Joint POS Tagging and Transition-based Constituent Parsing in Chinese with Non-local Features

Zhiguo Wang
Brandeis University
Waltham, MA, USA
zgwang@brandeis.edu

Nianwen Xue
Brandeis University
Waltham, MA, USA
xuen@brandeis.edu

Abstract

We propose three improvements to address the drawbacks of state-of-the-art transition-based constituent parsers. First, to resolve the error propagation problem of the traditional pipeline approach, we incorporate POS tagging into the syntactic parsing process. Second, to alleviate the negative influence of size differences among competing action sequences, we align parser states during beam-search decoding. Third, to enhance the power of parsing models, we enlarge the feature set with non-local features and semi-supervised word cluster features. Experimental results show that these modifications improve parsing performance significantly. Evaluated on the Chinese TreeBank (CTB), our final performance reaches 86.3% (F1) when trained on CTB 5.1, and 87.1% when trained on CTB 6.0, and these results outperform all state-of-the-art parsers.

1 Introduction

Constituent parsing is one of the most fundamental tasks in Natural Language Processing (NLP). It seeks to uncover the underlying recursive phrase structure of sentences. Most of the state-of-the-art parsers are based on the PCFG paradigm and chart-based decoding algorithms (Collins, 1999; Charniak, 2000; Petrov et al., 2006). Chart-based parsers perform exhaustive search with dynamic programming, which contributes to their high accuracy, but they also suffer from higher runtime complexity and can only exploit simple local structural information.

Transition-based constituent parsing (Sagae and Lavie, 2005; Wang et al., 2006; Zhang and Clark, 2009) is an attractive alternative. It utilizes a series of deterministic shift-reduce decisions to construct syntactic trees. Therefore, it runs in linear time and can take advantage of arbitrarily complex structural features from already constructed subtrees. The downside is that they only search a tiny fraction of the whole space and are therefore commonly considered to be less accurate than chart-based parsers. Recent studies (Zhu et al., 2013; Zhang et al., 2013) show, however, that this approach can also achieve the state-of-the-art performance with improved training procedures and the use of additional source of information as features.

However, there is still room for improvement for these state-of-the-art transition-based constituent parsers. First, POS tagging is typically performed separately as a preliminary step, and POS tagging errors will propagate to the parsing process. This problem is especially severe for languages where the POS tagging accuracy is relatively low, and this is the case for Chinese where there are fewer contextual clues that can be used to inform the tagging process and some of the tagging decisions are actually influenced by the syntactic structure of the sentence. This creates a chicken and egg problem that needs to be addressed when designing a parsing model. Second, due to the existence of unary rules in constituent trees, competing candidate parses often have different number of actions, and this increases the disambiguation difficulty for the parsing model. Third, transition-based parsers have the freedom to define arbitrarily complex structural features, but this freedom has not fully been taken advantage of and most of the present approaches only use simple structural features.

In this paper, we address these drawbacks to improve the transition-based constituent parsing for Chinese. First, we integrate POS tagging into the parsing process and jointly optimize these two processes simultaneously. Because non-local syntactic information is now available to POS tag

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pages 733–742,
Baltimore, Maryland, USA, June 23-25 2014.

determination, the accuracy of POS tagging improves, and this will in turn improve parsing accuracy. Second, we propose a novel state alignment strategy to align candidate parses with different action sizes during beam-search decoding. With this strategy, parser states and their unary extensions are put into the same beam, therefore the parsing model could decide whether or not to use unary actions within local decision beams. Third, we take into account two groups of complex structural features that have not been previously used in transition-based parsing: *non-local* features (Charniak and Johnson, 2005) and semi-supervised *word cluster* features (Koo et al., 2008). With the help of the non-local features, our transition-based parsing system outperforms all previous single systems in Chinese. After integrating semi-supervised word cluster features, the parsing accuracy is further improved to 86.3% when trained on CTB 5.1 and 87.1% when trained on CTB 6.0, and this is the best reported performance for Chinese.

The remainder of this paper is organized as follows: Section 2 introduces the standard transition-based constituent parsing approach. Section 3 describes our three improvements to standard transition-based constituent parsing. We discuss and analyze the experimental results in Section 4. Section 5 discusses related work. Finally, we conclude this paper in Section 6.

