Introducing More Features to Improve Chinese Shift-Reduce Parsing

Hongxian Wang, Qiang Zhou and Liou Chen
Center for Speech and Language Technologies,
Research Institute of Information Technology, Tsinghua University
Center for Speech and Language Technologies, Division of Technology Innovation and Development,
Tsinghua National Laboratory for Information Science and Technology
wanghongxian@gmail.com, zq-lxd@mail.tsinghua.edu.cn, chouou@gmail.com

***Abstract*—Recent years, shift-reduce parsing has gained popularity for its efficiency, but its performance still has a gap with the state of art parsers. In this paper, we construct a baseline Chinese shift-reduce parser using the common features. According to the error patterns in the baseline system, we explore the use of constituent label features and Lexical Relation Pair (LRP) information. The parser is trained and evaluated on the Tsinghua Chinese Treebank. After using the new features the boundary F-score arises from 85.19% to 86.25%. Additionally, we investigate the use of LRP extracted from raw text automatically, and the parser can also get a competitive result.**

I. INTRODUCTION

Parsing is always the research focus in the natural language processing (NLP) communities. Over the past decade, many statistical parsers have been developed and achieve high levels of accuracy. In the early years, the most popular data-driven parsers ([1], [2]) are based on generative models that are closely related to probabilistic context-free grammars (PCFG). These parsers reach a high accuracy, but with high complexity. Recently, Reference [3] presented a classifier-based dependency parser and showed that the deterministic parser can also achieve high levels of accuracy, in spite of its simplicity. Reference [4] showed that deterministic parsing is suitable for constituent parsing too.

The parser used in this paper is a transition based, also represented as shift-reduce parser. Compared with the traditional generative parser, a shift-reduce parser has a simple structure and is easy to implement. It can have a linear time complexity [3]. However, there is still a gap in performance between the shift-reduce parser and the state of art ones. The best F-score of the shift-reduce parser is about 5% lower than the chart parser. To narrow the gap, we can do work in two ways. One is to optimize the parsing strategy, such as expanding the search space using beam search [5] or best first search [4], or to integrate shift-reduce parser with other parsers [6]. The other is to apply more features to help the classifier to select the actions in parsing more accurately. Reference [7] used clausal information of a sentence to improve the performance of dependency shift-reduce parsing. In this paper we will focus on the affection of the features of lexical relation pairs and constituent label combinations in shift-reduce parsing.

A deterministic parser uses a series of actions to build a syntactic tree. It uses only the local information to determine which action should be applied. Since natural language is so complex, the local information cannot solve all the ambiguities. An assumption is that, if we can use more information in the action selection of the deterministic parser, more ambiguities can be solved.

The remaining sections of this paper are organized as follows: in section 2 we introduce the structure and feature schema of the baseline parser. We talk about the experiment in section 3 and make analysis of the experimental result and find out the frequent unrecalled patterns. In section 4, according to the error in the baseline system, we introduce two types of new features, and the performance improves. We compare our work with other's in section 5 and conclude the paper in section 6.

II. PARSER IMPLEMENTATION

The structure of this parser is similar to [8], adopting a greedy strategy. Some details of the parser will be illustrated in this section, first is action schema, and feature selection follows.

A. Parser Actions

A deterministic shift-reduce parser uses a series of actions to build a syntactic tree from the raw text. As our parser is based on the Tsinghua Chinese Treebank (TCT)[9], it is designed for constituent parsing. Shift-reduce parser is intrinsically incapable of handling multi-node constituent. To overcome this weakness, researchers usually do some preprocessing. Reference [8] transformed the multi-children node to binary node in training process and transformed the binary node back to multi-children node after parsing. Reference [10] transformed the constituent syntactic tree to dependency syntactic tree and transformed it back after parsing. Contrasted to these transform methods, we design a special action to handle multiple-node constituent parsing directly. This can make the parser structure simpler and run faster.

