Better Parser Combination

Xiao Chen[1*] and **Changning Huang**[2] and **Mu Li**[2] and **Chunyu Kit**[1]

[1]City University of Hong Kong
Tat Chee Avenue, Kowloon
Hong Kong SAR
x.chen@student.cityu.edu.hk
ctckit@cityu.edu.hk

[2]Microsoft Research Asia
No.49, Zhichun Road, Haidian District
Beijing, China
{v-cnh,muli}@microsoft.com

Abstract

A better parser combination approach is proposed in this paper to combine the n-best outputs of individual parsers for higher performance. Under the constituent weighting and reparsing framework, we propose a constrained constituent recombination method to address the "flat tree" problem of traditional constituent recombination. The better combination parser generated four runs ranked the top No.1 to No.4 in the CIPS ParsEval-2009.

1 Introduction

Parsing is a major task in natural language processing. Recently, it is widely adopted by the popular applications of natural language processing techniques, such as information extraction and machine translation. Parser combination is a technique to combine the output of multiple individual parsers to boost the performance. Several approaches of parser combination have been proposed by (Henderson and Brill, 1999), (Sagae and Lavie, 2006) and (Fossum and Knight, 2009). The parsing performance increased greatly in their works.

There are many ways to combine the output of multiple parsers: parse selection, constituent recombination and production recombination. (Fossum and Knight, 2009) have proved that constituent recombination is the best. The approach presented in this paper is based on the constituent recombination method. This better combination parser generated four different runs, which are finally ranked from the 1st place to the 4th place in the CIPS ParsEval-2009.

*This work is done within the internship of MSRA.

2 The Adaptation of Parsers

Three open source constituent parsers are adopted into the combination: Bikel's parser, Stanford parser and Berkeley parser.

Bikel's parser (Bikel, 2004) is a implementation of Collins' head-driven statistical model (Collins, 2003). The Stanford parser is based on the factored model described in (Klein and Manning, 2003a; Klein and Manning, 2003b). Berkeley parser is based on unlexicalized parsing model, described in (Petrov et al., 2006; Petrov and Klein, 2007).

All the three parsers are claimed to be multilingual parsers but only accept training data in UPenn Treebank format. To adapt these parsers to the Tsinghua Chinese Treebank(TCT) used in CIPS ParsEval-2009 task 5, we firstly transform the TCT training data into UPenn format. Then, some slight modifications have been made to the three parsers, so that they could fit in our task.

For Bikel's parser and Stanford parser, that are based on lexicalized model and factored model, a list of manually made head rules is required when trained on the UPenn Treebank, since there are no head position labels included in its representation framework. However, head position of all the constituents are overtly labeled in the TCT representation format. Our solution for this problem is to see head position labels as extension of constituent label. For example, let the "np" of different head position to be different kind of "np": np-1 $\rightarrow$ np1 and np-0 $\rightarrow$ np0, etc. The head rules for Treebank with such labels are rewritten in the Bikel and Stanford parsers.

Berkeley parser is an implementation of the unlexicalized parsing model, which is more generalized and almost language-independent. The heads of constituents are not useful to this parser, so we train it on TCT in the format that the head position labels are removed from the constituents.

The performance of the three parsers, evaluated on official testing set by the official evaluation program, are shown in Table 1.

Bracket(%)	Bikel	Stanford	Berkeley
Recall	81.48	83.86	86.89
Precision	82.20	82.72	86.43
F1	81.84	83.29	86.66

Table 1: The Performance of Individual Parsers

3 The Recombination of Constituents

3.1 Constituent Recombination

Our combination parser is based on the constituent recombination method described in (Sagae and Lavie, 2006) and (Fossum and Knight, 2009). The foundation of this method is that the correct constituents should be contained in the n-best outputs of individual parsers. According to our experiments, the 1-best output of Bikel's parser and 20-best output of the Berkeley and Stanford parser could recall about 98.34% of the correct constituents in the official testing set, therefore, the outputs of these individual parsers are not homogenous. As long as the combination process could select the correct constituents, the performance should be promising.

The combination process consists of two parts: the constituent weighting and reparsing.

