Dependency Parser for Chinese Constituent Parsing *

Xuezhe Ma, Xiaotian Zhang, Hai Zhao, Bao-Liang Lu
[1]Center for Brain-Like Computing and Machine Intelligence
Department of Computer Science and Engineering, Shanghai Jiao Tong University
[2]MOE-Microsoft Key Laboratory for Intelligent Computing and Intelligent Systems
Shanghai Jiao Tong University, 800 Dong Chuan Rd., Shanghai 200240, China
{xuezhe.ma,xtian.zh}@gmail.com, {zhaohai,blu}@cs.sjtu.edu.cn

Abstract

This paper presents our work for participation in the 2010 CIPS-ParsEval shared task on Chinese syntactic constituent tree parsing. We use dependency parsers for this constituent parsing task based on a formal dependency-constituent transformation method which converts dependency to constituent structures using a machine learning approach. A conditional random fields (CRF) tagger is adopted for head information recognition. Our experiments shows that acceptable parsing and head tagging results are obtained on our approaches.

1 Introduction

Constituent parsing is a challenging but useful task aiming at analyzing the constituent structure of a sentence. Recently, it is widely adopted by the popular applications of natural language processing techniques, such as machine translation (Ding and Palmer, 2005), synonym generation (Shinyama et al., 2002), relation extraction (Culotta and Sorensen, 2004) and lexical resource augmentation (Snow et al., 2004). A great deal of researches have been conducted on this topic with promising progress (Magerman, 1995; Collins, 1999; Charniak, 2000; Charniak and Johnson, 2005; Sagae and Lavie, 2006; Petrov and Klein, 2007; Finkel et al., 2008; Huang, 2008).

Recently, several effective dependency parsing algorithms has been developed and shows excellent performance in the responding parsing tasks (McDonald, 2006; Nivre and Scholz, 2004). Since graph structures of dependency and constituent parsing over a sentence are strongly related, they should be benefited from each other. It is true that constituent parsing may be smoothly altered to fit dependency parsing. However, due to the inconvenience from dependency to constituent structure, it is not so easy to adopt the latter for the former. This means that most of these popular and effective dependency parsing models can not be directly extended to constituents parsing. This paper proposes an formal method for such a conversion which adoptively solves the problem of ambiguity. Based on the proposed method, a dependency parsing algorithm can be used to solve tasks of constituent parsing.

A part of Tsinghua Chinese Treebank (TCT) (Zhou, 2004; Zhou, 2007; Chen et al., 2008) is used as the training and test data for the 2010 CIPS-ParsEval shared task. Being different from the annotation scheme of the Penn Chinese Treebank (CTB), the TCT has another annotation scheme, which combines both the constituent tree structure and the head information of each constituent. Specifically, there can be always multiple heads in a constituent. For the 2010 CIPS-ParsEval shared task, only segmented sentences are given in test data without part-of-speech (POS) tags, a POS tagger is required for this task. Therefore, we divide our system into three major cascade stages, namely POS tagging, constituent parsing and head information recognition, which are connected as a pipeline of processing. For the POS tagging, we adopt the SVMTool tagger (Gimenez and Marquez, 2004); for the constituent parsing, we use the Maximum Spanning Tree (MST) (McDonald, 2006) parser combined with a dependencies-to-constituents conversion; and for the head information recognition, we apply a sequence labeling method to label head information.

Section 2 presents the POS tagger in our approach. The details of our parsing method is presented in section 3. The head information recognition is described in section 4. The data and experimental results are shown in section 5. The last section is the conclusion and future work.

2 POS Tagging

The SVMTool tagger (Gimenez and Marquez, 2004) is used as our POS tagging tool for the first stage. It is a POS tagger based on SVM classifier, written in Perl. It can be trained on standardized collection of hand POS-tagged sentences. It uses SVM-Light[1] toolkit as the

*This work is partially supported by the National Natural Science Foundation of China (Grant No. 60903119, Grant No. 60773090 and Grant No. 90820018), the National Basic Research Program of China (Grant No. 2009CB320901), and the National High-Tech Research Program of China (Grant No.2008AA02Z315).