Reduce: this action will combine the item at the top of the stack and the item in the head of the buffer to be a new tree node. To do this, the item at the top of the stack is popped as item A, and the one in the head of the buffer is popped as item B. A new tree node is constructed, and the item A is as the left child and the item B as the right. At last, the new tree node is pushed into the head of the buffer.

Shift: This action is the simplest of all. It pops the item in the head of the buffer and pushes it to the top of the stack.

Reduce-Left: This is a variant of the action reduce. It is designed to process the multi-children node. When applying this action, the element at the top of the stack is popped out as A, and the one in the head of the buffer is popped out as B. This action makes B as the most right child of A, and push A to the head of the buffer.

Reduce-unary: This action is another variant of the action reduce. It is designed for the single-child node. As a result of this action, the element at the head is popped out and become a child of the new constructed tree. The tree, which has only one child, is pushed back to the head of the buffer.

The four actions above are illustrated in Table 1. In the table, the pipe line (vertical bar) means the separator between the stack and the buffer. The stack is on the left of the pipe line, and the right most is the top of the stack. On the right of the pipe line is the buffer and the left most is the head of the buffer. The letters a, b etc. represent words。

Table 1: Action Illustration

shift	[a b] | c d → [a b] c | d
reduce	[a b] | c d → |[[a b] c] d
reduce-left	[a b] | c d → |[a b c] d
reduce-unary	[a b] | c d → [a b] | [c] d

There are some restriction rules for the four actions to use:

(1) Only a shift or a reduce-unary action is allowed when the stack is empty.

(2) A reduce-left action is allowed only if the item at the top of the stack is a partial tree and the buffer is not empty.

(3) A reduce-unary action is not allowed if the item at the head of the buffer is a partial tree. It is to avoid the unary tree more than one hierarchy.

B. Classifiers

Since Maximum Entropy model is inherently suitable for multi-class classification, we use Maximum Entropy classifier as our default classifier. The Maximum Entropy toolkit in this paper is developed by Zhang Le (2004)[1].

C. Basic Features

In order to reveal the affection of the new features, first we use some basic features in the parser as baseline. Following other researchers([8], [10], [11]), we use the common basic features.

[1] http://homepages.inf.ed.ac.uk/lzhang10/maxent_toolkit.html

To explain the feature extraction better, we represent the stack by S, the item at the top of the stack is S(1), and the second item of the stack is S(2), and so on. Similarly, the buffer is represented as B, and the first item of the buffer is B(1), which is also called the analysis focus. The following are B(2), B(3), and so on. In the following sections, we assume the stack and the buffer are connected, like …S(2)S(1)B(1)B(2)B(3)…. We consider B(1) as the focus of analysis when rebuild the syntactic tree. B(1) is also the center of the window when we use refer to the size of a window.

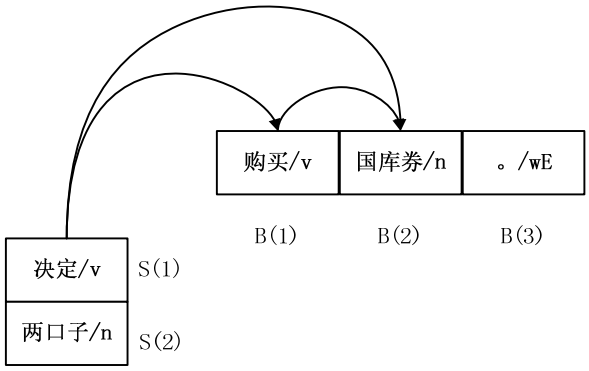

Figure 1: Illustration of feature extraction

Unigram Features: This feature is most simple feature of all. We extract all the words and POS tags as unigram feature in a window with size of 7 words. If the item is a partial tree, the left-most and right-most child's words and POS tags will be used instead.

