Robust and Efficient Chinese Word Dependency Analysis with Linear Kernel Support Vector Machines

Yu-Chieh Wu
Dept. of Computer Science and Information Engineering
National Central University
Taoyuan, Taiwan
bcbb@db.csie.ncu.edu.tw

Jie-Chi Yang
Graduate Institute of Network Learning Technology
National Central University
Taoyuan, Taiwan
yang@cl.ncu.edu.tw

Yue-Shi Lee
Dept. of Computer Science and Information Engineering
Ming Chuan University
Taoyuan, Taiwan
{leeys}@mcu.edu.tw

Abstract

Data-driven learning based on shift reduce parsing algorithms has emerged dependency parsing and shown excellent performance to many Treebanks. In this paper, we investigate the extension of those methods while considerably improved the runtime and training time efficiency via L_2-SVMs. We also present several properties and constraints to enhance the parser completeness in runtime. We further integrate root-level and bottom-level syntactic information by using sequential taggers. The experimental results show the positive effect of the root-level and bottom-level features that improve our parser from 81.17% to 81.41% and 81.16% to 81.57% labeled attachment scores with modified Yamada's and Nivre's method, respectively on the Chinese Treebank. In comparison to well-known parsers, such as MaltParser (80.74%) and MSTParser (78.08%), our methods produce not only better accuracy, but also drastically reduced testing time in 0.07 and 0.11, respectively.

1 Introduction

With the late development of Chinese Treebank (Xue et al. 2005), parsing Chinese is still an ongoing research issue. The goal of dependency parsing is to find the head-modifier (labeled) relations in texts. Though some of the parsing algorithms are language independent and show state-of-the-art performance on multilingual dependency Treebanks (Nivre et al., 2007; Buchholz and Marsi, 2006), they are often too slow for online purpose. Therefore, to develop an efficient and effective dependency parser is indispensable.

Over the past few years, several research studies had addressed the use of shift-reduce and edge-factored-based approaches attend fairly accurate performance in Chinese (Cheng et al., 2005; Hall, 2005; Wang et al., 2006). The former (shift-reduce) is a linear time algorithm, while the latter involves n^3 for decoding where n is the length of sentence. Even the shift-reduce approaches seems to be very efficient, most studies (Hall et al., 2007; Nivre et al., 2006) yet employ nonlinear kernel methods such as polynomial kernel support vector machines (SVMs). Furthermore, there is no research work directly compare with the two methods with Chinese Treebank. Nevertheless, the empirical training and testing time comparisons of those methods has not been reported yet.

In this paper, we present an efficient and robust parser for Chinese based on linear classifiers and shift-reduce parsing algorithms. We propose several useful properties to enhance the completeness of the two well-known shift-reduce algorithms, namely Nivre's shift reduce (NSR) (Nivre, 2003) and Yamada's shift reduce (YSR) (Yamada and Matsumoto, 2003) algorithms. To enhance the performance, we add root and bottom (neighbor) information by adopting sequential taggers. We also perform experiments on the Chinese Treebank and compare with two of the state-of-the-art parsers.

2 Parsing Algorithms

At the beginning, we briefly review the selected two parsing algorithm as follows. The NSR makes use of four parse actions to incrementally construct the dependency graph. By following the same notations as (Nivre, 2003), NSR initializes with $(S, I, A) = (\phi, W, \phi)$ where S is the stack (represented as a list), I is the queue initiated with all words, and A is the set of directed and labeled edges for the dependency graph. The stack is a list of words whose parent or child has not been found entirely. NSR incrementally parses a pair of words (one is the top of the stack and the other is the first word of the queue) and uses four parse actions to construct the dependency graph. The four parse actions are: {*Left-Arc* (*LA*), *Right-Arc* (*RA*), *Shift*, *Reduce*}. Both *LA* and *RA* could be parameterized with a dependence rela-

Coling 2008: Companion volume – Posters and Demonstrations, pages 135–138
Manchester, August 2008

tion type. By parsing a pair of words step-by-step, the parser terminates when the queue is empty.

Similar to NSR, YSR constructs the dependency graph by incrementally parse a pair of no_head words. The original YSR algorithm (Yamada and Matsumoto, 2003) makes use of three parse actions to parse a sentence left-to-right and involves n^2 parser transitions. Recently, Chang et al. (2006) showed that by adding an extra parse action *Wait-Left* and performing the "step-back" operation can accomplish parse in linear time. The step-back means that after an action determined, the parse pair moves back with except for *Shift* action. *Wait-Left* is mainly proposed to wait the next word until all its right children having attached to heads. In this paper, we employ such modification to form our basic YSR algorithm.

2.1 Useful Properties

We give more formal definitions of the dependency graph as follows.

