknown word. λ is an interpolation factor between w and w_{unk}, and should be small enough to cause no effect when the parser can generate the result without interpolation. Our implementation uses $\lambda = 10^{-10}$.

3.3 Probability Scaling

To solve the problem of underflow, we modify model probabilities as Equations (2) to (4) to avoid underflow without other expensive operations:

$$Q(X \to w) \equiv P'(X \to w)/s_l(w), \quad (2)$$
$$Q(X \to Y) \equiv P(X \to Y), \quad (3)$$
$$Q(X \to YZ) \equiv P(X \to YZ)/s_g, \quad (4)$$

where X, Y, Z are any non-terminal symbols (including pre-terminals) in the grammar, and w is any word. The result of parsing using Q is guaranteed to be the same as using original probabilities P and P', because Q maintains the same ordering of P and P' despite the fact that Q is not a probability. Values of Q are closer to 1 than the original values, reducing the risk of underflow. $s_l(w)$ is a scaling factor of a word w defined as the geometric mean of lexicon probabilities that generate w, $P'(X \to w)$, as in Equation (5):

$$s_l(w) \equiv \exp \sum_X P(X) \log P'(X \to w), \quad (5)$$

and s_g is the scaling factor of binary rules defined as the geometric mean of all binary rules in the grammar $P(X \to YZ)$ as in Equation (6):

$$s_g \equiv \exp \sum_X P(X) H(X), \quad (6)$$
$$H(X) \equiv \sum_{Y,Z} P(X \to YZ) \log P(X \to YZ). \quad (7)$$

Calculating $P(X)$ is not trivial, but we can retrieve these values using the graph propagation algorithm proposed by Petrov and Klein (2007).

Table 1: Dataset Summaries.

Type	#sent	#word
WSJ-train/dev	41.5 k	990 k
WSJ-test	2.42 k	56.7 k
NTCIR	3.08 M	99.0 M

4 Experiments

We evaluated parsing accuracies of our parser Ckylark and conventional PCFG-LA parsers: Berkeley Parser and Egret. Berkeley Parser is a conventional PCFG-LA parser written in Java with some additional optimization techniques. Egret is also a conventional PCFG-LA parser in C++ which can generate a parsing forest that can be used in downstream application such forest based machine translation (Mi et al., 2008).

4.1 Dataset and Tools

Table 1 shows summaries of each dataset.

We used GrammarTrainer in the Berkeley Parser to train a PCFG-LA grammar with the Penn Treebank WSJ dataset section 2 to 22 (WSJ-train/dev). Egret and Ckylark can use the same model as the Berkeley Parser so we can evaluate only the performance of the parsers using the same grammar. Each parser is run on a Debian 7.1 machine with an Intel Core i7 CPU (3.40GHz, 4 cores, 8MB caches) and 4GB RAM.

We chose 2 datasets to evaluate the performances of each parser. First, WSJ-test, the Penn Treebank WSJ dataset section 23, is a standard dataset

Table 2: Bracketing F1 scores of each parser.

Parser	**F1** (all)	**F1** ($\|\boldsymbol{w}\| \leq 40$)
Berkeley Parser	89.98	90.54
Egret	89.05	89.70
Ckylark (10^{-5})	89.44	90.07
Ckylark (10^{-7})	89.85	90.39

Table 3: Tagging accuracies of each parser.

Parser	**Acc** (all)	**Acc** ($\|\boldsymbol{w}\| \leq 40$)
Berkeley Parser	97.39	97.37
Egret	97.33	97.28
Ckylark (10^{-5})	97.37	97.35
Ckylark (10^{-7})	97.39	97.38

to evaluate parsing accuracy including about 2000 sentences. Second, we use NTCIR, a large English corpus including more than 3 million sentences, extracted from the NTCIR-8 patent translation task (Yamamoto and Shimohata, 2010).

Input sentences of each parser must be tokenized in advance, so we used a tokenization algorithm equivalent to the Stanford Tokenizer[5] for tokenizing the NTCIR dataset.

4.2 Results

Table 2 shows the bracketing F1 scores[6] of parse trees for each parser on the WSJ-test dataset and Table 3 also shows the part-of-speech tagging accuracies. We show 2 results for Ckylark with pruning threshold ϵ as 10^{-5} and 10^{-7}. These tables show that the result of Ckylark with $\epsilon = 10^{-7}$ achieves nearly the same parsing accuracy as the Berkeley Parser.

Table 4 shows calculation times of each parser on the WSJ-test dataset. When the pruning threshold ϵ is smaller, parsing takes longer, but in all cases Ckylark is faster than Egret while achieving higher accuracy. Berkeley Parser is the fastest of all parsers, a result of optimizations not included in the standard PCFG-LA parsing algorithm. Incorporating these techniques into Ckylark is future work.

Table 5 shows the number of parsing failures of each parser. All parsers generate no failure in the WSJ-test dataset, however, in the NTCIR dataset, 0.01% and 0.5% of sentences could not be parsed with the Berkeley Parser and Egret respectively. In contrast, our parser does not fail a single time.

[5] http://nlp.stanford.edu/software/tokenizer.shtml

[6] http://nlp.cs.nyu.edu/evalb/

Table 4: Calculation times of each parser.

Parser	Time [s]
Berkeley Parser	278
Egret	3378
Ckylark (10^{-5})	923
Ckylark (10^{-7})	2157

Table 5: Frequencies of parsing failure of each parser.

Parser	Failure			
	WSJ-test		NTCIR	
	(#)	(%)	(#)	(%)
Berkeley Parser	0	0	419	0.0136
Egret	0	0	17287	0.561
Ckylark (10^{-5})	0	0	0	0

Table 6: Number of failures of each coarse-to-fine level.

Smooth	Failure level						
	0	1	2	3	4	5	6
$\lambda = 0$	1741	135	24	11	5	57	1405
$\lambda = 10^{-10}$	0	130	19	8	4	51	1389

Table 6 shows the number of failures of Ckylark with $\epsilon = 10^{-5}$ and without the stopping approach; if the parser failed at the level l analysis then it returns the result of the $l - 1$ level. Thus, the stopping approach will never generate any failure, unless failure occurs at the initial level. The reason for failure at the initial level is only due to model mismatch, as no pruning has been performed. These errors can be prevented by lexicon smoothing at parsing time as shown in the case of level 0 with $\lambda = 10^{-10}$ in the table.

5 Conclusion

In this paper, we introduce Ckylark, a parser that makes three improvements over standard PCFG-LA style parsing to prevent parsing failure. Experiments show that Ckylark can parse robustly where other PCFG-LA style parsers (Berkeley Parser and Egret) fail. In the future, we plan to further speed up Ckylark, support forest output, and create interfaces to other programming languages.

Acknowledgement

Part of this work was supported by JSPS's Research Fellowship for Young Scientists.

References

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The Penn Treebank. *Computational linguistics*, 19(2).

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proc. ACL*.

Haitao Mi, Liang Huang, and Qun Liu. 2008. Forest-based translation. In *Proc. ACL-HLT*.

Graham Neubig and Kevin Duh. 2014. On the elements of an accurate tree-to-string machine translation system. In *Proc. ACL*.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proc. NAACL-HLT*.

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

Atsushi Fujii Masao Utiyama Mikio Yamamoto and Sayori Shimohata. 2010. Overview of the patent translation task at the NTCIR-8 workshop. In *Proc. NTCIR-8*.

Deniz Yuret, Aydin Han, and Zehra Turgut. 2010. Semeval-2010 task 12: Parser evaluation using textual entailments. In *Proc. SemEval*.