Bigram Features: This feature is generated from two neighbor single words or POS tags combined together. They are extracted from a window with a size of 7 words. For example in the parsing state in Figure 2, we can extract features: w1w2=购买_国库券, p1p2=v_n, and so on. If the item is a partial tree, it will be ignored.

Trigram Features: Similar to bigram feature, a trigram feature is generated from three neighbor words or POS tags combined together. The size of the window for trigram feature extraction is 3 words, that is to say there is only one trigram feature can be extracted in a step.

Action Features: Inspired from the work of [4], we add action feature as basic features. However, we find that only the previous action is not efficient, so we combine two neighbor previous actions together as a bigram action feature.

III. EXPERIMENTAL RESULTS AND ANALYSIS

A. Data Preparation

The data corpus used in our experiment is the training and testing set of CIPS-ParsEval-2010 Task 2 [12]. There are 17529 sentences in the training set and 1000 sentence in the testing set. The sentences in the training set are all gold-standard segmented and POS tagged, and each consistent in the tree is assigned with a label. The test data in CIPS-ParsEval-2010 are gold-standard segmented but without POS

tag. However, we use test data with gold-standard POS tagged sentence for convenience. The average length of the sentence in training data is 27.44 words, and 25.23 in testing data.

B. Evaluation Measure

Since our parser can give the boundary and label of the constituent simultaneously, we use two measures to evaluate the performance of the parser. First is the evaluation of the boundary. As many natural language processing tasks, we use three measures: Precision, recall and F-score, which are computed as follows:

$$Precision = 100\% \times \frac{\#\text{constituents with correct boundary}}{\#\text{all constituents in result}}$$

$$recall = 100\% \times \frac{\#\text{constituents with correct boundary}}{\#\text{all constituents in test data}}$$

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

To evaluate the performance of the parser on constituent labeling, we use the same measures as we do for boundary. The only difference is the constituent must be with correct boundary and correct label. Now we have six measures to evaluate the parser.

C. Experimental Results and Analysis

Table 2 shows the overall experimental results based on Maximum Entropy Classifier with the basic features.

Table 2: Results of Maximum Entropy Parser with basic features

Boundary	Precision (%)	84.87
	Recall (%)	85.51
	F-Score (%)	85.19
Boundary and Label	Precision (%)	79.67
	Recall (%)	80.28
	F-score (%)	79.97

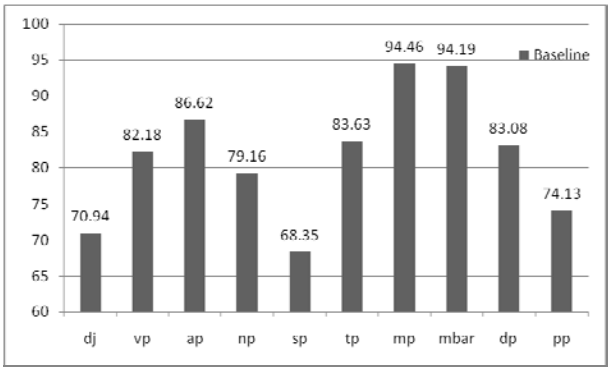

Figure 2: F-score on different constituents

Beside these overall results, we want to know some detail performance of the parser on different constituents[2]. Figure 2 shows the F-score on each constituent. We can see that, the performance differ on different constituents. It is quite high on some constituents (reaches 94.46%), while quite low on some others.

In order to improve the performance of the parser, we investigate the combination patterns of the unrecalled constituents, and get a list of the typical combination patterns of unrecalled constituents.

We select the most frequent error patterns of the constituents with low F-score, such as pp, sp, np, etc. They are listed in Table 3. It shows that, most of the unrecalled patterns are consist of a word and a constituent. However, in the basic feature schema, only word and POS features are extracted, such pattern can't be recognized. To overcome this problem, new features are needed.

