Transition-Based Parsing of the Chinese Treebank using a Global Discriminative Model

Yue Zhang
Oxford University
Computing Laboratory
yue.zhang@comlab.ox.ac.uk

Stephen Clark
Cambridge University
Computer Laboratory
stephen.clark@cl.cam.ac.uk

Abstract

Transition-based approaches have shown competitive performance on constituent and dependency parsing of Chinese. State-of-the-art accuracies have been achieved by a deterministic shift-reduce parsing model on parsing the Chinese Treebank 2 data (Wang et al., 2006). In this paper, we propose a global discriminative model based on the shift-reduce parsing process, combined with a beam-search decoder, obtaining competitive accuracies on CTB2. We also report the performance of the parser on CTB5 data, obtaining the highest scores in the literature for a dependency-based evaluation.

1 Introduction

Transition-based statistical parsing associates scores with each decision in the parsing process, selecting the parse which is built by the highest scoring sequence of decisions (Briscoe and Carroll, 1993; Nivre et al., 2006). The parsing algorithm is typically some form of bottom-up shift-reduce algorithm, so that scores are associated with actions such as *shift* and *reduce*. One advantage of this approach is that the parsing can be highly efficient, for example by pursuing a greedy strategy in which a single action is chosen at each decision point.

The alternative approach, exemplified by Collins (1997) and Charniak (2000), is to use a chart-based algorithm to build the space of possible parses, together with pruning of low-probability constituents and the Viterbi algorithm to find the highest scoring parse. For English dependency parsing, the two approaches give similar results (McDonald et al., 2005; Nivre et al., 2006). For English constituent-based parsing using the Penn Treebank, the best performing transition-based parser lags behind the current state-of-the-art (Sagae and Lavie, 2005). In contrast, for Chinese, the best dependency parsers are currently transition-based (Duan et al., 2007; Zhang and Clark, 2008). For constituent-based parsing using the Chinese Treebank (CTB), Wang et al. (2006) have shown that a shift-reduce parser can give competitive accuracy scores together with high speeds, by using an SVM to make a single decision at each point in the parsing process.

In this paper we describe a global discriminative model for Chinese shift-reduce parsing, and compare it with Wang et al.'s approach. We apply the same shift-reduce procedure as Wang et al. (2006), but instead of using a local classifier for each transition-based action, we train a generalized perceptron model over complete sequences of actions, so that the parameters are learned in the context of complete parses. We apply beam search to decoding instead of greedy search. The parser still operates in linear time, but the use of beam-search allows the correction of local decision errors by global comparison. Using CTB2, our model achieved Parseval F-scores comparable to Wang et al.'s approach. We also present accuracy scores for the much larger CTB5, using both a constituent-based and dependency-based evaluation. The scores for the dependency-based evaluation were higher than the state-of-the-art dependency parsers for the CTB5 data.

2 The Shift-Reduce Parsing Process

The shift-reduce process used by our beam-search decoder is based on the greedy shift-reduce parsers of Sagae and Lavie (2005) and Wang et al. (2006).

Proceedings of the 11th International Conference on Parsing Technologies (IWPT), pages 162–171,
Paris, October 2009.

The process assumes binary-branching trees; section 2.1 explains how these are obtained from the arbitrary-branching trees in the Chinese Treebank.

The input is assumed to be segmented and POS tagged, and the word-POS pairs waiting to be processed are stored in a queue. A stack holds the partial parse trees that are built during the parsing process. A parse *state* is defined as a ⟨stack,queue⟩ pair. Parser actions, including SHIFT and various kinds of REDUCE, define functions from states to states by shifting word-POS pairs onto the stack and building partial parse trees.

The actions used by the parser are:

- SHIFT, which pushes the next word-POS pair in the queue onto the stack;
- REDUCE–unary–X, which makes a new unary-branching node with label X; the stack is popped and the popped node becomes the child of the new node; the new node is pushed onto the stack;
- REDUCE–binary–{L/R}–X, which makes a new binary-branching node with label X; the stack is popped twice, with the first popped node becoming the right child of the new node and the second popped node becoming the left child; the new node is pushed onto the stack;
- TERMINATE, which pops the root node off the stack and ends parsing. This action is novel in our parser. Sagae and Lavie (2005) and Wang et al. (2006) only used the first three transition actions, setting the final state as all incoming words having been processed, and the stack containing only one node. However, there are a small number of sentences (14 out of 3475 from the training data) that have unary-branching roots. For these sentences, Wang's parser will be unable to produce the unary-branching roots because the parsing process terminates as soon as the root is found. We define a separate action to terminate parsing, allowing unary reduces to be applied to the root item before parsing finishes.