2 Transition-based Constituent Parsing

This section describes the transition-based constituent parsing model, which is the basis of Section 3 and the baseline model in Section 4.

2.1 Transition-based Constituent Parsing Model

A transition-based constituent parsing model is a quadruple $C = (S, T, s_0, S_t)$, where S is a set of parser *states* (sometimes called *configurations*), T is a finite set of *actions*, s_0 is an initialization function to map each input sentence into a unique *initial state*, and $S_t \in S$ is a set of *terminal states*. Each action $t \in T$ is a transition function to transit a state into a new state. A parser state $s \in S$ is defined as a tuple $s = (\sigma, \beta)$, where σ is a *stack* which is maintained to hold partial subtrees that are already constructed, and β is a *queue* which is used for storing word-POS pairs that remain unprocessed. In particular, the initial state has an empty stack σ and a queue β containing the entire input sentence (word-POS pairs), and the terminal states have an empty queue β and a stack σ containing only one complete parse tree. The task of transition-based constituent parsing is to scan the input POS-tagged sentence from left to right and perform a sequence of actions to transform the initial state into a terminal state.

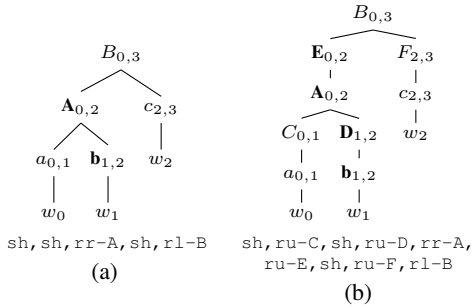

Figure 1: Two constituent trees for an example sentence $w_0w_1w_2$ with POS tags abc. The corresponding action sequences are given below, the spans of each nodes are annotated and the head nodes are written with **Bold** font type.

In order to construct lexicalized constituent parse trees, we define the following actions for the action set T according to (Sagae and Lavie, 2005; Wang et al., 2006; Zhang and Clark, 2009):

- SHIFT (`sh`): remove the first word-POS pair from β, and push it onto the top of σ;
- REDUCE-UNARY-X (`ru-x`): pop the top subtree from σ, construct a new unary node labeled with X for the subtree, then push the new subtree back onto σ. The head of the new subtree is inherited from its child;
- REDUCE-BINARY-{L/R}-X (`rl/rr-x`): pop the top two subtrees from σ, combine them into a new tree with a node labeled with X, then push the new subtree back onto σ. The left (L) and right (R) versions of the action indicate whether the head of the new subtree is inherited from its left or right child.

With these actions, our parser can process trees with unary and binary branches easily. For example, in Figure 1, for the input sentence $w_0w_1w_2$ and its POS tags abc, our parser can construct two parse trees using action sequences given below these trees. However, parse trees in Treebanks often contain an arbitrary number of branches. To

Type	Feature Templates
unigrams	p_0tc, p_0wc, p_1tc, p_1wc, p_2tc p_2wc, p_3tc, p_3wc, q_0wt, q_1wt q_2wt, q_3wt, $p_{0l}wc$, $p_{0r}wc$ $p_{0u}wc$, $p_{1l}wc$, $p_{1r}wc$, $p_{1u}wc$
bigrams	p_0wp_1w, p_0wp_1c, p_0cp_1w, p_0cp_1c p_0wq_0w, p_0wq_0t, p_0cq_0w, p_0cq_0t q_0wq_1w, q_0wq_1t, q_0tq_1w, q_0tq_1t p_1wq_0w, p_1wq_0t, p_1cq_0w, p_1cq_0t
trigrams	$p_0cp_1cp_2c$, $p_0wp_1cp_2c$, $p_0cp_1wq_0t$ $p_0cp_1cp_2w$, $p_0cp_1cq_0t$, $p_0wp_1cq_0t$ $p_0cp_1wq_0t$, $p_0cp_1cq_0w$

Table 1: Baseline features, where p_i represents the i_{th} subtree in the stack σ and q_i denotes the i_{th} item in the queue β. w refers to the head lexicon, t refers to the head POS, and c refers to the constituent label. p_{il} and p_{ir} refer to the left and right child for a binary subtree p_i, and p_{iu} refers to the child of a unary subtree p_i.

process such trees, we employ binarization and debinarization processes described in Zhang and Clark (2009) to transform multi-branch trees into binary-branch trees and restore the generated binary trees back to their original forms.