In the constituent weighting step, we toke the n-best output of the three parsers to be the candidates for recombination. Every constituent is defined as a tetrad: {label, start, end, weight}, in which the label is just the constituent label, start and end indicate the span of the constituent, and weight represents the confidence of the constituent being covering the span. Each constituent is weighted by summing the probability over all the parses that contains it. Then, the weight of a parse tree is the sum of weights of all its constituents.

The probability of an output parse of a parser is defined as follows:

$$Pr(p_{j,k}) = Pr(Parser_k)^{\lambda} Pr(p_{j,k}|Parser_k)^{1-\lambda} \quad (1)$$

in which, $Pr(p_{j,k})$ denotes the probability of the j-th parse of Parser k, $Pr(Parser_k)$ denotes the probability of Parser k, $Pr(p_{j,k}|Parser_k)$ denotes the probability of the j-th parse given Parser k.

We adopt the Mean Reciprocal Rank (MRR) to estimate the probability of a parser k. Given the 1-best result of all the parsers on the development set, the probability of parser k is calculated by:

$$Pr(Parser_k) = \frac{1}{N} \sum_{i=0}^{N} \frac{1}{Rank_{k,i}} \quad (2)$$

in which, N denotes the number of sentences in development set, $Rank_{k,i}$ denotes the rank of F1 score of i-th sentence in development set of parser k among all the individual parsers. Therefore, $Pr(Parser_k)$ is a measurement of the goodness of a given parser k. Another way to estimate the probability of a parser is introduced by (Fossum and Knight, 2009), which we also adopt in this evaluation and named it: First Place Rate (FPR). There is no big difference between these two strategies of estimating $Pr(Parser_k)$ in terms of affecting the performance of combination parser according to our experiments.

To estimate $Pr(p_{j,k}|Parser_k)$, we need the n-best output parses and their score(probability) given by the parser, $Pr(p_{j,k}|Parser_k)$ is estimated by:

$$Pr(p_{j,k}|Parser_k) = \frac{e^{\alpha * Score_{j,k}}}{\sum_{j'=1}^{n} e^{\alpha * Score_{j',k}}} \quad (3)$$

in which, $Score_{j,k}$ denotes the parse score of j-th parse in the n-best output of parser k for a sentence. α is a parameter which will be tuned on development data.

To capture the different effect of $Pr(Parser_k)$ and $Pr(p_{j,k}|Parser_k)$ in weighting, we add another parameter λ to combine the two factors in a log-linear manner, this parameter is also tuned on development set.

In the reparsing step, the combination parser will build edges[1] among constituents and search for the parse tree of highest weight. We employ the dynamic programming algorithm of (Huang and Chiang, 2005) to generate n-best parses of combination parser.

It should be noticed that the edge building of constituent recombination is not constrained by any grammar, any parse trees covering all the words without cross branching are all valid. As result of the weighting strategy of parse trees, the combination parser will be in favor of the parse tree of more constituents, in other word, the output parse tree will be of high recall but low precision.

[1] edge: links between parent node and its children, defined in (Klein and Manning, 2001) and (Huang and Chiang, 2005)

To address this problem, a cutoff threshold t is introduced into recombination process to prune the constituents with very low weights.

3.2 Constrained Constituent Recombination

A problem of the constituent recombination method is noticed by (Fossum and Knight, 2009). Parse trees generated by that method are usually much flatter than the gold standard trees.

To avoid this weakness, a production recombination method is proposed in (Fossum and Knight, 2009). This method is to recombine the context-free productions rather than constituents in the output of individual parsers. All the parses are converted into context-free productions such as "vp $\rightarrow$ v np". They are weighted by the same strategy as constituents in constituent recombination. The weight of a parse tree is defined to be the sum of the productions on it. However, this method did not beat constituent recombination in the experiments of that paper.

We proposed this constrained constituent recombination method to address the same problem in a different manner, aiming to generate better tree without loss of bracket F1.

The constrained constituent recombination is a method between the production recombination and the constituent recombination. We are still recombining constituents, while the reparsing step is different. The edges are built among constituent candidates under the constraint of the grammar learnt from training data. Since the edges are not directly taken from parser outputs, our method is more flexible than the production recombination. On the other hand, those edges are built according to the grammar, therefore, our method is not as arbitrary as constituent recombination.

Unlike constituent recombination, there is no threshold t to prune constituents, a unified penalty score p is introduced to encourage the parser to be inclined to the parse tree with fewer constituents, namely, to achieve higher precision. The p is subtracted from the total weight of the tree while each constituent is added to it. The experiment results in the following section will demonstrate the effectiveness of this method.