Let $R = \{r_1, r_2, r_3, \ldots, r_N\}$ be the finite set of dependency arc labels with N types. A dependency graph $G = (W, A)$ where W is the string of words $W = w_1, w_2, w_3$, etc. and A is the set of directed labeled arcs (w_x, r, w_y) where $r \in R$, and $w_x, w_y \in W$. For a parse pair w_x and w_y in a sentence, we introduce the following notation:

1. $w_x \rightarrow w_y$: w_x is the head of w_y, and $w_x \leftarrow w_y$: w_y is the head of w_x.
2. $w_x < w_y$: word w_x is on the left hand side of word w_y in the sentence.
3. (w_x, r, w_y): denotes the word w_y is the head of w_x with relation r.

Definition 1: Valid dependency graph

A dependency graph G is well formed and valid iff the following conditions are true.

1. G is connected
2. G is acyclic (cycle free)
3. G is projective
4. For each node in G, there is only one parent except the root word
5. G is a single rooted graph

Definition 2: Parse pair

When the parsing algorithm considers a pair of word (w_x, w_y), we name the pair "parse pair".

Definition 1 gives the formal definition of a valid dependency graph. Condition 3 and condition 5 are not always true for all languages. For example, there are multiple roots in Arabic dependency Treebank, while the dependency graph is usually non-projective according to the linguistic characteristics. Fortunately the dependency graphs in Chinese are fully projective and single rooted and thus compatible with Definition 1.

However, we can not always assume the classifier is perfect. During run-time, the classifier might make incorrect decision which leads to incorrect parse graph and even constructs an incomplete and invalid parse graph. For example, for NSR it is usually retain more than two words that are not attached to their heads in the stack. To solve it, we propose the following properties to enhance the completeness of the original NSR/YSR parsers.

Definition 3: One word sentence

If there is only one unparsed word, then it must be the root.

Proposition 4: Constrained parsing I

For a parse pair (w_x, w_y), if the head of w_x is not found previously, then the parse action *Reduce* is invalid.

Proof. The parse action *Reduce* will remove w_x from the stack and leads to an unconnected and multiple roots dependency graph (violates definition 1).

Proposition 5: Unique pair parsing

If there are only two unparsed words in G, then the parse action of this pair of words is limited to be $\{LA, RA\}$.

Proof. Clearly if the parse action is *Reduce*, then it violates Proposition 4 (unconnected graph). Similarly when applying *Shift*, the state does not change, i.e., there are still two unparsed words. Nonetheless, by applying either *LA* or *RA*, the two isolated words will be linked and gives a connected graph.

Proposition 6: Constrained parsing II

For a parse pair (w_x, w_y), if the head of w_x is found, then the parse action *RA* is invalid.

Proof. Assume the head of w_x is w_m. If the parser predicts *RA*, then it regards w_y as the head of w_x. Therefore it violates Definition 1 (for each word there should be at most one head in the sentence). On the other hand, actions *Reduce* and *Shift* do not change the structure of G and is intuitively valid parsing actions. In the case of *LA*, by adding the edge from w_x to w_y, the dependency graph does not violate definition 1. Thus, *LA* is also a valid action.

Definition 3 is very common and intuitively seen in the case of one word parsing at the final stage. The Proposition 4 limits the parser actions that bring about a single-rooted dependency graph. Proposition 5 is particularly useful when there are only two unparsed words in the stack for the NSR. On the basis of the original NSR algorithm, the parse work is done when the input queue is empty. However, words will be shift and put onto the stack if their heads are not found currently. Finally if the queue is empty and these words are still retained in the stack, then it will produce multiple roots and lead to an unconnected graph. Proposition 6 is proposed to avoid the case of multiple heads in the sentence when there are two no-head words. To handle more than two words on the stack, the "step-back" operation is used.

Some of the above properties can also be applied to YSR with slightly modifications. We skip the details here owing to the space constraints.

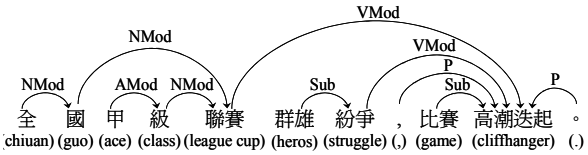

Figure 1: An example of Chinese dependency graph

3 Root and Neighbor Information

In general, the shift-reduce parsing algorithm incrementally parse a pair of words until the final parse graph has built. However, it is usually the case that when an error decision made at earlier stage, the real heads of the following words will be misattached. In particular the head is nearby the current pair of words. Similarly if the root word is misclassified as a child of other word, then the all nodes immediately modified by the root will attached to the wrong root.

