Punctuation Processing for Projective Dependency Parsing*

Ji Ma†, Yue Zhang‡ and Jingbo Zhu†⋆
†Northeastern University, Shenyang, China
‡Singapore University of Technology and Design, Singapore
⋆Hangzhou YaTuo Company, 358 Wener Rd., Hangzhou, China, 310012
majineu@gmail.com
yue_zhang@sutd.edu.sg
zhujingbo@mail.neu.edu.cn

Abstract

Modern statistical dependency parsers assign lexical heads to punctuations as well as words. Punctuation parsing errors lead to low parsing accuracy on words. In this work, we propose an alternative approach to addressing punctuation in dependency parsing. Rather than assigning lexical heads to punctuations, we treat punctuations as properties of their neighbouring words, used as features to guide the parser to build the dependency graph. Integrating our method with an arc-standard parser yields a 93.06% unlabelled attachment score, which is the best accuracy by a single-model transition-based parser reported so far.

1 Introduction

The task of dependency parsing is to identify the lexical head of each of the tokens in a string. Modern statistical parsers (McDonald et al., 2005; Nivre et al., 2007; Huang and Sagae, 2010; Zhang and Nivre, 2011) treat all the tokens equally, assigning lexical heads to punctuations as well as words. Punctuations arguably play an important role in syntactic analysis. However, there are a number of reasons that it is not necessary to parse punctuations:

First, the lexical heads of punctuations are not as well defined as those of words. Consequently, punctuations are not as consistently annotated in treebanks as words, making it harder to parse punctuations. For example, modern statistical parsers achieve above 90% unlabelled attachment score (UAS) on words. However, the UAS on punctuations are generally below 85%.

Moreover, experimental results showed that parsing accuracy of content words drops on sentences which contain higher ratios of punctuations. One reason for this result is that projective dependency parsers satisfy the "no crossing links" constraint, and errors in punctuations may prevent correct word-word dependencies from being created (see section 2). In addition, punctuations cause certain type of features inaccurate. Take valency features for example, previous work (Zhang and Nivre, 2011) has shown that such features are important to parsing accuracy, e.g., it may inform the parser that a verb already has two objects attached to it. However, such information might be inaccurate when the verb's modifiers contain punctuations.

Ultimately, it is the dependencies between words that provide useful information for real world applications. Take machine translation or information extraction for example, most systems take advantage of the head-modifier relationships between word pairs rather than word-punctuation pairs to make better predictions. The fact that most previous work evaluates parsing accuracies without taking punctuations into account is also largely due to this reason.

Given the above reasons, we propose an alternative approach to punctuation processing for dependency parsing. In this method, punctuations are not associated with lexical heads, but are treated as properties of their neighbouring words.

Our method is simple and can be easily incorporated into state-of-the-art parsers. In this work, we report results on an arc-standard transition-based parser. Experiments show that our method achieves about 0.90% UAS improvement over the greedy baseline parser on the standard Penn Treebank test set. Although the improvement becomes smaller as the beam width grows larger, we still achieved 93.06% UAS with a beam of width 64, which is the best result for transition-based parsers

*This work was done while the first author was visiting SUTD

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 791–796,
Baltimore, Maryland, USA, June 23-25 2014.

Length	$1 \sim 20$			$21 - 40$			$41 - 60$		
Punc %	$0 \sim 15$	$15 \sim 30$	> 30	$0 \sim 15$	$15 \sim 30$	> 30	$0 \sim 15$	$15 \sim 30$	> 30
E-F	94.56	92.88	87.67	91.84	91.82	83.87	89.83	88.01	–
A-S	93.87	92.00	90.05	90.81	90.15	75.00	88.06	**88.89**	–
A-S-64	95.28	94.43	88.15	92.96	92.63	76.61	90.78	88.76	–
MST	94.90	93.55	88.15	92.45	**93.11**	77.42	90.89	89.77	–

Table 2: Parsing accuracies vs punctuation ratios, on the development set

System	E-F	A-S	A-S-64	MST
Dev UAS	91.83	90.71	93.02	92.56
Test UAS	91.75	90.34	92.84	92.10
Dev UAS-p	83.20	79.69	84.80	84.42
Test UAS-p	84.67	79.64	87.80	85.67
Dev$^-$ UAS	90.64	89.55	91.87	90.11
Test$^-$ UAS	90.40	89.33	91.75	89.82

