K-Best Combination of Syntactic Parsers

Hui Zhang[1,2] **Min Zhang[1]** **Chew Lim Tan[2]** **Haizhou Li[1]**
[1]Institute for Infocomm Research   [2]National University of Singapore
zhangh1982@gmail.com   {mzhang, hli}@i2r.a-star.edu.sg   tancl@comp.nus.edu.sg

Abstract

In this paper, we propose a linear model-based general framework to combine k-best parse outputs from multiple parsers. The proposed framework leverages on the strengths of previous system combination and re-ranking techniques in parsing by integrating them into a linear model. As a result, it is able to fully utilize both the logarithm of the probability of each k-best parse tree from each individual parser and any additional useful features. For feature weight tuning, we compare the simulated-annealing algorithm and the perceptron algorithm. Our experiments are carried out on both the Chinese and English Penn Treebank syntactic parsing task by combining two state-of-the-art parsing models, a head-driven lexicalized model and a latent-annotation-based un-lexicalized model. Experimental results show that our F-Scores of 85.45 on Chinese and 92.62 on English outperform the previously best-reported systems by 1.21 and 0.52, respectively.

1   Introduction

Statistical models have achieved great success in language parsing and obtained the state-of-the-art results in a variety of languages. In general, they can be divided into two major categories, namely lexicalized models (Collins 1997, 1999; Charniak 1997, 2000) and un-lexicalized models (Klein and Manning 2003; Matsuzaki et al. 2005; Petrov et al. 2006; Petrov and Klein 2007). In lexicalized models, word information play a key role in modeling grammar rule generation, while un-lexicalized models usually utilize latent information derived from the parse structure diversity. Although the two models are different from each other in essence, both have achieved state-of-the-art results in a variety of languages and are complementary to each other (this will be empirically verified later in this paper). Therefore, it is natural to combine the two models for better parsing performance.

Besides individual parsing models, many system combination methods for parsing have been proposed (Henderson and Brill 1999; Zeman and Žabokrtský 2005; Sagae and Lavie 2006) and promising performance improvements have been reported. In addition, parsing re-ranking (Collins 2000; Riezler et al. 2002; Charniak and Johnson 2005; Huang 2008) has also been shown to be another effective technique to improve parsing performance. This technique utilizes a bunch of linguistic features to re-rank the k-best (Huang and Chiang 2005) output on the forest level or tree level. In prior work, system combination was applied on multiple parsers while re-ranking was applied on the k-best outputs of individual parsers.

In this paper, we propose a linear model-based general framework for multiple parsers combination. The proposed framework leverages on the strengths of previous system combination and re-ranking methods and is open to any type of features. In particular, it is capable of utilizing the logarithm of the parse tree probability from each individual parser while previous combination methods are unable to use this feature since the probabilities from different parsers are not comparable. In addition, we experiment on k-best combination while previous methods are only verified on 1-best combination. Finally, we apply our method in combining outputs from both the lexicalized and un-lexicalized parsers while previous methods only carry out experiments on multiple lexicalized parsers. We also compare two learning algorithms in tuning the feature weights for the linear model.

We perform extensive experiments on the Chinese and English Penn Treebank corpus. Experimental results show that our final results, an F-Score of 92.62 on English and 85.45 on Chinese, outperform the previously best-reported systems by 0.52 point and 1.21 point, respectively. This convincingly demonstrates the effectiveness of our proposed framework. Our study also shows that the simulated-annealing algorithm (Kirkpatrick et al. 1983) is more effective

than the perceptron algorithm (Collins 2002) for feature weight tuning.

The rest of this paper is organized as follows. Section 2 briefly reviews related work. Section 3 discusses our method while section 4 presents the feature weight tuning algorithm. In Section 5, we report our experimental results and then conclude in Section 6.

2 Related Work

As discussed in the previous section, system combination and re-ranking are two techniques to improve parsing performance by post-processing parsers' k-best outputs.

