# Exploiting Lexical Dependencies from Large-Scale Data for Better Shift-Reduce Constituency Parsing

*Muhua ZHU  Jingbo ZHU  Huizhen WANG*
Natural Language Processing Laboratory
Northeastern University, China
zhumuhua@gmail.com, {zhujingbo,wanghuizhen}@mail.neu.edu.cn

Abstract
This paper proposes a method to improve shift-reduce constituency parsing by using lexical dependencies. The lexical dependency information is obtained from a large amount of auto-parsed data that is generated by a baseline shift-reduce parser on unlabeled data. We then incorporate a set of novel features defined on this information into the shift-reduce parsing model. The features can help to disambiguate action conflicts during decoding. Experimental results show that the new features achieve absolute improvements over a strong baseline by 0.9% and 1.1% on English and Chinese respectively. Moreover, the improved parser outperforms all previously reported shift-reduce constituency parsers.

Title and Abstract in another language, $L_2$ (optional, and on same page)

## 利用大规模数据中词汇依存关系改进移进－归约成分句法分析

本文提出了一种利用词汇依存关系改进移进－归约成分句法分析的方法。首先，我们利用基准系统在大规模无标注数据上进行自动句法分析并且从分析结果中抽取得到词汇依存关系。其后，我们在词汇依存信息基础上定义了一组新特征并将这些特征整合到移进－归约句法分析模型 中。新特征可以用于移进－归约的动作冲突的消歧。实验结果表明，新特征在英文和中文数据上分别取得了0.9% 和1.1%的性能改进。最终得到的句法分析器的性能优于之前研究所报告的移进－归约句法分析器的性能。

Keywords: Shift-reduce Constituency Parsing, Lexical Dependencies, Large-scale Data.

Keywords in $L_2$: 移进－归约成分句法分析，词汇依存，大规模数据.

# 1 Introduction

Due to the simplicity and running efficiency, shift-reduce parsing has been studied extensively for a variety of grammars, ranging from constituency parsing (Sagae and Lavie, 2005, 2006; Zhang and Clark, 2009) through dependency parsing (Nivre, 2004; Yamada and Matsumoto, 2003; Zhang and Clark, 2008) to CCG parsing (Zhang and Clark, 2011). In dependency and CCG parsing, shift-reduce parsing is among the best-performing algorithms (Huang and Sagae, 2010; Zhang and Clark, 2011). However, compared to commonly-used statistical parsers available on the web such as Charniak-Jonhson (Charniak and Johnson, 2005) and Petrov-Klein (Petrov and Klein, 2007), shift-reduce constituency parsers still have room left for further improvements on parsing accuracy.

There exist at least two major directions to advance shift-reduce constituency parsing. One direction is to design better training and decoding algorithms. For example, in the respect of decoding, Sagae and Lavie (2006) proposed a best-first search strategy to expand the search space. In the respect of training, Zhang and Clark (2009) replaced local classifiers with a global learning algorithm. The other direction is to enrich feature representations for better shift-reduce constituency parsing, which will be the focus of this paper. In this direction, previous work has extensively studied a variety of features, all in the framework of supervised learning (Sagae and Lavie, 2005, 2006; Wang et al., 2006; Zhang and Clark, 2009).

How to further enrich feature representations for better shift-reduce constituency parsing becomes a very challenging problem. In this paper, we solve this issue by using the information of lexical head-modifier[1] relations (a.k.a. lexical dependencies) (Collins, 1996). Previous work on other constituency parsers have shown the effectiveness of lexical dependency information on disambiguating syntactic structures (Collins, 1996, 1997; Eisner and Satta, 1999). But in shift-reduce constituency parsing, such information is not fully used. For instance, Zhang and Clark (2009) completely neglected lexical dependency information. Sagae and Lavie (2005) and Wang et al. (2006) only incorporated as features the most recently recognized (left and right) modifiers of some designated words. Unlike previous work on shift-reduce constituency parsing, this paper aims to incorporate features that encode the information of whether words in an input sentence tend to have head-modifier relations. In addition, although it is feasible to get lexical dependency information from human-labeled treebank data by using head-finding rules (Collins, 1999), we find that lexical dependencies obtained from this source suffer from data sparseness (Section 5.1). We propose to solve this problem by utilizing additional large-scale unlabeled data.

The basic idea of our approach is to provide shift-reduce parsers with lexical dependency information that is obtained from large-scale auto-parsed data. To this end, we first parse unlabeled data with a baseline parser and afterwards extract bigram and trigram lexical dependencies from automatically parsed trees. Based on the extracted lexical dependencies, we finally design a set of features to enhance the baseline parser. The experiments in Section 5 show that new features can improve a strong baseline parser by 0.9% and 1.1% on English and Chinese data sets respectively. Moreover, our parser outperforms previously reported shift-reduce constituency parsers while maintaining efficiency.

Specifically, we make the following contributions in this paper:

- We propose a set of novel features for better shift-reduce constituency parsing that is based on lexical dependencies obtained from large-scale auto-parsed data;

[1] By 'head-modifier' we mean the linguistic notion that a word (*modifier*) modifies another word (*head*).

- We empirically compare two different sources for obtaining lexical dependencies: human-labeled treebank data and large-scale auto-parsed data respectively, and show the superiority of using auto-parsed data (Section 5.1);
- We empirically analyze major sources of shift-reduce parsing errors (Section 3.1) and verify the effectiveness of new features in resolving shift-reduce action conflicts (Section 5.6.2);

## 2 Baseline Parser

We use the beam-search shift-reduce parser (Zhang and Clark, 2009) as the baseline system in this paper. [2] In what follows, we describe the parser in brief.