Table 1: Parsing accuracies. "E-F" and "MST" denote easy-first parser and MSTparser, respectively. "A-S" and "A-S 64" denote our arc-standard parser with beam width 1 and 64, respectively. "UAS" and "UAS-p" denote word and punctuation unlabelled attachment score, respectively. "$^-$" denotes the data set with punctuations removed.

reported so far. Our code will be available at https://github.com/majineu/Parser/Punc/A-STD.

2 Influence of Punctuations on Parsing

In this section, we conduct a set of experiments to show the influence of punctuations on dependency parsing accuracies.

2.1 Setup

We use the Wall Street Journal portion of the Penn Treebank with the standard splits: sections 02-21 are used as the training set; section 22 and section 23 are used as the development and test set, respectively. Penn2Malt is used to convert bracketed structures into dependencies. We use our own implementation of the Part-Of-Speech (POS) tagger proposed by Collins (2002) to tag the development and test sets. Training set POS tags are generated using 10-fold jack-knifing. Parsing accuracy is evaluated using unlabelled attachment score (UAS), which is the percentage of words that are assigned the correct lexical heads.

To show that the influence of punctuations on parsing is independent of specific parsing algorithms, we conduct experiments using three parsers, each representing a different parsing methodology: the open source MSTParser[1](McDonald and Pereira, 2006), our own re-implementation of an arc-standard transition-based parser (Nivre, 2008), which is trained using global learning and beam-search (Zhang and Clark, 2008) with a rich feature set (Zhang and Nivre, 2011) [2], and our own re-implementation of the easy-first parser (Goldberg and Elhadad, 2010) with an extended feature set (Ma et al., 2013).

2.2 Punctuations and Parsing Accuracy

Our first experiment is to show that, compared with words, punctuations are more difficult to parse and to learn. To see this, we evaluate the parsing accuracies of the selected parsers on words and punctuations, separately. Results are listed in Table 1, where row 2 and row 3 list the UAS of words (all excluding punctuations) on the development and test set, respectively. Row 4 and row 5 list accuracies of punctuations (all excluding words) on the development and test set, respectively. We can see that although all the parsers achieve above 90% UAS on words, the UAS on punctuations are mostly below 85%.

As for learning, we calculate the percentage of parameter updates that are caused by associating punctuations with incorrect heads during training of the easy-first parser[3]. The result is that more than 31% of the parameter updates are caused due to punctuations, though punctuations account for only 11.6% of the total tokens in the training set.

The fact that parsers achieve low accuracies on punctuations is to some degree expected, because the head of a punctuation mark is linguistically less well-defined. However, a related problem is

[1]We trained a second order labelled parser with all the configurations set to the default value. The code is publicly available at http://sourceforge.net/projects/mstparser/

[2]Some feature templates in Zhang and Nivre (2011) involve head word and head POS tags which are not available for an arc-standard parser. Interestingly, without those features our arc-standard parser still achieves 92.84% UAS which is comparable to the 92.90% UAS obtained by the arc-eager parser of Zhang and Nivre (2011)

[3]For the greedy easy-first parser, whether a parameter update is caused by punctuation error can be determined with no ambiguity.

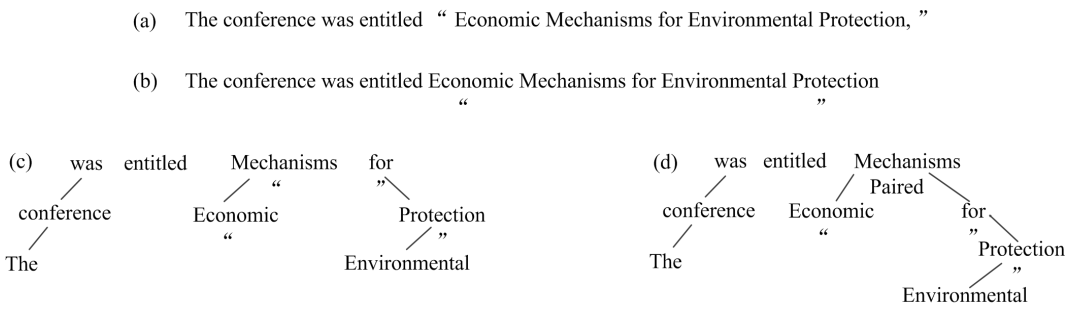

Figure 1: Illustration of processing paired punctuation. The property of a word is denoted by the punctuation below that word.