Regarding the system combination study, Henderson and Brill (1999) propose two parser combination schemes, one that selects an entire tree from one of the parsers, and one that builds a new tree by selecting constituents suggested by the initial trees. According to the second scheme, it breaks each parse tree into constituents, calculates the count of each constituent, then applies the majority voting to decide which constituent would appear in the final tree. Sagae and Lavie (2006) improve this second scheme by introducing a threshold for the constituent count, and search for the tree with the largest number of count from all the possible constituent combination. Zeman and Žabokrtský (2005) study four combination techniques, including voting, stacking, unbalanced combining and switching, for constituent selection on Czech dependency parsing. Promising results have been reported in all the above three prior work. Henderson and Brill (1999) combine three parsers and obtained an F1 score of 90.6, which is better than the score of 88.6 obtained by the best individual parser as reported in their paper. Sagae and Lavie (2006) combine 5 parsers to obtain a score of 92.1, while they report a score of 91.0 for the best single parser in their paper. Finally, Zeman and Žabokrtský (2005) reports great improvements over each individual parsers and show that a parser with very low accuracy can also help to improve the performance of a highly accurate parser. However, there are two major limitations in these prior works. First, only one-best output from each individual parsers are utilized. Second, none of these works uses the parse probability of each parse tree output from the individual parser.

Regarding the parser re-ranking, Collins (2000) proposes a dozen of feature types to re-rank k-best outputs of a single head-driven parser. He uses these feature types to extract around half a million different features on the training set, and then examine two loss functions, MRF and Boosting, to do feature selection. Charniak and Johnson (2005) generate a more accurate k-best output and adopt MaxEnt method to estimate the feature weights for more than one million features extracted from the training set. Huang (2008) further improves the re-ranking work of Charniak and Johnson (2005) by re-ranking on packed forest, which could potentially incorporate exponential number of k-best list. The re-ranking techniques also achieve great improvement over the original individual parser. Collins (2002) improves the F1 score from 88.2% to 89.7%, while Charniak and Johnson (2005) improve from 90.3% to 91.4%. This latter work was then further improved by Huang (2008) to 91.7%, by utilizing the benefit of forest structure. However, one of the limitations of these techniques is the huge number of features which makes the training very expensive and inefficient in space and memory usage.

3 K-best Combination of Lexicalized and Un-Lexicalized Parsers with Model Probabilities

In this section, we first introduce our proposed k-best combination framework. Then we apply this framework to the combination of two state-of-the-art lexicalized and un-lexicalized parsers with an additional feature inspired by traditional combination techniques.

3.1 K-best Combination Framework

Our proposed framework consists of the following steps:
1) Given an input sentence and N different parsers, each parser generates K-best parse trees.
2) We combine the N*K output trees and remove any duplicates to obtain M unique tress.
3) For each of the M unique trees, we re-evaluate it with all the N models which are used by the N parsers. It is worth noting that this is the key point (i.e. one of the major advantages) of our method since some parse trees are only generated from one or I (I<N) parsers. For example, if a tree is only generated from head-driven lexicalized model, then it only has the head-driven model score. Now we re-evaluate it with the latent-annotation un-lexicalized model to reflect the latent-

annotation model's confidence for this tree. This enables our method to effectively utilize the confidence measure of all the individual models without any bias. Without this re-evaluation step, the previous combination methods are unable to utilize the various model scores.

4) Besides model scores, we also compute some additional feature scores for each tree, such as the widely-used "constituent count" feature.

5) Then we adopt the linear model to balance and combine these feature scores and generate an overall score for each parse tree.

6) Finally we re-rank the M best trees and output the one with the highest score.

$$f(t) = \lambda_1 f_1(t) + \cdots + \lambda_N f_N(t) + {\lambda'}_1 {f'}_1(t) + \cdots + {\lambda'}_L {f'}_L(t)$$

The above is the linear function used in our method, where t is the tree to be evaluated, f_1 to f_N are the model confidence scores (in this paper, we use logarithm of the parse tree probability) from the N models, λ_1 to λ_N are their weights, ${f'}_1$ to ${f'}_L$ are the L additional features, ${\lambda'}_1$ to ${\lambda'}_L$ are their weights.