### 2.1 The Shift-Reduce Parsing Process

The shift-reduce process in the baseline parser assumes binary-branching trees, so binarization and debinarization are required for transforming training data and parsing output, respectively (Zhang and Clark, 2009). Given an input sentence (words and POS tags), any possible parse tree yielding the sentence corresponds *exactly* to one sequence of states. Formally, each state in the sequence is denoted by a tuple $\langle S, Q \rangle$, where $S$ is a stack containing partial parses and $Q$ is a queue of word-POS pairs that remain unprocessed. In particular, the initial state is $\langle \phi, w_1 \dots w_n \rangle$ where $S$ is empty and $Q$ contains the entire input sentence. The final state is $\langle S, \phi \rangle$ where $S$ contains a single parse tree with a pre-designated root label and $Q$ is empty. Thus, the shift-reduce parsing process is a transition process from the initial state to the final state by performing a sequence of the following actions.

1. shift, which moves a pair of word and POS tag from the head of the queue to the stack. Here the queue is required to be non-empty.
2. reduce-unary-X, which extends the top item on the stack by applying a unary rule and then replaces the top item with the newly generated constituent. Here X represents a treebank phrase label, such as *NP*, which is to be used as the root label of the new constituent.
3. reduce-binary-{L/R}-X, which moves top two items out of the stack and pushes a new item onto the stack. The new item has X as its root label and consists of two children with the first popped item becoming the right child and the second popped item becoming the left child. The switch L/R indicates whether the left (L) or the right (R) child becomes the head child.
4. terminate, which pops the root node off the stack and ends parsing. This action is applicable only when the stack contains a single parse and the queue is empty.

### 2.2 Beam Search Extension

The shift-reduce parsing process described above can be extended with beam search, as presented in Algorithm 1. The algorithm starts by initializing a beam of size $K$ with the initial state. In each iteration after the initialization, states are popped in turn out of the beam. For each popped state, all applicable actions are then evaluated with respect to the state. Scored action-state pairs are sorted in a temporary priority queue. When the beam gets empty, top $K$ highest-scored action-state pairs are fetched from the priority queue and next states corresponding to the action-state pairs are inserted back into the beam. If the highest-scored state in the beam is a final state, it will be returned as the parsing result; else the iteration continues. The algorithm has time complexity of $O(nK)$, where $n$ is the sentence length and $K$ is the beam size.

[2] `www.cl.cam.ac.uk/~yz360/zpar.html`

**Algorithm 1** Beam-search shift-reduce parsing

**Input:** a POS-tagged word sequence $w_1 \dots w_n$
beam size $K$ and *action set*

```
1: B ← {⟨ϕ, w_1 ... w_n⟩}        // initialize beam
2: loop
3:     priority queue P = []
4:     while B not empty do
5:         state ← pop(B)
6:         for all act ∈ action set do
7:             score ← evaluate act for state
8:             P·insert (⟨score, act, state⟩)
9:     for i = 0 to K do
10:        ⟨score, act, state⟩ ← Pop-Top (P)
11:        next ← apply act to state
12:        insert next to B
13:    best ← highest-scored state in B
14:    if best is complete then
15:        return best
```

## 2.3 Model and Learning Algorithm

To score an action $A$ with respect to a state $Y = \langle S, Q \rangle$, we use a linear model as defined by

$$Score(\langle A, Y \rangle) = \vec{w} \cdot \Phi(\langle A, Y \rangle) = \sum_i \lambda_i f_i(\langle A, Y \rangle)$$

where $f_i(\langle A, Y \rangle)$ are features extracted jointly from the action $A$ and state $Y$. To learn parameters $\lambda_i$, we use the generalized perceptron algorithm proposed in Collins (2002).

Generalized perceptron is an online learning algorithm that learns one instance at a time. The basic procedure is to use the beam-search parsing algorithm (Algorithm 1) to parse the yield of a gold parse tree. Whenever the gold partial parse is pruned from the beam, parameters will be updated immediately and the learner moves to the next training instance. Such a strategy is known as "early-update" (Collins and Roark, 2004). Finally, model parameters are set to be an average of the weight vectors obtained during the online learning.

## 2.4 Baseline Features

Features used in the baseline parser are similar as those used in Zhang and Clark (2009). For convenience of reference, we repeat the features in Table 1, where the symbol $S_i$ represents the $i_{th}$ item from the top of the stack S and the symbol $Q_i$ denotes the $i_{th}$ item from the front end of the queue $Q$. The symbol $w$ represents the lexical head for an item; $c$ represents the label for an item; and $t$ denotes POS of a lexical head. Note that Zhang and Clark (2009) also used bracket-related and separator features for Chinese parsing, which have been removed in the latest release of their parser. So in this article we choose to ignore such language specific features.

# 3 Our Approach

## 3.1 Motivation

We first empirically analyze major sources of shift-reduce parsing errors with the parsing results of the baseline parser on the English development set. The baseline parser is trained on human-labeled training data. Regarding the parsing results, we are especially concerned with *first mistakes* that the baseline parser makes because future mistakes are often caused by previous ones. There are

| Description | Templates |
|---|---|
| Unigrams | $S_0tc, S_0wc, S_1tc, S_1wc, S_2tc, S_2wc, S_3tc, S_3wc,$ $Q_0wt, Q_1wt, Q_2wt, Q_3wt,$ $S_0lwc, S_0rwc, S_0uwc, S_1lwc, S_1rwc, S_1uwc$ |
| Bigrams | $S_0wS_1w, S_0wS_1c, S_0cS_1w, S_0cS_1c,$ $S_0wQ_0w, S_0wQ_0t, S_0cQ_0w, S_0cQ_0t,$ $Q_0wN_1w, Q_0wQ_1t, Q_0tQ_1w, Q_0tQ_1t,$ $S_1wQ_0w, S_1wQ_0t, S_1cQ_0w, S_1cQ_0t$ |
| Trigrams | $S_0cS_1cS_2c, S_0wS_1cS_2c, S_0cS_1wQ_0t, S_0cS_1cS_2w,$ $S_0cS_1cQ_0t, S_0wS_1cQ_0t, S_0cS_1wQ_0t, S_0cS_1cQ_0w$ |