2.2 Modeling, Training and Decoding

To determine which action $t \in T$ should the parser perform at a state $s \in S$, we use a linear model to score each possible $\langle s, t \rangle$ combination:

$$score(s,t) = \vec{w} \cdot \phi(s,t) = \sum_i w_i f_i(s,t) \quad (1)$$

where $\phi(s,t)$ is the feature function used for mapping a state-action pair into a feature vector, and $\vec{w}$ is the weight vector. The score of a parser state s is the sum of the scores for all state-action pairs in the transition path from the initial state to the current state. Table 1 lists the feature templates used in our baseline parser, which is adopted from Zhang and Clark (2009). To train the weight vector $\vec{w}$, we employ the averaged perceptron algorithm with early update (Collins and Roark, 2004).

We employ the beam search decoding algorithm (Zhang and Clark, 2009) to balance the trade-off between accuracy and efficiency. Algorithm 1 gives details of the process. In the algorithm, we maintain a $beam$ (sometimes called $agenda$) to keep k best states at each step. The first $beam_0$

Algorithm 1 Beam-search Constituent Parsing

Input: A POS-tagged sentence, beam size k.
Output: A constituent parse tree.

```
1:  beam_0 ← {s_0}                          ▷ initialization
2:  i ← 0                                   ▷ step index
3:  loop
4:      P ← {}                              ▷ a priority queue
5:      while beam_i is not empty do
6:          s ← POP(beam_i)
7:          for all possible t ∈ T do
8:              s_new ← apply t to s
9:              score s_new with E.q (1)
10:             insert s_new into P
11:     beam_{i+1} ← k best states of P
12:     s_best ← best state in beam_{i+1}
13:     if s_best ∈ S_t then
14:         return s_best
15:     i ← i + 1
```

is initialized with the initial state s_0 (line 1). At step i, each of the k states in $beam_i$ is extended by applying all possible actions (line 5-10). For all newly generated states, only the k best states are preserved for $beam_{i+1}$ (line 11). The decoding process repeats until the highest scored state in $beam_{i+1}$ reaches a terminal state (line 12-14).

3 Joint POS Tagging and Parsing with Non-local Features

To address the drawbacks of the standard transition-based constituent parsing model (described in Section 1), we propose a model to jointly solve POS tagging and constituent parsing with non-local features.

3.1 Joint POS Tagging and Parsing

POS tagging is often taken as a preliminary step for transition-based constituent parsing, therefore the accuracy of POS tagging would greatly affect parsing performance. In our experiment (described in Section 4.2), parsing accuracy would decrease by 8.5% in F_1 in Chinese parsing when using automatically generated POS tags instead of gold-standard ones. To tackle this issue, we integrate POS tagging into the transition-based constituent parsing process and jointly optimize these two processes simultaneously. Inspired from Hatori et al. (2011), we modify the `sh` action by assigning a POS tag for the word when it is shifted:

- SHIFT-X (`sh-x`): remove the first word from

β, assign POS tag X to the word and push it onto the top of σ.

With such an action, POS tagging becomes a natural part of transition-based parsing. However, some feature templates in Table 1 become unavailable, because POS tags for the look-ahead words are not specified yet under the joint framework. For example, for the template q_0wt , the POS tag of the first word q_0 in the queue β is required, but it is not specified yet at the present state.

To overcome the lack of look-ahead POS tags, we borrow the concept of *delayed features* originally developed for dependency parsing (Hatori et al., 2011). Features that require look-ahead POS tags are defined as delayed features. In these features, look-ahead POS tags are taken as variables. During parsing, delayed features are extracted and passed from one state to the next state. When a `sh-x` action is performed, the look-ahead POS tag of some delayed features is specified, therefore these delayed features can be transformed into normal features (by replacing variable with the newly specified POS tag). The remaining delayed features will be transformed similarly when their look-ahead POS tags are specified during the following parsing steps.