Table 3: Examples of unrecalled patterns

Type	#Unrecalled	#All	UR rate
pp(overall)	130	817	0.16
p-np	43	284	0.15
p-sp	13	103	0.13
p-tp	11	83	0.13
sp(overall)	55	275	0.20
n-f	16	60	0.27
np-f	22	103	0.21
np(overall)	903	5379	0.17
dj-uJDE-n	20	52	0.38
np-uJDE-n	26	74	0.35
np-np	64	196	0.33
n-np	37	131	0.28

IV. NEW FEATURES

Since the basic features can't distinguish many ambiguities, such as the ambiguity listed in Table 3 We need to introduce some new features. There are two principles for the new features selection: (1) the new features should be pertinent to the error patterns, (2) the amount of the new features should be controlled, so as not to reduce the speed of the parser.

A. Constituent Combination Features

Complete Combination Feature: Since the basic features can't distinguish the word and constitute combination ambiguity, an intuitive idea is to add new features to represent the structure of the combination of words and constitutes. We introduce two types of new features to improve the performance of the parser:

- Combination of constituent label and neighbor POS tags
- Combination of constituent label and neighbor words

Table 4 list the overall result of the parser with the combination features. A notable thing is that, after adding the

[2] The consitutent tags used in the paper: dj-simple sentence, vp-verb phrase, ap-adjective phrase, np-noun phrase, sp-space phrse, tp-time phrase, mp-numeral phrase, mbar—numeral word, dp-adverb phrase, pp-preposition phrase.

new features, the model is so big that ME classifier can't load correctly, therefore we use a linear SVM classifier of liblinear[3] as a replacement. Our experiment shows that the parsers based on these two classifiers have similar performances.

Though new features are added, the performance decreases. Figure 3 shows the detail comparison with the baseline. We can see the np, vp, dj decrease much, the possible reason is that, these constitutes don't have regular combination patterns, the new features bring in little information but much more noise. Furthermore, adding all the combination features will also harm the speed of the parser. To avoid the noise we should add feature according to the unrecalled patterns listed in Table 3.

Table 4: Results of the parser with complete combination feature and a SVM classifier

		Baseline	Com-fea
Boundary	Precision (%)	84.07	65.54
	Recall (%)	85.15	81.51
	F-Score (%)	84.61	72.65
Boundary and Label	Precision (%)	78.92	60.87
	Recall (%)	79.94	75.70
	F-score (%)	79.43	67.48

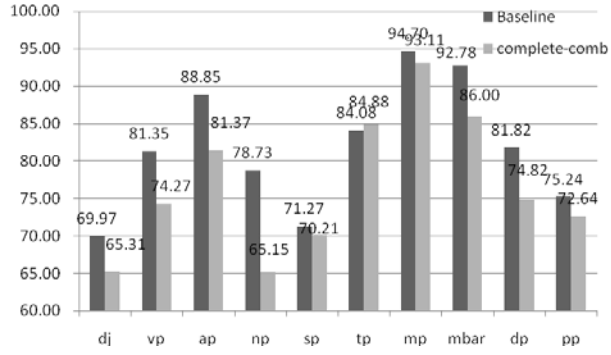

Figure 3: Detail result of the parser with complete combination feature and a SVM classifier

Selected Combination Feature: Experimental result shows adding combination features roughly will bring in noise and harm the speed of the parser. So we try to add the combination features only when they appear in the unrecalled patterns. This processing step will filter most of the noise features, and only a small amount of features are remained. The speed of the parser is not be harmed either.

Table 5 and Figure 4 show the performance difference of the parser after using the new features. Table 5 shows the overall difference, and Figure 4 shows the difference on each constituent. We can see the performance improves almost on every constituent, especially on pp, sp, np, etc. The reason for this is that these constituents usually have a regular combination pattern, and the new features can give more information to help the parser to select a suitable action.

Discussions: From the two experiments we can see that feature should be pertinent to the error patterns. A rough overall feature schema will bring in much noise, and will increase the scale of feature set. A possible way to improve the parser is to find more features according to the unrecalled patterns.