Table 1: A summary of baseline feature templates, where $S_i$ represents the $i_{th}$ item in stack $S$ and $Q_i$ denotes the $i_{th}$ item in the queue $Q$ from the front end.

| ID | Mistake Type | Ratio (Count) |
|---|---|---|
| 1 | shift vs. red-binary | 47.8% (451) |
| 2 | shift vs. red-unary | 18.1% (171) |
| 3 | red-binary vs. red-unary | 5.4% (92) |
| 4 | red-binary-L/R-{X vs. $X^*$} | 16.5% (156) |
| 5 | red-unary-{$X_1$ vs. $X_2$} | 5.7% (54) |

Table 2: Types and ratios of first mistakes made by the baseline parser on the English development set with auto-assigned POS.

944 first mistakes in total in the parsing results. Table 2 shows the types and ratios of the top 5 most frequent first mistakes. Cases 1-2 consist of conflicts between shift and reduce actions. Case 3 is comprised of conflicts between reduce-binary and reduce-unary actions. Mistakes in cases 4-5 are caused by wrong choices of labels where the symbols X, $X_1$, and $X_2$ refer to treebank phrase labels and the symbol $X^*$ denotes a temporary label which is introduced when a constituent with label $X$ is binarized. From the table we notice that action conflicts between shift and reduce-binary are the largest source of parsing errors, which cover nearly half of first mistakes.

Intuitively, lexical dependency information is beneficial to resolving shift and reduce-binary conflicts. In the following, we use a real example to make clear the intuition. Figure 1 illustrates the shift-reduce parsing process of the baseline parser on a sentence with auto-assigned POS tags. The baseline parser proceeds correctly until it reaches the state in Figure 1-(a). At that point, there is a conflict between reduce-binary (Figure 1-(b)) and shift (Figure 1-(c)) actions. We find that the baseline parser wrongly chooses the reduce-binary action because the word *bore* is (incorrectly) tagged as a verb.[3] The baseline parser tends to group the words preceding a verb as a constituent. However, if the parser is informed that the words *a* and *bore* have a lexical dependency relationship, the parser may correct its choice and switch to the shift action. In addition, we find that the human-labeled data used to train the baseline parser does not contain lexical dependencies between *a* and *bore*. We can see that extracting lexical dependencies solely from human-labeled training data has a data sparseness problem. This motivates us to utilize unlabeled data as an additional source for lexical dependency extraction.

[3]All the occurrences of *bore* in the training data of the POS tagger have the POS tag *VBD*.

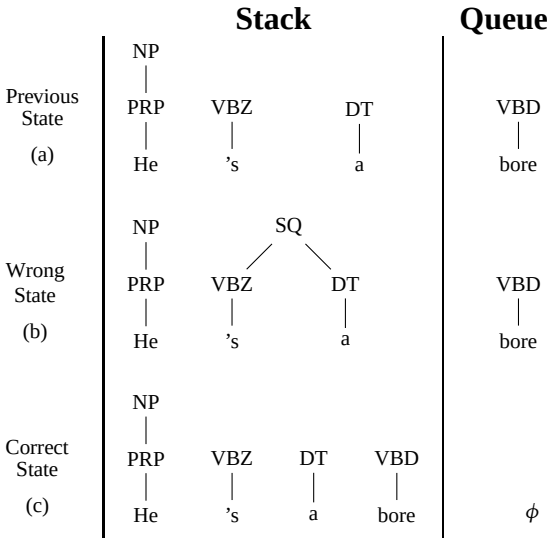



Figure 1: An example of shift-reduce conflicts, illustrating how lexical dependency information helps to disambiguate the conflicts.

## 3.2 Data Preprocessing

Before conducting lexical dependency extraction, we use the baseline parser to generate constituency parse trees from unlabeled data. Since shift-reduce parsers require POS tags as input, automatical POS tagging should be performed on unlabeled data before performing syntactic parsing. For unspaced languages such as Chinese, automatical word segmentation is also needed. To simplify the extraction process, we convert automatically parsed constituency trees into dependency trees with Penn2Malt (or other conversion tools).[4]

## 3.3 Extraction of Lexical Dependencies

After the tree conversion, the following lexical dependencies are read off from dependency trees. Here we restrict the dependencies to those between two words (bigram lexical dependencies) and those between three words (trigram lexical dependencies).

**Bigram Lexical Dependencies**

If two words are connected by an arc in a dependency tree, we claim these two words maintain a bigram lexical dependency. For pairs of words that have dependencies, we make a record of the words as well as their head-modifier relations. Formally, bigram lexical dependencies are denoted as $\langle w_1, w_2, L/R\rangle$ where $L/R$ indicates the direction of the dependency arc that connects $w_1$ and $w_2$. Moreover, lexical dependencies are word-order sensitive, that is, $\langle w_1, w_2, L\rangle$ is regarded to be different from $\langle w_2, w_1, R\rangle$.