4 Adding CRF Base-constituent Recognition

After analyzing the output of the three parsers, we decided to separate the constituents on a parse tree into to two subcategories: base-constituent and non-base-constituent. The base-constituents are the constituents whose children are all terminal nodes. The non-base-constituents are the constituents other than base-constituents.

To provide more good constituent candidates for recombination, we adopt the linear chain CRF model, which is widely used in chunking task, to recognize the base-constituents. The base-constituents are extracted from training data and transformed into the SJO (S: start a base-constituent; J: join a base-constituent; O: others) tagging format. A linear chain CRF model is trained with the feature templates in Table 2. In the experiments, we try to split the tags by their head position. For example, an "np-0" is represented by "np-L" since the head is on the left-most child. By parity of reasoning, there are "np-R" and "np-E" representing head on right most child and multi-head, respectively. Experiments show that this splitting of tags is better than un-split tags.

To compare our CRF recognizer with the individual parsers in recognizing base-constituent, we carried out an experiment on the official testing set. Results are shown in Table 3, the F1 is 0.78% higher than the best individual parser. Comparing the 20-best output of this CRF base-constituent recognition to the gold standard data, we found that the oracle of recall could be as high as 99.11%.

Base-constituent(%)	CRF	Bikel	Stanford	Berkeley
Recall	90.88	86.61	88.16	90.05
Precision	90.95	85.95	87.88	90.22
F1	90.92	86.28	88.02	90.14

Table 3: The Performance Comparison of Base-constituent Recognition

In sum, while recombination, we could weight the two groups of constituents separately. The weighting strategy is the same as last section, equation (1). The three parameters are defined separately: $\alpha 1$, $\lambda 1$, $t1$ for base-constituents; $\alpha 2$, $\lambda 2$, $t2$ for non-base-constituents. Since there is no threshold pruning in the constrained constituent combination method, $t1$ and $t2$ are replaced by unified penalty score p. They are all tuned on development data.

To evaluate the efficiency of adding base-constituent recognition, we split the official train-

Feature templates:
Word unigram : w(-1),w(0),w(1);
Word bigram: w(-1)w(0),w(0)w(1);
Pos tag unigram: pos(-3),pos(-2),pos(-1),pos(0),pos(1),pos(2),pos(3);
Pos tag bigram: pos(-2)pos(-1), pos(-1)pos(0), pos(0)pos(1),pos(1)pos(2);
Pos tag trigram: pos(-2)pos(-1)pos(0),pos(-1)pos(0)pos(1),pos(0)pos(1)pos(2);
Pos tag skip bigram: pos(-2)pos(0),pos(-1)pos(1),pos(0)pos(2);
Length: The length of sentence.(reduced into 4 types according to statistics on training data);

Table 2: Feature Templates for Base-constituent Recognition

ing data into three parts: training, development and testing set. The ratio of their sizes is 8:1:1. Three experiments have been done: Experiment 1 is just traditional constituent recombination (-separate, -base constituent). Experiment 2 is to weight the base-constituents and non-base-constituents separately (+separate -base constituent). Experiment 3 is adding the base-constituent recognition output into the experiment 2. All the result is shown in Table 4.

Bracket(%)	-separate -base-constituent	+separate -base-constituent	+separate +base-constituent
Recall	88.66	88.11	88.27
Precision	89.71	90.37	91.37
F1	89.18	89.23	89.79

Table 4: Comparison of Different Combination Strategy

Given this result, it is confirmed that adding the CRF base-constituent recognition could help to improve the performance of combination parser.

5 Head Finding

Head finding is a post process after parsing in our system. According to the statistics on the TCT Treebank, 92% of the constituents have only one head, while 8% of the constituents are multi-headed.

Our head finding procedure is to train a CRF classifier to classify each context-free production into several categories. For single head constituent the category is just the position of the head (0,1,...,6). For multi-head constituent, the category is "E", which means all the children except punctuations and conjunctions are all heads. The feature templates for the classification are in Table 5.

The head finding procedure is proceeded in the bottom-up fashion, so that the head words of productions in lower layers could be used as features for the productions of higher layers.