Table 5: Results of the parser with constituent combination features

		Baseline	Com-fea
Boundary	Precision (%)	84.87	85.85
	Recall (%)	85.51	86.65
	F-Score (%)	85.19	86.25
Boundary and Label	Precision (%)	79.67	80.92
	Recall (%)	80.28	81.68
	F-score (%)	79.97	81.29

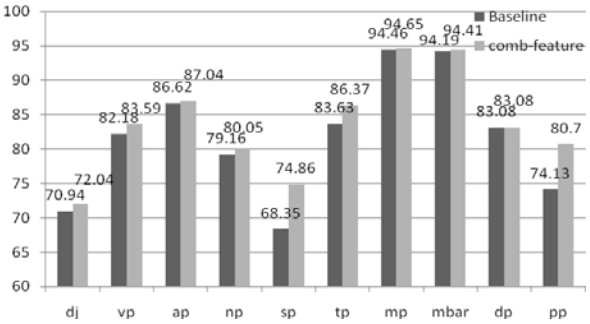

Figure 4: F1 score of two systems on different constituents

B. Lexical Relation Pair Features

Though the combination features of constituent label and neighbor words or POS tag can bring more information to help the parser to distinguish different conditions and improve the performance, we notice that there are some conditions that a word can be both combined with the left word or the right word in semantic view. For example, in the sentence : 两口子 /the couple 决定/decide 买/to buy 国库券/state treasury bill. The word '买/to buy' can be combined with its left adjacent word '决定/decide' and right adjacent word '国库券/state treasury bill' under this POS structure "v-v-n"(verb-verb-noun). But to understand the sentence correctly, only one combination is suitable. To recognize the correct combination, the parser needs to understand the sentence in meaning.

To overcome this problem, we introduce a new feature called Lexical Relation Pair (LRP). LRP is a triple like (w_i, w_j, r_{ij}), in which w_i and w_j is two words, and r_{ij} is a string representing the relation between the two words. Some researchers use Collocation to handle the ambiguity problem in natural language processing tasks ([13],[14]). "A collocation is an expression consisting of two or more words that correspond to some conventional way of saying things."

[3] http://www.csie.ntu.edu/~cjlin/liblinear/

[15]. Lexical Relation Pair is similar to the term: "Collocation", the difference is that the two words in Lexical Relation Pair usually have a semantic or syntactic relation and this relation is specified explicitly, while the words in a collocation only need to have a high co-occurrence rate. In the above example: 决定/v 购买/v 国库券/n(decide to buy state treasury bill), "购买/to buy 国库券/state treasury bill" will have a PO (Predicate Object) relation, this can help the parser combine the two words together.

We still follow the two principles at the beginning of this section when applying this feature. To meet the requirements, this feature is also pertinent to the unrecalled patterns. This feature is extracted only when the specific pattern appears. We using a trigram of the LRP to present the feature, $r_{S(1)B(1)} + r_{B(1)B(2)} + r_{S(1)B(2)}$, in which r_{ij} indicates the relation between words in position i and j, S(1) represents the first item in the stack, B(1) represents the first item in the buffer. Furthermore, we combine the trigrams of LRP with the current patterns.

Manually Extracted LRPs: The LRPs used in this paper are extracted from four manually annotated data sources, including Tsinghua Chinese Treebank, Chinese collocation dictionary, People's Daily annotated corpus and PKU grammatical dictionary. The final manual knowledge base contains 961907 LRP entries.