In this paper, we employ two individual parsing model scores and only one additional feature. Let f_1 be the head-driven model score, f_2 be the latent-annotation model score, ${f'}_1$ be the constituent count feature and ${\lambda'}_1$ is the weight of feature ${f'}_1$.

3.2 Confidences of Lexicalized and Un-lexicalized Model

The term "confidence" was used in prior parser combination studies to refer to the accuracy of each individual parser. This reflects how much we can trust the parse output of each parser. In this paper, we use the term "confidence" to refer to the logarithm of the tree probability computed by each model, which is a direct measurement of the model's confidence on the target tree being the best or correct parse output. In fact, the feature weight λ_i in our linear model functions similarly as the traditional "confidence". However, we do not directly use parser's accuracy as its value. Instead we tune it automatically on development set to optimize it against the parsing performance directly. In the following, we introduce the state-of-the-art head-driven lexicalized and latent-annotation un-lexicalized models (which are used as two individual models in this paper),

and describe how they compute the tree probability briefly.

Head-driven model is one of the most representative lexicalized models. It attaches the head word to each non-terminal and views the generation of each rule as a Markov process first from father to head child, and then to the head child's left and right siblings.

Take following rule r as example,

$$F \rightarrow L_n \dots L_2 L_1 M R_1 R_2 \dots R_m$$

F is the rule's left hand side (i.e. father label), M is the head child, L_i is M's left sibling and R_i is M's right sibling. Let h be M's head word, the probability of this rule is

$$P(r) = P(M|h, F) \cdot \prod_{i=1}^{n} P(L_i|L_{i-1}, M, h, F) \cdot \prod_{j=1}^{m} P(R_j|R_{j-1}, M, h, F)$$

The probability of a tree is just the product of the probabilities of all the rules in it. The above is the general framework of head-driven model. For a specific model, there may be some additional features and modification. For example, the model2 in Collins (1999) introduces sub-categorization and model3 introduces gap as additional features. Charniak (2000)'s model introduces pre-terminal as additional features.

The latent-annotation model (Matsuzaki et al. 2005; Petrov et al. 2006) is one of the most effective un-lexicalized models. Briefly speaking, latent-annotation model views each non-terminal in the Treebank as a non-terminal followed by a set of latent variables, and uses EM algorithms to automatically learn the latent variables' probability functions to maximize the probability of the given training data. Take the following binarized rule as example,

$$A \rightarrow B \ C$$

could be viewed as the set of rules

$$\{A_i \rightarrow B_j \ C_k | i, j, k \ are \ latent \ variables\}$$

The process of computing the probability of a normal tree is to first binarized all the rules in it, and then replace each rule to the corresponding set of rules with latent variables. Now the previous tree becomes a packed forest (Klein and Manning 2001; Petrov et al. 2007) in the latent-annotation model, and its probability is the inside probability of the root node. This model is quite different from the head-driven model in which

the probability of a tree is just the product all the rules' probability.

3.3 Constituent Counts

Besides the two model scores, we also adopt constituent count as an additional feature inspired by (Henderson and Brill 1999) and (Sagae and Lavie 2006). A constituent is a non-terminal node covering a special span. For example, "NP[2,4]" means a constituent labelled as "NP" which covers the span from the second word to the fourth word. If we have 100 trees and NP[2,4] appears in 60 of them, then its constituent count is 60. For each tree, its constituent count is the sum of all the counts of its constituent. However, as suggested in (Sagae and Lavie 2006), this feature favours precision over recall. To solve this issue, Sagae and Lavie (2006) use a threshold to balance them. For any constituent, we calculate its count if and only if it appears more than X times in the k-best trees; otherwise we set it as 0. In this paper, we normalize this feature by dividing the constituent count by the number of k-best. Note that the threshold value and the additional feature value are not independent. Once the threshold changes, the feature value has to be recalculated.

In conclusion, we have four parameters to estimate: two model score weights, one additional feature weight and a threshold for the additional feature.

4 Parameter Estimation

We adopt the minimum error rate principle to tune the feature weights by minimizing the error rate (i.e. maximizing the F1 score) on the development set. In our study, we implement and compare two algorithms, the simulated-annealing style algorithm and the average perceptron algorithm.