**Trigram Lexical Dependencies**

Trigram lexical dependencies encode a head-modifier relationship among three words. As with bigram lexical dependencies, trigram lexical dependencies are also word-order sensitive. In this

[4] `http://w3.msi.vxu.se/~nivre/research/Penn2Malt.html`

| Bigram Dependency Features | | | |
|---|---|---|---|
| $f_L(s_1w, s_0w)$ | $f_L(s_1w, s_0w) \circ s_1t \circ s_0t$ | $f_R(s_1w, s_0w)$ | $f_R(s_1w, s_0w) \circ s_1t \circ s_0t$ |
| $f_L(s_1w, q_0w)$ | $f_L(s_1w, q_0w) \circ s_1t \circ q_0t$ | $f_R(s_1w, q_0w)$ | $f_R(s_1w, q_0w) \circ s_1t \circ q_0t$ |
| $f_L(s_0w, q_0w)$ | $f_L(s_0w, q_0w) \circ s_0t \circ q_0t$ | $f_R(s_0w, q_0w)$ | $f_R(s_0w, q_0w) \circ s_0t \circ q_0t$ |
| **Trigram Dependency Features** | | | |
| $f_L(s_1w, s_1rdw, s_0w)$ | $f_L(s_1w, s_1rdw, s_0w) \circ s_1t \circ s_0t$ | $f_R(s_1w, s_0ldw, s_0w)$ | $f_R(s_1w, s_0ldw, s_0w) \circ s_1t \circ s_0t$ |
| $f_L(s_0w, s_0rdw, q_0w)$ | $f_L(s_0w, s_0rdw, q_0w) \circ s_0t \circ q_0t$ | $f_R(s_0w, NONE, q_0w)$ | $f_R(s_0w, NONE, q_0w) \circ s_0t \circ q_0t$ |

Table 3: New features designed on the basis of lexical dependencies. Here the symbol $w$ represents a word and the symbol $t$ represents a POS tag.

paper, we only consider the type of trigram lexical dependencies that have the first or the last word be the head and that require the other two words to be siblings among all the modifiers of the head. Such lexical dependencies can be represented formally as $\langle w_1, w_2, w_3, L/R \rangle$. Here the switch $L/R$ indicates the head among the three words. Specifically, the symbol $L$ specifies $w_1$ to be the head and the symbol $R$ designates $w_3$ to be the head. In addition, we also consider the special case that $w_2$ is *NONE*, which indicates that $w_1$ ($w_3$) is the rightmost (leftmost) modifier of $w_3$ ($w_1$).

## 3.4 Proposed Features

After extracting all lexical dependencies, we group bigram and trigram lexical dependencies *separately* into three categories according to their frequencies. Specifically, if a dependency relation is among top-10% most frequent records, then it receives the group tag *High Frequency (HF)*; else if it is in top-20%, then we use the tag *Middle Frequency (MF)*; else we use the tag *Low Frequency (LF)*. Although such a grouping strategy is heuristic in some sense, it has been proven effective in Chen et al. (2009). After the grouping, we finally get two lists, containing bigram and trigram lexical dependencies respectively.

Based on the bigram and trigram lexical dependency lists, we propose a set of dependency features which is described in detail in the following. Here $s_i$ denotes the $i_{th}$ item from the top of the stack $S$, and $q_i$ the $i_{th}$ item from the front end of the queue $Q$. In addition, $s_iw$ ($s_it$) refers to the head word (POS) of $s_i$ and $q_iw$ ($q_it$) refers to the word (POS) of $q_i$.

### 3.4.1 Bigram Dependency Features

Bigram dependency features have a generic form of $f_{L/R}(w_1, w_2)$ which returns a group tag (HF, MF, or LF) if the lexical dependency $\langle w_1, w_2, L/R \rangle$ is found in the bigram lexical dependency list; else it returns *NULL*. The above feature template is instantiated into three pairs of features: $\{f_L(s_1w, s_0w), f_R(s_1w, s_0w)\}$, $\{f_L(s_0w, q_0w), f_R(s_0w, q_0w)\}$, and $\{f_L(s_1w, q_0w), f_R(s_1w, q_0w)\}$.

We also combine the above features with POS tags of $w_1$ and $w_2$. Thus we have three more pairs of features in the generic form of $f_{L/R}(w_1, w_2) \circ t(w_1) \circ t(w_2)$, where $t(w_i)$ represents the POS tag of the word $w_i$. All the bigram dependency features are listed in Table 3.

### 3.4.2 Trigram Dependency Features

Trigram dependency features have the generic form of $f_{L/R}(w_1, w_2, w_3)$. In this paper, this feature template is instantiated into two pairs of features. The feature function $f_L(s_1w, s_1rdw, s_0w)$ returns a group tag if $\langle s_1w, s_1rdw, s_0w, L \rangle$ is found in the trigram lexical dependency list, where $s_1rdw$ denotes the rightmost modifier of $s_1w$ that has been recognized so far during the shift-reduce parsing process. Note that $s_1rdw$ might be *NONE* if no right modifiers have been recognized for $s_1w$. The

other trigram dependency features, $f_R(s_1w, s_0ldw, s_0w)$, $f_L(s_0w, s_0rdw, q_0w)$, and $f_R(s_0w,$ *NONE*, $q_0w)$ can be explained in a similar way. As with bigram dependency features, POS tags are combined with above features to obtain richer feature representations. Trigram dependency features used in the paper are summarized in Table 3.

## 3.5 Parsing with Proposed Features

To use the proposed dependency features, we only need to update the scoring function defined in Section 2.3. The new scoring function is shown in the following.

$$Score'(\langle A, Y\rangle) = \sum_i \lambda_i f_i(\langle A, Y\rangle) + \sum_j \lambda_j^d f_j^d$$

where $f_i(\langle A, Y\rangle)$ are the baseline features and $f_j^d$ refer to the dependency features defined above.

