A Three-step Deterministic Parser for Chinese Dependency Parsing

Kun Yu
Graduate School of Informatics
Kyoto University
kunyu@nlp.kuee.kyoto-u.ac.jp

Sadao Kurohashi
Graduate School of Informatics
Kyoto University
kuro@i.kyoto-u.ac.jp

Hao Liu
Graduate School of Information Science and Technology
The University of Tokyo
liuhao@kc.t.u-tokyo.ac.jp

Abstract

This paper presents a three-step dependency parser to parse Chinese deterministically. By dividing a sentence into several parts and parsing them separately, it aims to reduce the error propagation coming from the greedy characteristic of deterministic parsing. Experimental results showed that compared with the deterministic parser which parsed a sentence in sequence, the proposed parser achieved extremely significant improvement on dependency accuracy.

1 Introduction

Recently, as an attractive alternative to probabilistic parsing, deterministic parsing (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004) has drawn great attention with its high efficiency, simplicity and good accuracy comparable to the state-of-the-art generative probabilistic models. The basic idea of deterministic parsing is using a greedy parsing algorithm that approximates a globally optimal solution by making a sequence of locally optimal choices (Hall et al., 2006). This greedy idea guarantees the simplicity and efficiency, but at the same time it also suffers from the error propagation from the previous parsing choices to the left decisions.

For example, given a Chinese sentence, which means *Paternity test is a test that gets personal identity through DNA analysis, and it brings proof for finding lost children*, the correct dependency tree is shown by solid line (see Figure 1). But, if word 通过(through) is incorrectly parsed as depending on word 是(is) (shown by dotted line), this error will result in the incorrect parse of word 鉴定(a test) as depending on word 提供 (brings) (shown by dotted line).

This problem exists not only in Chinese, but also in other languages. Some efforts have been done to solve this problem. Cheng et al. (2005) used a root finder to divide one sentence into two parts by the root word and parsed them separately. But the two-part division is not enough when a sentence is composed of several coordinating sub-sentences. Chang et al. (2006) applied a pipeline framework in their dependency parser to make the local predictions more robust. While it did not show great help for stopping the error propagation between different parsing stages.

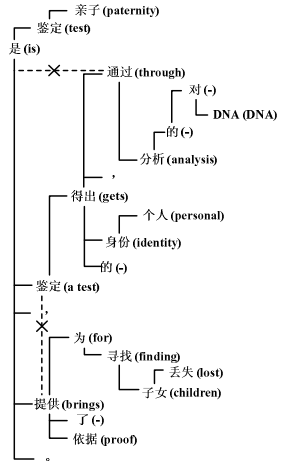

Figure 1. Dependency tree of a sentence (word sequence is top-down)

This paper focuses on resolving this issue for Chinese. After analyzing the dependency structure of sentences in Penn Chinese Treebank 5.1 (Xue et al., 2002), we found an interesting phenomenon: if we define a **main-root** as *the head of a sentence*, and define a **sub-sentence** as *a sequence of words separated by punctuations, and the head*[1] *of these words is the child of main-root or main-root itself*, then the punctuations that depend on main-root can be a separator of sub-sentences.

For example, in the example sentence there are three punctuations marked as PU_A, PU_B and PU_C, in which PU_B and PU_C depends on main-root but PU_A depends on word 得出(gets). According to our observation, PU_B and PU_C can be used for segmenting this sentence into two sub-sentences A and B (circled by dotted line in Figure 2), where the sub-root of A is main-root and the sub-root of B depends on main-root.

This phenomenon gives us a useful clue: *if we divide a sentence by the punctuations whose head is main-root, then the divided sub-sentences are basically independent of each other, which means we can parse them separately*. The shortening of sentence length and the recognition of sentence structure guarantee the robustness of deterministic parsing. The independent parsing of each sub-sentence also prevents the error-propagation. In

[1] *The head of sub-sentence* is defined as a **sub-root**.

Proceedings of NAACL HLT 2007, Companion Volume, pages 201–204,
Rochester, NY, April 2007.

addition, because the sub-root depends on main-root or is main-root itself, it is easy to combine the dependency structure of each sub-sentence to create the final dependency tree.

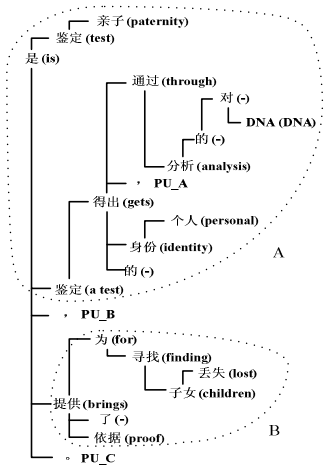

Figure 2. A segmentation of the sentence in Figure 1

Based on above analyses, this paper proposes a three-step deterministic dependency parser for Chinese, which works as:

Step1(Sentence Segmentation): Segmenting a sentence into sub-sentences by punctuations (sub-sentences do not contain the punctuations for segmentation);

Step2(Sub-sentence Parsing): Parsing each sub-sentence deterministically;

Step3(Parsing Combination): Finding main-root among all the sub-roots, then combining the dependency structure of sub-sentences by making main-root as the head of both the left sub-roots and the punctuations for sentence segmentation.