4.1 Simulated Annealing

Simulated-annealing algorithm has been proved to be a powerful and efficient algorithm in solving NP problem (Černý 1985). Fig 1 is the pseudo code of the simulated-annealing algorithm that we apply.

In a single iteration (line 4-11), the simulated algorithm selects some random points (the Markov link) for hill climbing. However, it accepts some bad points with a threshold probability controlled by the annealing temperature (line 7-10). The hill climbing nature gives this algorithm the ability of converging at local maximal point

and the random nature offers it the chance to jump from some local maximal points to global maximal point. We do a slight modification to save the best parameter so far across all the finished iterations and let it be the initial point for upcoming iterations (line 12-17).

RandomNeighbour(p) is the function to generate a random neighbor for the p (the four-tuple parameter to be estimated). F1(p) is the function to calculate the F1 score over the entire test set. Given a fixed parameter p, it selects the candidate tree with best score for each sentence and computes the F1 score with the PARSEVAL metrics.

Pseudo code 1. *Simulated-annealing algorithm*
Input: *k-best trees combined from two model output*
Notation:
    p: the current parameter value
    F1(p): the F1 score with the parameter value p
    TMF: the max F1 score of each iteration
    TMp: the optimal parameter value during iteration
    MaxF1: the max F1 score on dev set
    Rp: the parameter value which maximizes the F1 score
      of the dev set
    T: annealing temperature
    L: length of Markov link
Output: Rp

```
1.   MaxF1:= 0, Rp:= (0,0,0,0), T:=1, L=100 // initialize
2.   Repeat                                  // iteration
3.       TMp :=Rp
4.       for  i := 1 to L  do
5.           p := RandomNeighbour(TMp)
6.           d= F1(p)- TMF
7.           if d>0 or exp(d/T) > random[0,1) then
8.               TMF:=F1(p)
9.               TMp:=p
10.          end if
11.      end for
12.      if TMF > MaxF1 then
13.          MaxF:=TMF
14.          Rp:=TMp
15.      else
16.          TMp:=Rp
17.      end if
18.      T=T*0.9
19.  Until convergence
```

Fig 1. Simulated Annealing Algorithm

4.2 Averaged Perceptron

Another algorithm we apply is the averaged perceptron algorithm. Fig 2 is the pseudo code of this algorithm. Averaged perceptron is an online algorithm. It iterates through each instance. In each instance, it selects the candidate answer with the maximum function score. Then it updates the weight by the margin of feature value between the select answer and the oracle answer (line 5-9). After each iteration, it does average to generate a new weight (line 10). The averaged

perceptron has a solid theoretical fundamental and was proved to be effective across a variety of NLP tasks (Collins 2002).

However, it needs a slightly modification to adapt to our problem. Since the threshold and the constituent count are not independent, they are not linear separable. In this case, the perceptron algorithm cannot be guaranteed to converge. To solve this issue, we introduce an outer loop (line 2) to iterate through the value range of threshold with a fixed step length and in the inner loop we use perceptron to estimate the other three parameters. Finally we select the final parameter which has maximum F1 score across all the iteration (line 14-17).

Pseudo code 2. *Averaged perceptron algorithm*
Input: *k-best trees combined from two model output*
Notation:
    MaxF1, Rp: already defined in pseudo code 1
    T: the max number of iterations
    I: the number of instances
    Threshold: the threshold for constituent count
    w: the three feature weights other than threshold
    y': the candidate tree with max function score given a fixed weight w
    y^+: the candidate tree with the max F1 score (since the oracle tree may not appeared in our candidate set, we choose this one as the pseudo orcale tree)
    cand(i): the set of candidate tree for ith sentence
Output: Rp

1.   MaxF1:=0, T=30
2.   **for** Threshold :=0 to 1 with step 0.01 **do**
3.   Initialize w
4.   **for** iter : 1 to T **do**
5.   **for** i := 1 to I **do**
6. $y' := \mathrm{argmax}_{y\in cand(i)} w \cdot f(y)$
7. $w := w + f(y^+) - f(y')$
8. $w_i := w$
9.   **end for**
10. $w := \frac{\sum_{i=1}^{I} w_i}{I}$
11.   **if** converged **then break**
12.   **end for**
13.   p := (Threshold, w)
14.   **if** F1(p) > MaxF1 **then**
15.   MaxF1 := F1(p)
16.   Rp:=p
17.   **end if**
18. **end for**

Fig 2. Averaged Perceptron Algorithm

5 Experiments

We evaluate our method on both Chinese and English syntactic parsing task with the standard division on Chinese Penn Treebank Version 5.0 and WSJ English Treebank 3.0 (Marcus et al. 1993) as shown in Table 1.