# 4 Experimental Setup

## 4.1 Data Preparation

For English experiments, our labeled data came from the Wall Street Journal (WSJ) corpus of the Penn Treebank (Marcus et al., 1993). We used the standard divisions: sections 2-21 were used as training data, section 24 was used for system development, and section 23 was held out for performance evaluation. In terms of English unlabeled data, we used the TIPSTER corpus (LDC93T3A) which contains news articles from various sources, though in this paper we only used Wall Street Journal articles.

For Chinese experiments, we used Chinese Treebank (CTB) version 5.1 (Xue et al., 2005) as labeled data. Specifically, articles 001-270 and 440-1151 were used as training data, articles 271-300 were for evaluation, and articles 301-325 were development data. In the respect of Chinese unlabeled data, we utilized the corpus of Chinese Gigaword (LDC2003T09) after some basic cleanups.

We conducted necessary preprocessing on English and Chinese unlabeled data before they were fed to the baseline parser for automatic parsing. Specifically, we applied OpenNLP for English sentence boundary detection and tokenization.[5] For English POS tagging, SVMTool was used which achieves a per-token accuracy of 97.1% on section 23 of the WSJ corpus.[6] For the Chinese unlabeled data, we conducted sentence boundary detection simply according to sentence ending punctuations. Raw sentences were automatically segmented with a CRF-based word segmenter which achieves a segmentation accuracy of 97.2% on the testing data of CTB5.1. For automatic Chinese POS tagging, we utilized the Stanford POS tagger. [7] We trained the tagger on the CTB5.1 training data which has the accuracy of 95.4% on the CTB5.1 testing data.

Table 4 contains detailed data statistics of all the above corpora.

## 4.2 Performance Scoring

For performance evaluation in all the following experiments, we used *EVALB* to provide bracket scoring as well as complete match scoring.[8] For significance tests, we adopted the comparator developed by Daniel Bikel to compute *p-value*.[9]

[5] http://incubator.apache.org/opennlp/
[6] http://www.lsi.upc.edu/~nlp/SVMTool/
[7] http://nlp.stanford.edu/software/tagger.shtml
[8] http://nlp.cs.nyu.edu/evalb
[9] http://www.cis.upenn.edu/~dbikel/download/compare.pl

| Language | Statistics | Train | Dev | Test | Unlabeled |
|---|---|---|---|---|---|
| English | # sentences | 39.8k | 1.7k | 2.4k | 3,139.1k |
| | # words | 950.0k | 40.1k | 56.7k | 76,041.4k* |
| | # ave. length | 28.9 | 25.1 | 25.1 | 25.22* |
| Chinese | # sentences | 18.1k | 350 | 348 | 11,810.7k |
| | # words | 493.8k | 8.0k | 6.8k | 269,057.2k* |
| | # ave. length | 27.3 | 19.5 | 23.0 | 22.8* |

Table 4: Data statistics including the number of words and sentences, together with average sentence length. * The numbers are approximate due to the use of automatic preprocessing techniques.

| | English | | | Chinese | | |
|---|---|---|---|---|---|---|
| **Data Source** | **LR** | **LP** | **F1** | **LR** | **LP** | **F1** |
| Baseline | 88.2 | 88.2 | 88.2 | 83.7 | 84.4 | 84.0 |
| Human-Labeled | 88.5 | 88.7 | 88.6 | 83.8 | 84.4 | 84.1 |
| Auto-Parsed | 89.1 | 89.4 | 89.3 | 85.2 | 85.1 | 85.1 |
| Combined | 89.3 | 89.5 | 89.4 | 85.3 | 85.2 | 85.2 |

Table 5: Comparative results on English and Chinese developments sets with lexical dependencies extracted from diverse sources.

### 4.3 Running Parameters

We set the beam size to 16 in both training and decoding which maintains a good trade-off between parsing efficiency and accuracy (Zhang and Clark, 2009). With respect to the iteration number of perceptron learning, we tuned the parameter on the English and Chinese development sets and finally set the value to 21 for both English and Chinese experiments.

## 5 Experimental Results

### 5.1 Comparison of Different Sources

Table 5 shows the comparative results on English and Chinese development sets with lexical dependencies obtained from different sources. We experimented with four different settings: no lexical dependency information was used (Baseline), lexical dependencies were solely from human-labeled training data (Human-Labeled), lexical dependencies were solely from auto-parsed data (Auto-Parsed), and lexical dependencies were from the combination of human-labeled and auto-parsed data (Combined). For the latter three settings, all the dependency features listed in Table 3 were incorporated. In the data combination, we simply gave our human-labeled training data a relative weight of one.

From the results we can see that, although lexical dependency information from human-labeled training data can improve the performance on both English and Chinese, the improvement on Chinese is marginal (+0.1% in F1-score). One reason is that lexical dependencies from human-labeled training data have a data sparseness problem. By contrast, the use of large-scale auto-parsed data brings on much bigger improvements (+1.1% in F1-score on both English and Chinese). In addition, we find that data combination has trivial effect on the performance. One possible reason is that lexical dependencies form human-labeled training data are overwhelmed by those from auto-parsed data. We will leave further discussions on data combination to our future work and focus on the setting of obtaining lexical dependencies from auto-parsed data.

| Sentences with #Words ≤ 40 | | | | |
|---|---|---|---|---|
| **Features** | **LR** | **LP** | **F1** | **EX** |
| Baseline | 90.1 | 89.8 | 90.0 | 41.3 |
| +Bigram Features | 90.6 | 90.5 | 90.6 | 42.1 |
| +Trigram Features | 90.9 | 90.9 | 90.9 | 42.9 |
| Sentences with Unlimited Words | | | | |
| **Features** | **LR** | **LP** | **F1** | **EX** |
| Baseline | 89.6 | 89.4 | 89.5 | 39.0 |
| +Bigram Features | 90.1 | 90.1 | 90.1 | 39.7 |
| +Trigram Features | 90.4 | 90.5 | 90.4 | 40.7 |

Table 6: Main results on section 23 of the WSJ corpus, using automatically assigned POS tags and lexical dependencies extracted from auto-parsed data. Two types of dependency features are added **incrementally**.