[1]http://www.cs.cornell.edu/People/tj/svm_light/.

implementation of SVM classifier and achieves 97.2% accuracy on the Penn English Treebank. We test the accuracy of the SVMTool tagger on the development set of the TCT (see section 5.1) and achieve accuracy of 94.98%.

3 Parsing Constituents Using Dependency Parsing Algorithms

3.1 Convert Dependencies to Constituents

The conversion from constituent to dependency structures is straightforward with some specific rules based on linguistic theory. However, there is not an effective method which can accurately accomplish the opposite transformation, from the dependency structures back into constituent ones due to the existence of ambiguity introduced by the former transformation.

Aimed at the above difficulty, our solution is to introduce a formal dependency structure and a machine learning method so that the ambiguity from dependency structures to constituent structures can be dealt with automatically.

3.1.1 Binarization

We first transform constituent trees into the form that all productions for all subtrees are either unary or binary, before converting them to dependency structures. Due to the binarization, the target constituent trees of the conversion from dependency back to constituent structures are binary branching.

This binarization is done by the left-factoring approach described in (Charniak et al., 1998; Petrov and Klein, 2008), which converts each production with n children, where $n > 2$, into $n - 1$ binary productions. Additional non-terminal nodes introduced in this conversion must be clearly marked. Transforming the binary branching trees into arbitrary branching trees is accomplished by using the reverse process.

3.1.2 Using Binary Classifier

We train a classifier to decide which dependency edges should be transformed first at each step of conversion automatically. After the binarization described in the previous section, only one dependency edge should be transformed at each step. Therefore the classifier only need to decide which dependency edge should be transformed at each step during the conversion.

As a result of the projective property of constituent structures, this problem only happens in the cases that modifiers are at both sides of their heads. And for these cases that one head has multiple modifiers, only the leftmost or the rightmost dependency edge could be transformed first. Therefore, a binary classifier is always enough for the disambiguation at each step.

1.	Word form of the parent
2.	Part-of-speech (POS) tag of the parent
3.	Word form of the leftmost child
4.	POS tag of the leftmost child
5.	Dependency label of the leftmost child
6.	Word form of the rightmost child
7.	POS tag of the rightmost child
8.	Dependency label of the rightmost child
9.	Distance between the leftmost child and the parent
10.	Distance between the rightmost child and the parent

Table 1: Features used for conversion classifier.

Support Vector Machine (SVM) is adopted as the learning algorithm for the binary classifier and the features are in Table 1.

3.1.3 Convert Constituent Labels

The rest problem is that we should restore the label for each constituent when dependency structure trees are again converted to constituent structures. The problem is solved by storing constituent labels as labels of dependency types. The label for each constituent is just used as the label dependency type for each dependency edge.

The conversion method is tested on the development, too. Constituent trees are firstly converted into dependency structures using the head rules described in (Li and Zhou, 2009). Then, we transform those trees back to constituent structure using our conversion method and use the PARSEVAL (Black et al., 1991) measures to evaluate the performance of the conversion method. Our conversion method obtains 99.76% precision and 99.76% recall, which is a great performance.

3.2 Dependency Parser for Constituent Parsing

Based on the proposed conversion method, dependency parsing algorithms can be used for constituent parsing. This can be done by firstly transforming training data from constituents into dependencies and extract training instances to train a binary classifier for dependency-constituent conversion, then training a dependency parser using the transformed training data. On the test step, parse the test data using the dependency parser and convert output dependencies to constituents using the binary classifier trained in advance. In addition, since our conversion method needs dependency types, labeled dependency parsing algorithms are always required.

1.	Constituent label of the constituent
2.	Constituent label of each child of the constituent.
3.	Wether it is a terminal for each child of the constituent
4.	The leftmost word in the sentence of each child of the constituent.
5.	The leftmost word in the sentence of each child of the constituent.