One solution to improve this problem is to enhance the root and bottom (neighbor) information during parse. To obtain such information, we employ the sequential taggers to predict. That is one sequential tagger learns to determine whether the current word is the child of its left/right word or none of them while the other is to recognize the root word. For example, if the word is labeled as "left-Mod", then it means its left word is the head of it and the relation tag is "Mod".

Finding root in Chinese is even simpler, since there is only one root word in the same sentence in Chinese Treebank. Here, we adopt the same technology to label the root by using sequential taggers. Such solution had also been applied to English Treebank where a polynomial kernel SVM was used (Isozaki et al., 2004). However, there are two differences to our method. First, we enable our root tagger to incorporate bottom-level features. More precisely, the two taggers are cascaded combined. Second, to enhance the top-level syntactic information, our root tagger does not only recognize the root word, but also the words which belong to the immediate child of it. We give the following property to prove that attaching all root children to the root still leads to a valid dependency graph.

Proposition 7: Cycle-free for root tagger

The dependency graph is a cycle-free graph by linking root child to the root words.

Proof. The minimum cycle length in a valid dependency graph is two (two edges for two words by linked each other). Assume there are *K* children for a root. By attaching all children to the root, it leads to the out-degree of each child is 1, while the in-degree of the root is *K*. According to the Definition 1, the root word does not have any parent (out-degree of the root is exactly zero) and will never attach to any word in the sentence (include its children).

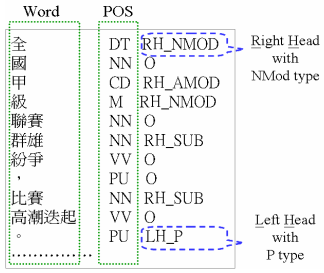

Figure 2: Attaching neighbor relations with sequental taggers

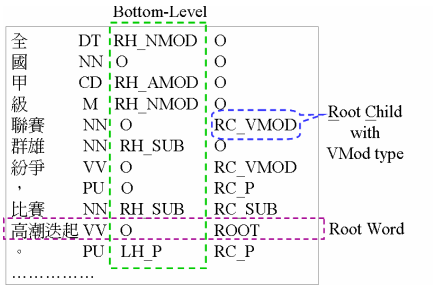

Figure 3: Attaching root words with sequential taggers

The sequential tagger used in this paper is CRF++ (Kudo et al., 2004). One advantage of conditional random fields (CRF) is that it is a structural learning method and can search optimal tag sequence with efficient Viterbi search algorithm.

Features used for the two taggers include word, part-of-speech tag, and prefix/suffix Chinese characters with context window = 3. Features that occurs less than twice in the training data is removed. Figure 1 shows an example of Chinese dependency graph. Figure 2 illustrates the sample of attaching neighbors with CRF++ by using the same sentence as in Figure 1. Figure 3 shows the example of identifying root and its children with CRF++.

Table 1: Feature set used for NSR and YSR

Feature type	Stack position	Queue position
Word	-1,0	0,+1,+2,+3
POS	-1,0	0,+1,+2,+3
BiWord	(-2,-1),(-1,0),(-2,0),(-1,+1)	(0,1),(1,2),(2,3),(0,2),(1,3)
BiPOS	(-2,-1),(-1,0),(-2,0),(-1,+1)	(0,1),(1,2),(2,3),(0,2),(1,3)
Neighbor (NSR)	-2,-1,0	0,+1,+2
Root (NSR)	-1,0	
Neighbor (YSR)	0	0
Root (YSR)	0	
History	-2,-1	
Child (Word)	0	0
Child (POS)	0	0

4 Experiments

We randomly select 90% of the Chinese Treebank 5.1 corpus for training and the remaining 10% is used for testing. Totally there are 0.45 million words in the training data and 50144 words for testing. By following (Hall et al., 2006), we use the same headword table to convert the CTB into dependency structure. The gold-tagged POS tags are

used in the experiments. All experimental results are evaluated by LAS (label attachment score), UAS (unlabeled attachment score), and root accuracy.

4.1 Settings

In this paper, we employ the MSTParser (McDonald et al., 2006) and MaltParser (Nivre, 2003) for comparison. We adopt the best settings for MaltParser with SVM and MBL learners as reported by (Hall et al., 2006)[2]. For MSTParser, the Eisner's decoding algorithm is used.

The learner we used in this paper is L_2-SVM with linear kernel (Keerthi and DeCoste, 2005). The one-versus-all (OVA) strategy is applied to handle multiclass problems. Features that appear less than twice are removed from the feature set. Table 4 lists the feature set for the NSR and YSR.