## 5.2 Main Results on English Data

Table 6 shows the main results on the English test set, where the two types of dependency features were added incrementally to the baseline parser. As the results show, both bigram and trigram dependency features have positive effect on the parsing accuracy. Specifically, on the whole test set the overall improvement over the baseline parser is +0.9% in F1-score, where bigram dependency features contribute an absolute 0.6% improvement and trigram dependency features further improve the performance by 0.3% over the results of using bigram dependency features. Significance tests show that the overall improvement on the whole test set is statistically significant on the level of $p < 10^{-4}$.

## 5.3 Comparative Results on English

Table 7 shows the comparison of our parser with a large body of representative related work. For fair comparison, here we disregarded parsers that are based on combination methods such as Petrov (2010) and Zhang et al. (2009). Following the taxonomy adopted in Huang et al. (2010), we grouped the related work into single parsers (SINGLE), discriminative reranking approaches (RE), and self-training (SELF). Note that our parser belongs to the category of self-training. From the results we can see that our parser outperforms all the single parsers listed in the table except Carreras et al. (2008). However, compared with Carreras et al. (2008), our parser has much smaller time-complexity: $O(nK)$ vs. $O(n^3G)$, where $K$ is the beam size used in our parser and $G$ is a grammar constant in Carreras et al. (2008). Compared with reranking and self-training parsers, our parser has relatively low parsing accuracy. But the reranking technique is actually complementary with our approach, so we might enhance our parser with this technique in the future.

## 5.4 Main Results on Chinese Data

In parallel to the results on the English test set, Table 8 shows the main results on the Chinese test set, using auto-assigned POS tags and lexical dependencies extracted from auto-parsed data. From the results on the whole test set we can see that dependency features contribute a bigger absolute improvement on Chinese than that on English (+1.1% vs. +0.9%). One possible reason is that the size of Chinese unlabeled data used in the paper is much bigger. Significance tests show that the overall improvement induced by bigram and trigram dependency features on the whole test is statistically significant on the level of $p < 10^{-3}$. These results indicate that the new features are very effective.

| Type | Parser | | LR | LP | F1 |
|---|---|---|---|---|---|
| SINGLE | Ratnaparkhi (1997) | | 86.3 | 87.5 | 86.9 |
| | Collins (1999) | | 88.1 | 88.3 | 88.2 |
| | Charniak (2000) | | 89.5 | 89.9 | 89.5 |
| | Sagae and Lavie (2005)* | | 86.1 | 86.0 | 86.0 |
| | Sagae and Lavie (2006)* | | 87.8 | 88.1 | 87.9 |
| | Petrov and Klein (2007) | | 90.1 | 90.2 | 90.1 |
| | Carreras et al. (2008) | | 90.7 | 91.4 | 91.1 |
| RE | Charniak and Johnson (2005) | | 91.2 | 91.8 | 91.0 |
| | Huang (2008) | | 92.2 | 91.2 | 91.7 |
| SELF | Huang and Harper (2009) | | 91.1 | 91.6 | 91.3 |
| | McClosky et al. (2006) | | 92.1 | 92.5 | 92.3 |
| | Huang et al. (2010)† | | 91.4 | 91.8 | 91.7 |
| | This Paper | Baseline | 89.6 | 89.4 | 89.5 |
| | | Auto-Parsed | 90.4 | 90.5 | 90.4 |

Table 7: Comparison with related work on section 23 of the WSJ corpus with automatically assigned POS tags. * The parsers based on shift-reduce parsing. † The results of self-training with a single latent annotation grammar.

Sentences with #Words≤ 40

| Features | LR | LP | F1 | EX |
|---|---|---|---|---|
| Baseline | 82.9 | 83.6 | 83.2 | 32.8 |
| +Bi-lexical Features | 84.3 | 85.1 | 84.7 | 32.8 |
| +Tri-lexical Features | 84.7 | 85.9 | 85.3 | 34.1 |
| Sentences with Unlimited Words | | | | |
| **Features** | **LR** | **LP** | **F1** | **EX** |
| Baseline | 79.5 | 80.7 | 80.1 | 28.2 |
| +Bi-lexical Features | 80.3 | 81.6 | 80.9 | 28.2 |
| +Tri-lexical Features | 80.6 | 81.9 | 81.2 | 29.3 |

Table 8: Main results on the CTB 5.1 test set, using automatically assigned POS tags and lexical dependencies extracted from auto-parsed data. Two types of dependency features are added **incrementally**.

## 5.5 Comparative Results on Chinese

Comparing Chinese constituency parsers is difficult in the sense that previously reported results were achieved frequently on different versions of CTB and/or with different data split standards. Zhang and Clark (2009) presented a detailed comparison between the baseline parser of this paper and a large body of related work on CTB 2.0. Here we only compared our parser with the parsers available on the web for Chinese parsing, as shown in Table 9. From the results we can see that, on Chinese parsing our parser outperforms Bikel (2004) and Charniak (2000) by 0.6% and 0.4%, respectively. However, our parser lags behind Petrov and Klein (2007) and the reranking parser (Charniak and Johnson, 2005).