that parsing accuracy on *words* tends to drop on the sentences which contain high ratio of *punctuations*. To see this, we divide the sentences in the development set into sub-sets according the punctuation ratio (percentage of punctuations that a sentence contains), and then evaluate parsing accuracies on the sub-sets separately.

The results are listed in Table 2. Since long sentences are inherently more difficult to parse, to make a fair comparison, we further divide the development set according to sentence lengths as shown in the first row[4]. We can see that most of the cases, parsing accuracies drop on sentences with higher punctuation ratios. Note that this negative effect on parsing accuracy might be overlooked since most previous work evaluates parsing accuracy *without* taking punctuations into account.

By inspecting the parser outputs, we found that error propagation caused by assigning incorrect head to punctuations is one of the main reason that leads to this result. Take the sentence shown in Figure 1 (a) for example, the word *Mechanisms* is a modifier of *entitled* according to the gold reference. However, if the quotation mark, “, is incorrectly recognized as a modifier of *was*, due to the “no crossing links” constraint, the arc between *Mechanisms* and *entitled* can never be created.

A natural question is whether it is possible to reduce such error propagation by simply removing all punctuations from parsing. Our next experiment aims at answering this question. In this experiment, we first remove all punctuations from the original data and then modify the dependency arcs accordingly in order to maintain word-word dependencies in the original data. We re-train the parsers on the modified training set and evaluate parsing accuracies on the modified data.

Results are listed in row 6 and row 7 of Table 1. We can see that parsing accuracies on the modified data drop significantly compared with that on the original data. The result indicates that by removing punctuations, we lose some information that is important for dependency parsing.

3 Punctuation as Properties

In our method, punctuations are treated as properties of its neighbouring words. Such properties are used as additional features to guide the parser to construct the dependency graph.

3.1 Paired Punctuation

Our method distinguishes *paired* punctuations from other punctuations. Here paired punctuations include brackets and quotations marks, whose Penn Treebank POS tags are the following four:

-LRB- -RRB- “ ”

The characteristics of paired punctuations include: (1) they typically exist in pairs; (2) they serve as *boundaries* that there is only one dependency arc between the words inside the boundaries and the words outside. Take the sentence in Figure 1 (a) for example, the only arc cross the boundary is (Mechanisms, entitled) where entitled is the head.

To utilize such boundary information, we further classify paired punctuations into two categories: those that serve as the beginning of the boundary, whose POS tags are either -LRB- or “, denoted by BPUNC; and those that serve as the end of the boundary, denoted by EPUNC.

Before parsing starts, a preprocessing step is used to first attach the paired punctuations as properties of their neighbouring words, and then remove them from the sentence. In particular,

[4] 1694 out of 1700 sentences on the development set with length no larger than 60 tokens

unigram	for p in $\beta_0, \beta_1, \beta_2, \beta_3, \sigma_0, \sigma_1, \sigma_2$	p_{punc}
	for p in $\beta_0, \beta_1, \beta_2, \sigma_0, \sigma_1$	$p_{punc} \odot p_w, p_{punc} \odot p_t$
bigram	for p, q in $(\sigma_0, \beta_0), (\sigma_0, \beta_1), (\sigma_0, \beta_2), (\sigma_0, \sigma_1), (\sigma_0, \sigma_2)$	$p_{punc} \odot q_{punc}, p_{punc} \odot q_t, p_{punc} \odot q_w$
	for p, q in $(\sigma_2, \sigma_0), (\sigma_1, \sigma_0), (\sigma_2, \sigma_0)$	$p_{punc} \odot q_t, p_{punc} \odot p_t \odot q_t$
	for p, q in $(\sigma_2, \sigma_0), (\sigma_1, \sigma_0), (\sigma_0, \beta_0)$	$d_{pq} \odot p_{punc} \odot p_t \odot q_t$