We use Satoshi Sekine and Michael Collins' EVALB script modified by David Ellis for accuracy evaluation. We use Charniak's parser (Charniak 2000) and Berkeley's parser (Petrov and Klein 2007) as the two individual parsers, where Charniak's parser represents the best performance of the lexicalized model and the Berkeley's parser represents the best performance of the un-lexicalized model. We retrain both of them according to the division in Table. 1. The number of EM iteration process for Berkeley's parser is set to 5 on English and 6 on Chinese. Both the Charniak's parser and Berkeley's parser provide function to evaluate an input parse tree's probability and output the logarithm of the probability.

Div. Lang.	Train	Dev	Test
English	Sec.02-21	Sec. 22	Sec. 23
Chinese	Art. 001-270, 400-1151	Art. 301-325	Art. 271-300

Table 1. Data division

5.1 Effectiveness of our Combination Method

This sub-section examines the effectiveness of our proposed methods. The experiment is set up as follows: 1) for each sentence in the dev and test sets, we generate 50-best from Charniak's parser (Charniak 2000) and Berkeley's parser (Petrov and Klein 2007), respectively; 2) the two 50-best trees are merged together and duplication was removed; 3) we tune the parameters on the dev set and test on the test set. (Without specific statement, we use simulated-annealing as default weight tuning algorithm.)

The results are shown in Table 2 and Table 3. "P" means precision, "R" means recall and "F" is the F1-measure (all is in % percentage metrics); "Charniak" represents the parser of (Charniak 2000), "Berkeley" represents the parser of (Petrov and Klein 2007), "Comb." represents the combination of the two parsers.

parser accuracy		Charniak	Berkeley	Comb.
<=40 words	P	85.20	86.65	**90.44**
	R	83.70	84.18	**85.96**
	F	84.44	85.40	**88.15**
All	P	82.07	84.63	**87.76**
	R	79.66	81.69	**83.27**
	F	80.85	83.13	**85.45**

Table 2. Results on Chinese

parser accuracy		Charniak	Berkeley	Comb.
<=40 words	P	90.45	90.27	**92.36**
	R	90.14	89.76	**91.42**
	F	90.30	90.02	**91.89**
All	P	89.86	89.77	**91.89**
	R	89.53	89.26	**90.97**
	F	89.70	89.51	**91.43**

Table 3. Results on English

From Table 2 and Table 3, we can see our method outperforms the single systems in all test cases with all the three evaluation metrics. Using the entire Chinese test set, our method improves the performance by 2.3 (85.45-83.13) point in F1-Score, representing 13.8% error rate reduction. Using the entire English test set, our method improves the performance by 1.7 (91.43-89.70) point in F1-Score, representing 16.5% error rate reduction. These improvements convincingly demonstrate the effectiveness of our method.

5.2 Effectiveness of K

Fig 3 and Fig. 4 show the relationship between F1 score and the number of K-best used when doing combination on Chinese and English respectively.