## 5.6 Additional Analysis

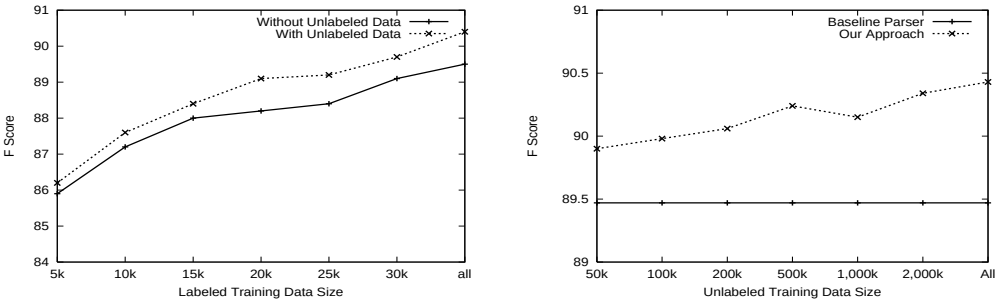
We performed several types of analysis, focusing on English, to investigate the effect of different sizes of human-labeled training data and auto-parsed data, as well as how lexical dependency information changes the distribution of first mistakes made by the baseline parser.

| Type | Parser | | LP | LR | F1 |
|---|---|---|---|---|---|
| SINGLE | Charniak (2000)* | | 79.6 | 82.1 | 80.8 |
| | Bikel (2004)† | | 79.3 | 82.0 | 80.6 |
| | Petrov and Klein (2007) | | 81.9 | 84.8 | 83.3 |
| RE | Charniak and Johnson (2005)* | | 80.8 | 83.8 | 82.3 |
| | This Paper | Baseline | 79.5 | 80.7 | 80.1 |
| | | Auto-Parsed | 80.6 | 81.9 | 81.2 |

Table 9: Comparison with related work on the test set of CTB 5.1 with automatically assigned POS tags. * Huang (2009) adapted the parsers to Chinese parsing on CTB 5.1. † We run the parser on CTB 5.1 to get the results.

### 5.6.1 Effect of Different Sizes of Labeled and Unlabeled Data

We studied the effect of varying sizes of human-labeled training data by a randomly sampling from sections 2-21. Meanwhile, we always used the whole set of auto-parsed data. The results are depicted in Figure 2-(a). As the results show, auto-parsed data improves parsing accuracy even when the human-labeled training data is small in size. In addition, by using our approach, a fraction of sections 2-21 plus the whole set of auto-parsed data is sufficient to achieve the F1-scores obtained by the parser trained solely on the whole sections 2-21.




(a) varying labeled training data (b) varying unlabeled data

Figure 2: Results with varying sizes of human-labeled training data and unlabeled data.

We also examined the effect of varying sizes of unlabeled data. In this experiment, we used sections 2-21 as human-labeled training data and changed the size of unlabeled data through random sampling. The results are depicted in Figure 2-(b). From the results we can see that improvements achieved by using unlabeled data are enlarged with the increment of the size of unlabeled data until the performance finally levels off.

### 5.6.2 Reduction on First Mistakes

The baseline parser made 1,522 first mistakes on the English test set. We analyzed how our approach changed the first mistakes. We grouped the changes into four cases. *No-Change* (1,090) refers to the case that the baseline and our parser make the same first mistakes at the same positions. In the case of *Correct* (249), first mistakes made by the baseline parser are corrected by our parser. *Wrong* (121) means that our parser makes first mistakes earlier than the baseline parser. Finally, *Others* (47) refers to the case that our parser and the baseline parser make first mistakes of different

types at the same positions. In addition, we are especially interested in how our approach reduces first mistakes of the type *shift vs. reduce-binary*. So we compared the numbers of first mistakes of this type in the *Correct* and *Wrong* cases, which are 168 and 78 respectively.

## 6 Related Work

Shift-reduce parsing has been widely studied for constituency parsing. Sagae and Lavie (2005) proposed a classifier-based shift-reduce parsing algorithm which was extended with a best-first search strategy in Sagae and Lavie (2006). Wang et al. (2006) adapted the parser in Sagae and Lavie (2005) to Chinese parsing and compared some representative classifiers. Zhang and Clark (2009) proposed a global learning algorithm to replace local classifiers. Shift-reduce parsing has also widely applied to parsing with other grammars (Nivre, 2004; Zhang and Clark, 2008; Huang and Sagae, 2010; Zhang and Clark, 2011). In this paper we focus on improving Zhang and Clark (2009) with a set of novel features defined on lexical dependencies obtained from auto-parsed data.

The approach used in this paper belongs to the category of semi-supervised learning. In the respect of semi-supervised learning for constituency parsing, self-training has been extensively studied (McClosky et al., 2006; Huang and Harper, 2009; Huang et al., 2010). The difference is that we use partial information derived from auto-parsed data instead of entire automatically parsed trees. Chen et al. (2009) and Noord (2007) exploited lexical dependencies from unlabeled data for dependency and HPSG parsing, respectively. In this paper we for the first time use lexical dependency information for advancing the state-of-the-art shift-reduce constituency parsers. It is noteworthy that, although Chen et al. (2009) and our work use the same strategy to extract lexical dependencies, features defined on lexical dependencies are distinct. Specifically, Chen et al. (2009) proposed features for a graph-based dependency parser while this paper focuses on a transition-based constituency parser. More recently, Bansal and Klein (2011) proposed features for both dependency and constituency parsing based on Web counts from the Google n-grams corpus. By contrast, lexical dependency information used in this paper is derived from auto-parsed data. Moreover, Web counts from the Google n-grams corpus represent surface evidence of lexical affinities while lexical dependencies are able to encode information on deep syntactic structures.