The trees built by the parser are lexicalized, using the head-finding rules from Zhang and Clark (2008). The left (L) and right (R) versions of the REDUCE-binary rules indicate whether the head of the new node is to be taken from the left or right child. Note also that, since the parser is building binary trees, the X label in the REDUCE rules can be one of the temporary constituent labels, such as NP*, which are needed for the binarization process described in Section 2.1. Hence the number of left and right binary reduce rules is the number of constituent labels in the binarized grammar.

for node $Y = X_1..X_m \in T$:
  if $m > 2$:
    find the head node $X_k (1 \leq k \leq m)$ of Y
    $m' = m$
    while $m' > k$ and $m' > 2$:
      new node $Y^* = X_1..X_{m'-1}$
      $Y \leftarrow Y^* X_{m'}$
      $m' = m' - 1$
    $n' = 1$
    while $n' < k$ and $k - n' > 1$:
      new node $Y^* = X_{n'}..X_k$
      $Y \leftarrow X_{n'} Y^*$
      $n' = n' + 1$

Figure 2: the binarization algorithm with input T

Wang et al. (2006) give a detailed example showing how a segmented and POS-tagged sentence can be incrementally processed using the shift-reduce actions to produce a binary tree. We show this example in Figure 1.

2.1 The binarization process

The algorithm in Figure 2 is used to map CTB trees into binarized trees, which are required by the shift-reduce parsing process. For any tree node with more than two child nodes, the algorithm works by first finding the head node, and then processing its right-hand-side and left-hand-side, respectively. The head-finding rules are taken from Zhang and Clark (2008). $Y = X_1..X_m$ represents a tree node Y with child nodes $X_1...X_m (m \geq 1)$.

The label of the newly generated node Y^* is based on the constituent label of the original node Y, but marked with an asterix. Hence binarization enlarges the set of constituent labels. We call the constituents marked with * *temporary* constituents. The binarization process is reversible, in that output from the shift-reduce parser can be unbinarized into CTB format, which is required for evaluation.