Table 3: Feature templates. For an element p either on σ or β of an arc-standard parser, we use p_{punc}, p_w and p_t to denote the punctuation property, head word and head tag of p, respectively. d_{pq} denotes the distance between the two elements p and q.

we attach BPUNCs to their right neighbours and EPUNCs to their left neighbours, as shown in Figure 1 (b). Note that in Figure 1 (a), the left neighbour of ” is also a punctuation. In such cases, we simply remove these punctuations since the existence of paired punctuations already indicates that there should be a boundary.

During parsing, when a dependency arc with lexical head w_h is created, the property of w_h is updated by the property of its left (or right) most child to keep track whether there is a BPUNC (or EPUNC) to the left (or right) side of the sub-tree rooted at w_h, as shown in Figure 1 (c). When BPUNCs and EPUNCs meet each other at w_h, a PAIRED property is assigned to w_h to capture that the words within the paired punctuations form a sub-tree, rooted at w_h. See Figure 1 (d).

3.2 Practical Issues

It is not uncommon that two BPUNCS appear adjacent to each other. For example,

(“Congress’s Environmental Buccaneers,” Sept. 18).

In our implementation, BPUNC or EPUNC properties are implemented using flags. In the example, we set two flags “ and (on the word Congrees’s. When BPUNC and EPUNC meet each other, the corresponding flags are turned off. In the example, when Congrees’s is identified as a modifier of Buccaneers, the ” flag of Buccaneers is turned off. However, we do not assign a PAIRED property to Buccaneers since its (flag is still on. The PAIRED property is assigned only when all the flags are turned off.

3.3 Non-Paired Punctuations

Though some types of non-paired punctuations may capture certain syntactic patterns, we do not make further distinctions between them, and treat these punctuations uniformly for simplicity.

Before parsing starts and after the preprocessing step for paired punctuations, our method employs a second preprocessing step to attach non-paired punctuations to their left neighbouring words. It is guaranteed that the property of the left neighbouring words of non-paired punctuations must be empty. Otherwise, it means the non-paired punctuation is adjacent to a paired punctuation. In such cases, the non-paired punctuation would be removed in the first processing step.

During parsing, non-paired punctuations are also passed bottom-up: the property of w_h is updated by its *right*-most dependent to keep track whether there is a punctuation to the right side of the tree rooted at w_h. The only special case is that if w_h already contains a BPUNC property, then our method simply ignores the non-paired property since we maintain the boundary information with the highest priority.

3.4 Features

We incorporate our method into the arc-standard transition-based parser, which uses a stack σ to maintain partially constructed trees and a buffer β for the incoming words (Nivre, 2008). We design a set of features to exploit the potential of using punctuation properties for the arc-standard parser.

The feature templates are listed in Table 3. In addition to the features designed for paired punctuations, such as bigram punctuation features listed in line 3 of Table 3, we also design features for non-paired punctuations. For example, the distance features in line 5 of Table 3 is used to capture the pattern that if a word w with comma property is the left modifier of a noun or a verb, the distance between w and its lexical head is often larger than 1. In other words, they are not adjacent.

4 Results

Our first experiment is to investigate the effect of processing paired punctuations on parsing accuracy. In this experiment, the method introduced in Section 3.1 is used to process paired punctuations, and the non-paired punctuations are left un-

s	Baseline	Paired	All
1	90.76	91.25	91.47
2	91.88	92.06	92.34
4	92.50	92.61	92.70
8	92.73	92.76	92.82
16	92.90	92.94	92.99
64	92.99	93.04	93.10

Table 4: Parsing accuracies on the development set. s denotes the beam width.

touched. Feature templates used in this experiment are those listed in the top three rows of Table 3 together with those used for the baseline arc-standard parser.

Results on the development set are shown in the second column of Table 4. We can see that when the beam width is set to 1, our method achieves an 0.49 UAS improvement. By comparing the outputs of the two parsers, two types of errors made by the baseline parser are effectively corrected.