3.2 State Alignment

Assuming an input sentence contains n words, in order to reach a terminal state, the initial state requires n `sh-x` actions to consume all words in β, and $n-1$ `rl`/`rr-x` actions to construct a complete parse tree by consuming all the subtrees in σ. However, `ru-x` is a very special action. It only constructs a new unary node for the subtree on top of σ, but does not consume any items in σ or β. As a result, the number of `ru-x` actions varies among terminal states for the same sentence. For example, the parse tree in Figure 1a contains no `ru-x` action, while the parse tree for the same input sentence in Figure 1b contains four `ru-x` actions. This makes the lengths of complete action sequences very different, and the parsing model has to disambiguate among terminal states with varying action sizes. Zhu et al. (2013) proposed a *padding* method to align terminal states containing different number of actions. The idea is to append some *IDLE* actions to terminal states with shorter action sequence, and make sure all terminal states contain the same number of actions (including *IDLE* actions).

Algorithm 2 Beam-search with State Alignment

Input: A word-segmented sentence, beam size k.
Output: A constituent parse tree.

1: $beam_0 \leftarrow \{s_0\}$ ▷ initialization
2: **for** $i \leftarrow 0$ to $2n-1$ **do** ▷ n is sentence length
3: $P_0 \leftarrow \{\}$, $P_1 \leftarrow \{\}$ ▷ two priority queues
4: **while** $beam_i$ is not empty **do**
5: $s \leftarrow$ POP($beam_i$)
6: **for** $t \in \{$`sh-x`, `rl-x`, `rr-x`$\}$ **do**
7: $s_{new} \leftarrow$ apply t to s
8: score s_{new} with E.q (1)
9: insert s_{new} into P_0
10: **for** all state s in P_0 **do**
11: **for** all possible $t \in \{$`ru-x`$\}$ **do**
12: $s_{new} \leftarrow$ apply t to s
13: score s_{new} with E.q (1)
14: insert s_{new} into P_1
15: insert all states of P_1 into P_0
16: $beam_{i+1} \leftarrow k$ best states of P_0
17: **return** the best state in $beam_{2n-1}$

We propose a novel method to align states during the parsing process instead of just aligning terminal states like Zhu et al. (2013). We classify all the actions into two groups according to whether they consume items in σ or β. `sh-x`, `rl-x`, and `rr-x` belong to *consuming* actions, and `ru-x` belongs to *non-consuming* action. Algorithm 2 gives the details of our method. It is based on the beam search decoding algorithm described in Algorithm 1. Different from Algorithm 1, Algorithm 2 is guaranteed to perform $2n-1$ parsing steps for an input sentence containing n words (line 2), and divides each parsing step into two parsing phases. In the first phase (line 4-9), each of the k states in $beam_i$ is extended by consuming actions. In the second phase (line 10-14), each of the newly generated states is further extended by non-consuming actions. Then, all these states extended by both consuming and non-consuming actions are considered together (line 15), and only the k highest-scored states are preserved for $beam_{i+1}$ (line 16). After these $2n-1$ parsing steps, the highest scored state in $beam_{2n-1}$ is returned as the final result (line 17). Figure 2 shows the states aligning process for the two trees in Figure 1. We find that our new method aligns states with their `ru-x` extensions in the same $beam$, therefore the parsing model could make decisions on whether using `ru-x` actions or not within local decision

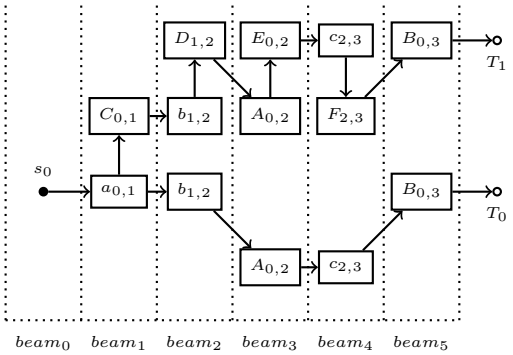

Figure 2: State alignment for the two trees in Figure 1, where s_0 is the initial state, T_0 and T_1 are terminal states corresponding to the two trees in Figure 1. For clarity, we represent each state as a rectangle with the label of top subtree in the stack σ. We also denote `sh-x` with $\rightarrow$, `ru-x` with $\uparrow$ or $\downarrow$, `rl-x` with $\nearrow$, and `rr-x` with $\searrow$.

beams.