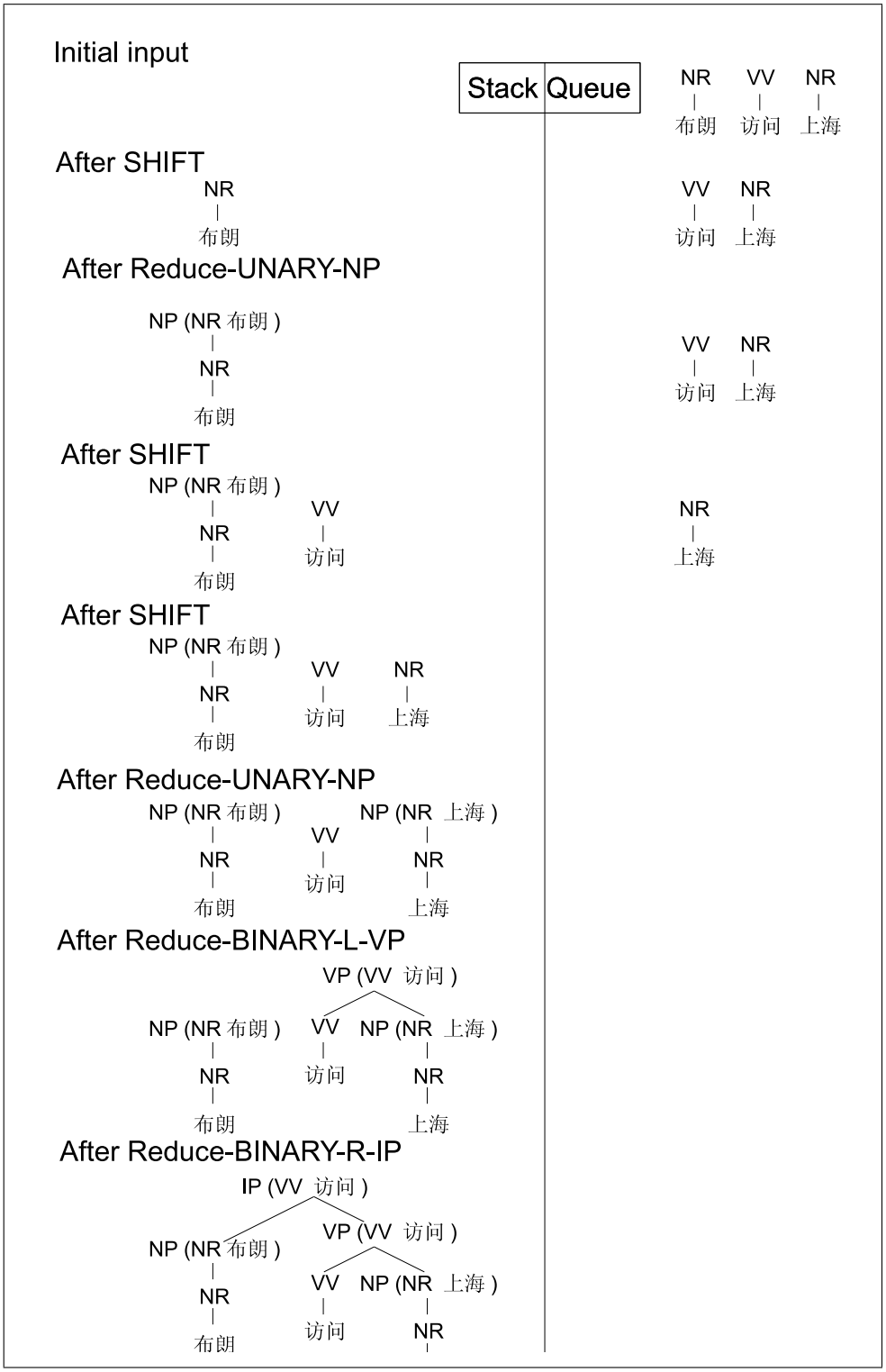

Figure 1: An example shift-reduce parsing process, adopted from Wang et al. (2006)

2.2 Restrictions on the sequence of actions

Not all sequences of actions produce valid binarized trees. In the deterministic parser of Wang et al. (2006), the highest scoring action predicted by the classifier may prevent a valid binary tree from being built. In this case, Wang et al. simply return a partial parse consisting of all the subtrees on the stack.

In our parser a set of restrictions is applied which guarantees a valid parse tree. For example, two simple restrictions are that a SHIFT action can only be applied if the queue of incoming words

Variables: state item $item = (S, Q)$, where
S is stack and Q is incoming queue;
the agenda $agenda$;
list of state items $next$;
Algorithm:
for $item \in agenda$:
if $item$.score = $agenda$.bestScore and $item$.isFinished:
$rval = item$
break
next = []
for $move \in item$.legalMoves:
next.push(item.TakeAction(move))
agenda = next.getBBest()
Outputs: $rval$

Figure 3: the beam-search decoding algorithm

is non-empty, and the binary reduce actions can only be performed if the stack contains at least two nodes. Some of the restrictions are more complex than this; the full set is listed in the Appendix.

3 Decoding with Beam Search

Our decoder is based on the incremental shift-reduce parsing process described in Section 2. We apply beam-search, keeping the B highest scoring state items in an agenda during the parsing process. The agenda is initialized with a state item containing the starting state, i.e. an empty stack and a queue consisting of all word-POS pairs from the sentence.

At each stage in the decoding process, existing items from the agenda are progressed by applying legal parsing actions. From all newly generated state items, the B highest scoring are put back on the agenda. The decoding process is terminated when the highest scored state item in the agenda reaches the final state. If multiple state items have the same highest score, parsing terminates if any of them are finished. The algorithm is shown in Figure 3.

4 Model and Learning Algorithm

We use a linear model to score state items. Recall that a parser state is a ⟨stack,queue⟩ pair, with the stack holding subtrees and the queue holding incoming words waiting to be processed. The score

Inputs: training examples (x_i, y_i)
Initialization: set $\vec{w} = 0$
Algorithm:
for $t = 1..T$, $i = 1..N$:
$z_i = \mathit{parse}(x_i, \vec{w})$
if $z_i \neq y_i$:
$\vec{w} = \vec{w} + \Phi(y_i) - \Phi(z_i)$
Outputs: $\vec{w}$

Figure 4: the perceptron learning algorithm

for state item Y is defined by:

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

where $\Phi(Y)$ is the global feature vector from Y, and $\vec{w}$ is the weight vector defined by the model. Each element from $\Phi(Y)$ represents the global count of a particular feature from Y. The feature set consists of a large number of features which pick out various configurations from the stack and queue, based on the words and subtrees in the state item. The features are described in Section 4.1. The weight values are set using the generalized perceptron algorithm (Collins, 2002).