2 Sentence Segmentation

As mentioned in section 1, the punctuations depending on main-root can be used to segment a sentence into several sub-sentences, whose sub-root depends on main-root or is main-root. But by analysis, we found only several punctuations were used as separator commonly. To ensure the accuracy of sentence segmentation, we first define the punctuations which are possible for segmentation as **valid punctuation**, which includes *comma, period, colon, semicolon, question mark, exclamatory mark and ellipsis*. Then the task in step 1 is to find punctuations which are able to segment a sentence from all the valid punctuations in a sentence, and use them to divide the sentence into two or more sub-sentences.

We define a classifier (called as **sentence segmenter**) to classify the valid punctuations in a sentence to be good or bad for sentence segmentation. SVM (Sebastiani, 2002) is selected as classification model for its robustness to over-fitting and high performance.

Table 1 shows the binary features defined for sentence segmentation. We use a lexicon consisting of all the words in Penn Chinese Treebank 5.1 to lexicalize word features. For example, if word 为 (for) is the 27150_{th} word in the lexicon, then feature $Word_1$ of PU_B (see Figure 2) is '27150:1'. The pos-tag features are got in the same way by a pos-tag list containing 33 pos-tags, which follow the definition in Penn Chinese Treebank. Such method is also used to get word and pos-tag features in other modules.

Table 1. Features for sentence segmenter

Feature	Description
$Word_n/Pos_n$	word/pos-tag in different position, n=-2,-1,0,1,2
Word_left/ Pos_left	word/pos-tag between the first left valid punctuation and current punctuation
Word_right/ Pos_right	word/pos-tag between current punctuation and the first right valid punctuation
#Word_left/ #Word_right	if the number of words between the first left/right valid punctuation and current punctuation is higher than 2, set as 1; otherwise set as 0
V_left/ V_right	if there is a verb between the first left/right valid punctuation and current punctuation, set as 1; otherwise set as 0
N_leftFirst/ N_rightFirst	if the left/right neighbor word is a noun, set as 1; otherwise set as 0
P_rightFirst/ CS_rightFirst	if the right neighbor word is a preposition/subordinating conjunction, set as 1; otherwise set as 0

3 Sub-sentence Parsing

3.1 Parsing Algorithm

The parsing algorithm in step 2 is a shift-reduce parser based on (Yamada and Matsumoto, 2003). We call it as **sub-sentence parser**.

Two stacks *P* and *U* are defined, where stack *P* keeps the words under consideration and stack *U* remains all the unparsed words. All the dependency relations created by the parser are stored in queue *A*.

At start, stack *P* and queue *A* are empty and stack *U* contains all the words. Then word on the top of stack *U* is pushed into stack *P*, and a trained classifier finds probable action for word pair <*p*,*u*> on the top of the two stacks. After that, according to different actions, dependency relations are created and pushed into queue *A*, and the elements in the two stacks move at the same time. Parser stops when stack *U* is empty and the dependency tree can be drawn according to the relations stored in queue *A*.

Four actions are defined for word pair <*p*, *u*>:

LEFT: if word *p* modifies word *u*, then push $p \rightarrow u$ into *A* and push *u* into *P*.

RIGHT: if word *u* modifies word *p*, then push $u \rightarrow p$ into *A* and pop *p*.

REDUCE: if there is no word *u'* ($u' \in U$ and $u' \neq u$) which modifies *p*, and word next to *p* in stack *P* is *p*'s head, then pop *p*.

SHIFT: if there is no dependency relation between *p* and *u*, and word next to *p* in stack *P* is not *p*'s head, then push *u* into stack *P*.

We construct a classifier for each action separately, and classify each word pair by all the classifiers. Then the action with the highest classification score is selected. SVM is used as the classifier, and *One vs. All* strategy (Berger, 1999) is applied for its good efficiency to extend binary classifier to multi-class classifier.

3.2 Features

Features are crucial to this step. First, we define some features based on local context (see F_{local} in Table 2), which are often used in other deterministic parsers (Yamada and Matsumoto, 2003; Nivre et al., 2006). Then, to get top-down information, we add some global features (see F_{global} in Table 2). All the features are binary features, except that *Distance* is normalized between 0-1 by the length of sub-sentence.