3.3 Feature Extension

One advantage of transition-based constituent parsing is that it is capable of incorporating arbitrarily complex structural features from the already constructed subtrees in σ and unprocessed words in β. However, all the feature templates given in Table 1 are just some simple structural features. To further improve the performance of our transition-based constituent parser, we consider two group of complex structural features: *non-local* features (Charniak and Johnson, 2005; Collins and Koo, 2005) and semi-supervised *word cluster* features (Koo et al., 2008).

Table 2 lists all the non-local features we want to use. These features have been proved very helpful for constituent parsing (Charniak and Johnson, 2005; Collins and Koo, 2005). But almost all previous work considered non-local features only in parse reranking frameworks. Instead, we attempt to extract non-local features from newly constructed subtrees during the decoding process as they become incrementally available and score newly generated parser states with them. One difficulty is that the subtrees built by our baseline parser are binary trees (only the complete parse tree is debinarized into its original multi-branch form), but most of the non-local features need to be extracted from their original multi-branch forms. To resolve this conflict, we integrate the debinarization process into the parsing process, i.e., when a new subtree is constructed during parsing, we debinarize it immediately if it is not rooted with an *intermediate* node [1]. The other subtrees for subsequent parsing steps will be built based on these debinarized subtrees. After the modification, our parser can extract non-local features incrementally during the parsing process.

(Collins and Koo, 2005)	(Charniak and Johnson, 2005)	
Rules	CoPar	HeadTree
Bigrams	CoLenPar	
Grandparent Rules	RightBranch	
Grandparent Bigrams	Heavy	
Lexical Bigrams	Neighbours	
Two-level Rules	NGramTree	
Two-level Bigrams	Heads	
Trigrams	Wproj	
Head-Modifiers	Word	

Table 2: Non-local features for constituent parsing.

Semi-supervised word cluster features have been successfully applied to many NLP tasks (Miller et al., 2004; Koo et al., 2008; Zhu et al., 2013). Here, we adopt such features for our transition-based constituent parser. Given a large-scale unlabeled corpus (word segmentation should be performed), we employ the Brown cluster algorithm (Liang, 2005) to cluster all words into a binary tree. Within this binary tree, words appear as leaves, left branches are labeled with 0 and right branches are labeled with 1. Each word can be uniquely identified by its path from the root, and represented as a bit-string. By using various length of prefixes of the bit-string, we can produce word clusters of different granularities (Miller et al., 2004). Inspired from Koo et al. (2008), we employ two types of word clusters: (1) taking 4 bit-string prefixes of word clusters as replacements of POS tags, and (2) taking 8 bit-string prefixes as replacements of words. Using these two types of clusters, we construct semi-supervised word cluster features by mimicking the template structure of the original baseline features in Table 1.

4 Experiment

4.1 Experimental Setting

We conducted experiments on the Penn Chinese Treebank (CTB) version 5.1 (Xue et al., 2005): Articles 001-270 and 400-1151 were used as the training set, Articles 301-325 were used as the development set, and Articles 271-300 were used

[1] Intermediate nodes are produced by binarization process.

as the test set. Standard corpus preparation steps were performed before our experiments: empty nodes and functional tags were removed, and the unary chains were collapsed to single unary rules as Harper and Huang (2011). To build word clusters, we used the unlabeled Chinese Gigaword (LDC2003T09) and conducted Chinese word segmentation using a CRF-based segmenter.

We used EVALB [2] tool to evaluate parsing performance. The metrics include labeled precision (LP), labeled recall (LR), bracketing F_1 and POS tagging accuracy. We set the beam size k to 16, which brings a good balance between efficiency and accuracy. We tuned the optimal number of iterations of perceptron training algorithm on the development set.