Table 2: CRF features for head information recognition.

1.	Word form and POS tag of the parent.
2.	Word form and POS tag of each child.
3.	POS tag of the leftmost child of each child.
4.	POS tag of the rightmost child of each child.
5.	Dependency label between the parent and its parent

Table 3: CRF features for dependency type labeling.

4 Head Information Recognition

Since head information of each constituent is always determined by the syntactic label of its own and the categories of the constituents in subtrees, the order and relations between the productions of each constituent strongly affects the head information labeling. It is natural to apply a sequential labeling strategy to tackle this problem. The linear chain CRF model is adopted for the head information labeling, and the implementation of CRF model we used is the 0.53 version of the CRF++ toolkit[2]. We assume that head information is independent between different constituents, which could decrease the length of sequence to be labeled for the CRF model.

We use a binary tag set to determine whether a constituent is a head, e.g. H for a head, O for a non-head, which is the same as (Song and Kit, 2009). The features in Table 2 are used for CRF model.

To test our CRF tagger, we remove all head information from the development set, and use the CRF tagger to retrieve the head. The result strongly proves its effectiveness by showing an accuracy of 99.52%.

5 Experiments

All experiments reported here were performed on a Core 2 Quad 2.83Ghz CPU with 8GB of RAM.

[2]The CRF++ toolkit is publicly available from `http://crfpp.sourceforge.net/`.

5.1 Data

There are 37,219 short sentences in official released training data for the first sub-task and 17,744 long sentences for the second sub-task (for the second sub-task, one line in the training data set may contain more than one sentence). We split one eighth of the data as our development set. On the other hand, there are both 1,000 sentences in released test data for the first and second sub-tasks.

5.2 Constituent Parsing

As mentioned in section 3, constituent parsing is done by using a dependency parser combined with our conversion method. We choose the second order maximum spanning tree parser with k-best online large-margin learning algorithm (Crammer and Singer, 2003; Crammer et al., 2003). The MST parser we use is in the form of an open source program implemented in C++[3].

The features used for MST parser is the same as (McDonald, 2006). Both the single-stage and two-stage dependency type labeling approaches are applied in our experiments. For the two-stage dependency type labeling, The linear chain CRF model is adopted instead of the first-order Markov model used in (McDonald, 2006). The features in Table 3 are used for CRF model. It takes about 7 hours for training the MST parser, and about 24 hours for training the CRF model.

As mentioned in section 3.1.2, SVM is adopted as the learning algorithm for the binary classifier. There are about 40,000 training instances in the first sub-task and about 80,000 in the second sub-task. Development sets are used for tuning parameter C of SVM and the training time of the SVM classifier for the first and second sub-task is about 8 and 24 hours, respectively. However, the conversion from dependencies to constituents is extremely fast. Converting more than 2,000 trees takes less than 1 second.

To transform the constituent trees in training set into dependency structures, we use the head rules of (Li and Zhou, 2009).

5.3 Results

The evaluation metrics used in 2010 CIPS-ParsEval shared task is shown in following:

1. syntactic parsing

$$\textbf{Precision} = \frac{\text{number of correct constituents in proposed parse}}{\text{number of constituents in proposed parse}}$$

$$\textbf{Recall} = \frac{\text{number of correct constituents in proposed parse}}{\text{number of constituents in standard parse}}$$

$$\textbf{F1} = \frac{2*\text{Precision}*\text{Recall}}{\text{Precision+Recall}}$$

[3]The Max-MSTParser is publicly available from `http://max-mstparser.sourceforge.net/`.

	without head			with head		
	Precision	Recall	F1	Precision	Recall	F1
single-stage	77.78	78.13	77.96	75.78	76.13	75.95
two-stage	78.61	78.76	78.69	76.61	76.75	76.68

Table 4: Official scores of syntactic parsing. single-stage and two-stage are for single-stage and two-stage dependency type labeling approached, respectively.