The perceptron algorithm is shown in Figure 4. It initializes weight values as all zeros, and uses the current model to decode training examples (the *parse* function in the pseudo-code). If the output is correct, it passes on to the next example. If the output is incorrect, it adjusts the weight values by adding the feature vector from the gold-standard output and subtracting the feature vector from the parser output. Weight values are updated for each example (making the process *online*) and the training data is iterated over T times. In order to avoid overfitting we used the now-standard averaged version of this algorithm (Collins, 2002).

We also apply the *early update* modification from Collins and Roark (2004). If the agenda, at any point during the decoding process, does not contain the correct partial parse, it is not possible for the decoder to produce the correct output. In this case, decoding is stopped early and the weight values are updated using the highest scoring partial parse on the agenda.

4.1 Feature set

Table 1 shows the set of feature templates for the model. Individual features are generated from

Description	Feature templates
Unigrams	S_0tc, S_0wc, S_1tc, S_1wc,
	S_2tc, S_2wc, S_3tc, S_3wc,
	N_0wt, N_1wt, N_2wt, N_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_0wN_0w, S_0wN_0t, S_0cN_0w, S_0cN_0t,
	N_0wN_1w, N_0wN_1t, N_0tN_1w, N_0tN_1t
	S_1wN_0w, S_1wN_0t, S_1cN_0w, S_1cN_0t,
Trigrams	$S_0cS_1cS_2c$, $S_0wS_1cS_2c$,
	$S_0cS_1wS_2c$, $S_0cS_1cS_2w$,
	$S_0cS_1cN_0t$, $S_0wS_1cN_0t$,
	$S_0cS_1wN_0t$, $S_0cS_1cN_0w$
Bracket	S_0wb, S_0cb
	S_0wS_1cb, S_0cS_1wb, S_0cS_1cb
	S_0wN_0tb, S_0cN_0wb, S_0cN_0tb
Separator	S_0wp, S_0wcp, S_0wq, S_0wcq,
	S_1wp, S_1wcp, S_1wq, S_1wcq
	S_0cS_1cp, S_0cS_1cq

Table 1: Feature templates

these templates by first instantiating a template with particular labels, words and tags, and then pairing the instantiated template with a particular action. In the table, the symbols S_0, S_1, S_2, and S_3 represent the top four nodes on the stack, and the symbols N_0, N_1, N_2 and N_3 represent the first four words in the incoming queue. S_0L, S_0R and S_0U represent the left and right child for binary branching S_0, and the single child for unary branching S_0, respectively; w represents the lexical head token for a node; c represents the label for a node. When the corresponding node is a terminal, c represents its POS-tag, whereas when the corresponding node is non-terminal, c represents its constituent label; t represents the POS-tag for a word.

The context S_0, S_1, S_2, S_3 and N_0, N_1, N_2, N_3 for the feature templates is taken from Wang et al. (2006). However, Wang et al. (2006) used a polynomial kernel function with an SVM and did not manually create feature combinations. Since we used the linear perceptron algorithm we manually combined Unigram features into Bigram and Trigram features.

The "Bracket" row shows bracket-related features, which were inspired by Wang et al. (2006). Here brackets refer to left brackets including "（", "“" and "《" and right brackets including "）", "”" and "》". In the table, b represents the matching status of the last left bracket (if any) on the stack. It takes three different values: 1 (no matching right bracket has been pushed onto stack), 2 (a matching right bracket has been pushed onto stack) and 3 (a matching right bracket has been pushed onto stack, but then popped off).

The "Separator" row shows features that include one of the separator punctuations (i.e. "，", "。", "、" and "；") between the head words of S_0 and S_1. These templates apply only when the stack contains at least two nodes; p represents a separator punctuation symbol. Each unique separator punctuation between S_0 and S_1 is only counted once when generating the global feature vector. q represents the count of any separator punctuation between S_0 and S_1.

Whenever an action is being considered at each point in the beam-search process, templates from Table 1 are matched with the context defined by the parser state and combined with the action to generate features. Negative features, which are the features from incorrect parser outputs but not from any training example, are included in the model. There are around a million features in our experiments with the CTB2 dataset.