The first is that our method is able to capture the pattern that there is only one dependency arc between the words within the paired-punctuations and the words outside, while the baseline parser sometimes creates more dependency arcs that cross the boundary.

The second is more interesting. Our method is able to capture that the root, w_h, of the sub-tree within the paired-punctuation, such as "Mechanisms" in Figure 1, generally serves as a modifier of the words outside, while the baseline parser occasionally make w_h as the head of the sentence.

As we increase the beam width, the improvement of our method over the baseline becomes smaller. This is as expected, since beam search also has the effect of reducing error propagation (Zhang and Nivre, 2012), thereby alleviating the errors caused by punctuations.

In the last experiment, we examine the effect of incorporating all punctuations using the method introduced in Section 2. In this experiment, we use all the feature templates in Table 3 and those in the baseline parser. Results are listed in the fourth column of Table 4, which shows that parsing accuracies can be further improved by also processing non-paired punctuations. The overall accuracy improvement when the beam width is 1 reaches 0.91%. The extra improvements mainly come from better accuracies on the sentences with comma. However, the exact type of errors that are corrected by using non-paired punctuations is more difficult to summarize.

system	UAS	Comp	Root
Baseline	90.38	37.71	89.45
All-Punc	91.32	41.35	92.43
Baseline-64	92.84	46.90	**95.57**
All-Punc-64	**93.06**	**48.55**	95.53
Huang 10	92.10	–	–
Zhang 11	92.90	48.00	91.80
Choi 13	92.96	–	–
Bohnet 12	93.03	–	–

Table 5: Final result on the test set.

The final results on the test set are listed in Table 5[5]. Table 5 also lists the accuracies of state-of-the-art transition-based parsers. In particular, "Huang 10" and "Zhang 11" denote Huang and Sagae (2010) and Zhang and Nivre (2011), respectively. "Bohnet 12" and "Choi 13" denote Bohnet and Nivre (2012) and Choi and Mccallum (2013), respectively. We can see that our method achieves the best accuracy for single-model transition-based parsers.

5 Conclusion and Related Work

In this work, we proposed to treat punctuations as properties of context words for dependency parsing. Experiments with an arc-standard parser showed that our method effectively improves parsing performance and we achieved the best accuracy for single-model transition-based parser.

Regarding punctuation processing for dependency parsing, Li et al. (2010) proposed to utilize punctuations to segment sentences into small fragments and then parse the fragments separately. A similar approach is proposed by Spitkovsky et al. (2011) which also designed a set of constraints on the fragments to improve unsupervised dependency parsing.

Acknowledgements

We highly appreciate the anonymous reviewers for their insightful suggestions. This research was supported by the National Science Foundation of China (61272376; 61300097; 61100089), the Fundamental Research Funds for the Central Universities (N110404012), the research grant T2MOE1301 from Singapore Ministry of Education (MOE) and the start-up grant SRG ISTD2012038 from SUTD.

[5]The number of training iteration is determined using the development set.

References

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1455–1465, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jinho D. Choi and Andrew Mccallum. 2013. Transition-based dependency parsing with selectional branching. In *In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*.

Michael Collins. 2002. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 742–750, Stroudsburg, PA, USA. Association for Computational Linguistics.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In Jan Hajic, Sandra Carberry, and Stephen Clark, editors, *ACL*, pages 1077–1086. The Association for Computer Linguistics.

Zhenghua Li, Wanxiang Che, and Ting Liu. 2010. Improving dependency parsing using punctuation. In Minghui Dong, Guodong Zhou, Haoliang Qi, and Min Zhang, editors, *IALP*, pages 53–56. IEEE Computer Society.

Ji Ma, Jingbo Zhu, Tong Xiao, and Nan Yang. 2013. Easy-first pos tagging and dependency parsing with beam search. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–114, Sofia, Bulgaria, August. Association for Computational Linguistics.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)*, volume 6, pages 81–88.

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

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2011. Punctuation: Making a point in unsupervised dependency parsing. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning (CoNLL-2011)*.

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, October. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA, June. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2012. Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *Proceedings of COLING 2012: Posters*, pages 1391–1400, Mumbai, India, December. The COLING 2012 Organizing Committee.