From Fig 3 and Fig. 4, we could see that the F1 score first increases with the increasing of K (there are some vibration points, this may due to statistical noise) and reach the peak when K is around 30-50, then it starts to drop. It shows that k-best list did provide more information than one-best and thus can help improve the accuracy; however more k-best list may also contain more noises and these noises may hurt the final combination quality.
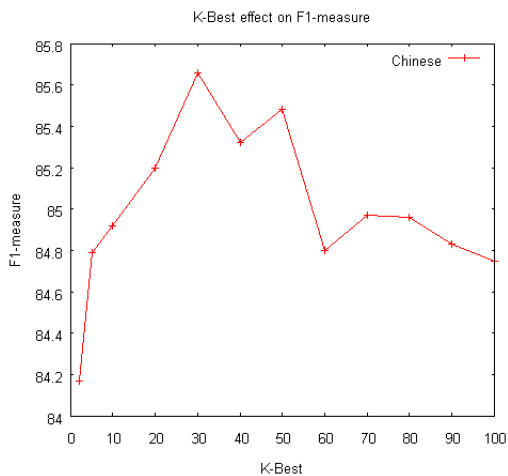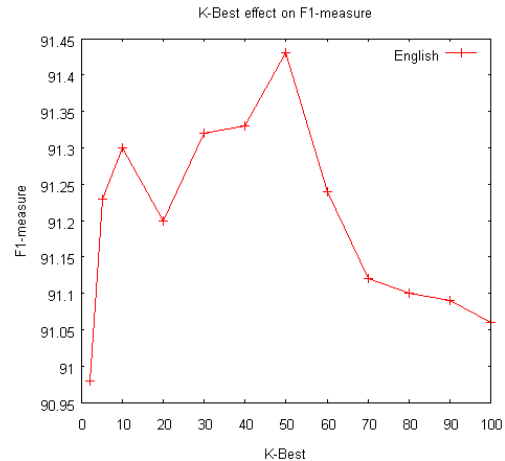

Fig 3. F1-measure vs. K on Chinese

Fig 4. F1-measure vs. K on English

5.3 Diversity on the K-best Output of the Head-driven and Latent-annotation-driven Model

In this subsection, we examine how different of the 50-best trees generated from Charnriak's parser (head-driven model) (Charnriak, 2000) and Berkeley's parser (latent-annotation model) (Petrov and Klein, 2007).

Table 4 reports the statistics on the 50-best output for Chinese and English test set. Since for some short sentences the parser cannot generate up to 50 best trees, the average number of trees is less than 50 for each sentence. Each cell reports the total number of trees generated over the entire test set followed by the average count for each sentence in bracket. "Total" means simply combine the number of trees from the two parsers while "Unique" means the number after removing the duplicated trees for each sentence. In the last row, we report the averaged redundant rate for each sentence, which is derived by dividing the figures in the row "Duplicated" by those in the row "Total".

	Chinese	English
Charniak	14577 (41.9)	120438 (49.9)
Berkeley	14524 (41.7)	114299 (47.3)
Total	29101 (83.6)	234737 (97.2)
Unique	27747 (79.7)	221633 (91.7)
Duplicated	1354 (3.9)	13104 (5.4)
Redundant rate	4.65%	5.58%

Table 4. The statistics on the 50-best output for Chinese and English test set.

The small redundant rate clearly suggests that the two parsing models are quite different and are complementary to each other.

1557

parser / Oracle		Charniak	Berkeley	Comb.
Chinese	P	88.95	90.07	**92.45**
	R	86.51	87.12	**89.67**
	F	87.71	88.57	**91.03**
English	P	97.06	95.86	**98.10**
	R	96.57	95.53	**97.68**
	F	96.82	95.70	**97.89**

Table 5. The oracle over 50-best output for individual parser and our method

The k-best oracle score is the upper bound of the quality of the k-best trees. Table 5 reports the oracle score for the 50-best of the two individual parsers and our method. Similar to Table 4, Table 5 shows again that the two models are complementary to each other and our method is able to take the strength of the two models.

5.4 Effectiveness of Model Confidence

One of the advantages of our method that we claim is that we can utilize the feature of the model confidence score (logarithm of the parse tree probability).

Table 6 shows that all the three features contribute to the final accuracy improvement. Even if we only use the "B+C" confidence scores, it also outperforms the baseline individual parser (as reported in Table 2 and Table 3) greatly. All these together clearly verify the effective of the model confidence feature and our method can effectively utilize this feature.

Feat. / Lang	I	B+C	B+C+I
Chinese	82.34	84.67	**85.45**
English	90.20	91.02	**91.43**

Table 6. F1 score on 50-best combination with different feature configuration. "I" means the constituent count, "B" means Berkeley parser confidence score and "C" means Charniak parser confidence score.