Before parsing, we use a root finder (i.e. the sub-sentence root finder introduced in Section 4) to get $Root_n$ feature, and develop a **baseNP chunker** to get $BaseNP_n$ feature. In the baseNP chunker, *IOB* representation is applied for each word, where *B* means the word is the beginning of a baseNP, *I* means the word is inside of a baseNP, and *O* means the word is outside of a baseNP. Tagging is performed by SVM with *One vs. All* strategy. Features used in baseNP chunking are current word, surrounding words and their corresponding pos-tags. Window size is 5.

Table 2. Features for sub-sentence parser

Feature		Description
Local Feature (F_{local})	$Word_n$/ Pos_n	word/pos-tag in different position, n= P_0, P_1, P_2, U_0, U_1, U_2 (P_i/U_i mean the i_{th} position from top in stack *P*/*U*)
	$Word\_child_n$/ Pos_child_n	the word/pos-tag of the children of $Word_n$, n= P_0, P_1, P_2, U_0, U_1, U_2
	Distance	distance between p and u in sentence
Global Feature (F_{global})	$Root_n$	if $Word_n$ is the sub-root of this sub-sentence, set as 1; otherwise set as 0
	$BaseNP_n$	baseNP tag of $Word_n$

Table 3. Features for sentence/sub-sentence root finder

Feature	Description
$Word_n$/Pos_n	words in different position, n=-2,-1,0,1,2
Word_left/Pos_left	$word_n$/pos_n where n<-2
Word_right/Pos_right	$word_n$/pos_n where n>2
#Word_left/ #Word_right	if the number of words between the start/end of sentence and current word is higher than 2, set as 1; otherwise set as 0
V_left/V_right	if there is a verb between the start/end of sentence and current word, set as 1; otherwise set as 0
$Noun_n$/$Verb_n$/Adj_n	if the word in different position is a noun/verb/adjective, set as 1; otherwise set as 0. n=-2,-1,1,2
Dec_right	if the word next to current word in right side is 的(of), set as 1; otherwise set as 0
CC_left	if there is a coordinating conjunction between the start of sentence and current word, set as 1; otherwise set as 0
$BaseNP_n$	baseNP tag of $Word_n$

4 Parsing Combination

A root finder is developed to find main-root for parsing combination. We call it as **sentence root finder**. We also retrain the same module to find the sub-root in step 2, and call it as **sub-sentence root finder**.

We define the root finding problem as a classification problem. A classifier, where we still select SVM, is trained to classify each word to be root or not. Then the word with the highest classification score is chosen as root. All the binary features for root finding are listed in Table 3. Here the baseNP chunker introduced in section 3.2 is used to get the $BaseNP_n$ feature.

5 Experimental Results

5.1 Data Set and Experimental Setting

We use Penn Chinese Treebank 5.1 as data set. To transfer the phrase structure into dependency structure, head rules are defined based on Xia's head percolation table (Xia and Palmer, 2001). 16,984 sentences and 1,292 sentences are used for training and testing. The same training data is also used to train the sentence segmenter, the baseNP chunker, the sub-sentence root finder, and the sentence root finder. During both training and testing, the gold-standard word segmentation and pos-tag are applied.

TinySVM is selected as a SVM toolkit. We use a polynomial kernel and set the degree as 2 in all the experiments.

5.2 Three-step Parsing vs. One-step Parsing

First, we evaluated the dependency accuracy and root accuracy of both three-step parsing and one-step parsing. Three-step parsing is the proposed parser and one-step parsing means parsing a sentence in sequence (i.e. only using step 2). Local and global features are used in both of them.

Results (see Table 4) showed that because of the shortening of sentence length and the prevention of error propagation three-step parsing got 2.14% increase on dependency accuracy compared with one-step parsing. Based on McNemar's test (Gillick and Cox, 1989), this improvement was considered extremely statistically significant ($p<0.0001$). In addition, the proposed parser got 1.01% increase on root accuracy.

Table 4. Parsing result of three-step and one-step parsing

Parsing Strategy	Dep.Accu. (%)	Root Accu. (%)	Avg. Parsing Time (sec.)
One-step Parsing	82.12	74.92	22.13
Three-step Parsing	84.26 (+2.14)	75.93 (+1.01)	24.27 (+2.14)

Then we tested the average parsing time for each sentence to verify the efficiency of proposed parser. The average sentence length is 21.68 words. Results (see Table 4) showed that compared with one-step parsing, the proposed parser only used 2.14 more seconds aver-

agely when parsing one sentence, which did not affect efficiency greatly.

To verify the effectiveness of proposed parser on complex sentences, which contain two or more sub-sentences according to our definition, we selected 665 such sentences from testing data set and did evaluation again. Results (see Table 5) proved that our parser outperformed one-step parsing successfully.