4.2 Pipeline Approach *vs* Joint POS Tagging and Parsing

In this subsection, we conducted some experiments to illustrate the drawbacks of the pipeline approach and the advantages of our joint approach. We built three parsing systems: *Pipeline-Gold* system is our baseline parser (described in Section 2) taking gold-standard POS tags as input; *Pipeline* system is our baseline parser taking as input POS tags automatically assigned by Stanford POS Tagger [3]; and *JointParsing* system is our joint POS tagging and transition-based parsing system described in subsection 3.1. We trained these three systems on the training set and evaluated them on the development set. The second, third and forth rows in Table 3 show the parsing performances. We can see that the parsing F_1 decreased by about 8.5 percentage points in F_1 score when using automatically assigned POS tags instead of gold-standard ones, and this shows that the pipeline approach is greatly affected by the quality of its preliminary POS tagging step. After integrating the POS tagging step into the parsing process, our *JointParsing* system improved the POS tagging accuracy to 94.8% and parsing F_1 to 85.8%, which are significantly better than the *Pipeline* system. Therefore, the joint parsing approach is much more effective for transition-based constituent parsing.

4.3 State Alignment Evaluation

We built two new systems to verify the effectiveness of our state alignment strategy proposed in Subsection 3.2. The first system *Padding* extends our *JointParsing* system by aligning terminal states with the padding strategy proposed in Zhu et al. (2013), and the second system *StateAlign* extends the *JointParsing* system with our state alignment strategy. The fifth and sixth rows of Table 3 give the performances of these two systems. Compared with the *JointParsing* system which does not employ any alignment strategy, the *Padding* system only achieved a slight improvement on parsing F_1 score, but no improvement on POS tagging accuracy. In contrast, our *StateAlign* system achieved an improvement of 0.6% on parsing F_1 score and 0.4% on POS tagging accuracy. All these results show us that our state alignment strategy is more helpful for beam-search decoding.

System	LP	LR	F_1	POS
Pipeline-Gold	92.2	92.5	92.4	100
Pipeline	83.9	83.8	83.8	93.0
JointParsing	85.1	86.6	85.8	94.8
Padding	85.4	86.4	85.9	94.8
StateAlign	86.9	85.9	86.4	95.2
Nonlocal	88.0	86.5	87.2	95.3
Cluster	89.0	88.3	88.7	96.3
Nonlocal&Cluster	89.4	88.7	89.1	96.2

Table 3: Parsing performance on Chinese development set.

4.4 Feature Extension Evaluation

In this subsection, we examined the usefulness of the new non-local features and the semi-supervised word cluster features described in Subsection 3.3. We built three new parsing systems based on the *StateAlign* system: *Nonlocal* system extends the feature set of *StateAlign* system with non-local features, *Cluster* system extends the feature set with semi-supervised word cluster features, and *Nonlocal&Cluster* system extend the feature set with both groups of features. Parsing performances of the three systems are shown in the last three rows of Table 3. Compared with the *StateAlign* system which takes only the baseline features, the non-local features improved parsing F_1 by 0.8%, while the semi-supervised word cluster features result in an improvement of 2.3% in parsing F_1 and an 1.1% improvement on POS tagging accuracy. When integrating both groups of features, the final parsing F_1 reaches 89.1%. Al-

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

[3] http://nlp.stanford.edu/downloads/tagger.shtml

Type	System	LP	LR	F_1	POS
Our Systems	Pipeline	80.0	80.3	80.1	94.0
	JointParsing	82.4	83.0	82.7	95.1
	Padding	82.7	83.6	83.2	95.1
	StateAlign	84.2	82.9	83.6	95.5
	Nonlocal	85.6	84.2	84.9	95.9
	Cluster	85.2	84.5	84.9	95.8
	Nonlocal&Cluster	**86.6**	**85.9**	**86.3**	**96.0**
Single Systems	Petrov and Klein (2007)	81.9	84.8	83.3	-
	Zhu et al. (2013)	82.1	84.3	83.2	-
Reranking Systems	Charniak and Johnson (2005)*	80.8	83.8	82.3	-
	Wang and Zong (2011)	-	-	85.7	-
Semi-supervised Systems	Zhu et al. (2013)	84.4	86.8	85.6	-

Table 4: Parsing performance on Chinese test set. *Huang (2009) adapted the parse reranker to CTB5.

l these results show that both the non-local features and the semi-supervised features are helpful for our transition-based constituent parser.

4.5 Final Results on Test Set

In this subsection, we present the performances of our systems on the CTB test set. The corresponding results are listed in the top rows of Table 4. We can see that all these systems maintain a similar relative relationship as they do on the development set, which shows the stability of our systems.