Table 6: Overall results after adding LRP features

		Com-fea	LRP
Boundary	Precision (%)	85.85	86.19
	Recall (%)	86.65	87.15
	F-Score (%)	86.25	86.67
Boundary and Label	Precision (%)	80.92	81.53
	Recall (%)	81.68	82.44
	F-score (%)	81.29	81.98

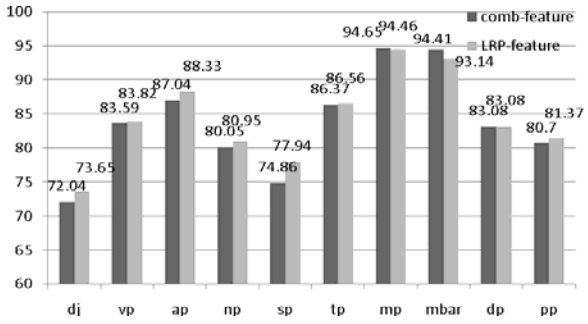

Figure 5: Detail results comparison before and after adding LRP features

The overall results after adding LRP features are listed in Table 6, and the detail results are showed in Figure 5. The results are compared with the ones before adding LRP features. Along with the overall improvement, the parser improves on almost all consistent, especially on the constituents of dj, np and sp. The reason for this is that, these phrases usually have a complicated structure, and the sub-constituents in these phrases often have combination ambiguities, the LRP features can help the parser to select the correct actions.

Automatically Extracted LRPs: The experimental results show that LRP features are useful in improving the performance of the parser. The LRP knowledge base used in last subsection are extracted from manually annotated corpus, however using manual LRP knowledge base has some shortages: 1) Manual Annotation are very expensive and the corpus is usually small; 2) NLP task is often domain related, so it is difficult to apply the manual LRP to the new domain.

A reasonable idea is to use automatically extracted LRP knowledge base. We use a LRP corpus [16] extracted automatically from the corpus of 1998 and 2000 People's Daily[4]. The extraction method is based on a Basic Chunk Parser[17]. The automatically extracted data has 561,119 LRP entries.

We use the automatically extracted LRPs as a replacement of the manual LRP Knowledge base. After applying it to the parser, we find the result almost the same as the ones when using the manual LRP knowledge base. Table 7 and Figure 6 display the comparison of results when using manual and automatic LRP knowledge base. Figure 6 shows the detail comparison on each constituent. The performances of the parser are similar almost on each constituent when using two different knowledge bases. We also note that, in the overall results showed in Table 7, though the precision is a little higher when using manual LRPs, the recall is a little higher when using automatic LRPs.

Table 7: Comparison of the manual LRP and Automatic LRP

		Man-LRP	Auto-LRP
Boundary	Precision (%)	86.19	86.07
	Recall (%)	87.15	87.18
	F-Score (%)	86.67	86.62
Boundary and Label	Precision (%)	81.53	81.47
	Recall (%)	82.44	82.53
	F-score (%)	81.98	82.00

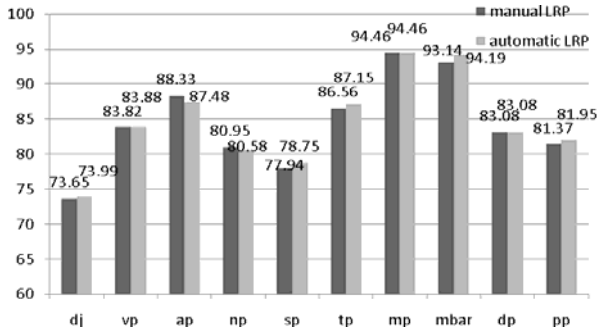

Figure 6: Comparison of the manual LRPs and Automatic LRPs in detail

[4] http://icl.pku.edu.cn/icl_res/default_en.asp

Discussion: The real factor which affects the performance of the parser is the coverage of the LRP knowledge base rather than the scale. We calculate the coverage of the manual and automatic LRP knowledgebase. The coverage of manual LRPs is 41.38%, while the one of automatic LRPs is 53.41%. That is why the automatic LRP knowledge base with a smaller scale has comparable result with the manual LRPs. The manually extracted LRPs are more credible, so the parser has a higher precision even with a lower coverage. In spite of the automatic LRPs' lower confidence, its wider coverage helps the recall much more.