Wang et al. (2006) used a range of other features, including rhythmic features of S_0 and S_1 (Sun and Jurafsky, 2003), features from the most recently found node that is to the left or right of S_0 and S_1, the number of words and the number of punctuations in S_0 and S_1, the distance between S_0 and S_1 and so on. We did not include these features in our parser, because they did not lead to improved performance during development experiments.

5 Experiments

The experiments were performed using the Chinese Treebank 2 and Chinese Treebank 5 data. Standard data preparation was performed before the experiments: empty terminal nodes were removed; any non-terminal nodes with no children were removed; any unary $X \rightarrow X$ nodes resulting from the previous steps were collapsed into one X node.

For all experiments, we used the EVALB tool[1] for evaluation, and used labeled recall (LR), labeled precision (LP) and $F1$ score (which is the

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

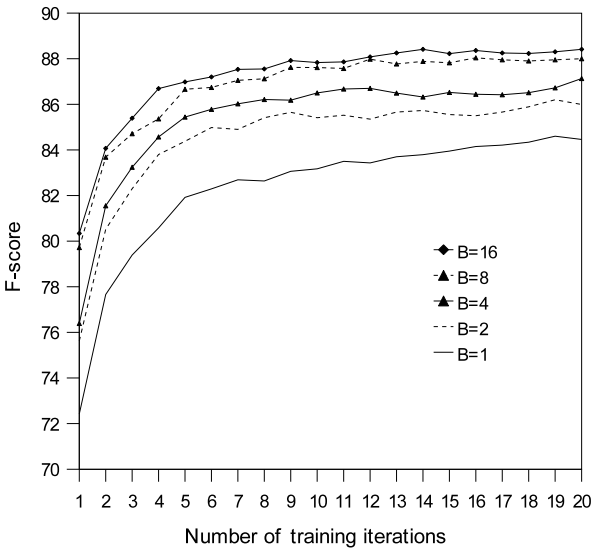

Figure 5: The influence of beam-size

	Sections	Sentences	Words
Training	001–270	3475	85,058
Development	301–325	355	6,821
Test	271–300	348	8,008

Table 2: The standard split of CTB2 data

harmonic mean of LR and LP) to measure parsing accuracy.

5.1 The influence of beam-size

Figure 5 shows the accuracy curves using different beam-sizes for the decoder. The number of training iterations is on the x-axis with F-score on the y-axis. The tests were performed using the development test data and gold-standard POS-tags. The figure shows the benefit of using a beam size greater than 1, with comparatively little accuracy gain being obtained beyond a beam size of 8. Hence we set the beam size to 16 for the rest of the experiments.

5.2 Test results on CTB2

The experiments in this section were performed using CTB2 to allow comparison with previous work, with the CTB2 data extracted from Chinese Treebank 5 (CTB5). The data was split into training, development test and test sets, as shown in Table 2, which is consistent with Wang et al. (2006) and earlier work. The tests were performed using both gold-standard POS-tags and POS-tags automatically assigned by a POS-tagger. We used our own implementation of the perceptron-based tagger from Collins (2002).

Model	LR	LP	$F1$
Bikel Thesis	80.9%	84.5%	82.7%
Wang 2006 SVM	87.2%	88.3%	87.8%
Wang 2006 Stacked	88.3%	88.1%	88.2%
Our parser	89.4%	90.1%	89.8%

Table 3: Accuracies on CTB2 with gold-standard POS-tags

The results of various models measured using sentences with less than 40 words and using gold-standard POS-tags are shown in Table 3. The rows represent the model from Bikel and Chiang (2000), Bikel (2004), the SVM and ensemble models from Wang et al. (2006), and our parser, respectively. The accuracy of our parser is competitive using this test set.

The results of various models using automatically assigned POS-tags are shown in Table 4. The rows in the table represent the models from Bikel and Chiang (2000), Levy and Manning (2003), Xiong et al. (2005), Bikel (2004), Chiang and Bikel (2002), the SVM model from Wang et al. (2006) and the ensemble system from Wang et al. (2006), and the parser of this paper, respectively. Our parser gave comparable accuracies to the SVM and ensemble models from Wang et al. (2006). However, comparison with Table 3 shows that our parser is more sensitive to POS-tagging errors than some of the other models. One possible reason is that some of the other parsers, e.g. Bikel (2004), use the parser model itself to resolve tagging ambiguities, whereas we rely on a POS tagger to accurately assign a single tag to each word. In fact, for the Chinese data, POS tagging accuracy is not very high, with the perceptron-based tagger achieving an accuracy of only 93%. The beam-search decoding framework we use could accommodate joint parsing and tagging, although the use of features based on the tags of incoming words complicates matters somewhat, since these features rely on tags having been assigned to all words in a pre-processing step. We leave this problem for future work.