## 7 Conclusion

We have presented a method to utilize lexical dependency information to improve shift-reduce constituency parsing. A set of new features was proposed based on the lexical dependency information and integrated into the shift-reduce parser. Our method well addressed the action conflict problem. We evaluated the proposed method on English and Chinese data. The results show that our new parsers provide comparable accuracies with state-of-the-part parsers while maintaining the advantage in parsing speed.

## Acknowledgments

We would like to thank Wenliang Chen for his help in extracting lexical dependencies and discussions on designing features. This work was supported in part by the National Science Foundation of China (61073140, 61272376, 61100089, 61003159), Specialized Research Fund for the Doctoral Program of Higher Education (20100042110031) and the Fundamental Research Funds for the Central Universities.

## References


Bansal, M. and Klein, D. (2011). Web-scale features for full-scale parsing. In *the 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)*, pages 693–702.

Bikel, D. M. (2004). On the parameter space of generative lexicalized statistical parsing models. In *Ph.D. thesis, University of Pennsylvania*.

Carreras, X., Collins, M., and Koo, T. (2008). Tag, dynamic programming and the perceptron for efficient, feature-rich parsing. In *Conference on Computational Natural Language Learning (CoNLL 2008)*, pages 9–16.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2000)*, pages 132–139.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In *the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, pages 173–180.

Chen, W., Kazama, J., Uchimoto, K., and Torisawa, K. (2009). Improving dependency parsing with subtrees from auto-parsed data. In *the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP 2009)*, pages 570–579.

Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. In *the 34th Annual Meeting of the Associaton for Computational Linguistics (ACL 1996)*.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *the 35th Annual Meeting of the Associaton for Computational Linguistics (ACL 1997)*.

Collins, M. (1999). Head-driven statistical models for natural language parsing. In *Ph.D. thesis, University of Pennsylvania*.

Collins, M. (2002). Discriminative training methods for hidden markov models: theory and experiemnts with perceptron algorithm. In *the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 1–8.

Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *the 32rd Annual Meeting of the Association for Computational Linguistics (ACL 2004)*.

Eisner, J. and Satta, G. (1999). Efficient parsing for bilexical context-free grammars and head automaton grammars. In *the 37th Annual Meeting of the Association for Computational Linguistics (ACL 1999)*.

Huang, L. and Sagae, K. (2010). Dynamic programming for linear-time incremental parsing. In *the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010)*, pages 1077–1086.

Huang, L.-Y. (2009). Improve chinese parsing with max-ent reranking parser. In *Master Project Report, Brown University*.

Huang, Z. and Harper, M. (2009). Self-training PCFG grammars with latent annotations across languages. In *the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP 2009)*, pages 832–841.

Huang, Z., Harper, M., and Petrov, S. (2010). Self-training with products of latent variable grammars. In *the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP 2010)*, pages 12–22.

Marcus, M. P., Santorini, B., and Marcinkiewiz, M. A. (1993). Building a large annotated corpus of English. In *Computational Linguistics, 19(2)*, pages 313–330.

McClosky, D., Charniak, E., and Johnson, M. (2006). Reranking and self-training for parser-adaptation. In *the 44th Annual Meeting of the Association for Computational Linguistics and 21st INternational Conference on Computational Linguistics (ACL-COLING 2006)*, pages 337–344.

Nivre, J. (2004). Incrementality in deterministic dependency parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together: Workshop at ACL 2004*.

Noord, G. (2007). Using self-trained bilexical preferences to improve disambiguation accuracy. In *the 10th International Conference on Parsing Technologies (IWPT 2007)*.

Petrov, S. (2010). Products of random latent variable grammars. In *Human Language Technologies: The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2010)*, pages 19–27.

Petrov, S. and Klein, D. (2007). Improved inference for unlexicalized parsing. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2007)*, pages 404–411.

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. In *the 1997 Conference on Empirical Methods in Natural Language Processing (EMNLP 1997)*.

Sagae, K. and Lavie, A. (2005). A classifier-based parser with linear run-time complexity. In *the 9th International Conference on Parsing Technologies (IWPT 2005)*, pages 125–132.

Sagae, K. and Lavie, A. (2006). A best-first probabilistic shift-reduce parser. In *the 44th Annual Meeting of the Association for Computational Linguistics and 21st INternational Conference on Computational Linguistics (ACL-COLING 2006)*, pages 691–698.

Wang, M., Sagae, K., and Mitamura, T. (2006). A fast, accurate deterministic parser for Chinese. In *the 44th Annual Meeting of the Association for Computational Linguistics and 21st INternational Conference on Computational Linguistics (ACL-COLING 2006)*, pages 25–32.

Xue, N., Xia, F., dong Chiou, F., and Palmer, M. (2005). The Penn Chinese treebank: phrase structure annotation of a large corpus. In *Natural Language Engineering, 11(2)*, pages 207–238.

Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *the 8th International Conference on Parsing Technologies (IWPT 2003)*, pages 195–206.

Zhang, H., Zhang, M., Tan, C. L., and Li, H. (2009). K-best combination of syntactic parsers. In *the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP 2009)*, pages 1552–1560.

Zhang, Y. and Clark, S. (2008). A Tale of two parsers: investigating and combining graph-based and transition-based dependency parsing. In *the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP 2008) (EMNLP 2008)*, pages 562–571.

Zhang, Y. and Clark, S. (2009). Transition-based parsing of the Chinese treebank using a global discriminative model. In *the 11th International Conference on Parsing Technologies (IWPT 2009)*, pages 162–171.

Zhang, Y. and Clark, S. (2011). Shift-reduce CCG parsing. In *the 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)*, pages 683–692.