To further illustrate the effectiveness of our systems, we compare them with some state-of-the-art systems. We group parsing systems into three categories: single systems, reranking systems and semi-supervised systems. Our *Pipeline*, *JointParsing*, *Padding*, *StateAlign* and *Nonlocal* systems belong to the category of single systems, because they don't utilize any extra processing steps or resources. Our *Cluster* and *Nonlocal&Cluster* systems belong to semi-supervised systems, because both of them have employed semi-supervised word cluster features. The parsing performances of state-of-the-art systems are shown in the bottom rows of Table 4. We can see that the final F_1 of our *Nonlocal* system reached 84.9%, and it outperforms state-of-the-art single systems by more than 1.6%. As far as we know, this is the best result on the CTB test set acquired by single systems. Our *Nonlocal&Cluster* system further improved the parsing F_1 to 86.3%, and it outperforms all reranking systems and semi-supervised systems. To our knowledge, this is the best reported performance in Chinese parsing.

System	F_1
Huang and Harper (2009)	85.2
Nonlocal&Cluster	**87.1**

Table 5: Parsing performance based on CTB 6.

All previous experiments were conducted on CTB 5. To check whether more labeled data can further improve our parsing system, we evaluated our *Nonlocal&Cluster* system on the Chinese TreeBank version 6.0 (CTB6), which is a super set of CTB5 and contains more annotated data. We used the same development set and test set as CTB5, and took all the remaining data as the new training set. Table 5 shows the parsing performances on CTB6. Our *Nonlocal&Cluster* system improved the final F_1 to 87.1%, which is 1.9% better than the state-of-the-art performance on CTB6 (Huang and Harper, 2009). Compared with its performance on CTB5 (in Table 4), our *Nonlocal&Cluster* system also got 0.8% improvement. All these results show that our approach can become more powerful when given more labeled training data.

4.6 Error Analysis

To better understand the linguistic behavior of our systems, we employed the berkeley-parser-analyser tool [4] (Kummerfeld et al., 2013) to categorize the errors. Table 6 presents the average

[4] http://code.google.com/p/berkeley-parser-analyser/

System	NP Int.	Unary	1-Word Span	Coord	Mod. Attach	Verb Args	Diff Label	Clause Attach	Noun Edge
Worst	1.75	0.74	0.44	0.49	0.39	0.37	0.29	0.15	0.14
Pipeline									
JointParsing									
Padding									
StateAlign									
Nonlocal									
Cluster									
Nonlocal&Cluster									
Best	1.33	0.42	0.28	0.29	0.19	0.21	0.17	0.07	0.09

Table 6: Parse errors on Chinese test set. The shaded area of each bar indicates average number of that error type per sentence, and the completely full bar indicates the number in the *Worst* row.

System	VV→NN	NN→VV	DEC→DEG	JJ→NN	NR→NN	DEG→DEC	NN→NR	NN→JJ
Worst	0.26	0.18	0.15	0.09	0.08	0.07	0.06	0.05
Pipeline								
JointParsing								
Padding								
StateAlign								
Nonlocal								
Cluster								
Nonlocal&Cluster								
Best	0.14	0.10	0.03	0.07	0.05	0.03	0.03	0.02

Table 7: POS tagging error patterns on Chinese test set. For each error pattern, the left hand side tag is the gold-standard tag, and the right hand side is the wrongly assigned tag.

number of errors for each error type by our parsing systems. We can see that almost all the *Worst* numbers are produced by the *Pipeline* system. The *JointParsing* system reduced errors of all types produced by the *Pipeline* system except for the coordination error type (Coord). The *StateAlign* system corrected a lot of the NP-internal errors (NP Int.). The *Nonlocal* system and the *Cluster* system produced similar numbers of errors for all error types. The *Nonlocal&Cluster* system produced the *Best* numbers for all the error types. NP-internal errors are still the most frequent error type in our parsing systems.

Table 7 presents the statistics of frequent POS tagging error patterns. We can see that *JointParsing* system disambiguates {VV, NN} and {DEC, DEG} better than *Pipeline* system, but cannot deal with the NN→JJ pattern very well. *StateAlign* system got better results in most of the patterns, but cannot disambiguate {NR, NN} well. *Nonlocal&Cluster* system got the best results in disambiguating the most ambiguous POS tag pairs of {VV, NN}, {DEC, DEG}, {JJ, NN} and {NN, NR}.