In a recent paper, Petrov and Klein (2007) reported LR and LP of 85.7% and 86.9% for sentences with less than 40 words and 81.9% and 84.8% for all sentences on the CTB2 test set, re-

	≤ 40 words				≤ 100 words				Unlimited			
	LR	LP	$F1$	POS	LR	LP	$F1$	POS	LR	LP	$F1$	POS
Bikel 2000	76.8%	77.8%	77.3%	-	73.3%	74.6%	74.0%	-	-	-	-	-
Levy 2003	79.2%	78.4%	78.8%	-	-	-	-	-	-	-	-	-
Xiong 2005	78.7%	80.1%	79.4%	-	-	-	-	-	-	-	-	-
Bikel Thesis	78.0%	81.2%	79.6%	-	74.4%	78.5%	76.4%	-	-	-	-	-
Chiang 2002	78.8%	81.1%	79.9%	-	75.2%	78.0%	76.6%	-	-	-	-	-
Wang 2006 SVM	78.1%	81.1%	79.6%	92.5%	75.5%	78.5%	77.0%	92.2%	75.0%	78.0%	76.5%	92.1%
Wang 2006 Stacked	79.2%	81.1%	80.1%	92.5%	76.7%	78.4%	77.5%	92.2%	76.2%	78.0%	77.1%	92.1%
Our parser	80.2%	80.5%	80.4%	93.5%	76.5%	77.7%	77.1%	93.1%	76.1%	77.4%	76.7%	93.0%

Table 4: Accuracies on CTB2 with automatically assigned tags

≤ 40 words				Unlimited			
LR	LP	$F1$	POS	LR	LP	$F1$	POS
87.9%	87.5%	87.7%	100%	86.9%	86.7%	86.8%	100%
80.2%	79.1%	79.6%	94.1%	78.6%	78.0%	78.3%	93.9%

Table 5: Accuracies on CTB5 using gold-standard and automatically assigned POS-tags

	Sections	Sentences	Words
Set A	001–270	3,484	84,873
Set B	Set A; 400–699	6,567	161,893
Set C	Set B; 700–931	9,707	236,051

Table 6: Training sets with different sizes

spectively. These results are significantly better than any model from Table 4. However, we did not include their scores in the table because they used a different training set from CTB5, which is much larger than the CTB2 training set used by all parsers in the table. In order to make a comparison, we split the data in the same way as Petrov and Klein (2007) and tested our parser using automatically assigned POS-tags. It gave LR and LP of 82.0% and 80.9% for sentences with less than 40 words and 77.8% and 77.4% for all sentences, significantly lower than Petrov and Klein (2007), which we partly attribute to the sensitivity of our parser to pos tag errors (see Table 5).

5.3 The effect of training data size

CTB2 is a relatively small corpus, and so we investigated the effect of adding more training data from CTB5. Intuitively, more training data leads to higher parsing accuracy. By using increased amount of training sentences (Table 6) from CTB5 with the same development test data (Table 2), we draw the accuracy curves with different number of training iterations (Figure 6). This experiment confirmed that the accuracy increases with the amount of training data.

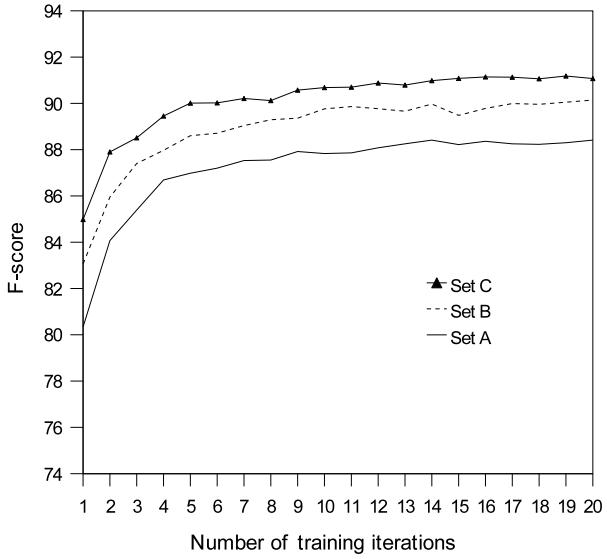

Figure 6: The influence of the size of training data

Another motivation for us to use more training data is to reduce overfitting. We invested considerable effort into feature engineering using CTB2, and found that a small variation of feature templates (e.g. changing the feature template S_0cS_1c from Table 1 to S_0tcS_1tc) can lead to a comparatively large change (up to 1%) in the accuracy. One possible reason for this variation is the small size of the CTB2 training data. When performing experiments using the larger set B from Table 6, we observed improved stability relative to small feature changes.