	Micro-R	Macro-R
single-stage	62.74	62.47
two-stage	63.14	62.48

Table 5: Official scores of event recognition

The correctness of syntactic constituents is judged based on the following two criteria:

(a) the boundary, the POS tags of all the words in the constituent and the constituent type label should match that of the constituent in the gold standard data.

(b) the boundary, the POS tags of all the words in the constituent, the constituent type label and head child index of the constituent should match that of the constituent in the gold standard data. (if the constituent contains more than one head child index, at least one of them should be correct.)

2. event pattern recognition

$$\textbf{Micro-R} = \frac{\text{number of all correct events in proposed parse}}{\text{number of all events in standard parse}}$$

$$\textbf{Macro-R} = \frac{\text{sum of recall of different target verbs}}{\text{number of target verbs}}$$

Here the event pattern of a sentence is defined to be the sequence of event blocks controlled by the target verb in a sentence. The criteria for judging the correctness of event pattern recognition is:

- the event pattern should be completely consistent with gold standard data (information of each event block should completely match and the order of event blocks should also consistent).

There are both two submissions for the first and second sub-tasks. One is using the single-stage dependency type labeling and the other is two-stage. Since there are some mistakes in our models for the second sub-task, the results of our submissions are unexpectedly poor and are not shown in this paper. All the results in this paper is reported by the official organizer of the 2010 CIPS-ParsEval shared task.

The accuracy of POS tagging on the official test data is 92.77%. The results of syntactic parsing for the first sub-task is shown in Table 4. And results of event recognition is shown in Table 5.

From the Table 4 and 5, we can see that our system achieves acceptable parsing and head tagging results, and the results of event recognition is also reasonably high.

5.4 Comparison with Previous Works

We comparison our approach with previous works of 2009 CIPS-ParsEval shared task. The data set and evaluation measures of 2009 CIPS-ParsEval shared task, which are quite different from that of 2010 CIPS-ParsEval shared task, are used in this experiment for the comparison purpose. Table 6 shows the comparison.

We compare our method with several main parsers on the official data set of 2009 CIPS-ParsEval shared task. All these results are evaluated with official evaluation tool by the 2009 CIPS-ParsEval shared task. Bikel's parser[4] (Bikel, 2004) in Table 6 is a implementation of Collins' head-driven statistical model (Collins, 2003). The Stanford parser[5] is based on the factored model described in (Klein and Manning, 2002). The Charniak's parser[6] is based on the parsing model described in (Charniak, 2000). Berkeley parser[7] is based on unlexicalized parsing model described in (Petrov and Klein, 2007). According to Table 6, the performance of our method is better than all the four parsers described above. Chen et al. (2009) and Jiang et al. (2009) both make use of combination of multiple parsers and achieve considerably high performance.

[4] http://www.cis.upenn.edu/~dbikel/software.html

[5] http://nlp.stanford.edu/software/lex-parser.shtml/

[6] ftp://ftp.cs.brown.edu/pub/nlparser/

[7] http://nlp.cs.berkeley.edu/Main.html

	F1
Bikel's parser	81.8
Stanford parser	83.3
Charniak's parser	83.9
Berkeley parser	85.2
this paper	**85.6**
Jiang et al (2009).	87.2
Chen et al (2009).	88.8

Table 6: Comparison with previous works

6 Conclusion

This paper describes our approaches for the parsing task in CIPS-ParsEval 2010 shared task. A pipeline system is used to solve the POS tagging, constituent parsing and head information recognition. SVMTool tagger is used for the POS tagging. For constituent parsing, we proposes a conversion based method, which can use dependency parsers for constituent parsing. MST parser is chosen as our dependency parser. A CRF tagger is used for head information recognition. The official scores indicate that our system obtains acceptable results on constituent parsing and high performance on head information tagging.

One of future work should apply parser combination and reranking approaches to leverage this in producing more accurate parsers.