Table 5. Parsing result of complex sentence

Parsing Strategy	Dep.Accu. (%)	Root Accu. (%)
One-step Parsing	82.56	78.95
Three-step Parsing	84.94 (+2.38)	79.25 (+0.30)

5.3 Comparison with Others' Work

At last, we compare the proposed parser with Nivre's parser (Hall et al., 2006). We use the same head rules for dependency transformation as what were used in Nivre's work. We also used the same training (section 1-9) and testing (section 0) data and retrained all the modules. Results showed that the proposed parser achieved 84.50% dependency accuracy, which was 0.20% higher than Nivre's parser (84.30%).

6 Discussion

In the proposed parser, we used five modules: sentence segmenter (step1); sub-sentence root finder (step2); baseNP chunker (step2&3); sub-sentence parser (step2); and sentence root finder (step3).

The robustness of the modules will affect parsing accuracy. Thus we evaluated each module separately. Results (see Table 6) showed that all the modules got reasonable accuracy except for the sentence root finder. Considering about this, in step 3 we found main-root only from the sub-roots created by step 2. Because the sub-sentence parser used in step 2 had good accuracy, it could provide relatively correct candidates for main-root finding. Therefore it helped decrease the influence of the poor sentence root finding to the proposed parser.

Table 6. Evaluation result of each module

Module	F-score(%)	Dep.Accu(%)
Sentence Segmenter (M1)	88.04	---
Sub-sentence Root Finder (M2)	88.73	---
BaseNP Chunker (M3)	89.25	---
Sub-sentence Parser (M4)	---	85.56
Sentence Root Finder (M5)	78.01	---

Then we evaluated the proposed parser assuming using gold-standard modules (except for sub-sentence parser) to check the contribution of each module to parsing. Results (see Table 7) showed that (1) the accuracy of current sentence segmenter was acceptable because only small increase on dependency accuracy and root accuracy was got by using gold-standard sentence segmentation; (2) the correct recognition of baseNP could help improve dependency accuracy but gave a little contribution to root accuracy; (3) the accuracy of both sub-sentence root finder and sentence root finder was most crucial to parsing. Therefore improving the two root finders is an important task in our future work.

Table 7. Parsing result with gold-standard modules

Gold-standard Module	Dep.Accu(%)	Root.Accu(%)
w/o	84.26	75.93
M1	84.51	76.24
M1+M2	86.57	80.34
M1+M2+M3	88.63	80.57
M1+M2+M3+M5	91.25	91.02

7 Conclusion and Future Work

We propose a three-step deterministic dependency parser for parsing Chinese. It aims to solve the error propagation problem by dividing a sentence into independent parts and parsing them separately. Results based on Penn Chinese Treebank 5.1 showed that compared with the deterministic parser which parsed a sentence in sequence, the proposed parser achieved extremely significant increase on dependency accuracy.

Currently, the proposed parser is designed only for Chinese. But we believe it can be easily adapted to other languages because no language-limited information is used. We will try this work in the future. In addition, improving sub-sentence root finder and sentence root finder will also be considered in the future.

Acknowledgement

We would like to thank Dr. Daisuke Kawahara and Dr. Eiji Aramaki for their helpful discussions. We also thank the three anonymous reviewers for their valuable comments.

Reference

A.Berger. Error-correcting output coding for text classification. 1999. In *Proceedings of the IJCAI-99 Workshop on Machine Learning for Information Filtering*.

M.Chang, Q.Do and D.Roth. 2006. A Pipeline Framework for Dependency Parsing. In *Proceedings of Coling-ACL 2006*.

Y.Cheng, M.Asahara and Y.Matsumoto. 2005. Chinese Deterministic Dependency Analyzer: Examining Effects of Global Features and Root Node Finder. In *Proceedings of IJCNLP 2005*.

L.Gillick and S.J.Cox. 1989. Some Statistical Issues in the Comparison of Speech Recognition Algorithms. In *Proceedings of ICASSP*.

J.Hall, J.Nivre and J.Nilsson. 2006. Discriminative Classifiers for Deterministic Dependency Parsing. In *Proceedings of Coling-ACL 2006*. pp. 316-323.

J.Nivre and M.Scholz. 2004. Deterministic Dependency Parsing of English Text. In *Proceedings of Coling 2004*. pp. 64-70.

F.Sebastiani. 2002. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1): 1-47.

F.Xia and M.Palmer. 2001. Converting Dependency Structures to Phrase Structures. In *HLT-2001*.

N.Xue, F.Chiou and M.Palmer. 2002. Building a Large-Scale Annotated Chinese Corpus. In P*roceedings of COLING 2002*.

H.Yamada and Y.Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of IWPT. 2003*.