	Sections	Sentences	Words
Training	001–815; 1001–1136	16,118	437,859
Dev	886–931; 1148–1151	804	20,453
Test	816–885; 1137–1147	1,915	50,319

Table 7: Standard split of CTB5 data

	Non-root	Root	Complete
Zhang 2008	86.21%	76.26%	34.41%
Our parser	86.95%	79.19%	36.08%

Table 8: Comparison with state-of-the-art dependency parsing using CTB5 data

5.4 Test accuracy using CTB5

Table 5 presents the performance of the parser on CTB5. We adopt the data split from Zhang and Clark (2008), as shown in Table 7. We used the same parser configurations as Section 5.2.

As an additional evaluation we also produced dependency output from the phrase-structure trees, using the head-finding rules, so that we can also compare with dependency parsers, for which the highest scores in the literature are currently from our previous work in Zhang and Clark (2008). We compare the dependencies read off our constituent parser using CTB5 data with the dependency parser from Zhang and Clark (2008). The same measures are taken and the accuracies with gold-standard POS-tags are shown in Table 8. Our constituent parser gave higher accuracy than the dependency parser. It is interesting that, though the constituent parser uses many fewer feature templates than the dependency parser, the features do include constituent information, which is unavailable to dependency parsers.

6 Related work

Our parser is based on the shift-reduce parsing process from Sagae and Lavie (2005) and Wang et al. (2006), and therefore it can be classified as a transition-based parser (Nivre et al., 2006). An important difference between our parser and the Wang et al. (2006) parser is that our parser is based on a discriminative learning model with global features, whilst the parser from Wang et al. (2006) is based on a local classifier that optimizes each individual choice. Instead of greedy local decoding, we used beam search in the decoder.

An early work that applies beam search to constituent parsing is Ratnaparkhi (1999). The main difference between our parser and Ratnaparkhi's is that we use a global discriminative model, whereas Ratnaparkhi's parser has separate probabilities of actions chained together in a conditional model.

Both our parser and the parser from Collins and Roark (2004) use a global discriminative model and an incremental parsing process. The major difference is the use of different incremental parsing processes. To achieve better performance for Chinese parsing, our parser is based on the shift-reduce parsing process. In addition, we did not include a generative baseline model in the discriminative model, as did Collins and Roark (2004).

Our parser in this paper shares similarity with our transition-based dependency parser from Zhang and Clark (2008) in the use of a discriminative model and beam search. The main difference is that our parser in this paper is for constituent parsing. In fact, our parser is one of only a few constituent parsers which have successfully applied global discriminative models, certainly without a generative baseline as a feature, whereas global models for dependency parsing have been comparatively easier to develop.

7 Conclusion

The contributions of this paper can be summarized as follows. First, we defined a global discriminative model for Chinese constituent-based parsing, continuing recent work in this area which has focused on English (Clark and Curran, 2007; Carreras et al., 2008; Finkel et al., 2008). Second, we showed how such a model can be applied to shift-reduce parsing and combined with beam search, resulting in an accurate linear-time parser. In standard tests using CTB2 data, our parser achieved comparable Parseval F-score to the state-of-the-art systems. Moreover, we observed that more training data lead to improvements on both accuracy and stability against feature variations, and reported performance of the parser using CTB5 data. By converting constituent-based output to dependency relations using standard head-finding rules, our parser also obtained the highest scores for a CTB5 dependency evaluation in the literature.

Due to the comparatively low accuracy for Chinese POS-tagging, the parsing accuracy dropped

significantly when using automatically assigned POS-tags rather than gold-standard POS-tags. In our further work, we plan to investigate possible methods of joint POS-tagging and parsing under the discriminative model and beam-search framework.