References

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

Black, Ezra W., Steven P. Abney, Daniel P. Flickinger, Cluadia Gdaniec, Ralph Grishman, Philio Harrison, Donald Hindle, Robert J.P. Inqria, Frederick Jelinek, Judith L. Klavans, Mark Y. Liberman, Mitchell P. Marcus, Salim Roukos, and B Santorini. 1991. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of the February 1991 DARPA Speech and Natural Language Workshop*.

Charniak, Eugene and Mark Johnson. 2005. Coarse-to-fine-grained n-best parsing and discriminative reranking. In *Proceedings of the 43rd ACLL*, pages 132–139.

Charniak, Eugene, Sharon Goldwater, and Mark Johnson. 1998. Edge-based best-first chart parsing. In *Proceedings of the Sixth Workshop on Very Large Corpora*.

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

Chen, Yi, Qiang Zhou, and Hang Yu. 2008. Analysis of the hierarchical Chinese funcitional chunk bank. *Journal of Chinese Information Processing*, 22(3):24–31.

Chen, Xiao, Changning Huang, Mu Li, and Chunyu Kit. 2009. Better parser combination. In *CIPS-ParsEval-2009 shared task*.

Collins, Michael. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

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

Crammer, Koby and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learining*.

Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2003. Online passive aggressive algorithms. In *Proceedings of NIPS*.

Culotta, Aron and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of ACL*.

Ding, Yuan and Martha Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of ACL*.

Finkel, Jenny Rose, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. pages 959–967, The Ohio State University, Columbus, Ohio, USA.

Gimenez and Marquez. 2004. Svmtool: A general POS tagger generator based on support vector machines. In *Proceedings of the 4th International Conference of Language Resources and Evaluation*, Lisbon, Portugal.

Huang, Liang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL/HLT*.

Jiang, Wenbin, Hao Xiong, and Qun Liu. 2009. Mutipath shift-reduce parsing with online training. In *CIPS-ParsEval-2009 shared task*.

Klein, Dan and Christopher Manning. 2002. Fast exact inference with a factored model for natural language parsing. In *In Advances in NIPS 2002*, pages 3–10.

Li, Junhui and Guodong Zhou. 2009. Soochow university report for the 1st china workshop on syntactic parsing. In *CIPS-ParsEval-2009 shared task*.

Magerman, David M. 1995. Statistical decision-tree models for parsing. In *Proceedings of ACL*, pages 276–283, MIT, Cambridge, Massachusetts, USA.

McDonald, Ryan. 2006. *Discriminative Learning Spanning Tree Algorithm for Dependency Parsing*. Ph.D. thesis, University of Pennsylvania.

Nivre, Joakim and Mario Scholz. 2004. Deterministic dependency parsing of english text. In *Proceedings of the 20th international conference on Computational Linguistics (COLING-2004)*, pages 64–70, Geneva, Switzerland, August 23rd-27th.

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

Petrov, Slav and Dan Klein. 2008. Discriminative log-linear grammars with latent variables. In *Proceedings of NIPS 20*.

Sagae, Kenji and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of COLING/ACL*, pages 689–691, Sydney, Australia.

Shinyama, Yusuke, Satoshi Sekine, and Kiyoshi Sudo. 2002. Automatic paraphrase acquisition from news articles. In *HLT-2002*.

Snow, Rion, Daniel Jurafsky, and Andrew Y. Ng. 2004. Learning syntactic patterns for automatic hypernym discovery. In *Proceedings of NIPS*.

Song, Yan and Chunyu Kit. 2009. PCFG parsing with crf tagging for head recognition. In *CIPS-ParsEval-2009 shared task*.

Zhou, Qiang. 2004. Annotation scheme for Chinese treebank. *Journal of Chinese Information Processing*, 18(4):1–8.

Zhou, Qiang. 2007. Base chuck scheme for the Chinese language. *Journal of Chinese Information Processing*, 21(3):21–27.