The experimental result here enables us to apply the parser to different domains even without a homologous Treebank. We need only to apply a chunk parser on the raw domain texts and extract enough LRPs to improve the parser's performance on the new domain.

V. Related Work

Due to its lower time complexity and simplicity in implementation, shift-reduce parser attracts much attention. However, its performance still has a gap with the state of art parsers. To narrow the gap, researchers have made great efforts.

Shift-reduce parser adopts a greedy strategy, so it is difficult to correct a previous error. To overcome this problem, reference [4]used best-first search to enlarge the search space, and get a better result. Similar to them, reference [5] used beam search to improve the performance of the parser. Reference [7] maintained multiple transition paths after each transition step to alleviate effect of the previous error. All these work improve the performance more or less. But they all tried to enlarge the search space of the parser, which inevitably harms the speed of the parser. Contrast to their work, we propose a different way: introducing new features to get more information from the state of the parser. Benefit by the pertinence of the new features, we add a small amount of features, and the performance improves without speed reduction. Table 8 lists the speed of the parser in each feature schema. The latter features are added to previous ones directly.

Table 8: Speed of the parser in different features schemas

Feature Schema	Speed
Baseline	50 sentences/s
+Combination Feature	50 sentences/s
+LRP Features	50 sentences/s

Performance Comparison: CIPS-ParsEval-2010 used automatic POS annotation, but we use manual POS annotation, so the results cannot be compared directly. Noticing that CIPS-ParsEval-2009 [18] used manual POS annotation, we select two best single model systems to compare with. Reference [20] used a traditional PCFG method, and reference [19]used multipath transition state and online training at the same time and achieved a high score on both boundary F-score and boundary&label F-score. Table 9 shows the comparison. Though we do not do any optimization on the CIP-ParsEval-2009 corpus, the performance of the parser improves by applying new features, and gets a comparable result with the traditional PCFG method. Although it still has a gap with the method using enlarging search space method, but we achieve this without any speed reduction.

Table 9: Comparison with other's work on the ParEval 2009 corpus

	Boundary	Boundary&Label
(Jiang 2009)	90.09	87.15
(Song 2009)	87.10	83.51
Our baseline	86.08	81.13
Our best	87.02	82.73

Speed comparison: Efficiency is a high light of shift reduce parser. We select two high efficiency systems to compare with in the speed in similar hardware conditions. One is a CCG parser presented in the summer workshop 2009 in John Hopkins University[21], another is also a shift reduce parser presented by[11]. These are two fastest parsers we can find. Table 10 shows the comparison. We can see our speed exceeds the shift reduce parser by, but lags behind the CCG parser. Since their goal is to implement a fast parser, they did much optimization on speed, however we focus on improving the performance of the parser without speed reduction.

Table 10: Comparison with other's work in speed

	Corpus	Speed (sent/s)
JHU 2009	Penn Treebank	60-100
(Wang, 2006)	Penn CTB	29
Our System	ParsEval 2010	50

VI. Conclusions

Shift-reduce Parser has a very high efficiency but its performance lag behind the state of art parser. In this paper, we improve the performance of the parser without speed reduction by introducing new features.

This paper has two main contributions:

1) We analyze the result of the baseline parser, and report the frequent unrecalled constituent pattern. Depending on the analysis, we introduce constituent label combination feature and LRP feature and the F-score of the boundary improves from 85.19% to 86.25% without any speed reduction.

2) We explore the use of automatically extracted LRP in shift-reduce parsing, and prove that it can achieve a comparable result with the one when using manually extracted LRP. This encourages us to apply the parser to different domains.

ACKNOWLEDGMENT

This work was supported by National Natural Science Foundation of China (Granted No.: 60873173), Tsinghua-Intel Joint Research Project.