5 Related Work

Joint POS tagging with parsing is not a new idea. In PCFG-based parsing (Collins, 1999; Charniak, 2000; Petrov et al., 2006), POS tagging is considered as a natural step of parsing by employing lexical rules. For transition-based parsing, Hatori et al. (2011) proposed to integrate POS tagging with *dependency* parsing. Our joint approach can be seen as an adaption of Hatori et al. (2011)'s approach for *constituent* parsing. Zhang et al. (2013) proposed a transition-based constituent parser to process an input sentence from the character level. However, manual annotation of the word-internal structures need to be added to the original Treebank in order to train such a parser.

Non-local features have been successfully used for constituent parsing (Charniak and Johnson, 2005; Collins and Koo, 2005; Huang, 2008). However, almost all of the previous work use non-local features at the parse reranking stage. The reason is that the single-stage chart-based parser cannot use non-local structural features. In contrast, the transition-based parser can use arbitrarily complex structural features. Therefore, we can concisely utilize non-local features in a single-

stage parsing system.

6 Conclusion

In this paper, we proposed three improvements to transition-based constituent parsing for Chinese. First, we incorporated POS tagging into transition-based constituent parsing to resolve the error propagation problem of the pipeline approach. Second, we proposed a state alignment strategy to align competing decision sequences that have different number of actions. Finally, we enhanced our parsing model by enlarging the feature set with non-local features and semi-supervised word cluster features. Experimental results show that all these methods improved the parsing performance substantially, and the final performance of our parsing system outperformed all state-of-the-art systems.

Acknowledgments

We thank three anonymous reviewers for their cogent comments. This work is funded by the DAPRA via contract HR0011-11-C-0145 entitled "Linguistic Resources for Multilingual Processing". All opinions expressed here are those of the authors and do not necessarily reflect the views of DARPA.

References

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. Association for Computational Linguistics.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Association for Computational Linguistics.

Michael Collins and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July.

Michael Collins. 1999. *HEAD-DRIVEN STATISTICAL MODELS FOR NATURAL LANGUAGE PARSING*. Ph.D. thesis, University of Pennsylvania.

Mary Harper and Zhongqiang Huang. 2011. Chinese statistical parsing. *Handbook of Natural Language Processing and Machine Translation*.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2011. Incremental joint pos tagging and dependency parsing in chinese. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1216–1224, Chiang Mai, Thailand, November. Asian Federation of Natural Language Processing.

Zhongqiang Huang and Mary Harper. 2009. Self-training pcfg grammars with latent annotations across languages. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 832–841. Association for Computational Linguistics.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *ACL*, pages 586–594.

Ling-Ya Huang. 2009. Improve chinese parsing with max-ent reranking parser. *Master Project Report, Brown University*.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, Columbus, Ohio, June. Association for Computational Linguistics.

Jonathan K. Kummerfeld, Daniel Tse, James R. Curran, and Dan Klein. 2013. An empirical examination of challenges in chinese parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 98–103, Sofia, Bulgaria, August. Association for Computational Linguistics.

Percy Liang. 2005. *Semi-supervised learning for natural language*. Ph.D. thesis, Massachusetts Institute of Technology.

Scott Miller, Jethran Guinness, and Alex Zamanian. 2004. Name tagging with word clusters and discriminative training. In *HLT-NAACL*, volume 4, pages 337–342. Citeseer.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *HLT-NAACL*, pages 404–411.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132. Association for Computational Linguistics.

Zhiguo Wang and Chengqing Zong. 2011. Parse reranking based on higher-order lexical dependencies. In *IJCNLP*, pages 1251–1259.

Mengqiu Wang, Kenji Sagae, and Teruko Mitamura. 2006. A fast, accurate deterministic parser for chinese. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 425–432. Association for Computational Linguistics.

Naiwen 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.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 162–171. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2013. Chinese parsing exploiting characters. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 125–134, Sofia, Bulgaria, August. Association for Computational Linguistics.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria, August. Association for Computational Linguistics.