To evaluate the accuracy of pure head finding, we perform the head finding procedure on a randomly selected held-out development set, the head position labels are removed leaving only the gold parse tree. Experiment shows the accuracy could reach 98.93%, which is promising. As long as the substructures in the parse trees are correct, the final performance would not drop much after head finding.

6 Conclusion & Discussion

After the deadline, we fixed a bug in our program. The updated results, shown in Table 6 and 7, are slightly better than those posted on CIPS ParsEval-2009 website. All the evaluation results in this paper, except those in section 4, are all calculated by the official evaluation program.

Bracket(%)	A(mc)	B(mr)	C(fc)	D(fr)
Recall	87.14	88.13	87.11	88.00
Precision	90.52	89.15	90.47	89.68
F1	88.80	88.63	88.76	88.83

Table 6: The Evaluation of Bracket

Bracket(%)	A(mc)	B(mr)	C(fc)	D(fr)
Recall	79.07	82.36	79.06	82.14
Precision	82.14	83.31	82.11	83.70
F1	80.58	82.83	80.56	82.91

Table 7: The Evaluation of Bracket with Head Position Labels(complete match)

In the Tables 6, 7, and 8: **m** stands for using MRR to estimate the parser probability; **f** stands for using FPR to estimate the parser probability; **c** stands for using constituent recombination; **r** stands for using constrained constituent recombination.

To assess the performance drop caused by head

Feature templates:
The label of the current constituent;
The label of the left most child, the middle child and the right most child;
The head word of the left most child, the middle child and the right most child;
The POS tag of the head word of the left most child, the middle child and the right most child;
Bigram of label, head word and POS tag of head word of the children: L/M, M/R; [a]
Trigram of label, head word and POS tag of head word of the children: L/M/R;
The number of children;

Table 5: Feature Templates for Head Finding

[a]For the constituents have more than 3 children, the middle children are replaced by a universal multi-children node.

finding, we set up another evaluation method. All the brackets are represented by its span, constituent label and head position, therefore, a bracket is correctly recognized only when the three features are all matched. These results are more comparable to those evaluated without head position, because the number of brackets are the same. The evaluation is conducted by removing the "-" in the constituent labels and runing the official evaluation program in bracket evaluation mode. The results are in Table 8.

Bracket(%)	A(mc)	B(mr)	C(fc)	D(fr)
Recall	83.03	84.42	83.01	84.23
Precision	86.25	85.40	86.22	85.83
F1	84.61	84.91	84.59	85.02

Table 8: The Evaluation of Bracket with Head Position Labels(complete match, no hyphen)

The bracket F1 of the combination parser is 2% higher than the best individual parser, the Berkeley parser, and the recall and precision are also elevated. This is indicating that our recombination method could boost the performance. However, the gap between recall and precision of the combination result is larger than the individual parsers especially for the constituent recombination method, which denoting that the methods elevated much more precision than recall to achieve a higher F1.

More importantly, the constrained constituent recombination outperforms constituent recombination while evaluated with or without head position, which is consistent with our original intention that this method could output better parse trees.

It is also worth notice that, although, the result B underperforms other results on bracket F1 without head position matching, it outperforms the two constituent recombination results while evaluated with head position. Here raising a question, whether the bracket recall, precision and F1 are enough to evaluate the parsing performance?

Although the format of TCT is quite different from UPenn Treebank, most constituent parsers based on PCFG are all adaptable. Constituent recombination is an open framework to combine advantages of different parsing models, so that it could achieve the state-of-art performance.

References

Daniel Bikel. 2004. Intricacies of collins parsing model. *Computational Linguistics*, 30(4):480–511.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.

Victoria Fossum and Kevin Knight. 2009. Combining constituent parsers. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 253–256, Boulder, Colorado, June. Association for Computational Linguistics.

John Henderson and Eric Brill. 1999. Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 187–194.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64, Vancouver, British Columbia, October. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2001. Parsing and hypergraphs. In *The Seventh International Workshop on Parsing Technologies*, Beijing, China, October. Tsinghua University Press.

Dan Klein and Christopher Manning. 2003a. Fast exact inference with a factored model for natural language parsing. In *In Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pages 3–10. MIT Press.

Dan Klein and Christopher D. Manning. 2003b. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July. Association for Computational Linguistics.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York, April. Association for Computational Linguistics.

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

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132, New York City, USA, June. Association for Computational Linguistics.