4.2 Results

Table 2 summarizes overall experimental results. The final two rows list the entire training and testing time of the corresponding methods. From this table, we can see that our method (both NSR and YSR) achieve the best and second best parsing accuracy in terms of LAS, UAS, and root accuracy. For testing time efficiency, both our NSR and YSR also outperform the other methods. Meanwhile there is no significant difference between NSR and YSR from the aspect of run time efficiency view. In comparison to MaltParser, NSR yields 14 times faster in parsing speed.

Next, we analyze the effect of the two sequential taggers. The pure system performance of the neighbor tagger is 88.54 in $F_{(\beta)}$ rate, while the root tagger only achieves 61.67 $F_{(\beta)}$ score. The entire training time of the two taggers takes about 10 hours. Table 3 shows the compared results. It is clear that adding the two taggers leads better parsing accuracy than pure NSR and YSR. For example, it enhances the LAS score from 81.17 to 81.41 for NSR. Furthermore, the pure NSR and YSR still produce better parsing accuracy than MaltParser and MSTParser.

5 Conclusion

This paper presents an efficient and robust Chinese dependency parsing based on shift reduce parsing algorithms. We employ two sequential taggers to label the root and neighbor information as features. Experimental results show that our methods outperform two top-performed parsers, MaltParser and MSTParser in both accuracy and run-time efficiency. In the future, we will to investigate the effect of full parsing Chinese by applying shift-reduce-like approaches.

[2] http://w3.msi.vxu.se/~nivre/research/chiMaltSVM.html

Table 2: Parsing accuracy of each parsing algorithm

Evaluation Metrics	MaltParser (SVM)	MaltParser (MBT)	MST	this paper	
				NSR	YSR
LAS	80.74	73.53	78.08	81.41	**81.57**
UAS	81.98	75.40	79.53	82.60	**82.76**
LAC	91.28	86.26	89.21	92.26	**92.37**
Root	65.88	69.36	73.71	74.93	**77.61**
Sentence	33.12	25.67	24.07	32.85	**33.44**
TrainTime	6.74hr	**3.42min**	7.51hr	2.76hr	2.24hr
TestTime	15.92min	3.22min	10.15min	**1.12min**	1.15min

Table 3: Effective of the additional root and neighbor information

Improvement rate	NSR	YSR
LAS	81.17→81.41	81.16→81.57
UAS	82.33→82.60	82.37→82.76
LAC	92.07→92.26	92.15→92.37
Root_ Accuracy	74.14→74.93	76.24→77.61

References

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proc. of CoNLL*, pp. 149-164.

Ming-Wei Chang, Quang Do, and Dan Roth. 2006. Multilingual dependency parsing: a pipeline approach. *Recent Advances in Natural Language Processing*, pp. 195-204.

Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2005. Chinese deterministic dependency analyzer: examining effects of global features and root node finder. In *Proc. of SIGHAN*, pp. 17-24.

Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2006. Multilingual Dependency Parsing at NAIST. In Proc. of CoNLL, pp. 191-195.

Jason Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proc. of COLING*, pp. 340-345.

Johan Hall, Joakim Nivre, and Jens Nilsson. 2006. Discriminative Classifiers for Deterministic Dependency Parsing. In *Proc. of COLING-ACL Main Conference Poster Sessions*, pp. 316-323.

Hideki Isozaki, Hideto Kazawa, and Tsutomu Hirao. 2004. A deterministic word dependency analyzer enhanced with preference learning. In *Proc. of COLING*, pp. 275-281.

Sathiya Keerthi and Dennis DeCoste. 2005. A modified finite Newton method for fast solution of large scale linear SVMs, JMLR, 6: 341-361.

Taku Kudo and Yuji Matsumoto. 2003. Fast methods for kernel-based text analysis. In Proc. of ACL, pp. 24-31.

Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. 2004. Applying conditional random fields to Japanese morphological analysis, In *Proc. of EMNLP*, pp. 230-237.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative. In *Proc. of CoNLL*, pp. 216-220.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proc. of IWPT*, pp. 149-160.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryigit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proc. of CoNLL*, pp. 221-226.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proc. of EMNLP-CoNLL*, pp. 915-932.

Qin Iris Wang, Dekang Lin, and Dale Schuurmans. 2007. Simple training of dependency parsers via structured boosting. In *Proc. of IJCAI*, pp. 1756-1762.

Yu-Chieh Wu, Jie-Chi Yang, and Yue-Shi Lee. 2007. Multilingual deterministic dependency parsing framework using modified finite Newton method support vector machines. In *Proc. of EMNLP-CoNLL*, pp. 1175-1181.

Nianwen Xue, Fei Xia, Fu-Dong Chiou and Martha Palmer. 2005. The Penn Chinese Treebank: phrase structure annotation of a large corpus. Natural Language Engineering, 11(2):207-238.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. of IWPT*, pp. 195-206.