A discriminative model allows consistent training of a wide range of different features. We showed in Zhang and Clark (2008) that it was possible to combine graph and transition-based dependency parser into the same global discriminative model. Our parser framework in this paper allows the same integration of graph-based features. However, preliminary experiments with features based on graph information did not show accuracy improvements for our parser. One possible reason is that the transition actions for the parser in this paper already include graph information, such as the label of the newly generated constituent, while for the dependency parser in Zhang and Clark (2008), transition actions do not contain graph information, and therefore the use of transition-based features helped to make larger improvements in accuracy. The integration of graph-based features for our shift-reduce constituent parser is worth further study.

The source code of our parser is publicly available at http://www.sourceforge.net/projects/zpar.[2]

Appendix

The set of restrictions which ensures a valid binary tree is shown below. The restriction on the number of consecutive unary rule applications is taken from Sagae and Lavie (2005); it prevents infinite running of the parser by repetitive use of unary reduce actions, and ensures linear time complexity in the length of the sentence.

- the shift action can only be performed when the queue of incoming words is not empty;
- when the node on top of the stack is temporary and its head word is from the right child, no shift action can be performed;
- the unary reduce actions can be performed only when the stack is not empty;
- a unary reduce with the same constituent label ($Y \rightarrow Y$) is not allowed;
- no more than three unary reduce actions can be performed consecutively;
- the binary reduce actions can only be performed when the stack contains at least two nodes, with at least one of the two nodes on top of stack (with R being the topmost and L being the second) being non-temporary;
- if L is temporary with label X^*, the resulting node must be labeled X or X^* and left-headed (i.e. to take the head word from L); similar restrictions apply when R is temporary;
- when the incoming queue is empty and the stack contains only two nodes, binary reduce can be applied only if the resulting node is non-temporary;
- when the stack contains only two nodes, temporary resulting nodes from binary reduce must be left-headed;
- when the queue is empty and the stack contains more than two nodes, with the third node from the top being temporary, binary reduce can be applied only if the resulting node is non-temporary;
- when the stack contains more than two nodes, with the third node from the top being temporary, temporary resulting nodes from binary reduce must be left-headed;
- the terminate action can be performed when the queue is empty, and the stack size is one.

[2]The make target for the parser in this paper is chinese.conparser.

References

Daniel M. Bikel and David Chiang. 2000. Two statistical parsing models applied to the Chinese Treebank. In *Proceedings of SIGHAN Workshop*, pages 1–6, Morristown, NJ, USA.

Daniel M. Bikel. 2004. *On the Parameter Space of Generative Lexicalized Statistical Parsing Models.* Ph.D. thesis, University of Pennsylvania.

Ted Briscoe and John Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

Xavier Carreras, Michael Collins, and Terry Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of CoNLL*, pages 9–16, Manchester, England, August.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL*, pages 132–139, Seattle, WA.

David Chiang and Daniel M. Bikel. 2002. Recovering latent information in treebanks. In *Proceedings of COLING*.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*, pages 111–118, Barcelona, Spain, July.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Meeting of the ACL*, pages 16–23, Madrid, Spain.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, Philadelphia, USA, July.

Xiangyu Duan, Jun Zhao, and Bo Xu. 2007. Probabilistic models for action-based Chinese dependency parsing. In *Proceedings of ECML/ECPPKDD*, Warsaw, Poland, September.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL/HLT*, pages 959–967, Columbus, Ohio, June. Association for Computational Linguistics.

Roger Levy and Christopher D. Manning. 2003. Is it harder to parse Chinese, or the Chinese Treebank? In *Proceedings of ACL*, pages 439–446, Sapporo, Japan, July.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98, Ann Arbor, Michigan, June.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of CoNLL*, pages 221–225, New York City, June.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of HLT/NAACL*, pages 404–411, Rochester, New York, April. Association for Computational Linguistics.

Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of IWPT*, pages 125–132, Vancouver, British Columbia, October.

Honglin Sun and Daniel Jurafsky. 2003. The effect of rhythm on structural disambiguation in Chinese. In *Proceedings of SIGHAN Workshop*.

Mengqiu Wang, Kenji Sagae, and Teruko Mitamura. 2006. A fast, accurate deterministic parser for Chinese. In *Proceedings of COLING/ACL*, pages 425–432, Sydney, Australia.

Deyi Xiong, Shuanglong Li, Qun Liu, Shouxun Lin, and Yueliang Qian. 2005. Parsing the Penn Chinese Treebank with semantic knowledge. In *Proceedings of IJCNLP*.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*, pages 562–571, Hawaii, USA, October.