5.5 Comparison of the Weight Tuning Algorithms

In this sub-section, we compare the two weight tuning algorithms on 50-best combination tasks on both Chinese and English. Dan Bikel's randomized parsing evaluation comparator (Bikel 2004) was used to do significant test on precision and recall metrics. The results are shown in Table 7.

We can see, simulated annealing outperforms the averaged perceptron significantly in both precision ($p<0.005$) and recall ($p<0.05$) metrics of Chinese task and precision ($p<0.005$) metric of English task. Though averaged perceptron got slightly better recall score on English task, it is not significant according to the p-value ($p>0.2$).

From table 8, we could see the simulated annealing algorithm is around 2-4 times slower than averaged perceptron algorithm.

Algo. / Lang		SA.	AP.	P-value
Chinese	P	**87.76**	86.85	0.003
	R	**83.27**	82.90	0.030
English	P	**91.89**	91.72	0.004
	R	90.97	**91.02**	0.205

Table 7. Precision and Recall score on 50-best combination by the two parameter estimation algorithms with significant test; "SA." is simulated annealing, "AP." is averaged perceptron, "P-value" is the significant test p-value.

Algo. / Lang	Simulated Annealing	Averaged Perceptron
Chinese	2.3	0.6
English	12	6

Table 8. Time taken (in minutes) on 50-best combination of the two parameter estimation algorithms

5.6 Performance-Enhanced Individual Parsers on English

For Charniak's lexicalized parser, there are two techniques to improve its performance. One is re-ranking as explained in section 2. The other is the self-training (McClosky et al. 2006) which first parses and reranks the NANC corpus, and then use them as additional training data to re-train the model. In this sub-section, we apply our method to combine the Berkeley parser and the enhanced Charniak parser by using the new model confidence score output from the enhanced Charniak parser.

Table 9 and Table 10 show that the Charniak parser enhanced by re-ranking and self-training is able to help to further improve the performance of our method. This is because that the enhanced Charniak parser provides more accurate model confidence score.

parser / accuracy		reranking	Comb.	baseline
<=40 words	P	92.34	**93.41**	92.36
	R	91.61	**92.15**	91.42
	F	91.97	**92.77**	91.89
All	P	91.78	**92.92**	91.89
	R	91.03	**91.70**	90.97
	F	91.40	**92.30**	91.43

Table 9. Performance with Charniak parser enhanced by re-ranking; "baseline" is the performance of the combination of Table 3.

parser / accuracy		self-train+ reranking	Comb.	baseline
<=40 words	P	92.87	**93.69**	92.36
	R	92.12	**92.44**	91.42
	F	92.49	**93.06**	91.89
All	P	92.41	**93.25**	91.89
	R	91.64	**92.00**	90.97
	F	92.02	**92.62**	91.43

Table 10. Performance with Charniak parser enhanced by re-ranking plus self-training

5.7 Comparison with Other State-of-the-art Results

Table 11 and table 12 compare our method with the other state-of-the-art methods; we use I, B, R, S and C to denote individual model (Charniak 2000; Collins 2000; Bod 2003; Petrov and Klein 2007), bilingual-constrained model (Burkett and Klein 2008)[1], re-ranking model (Charniak and Johnson 2005, Huang 2008), self-training model (David McClosky 2006) and combination model (Sagae and Lavie 2006) respectively. The two tables clearly show that our method advance the state-of-the-art results on both Chinese and English syntax parsing.

	System	F1-Measure
I	Charniak (2000)	80.85
	Petrov and Klein (2007)	83.13
B	Burkett and Klein (2008)[1]	84.24
C	**Our method**	**85.45**

Table 11. Accuracy comparison on Chinese

	System	F1-Measure
I	Petrov and Klein (2007)	89.5
	Charniak (2000)	89.7
	Bod (2003)	90.7
R	Collins (2000)	89.7
	Charniak and Johnson (2005)	91.4
	Huang (2008)	91.7
S	David McClosky (2006)	92.1
C	Sagae and Lavie (2006)	92.1
	Our method	**92.6**

Table 12. Accuracy comparison on English.