REFERENCES

[1] E. Charniak,. "A maximum-entropy-inspired parser". In *Proc.N. American ACL (NAACL)*, pp.132–139. 2000.

[2] M. Collins. "Three generative, lexicalised models for statistical parsing". In *Proc. of ACL*, pp. 16–23. 1997.

[3] J. Nivre and M. Scholz, "Deterministic dependency parsing of English text". In *Proc. of COLING*, 2004.

[4] K. Sagae and A. Lavie, "A best-first probabilistic shift-reduce parser". In *Proc. of the COLING/ACL*, pp. 691-698. 2006.

[5] Y. Zhang and S. Clark, "Transition-based parsing of the Chinese treebank using a global discriminative model". In *Proc. of the 11th International Conference on Parsing Technologies*, pp. 162-171. 2009.

[6] J. Nivre and R. McDonald. "Integrating graph-based and transition-based dependency parsers". In *Proc. of ACL-HLT*, pp. 950-958, 2008.

[7] P. Gadde, K. Jindal, S. Husain, D. M. Sharma and R. Sangal, "Improving Data Driven Dependency Parsing using Clausal Information". In *Proc. of NAACL-HLT*, 2010.

[8] K. Sagae and A. Lavie, "A classifier-based parser with linear run-time complexity". In *Proc. of the Ninth International Workshop on Parsing Technology*, pp. 125-132. 2005.

[9] Q. Zhou, "Chinese Treebank Annotation Scheme". Journal of Chinese Information. 18(4), pp. 1-8. 2004. (in Chinese)

[10] X. Ma, X. Zhang, H. Zhao and B. L. Lu, "Dependency Parser for Chinese Constituent Parsing" In *Proc. of CIPS-SIGHAN Joint Conference on Chinese Language Processing*. 2010.

[11] M. Wang, K. Sagae and T. Mitamura, "A fast, accurate deterministic parser for Chinese". In *Proc. of COLING-ACL*, pp. 425-432. 2006.

[12] Q. Zhou and J. Zhu, "Chinese Parsing Evaluation". In *Proc. of CIPS-SIGHAN Joint Conference on Chinese Language Processing*, 2010.

[13] J. Chen and M. Palmer, "Towards Robust High Performance Word Sense Disambiguation of English Verbs Using Rich Linguistic Features". In *Proc. of IJCNLP*, pp. 933-944. 2005

[14] D. Xiong, Q. Liu and S. Lin, "Maximum entropy based phrase reordering model for statistical machine translation".In *Proc. of ACL-44*, pp. 521-528. 2006.

[15] C. D. Manning, H. Schütze and MITCogNet, Foundations of statistical natural language processing, vol.59. MIT Press, 1999.

[16] H. Qiu, " Automatic Extraction of Chinese Lexical Relation Pair and Basic Concept Unit," In Proc. of NCMMSC (in press), 2011

[17] H. Yu, "Research on Automatic Chinese Chunk Parsing," Bachelor's thesis dissertation, Department of Computer Science and Technology, Tsinghua University, Beijing, 2007.

[18] Q. Zhou and Y. Li, "First Workshop on Chinese Syntactic Parsing Evalation Report," In *Proc. of the First Workshop on Chinese Syntactic Parsing Evaluation*. 2009.

[19] W. Jiang, H. Xiong and Q. Liu, "Muti-Path Shift-Reduce Parsing with Online Training," in *Proc. of 1st Workshop on Chinese Syntactic Parsing Evaluation*. 2009.

[20] Y. Song and C. Kit, "PCFG Parsing with CRF tagging for head recognition". In *Proc. of 1st Workshop on Chinese Syntactic Parsing Evaluation*. 2009.

[21] S. Clark, A. Copestake, J. R. Currany, Y. Zhangz, A. Herbelot and J. Haggertyy, et al., "Large-Scale Syntactic Processing: Parsing the Web," *JHU Summer Research Workshop*, 2009.