6 Conclusions

In this paper, we propose a linear model-based general framework for multiple parser combination. Compared with previous methods, our method is able to use diverse features, including logarithm of the parse tree probability calculated by the individual systems. We verify our method by combining the two representative parsing models, lexicalized model and un-lexicalized model, on both Chinese and English. Experimental results show our method is very effective and advance the state-of-the-art results on both Chinese and English syntax parsing. In the future, we will explore more features and study the forest-based combination methods for syntactic parsing.

References

Dan Bikel. 2004. *On the Parameter Space of Generative Lexicalized Statistical Parsing Models.* Ph.D. Thesis, University of Pennsylvania 2004.

Rens Bod. 2003. *An efficient implementation of a new DOP model.* EACL-04.

David Burkett and Dan Klein. 2008. *Two Languages are Better than One (for Syntactic Parsing).* EMNLP-08.

---

[1] Burkett and Klein (2008) use the additional knowledge from Chinese-English parallel Treebank to improve Chinese parsing accuracy.

The corresponding authors of this paper are Hui Zhang (zhangh1982@gmail.com) and Min Zhang (mzhang@i2r.a-star.edu.sg)

V Černý 1985. *Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm.* Journal of Optimization Theory and Applications, 45:41-51.1985.

Eugene Charniak. 1997. *Statistical parsing with a context-free grammar and word statistics.* AAAI-97, pages 598-603.

Eugene Charniak. 2000. *A maximum-entropy-inspired parser.* NAACL-2000.

Eugene Charniak and Mark Johnson. 2005. *Coarse-to-fine-grained n-best parsing and discriminative reranking.* ACL-05, Pages 173-180.

Michael Collins. 1997. *Three generative, lexicalised models for statistical parsing.* ACL-97, pages 16-23.

Michael Collins.1999. *Head-drivenstatistical models for natural language parsing.* Doctoral Dissertation, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia 1999.

Michael Collins. 2000. *Discriminative reranking for natural language parsing.* ICML-00, pages 175-182.

Michael Collins. 2002. *Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms.* EMNLP-02.

Liang Huang. 2008. *Forest Reranking: Discriminative Parsing with Non-Local Features.* ACL-HLT-08, pages 586-594.

Liang Huang and David Chiang. 2005. *Better k-best Parsing.* IWPT-05.

S. Kirkpatrick, C. D. Gelatt, Jr. and M. P. Vecchi. 1983. *Optimization by Simulated Annealing.* Science. New Series 220 (4598): 671-680.

Dan Klein and Christopher D. Manning. 2001. *Parsing and Hypergraphs.* IWPT-01.

Dan Klein and Christopher D. Manning. 2003. *Accurate unlexicalized parsing.* ACL-03, pages 423-430.

John Henderson and Eric Brill. 1999. *Exploiting diversity in natural language processing: combining parsers.* EMNLP-99.

Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz. 1993. *Building a large annotated corpus of English: the Penn Treebank.* Computational Linguistics, 19:313-330.

Takuya Matsuzaki. Yusuke Miyao and Jun'ichi Tsujii. 2005. *Probabilistic CFG with latent annotations.* ACL-05, pages 75-82.

David McClosky, Eugene Charniak and Mark Johnson. 2006. *Effective self-training for parsing.* NAACL-06, pages 152-159.

Slav Petrov, Leon Barrett, Romain Thibaux and Dan Klein. 2006. *Learning accurate, compact, and interpretable tree annotation.* COLING-ACL-06, pages 443-440.

Slav Petrov and Dan Klein. 2007. *Improved inference for unlexicalized parsing.* HLT-NAACL-07, pages 401-411.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. III Maxwell and Mark Johnson. 2002. *Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques.* ACL-02, pages 271–278.

Kenji Sagae and Alon Lavie. 2006. *Parser combination by reparsing.* HLT-NAACL-06, pages 129-132.

Daniel Zeman and Zdeněk Žabokrtský. *Improving Parsing Accuracy by Combining Diverse Dependency Parsers.* IWPT-05.