Combining Data-Driven Constituent and Dependency Parsers for CIPS-ParsEval-2009

WANG Rui
Computational Linguistics Department
Saarland University, Germany
rwang@coli.uni-sb.de

ZHANG Yi
DFKI GmbH
Saarbruecken, Germany
yzhang@coli.uni-sb.de

QIU Wei, CHEN Mosha, LI Tingyu, ZHANG Wenbo, and YAO Tianfang
UdS-SJTU Joint Research Lab for Language Technology
Dept. of Computer Science and Engineering
Shanghai Jiao Tong University, Shanghai 200240, China.
{jiuren, mosha, hh190, hellowenniu}@sjtu.edu.cn; yao-tf@cs.sjtu.edu.cn

Abstract

In this paper, we describe our Chinese parsing system participated in the CIPS-ParsEval-2009 evaluation task 5. Two open source data driven parsing systems are used: constituent tree parsing with Berkeley's Parser, and dependency parsing with MSTParser. The results are combined to produce the phrase structure trees with head indices. Results show that the Berkeley parser achieved close to state-of-the-art performance on constituent tree parsing without head indices. And the combination gives us good partial-head identification accuracy. However the poor performance on full head identification indicates that the combination cannot handle the cases well where multiple heads exist.

1 Introduction

Chinese parsing has reportedly reached high accuracy in recent years, based mostly on the studies carried out with the Penn Chinese Treebank (hence CPTB). While the numbers are reassuring (e.g. comparing to the frustratingly low accuracy on Arabic), the worry of underestimating the complexity of Chinese syntax arises, in the fear of being over-committed to a particular annotation scheme.

While encoding linguistic knowledge into syntactic parsing systems has proved to help with parser performance, several statistical parsing systems are reluctant to go in this direction, in order to stay neutral to any linguistic assumption. This allows one to investigate the parsing system in a pure data-driven sense, and the resulting system can be easily adapted to different languages (or, rather just dataset from the pure data-driven view).

In very recent years, several of these “pure” data-driven parsing systems have emerged with the (close-to) state-of-the-art performance. And in the case of parsing Mandarin Chinese, most results are just reported on the CPTB. To our great interest is to see if these systems can easily achieve high performance on a dataset with different text genre and annotation scheme.

The main motivation of our participation is to see how much we can gain from the pure data-driven parsing models (mainly developed for English, but in principle, language-independent). Being different from the CPTB annotation, the Tsinghua Chinese Treebank (TCT) has another annotation scheme, which combines both the constituent tree structure and the head information of each constituent.

In practice, we take the strategy of combining two state-of-the-art parsing systems, one constituent parser and one dependency parser, to solve the problem. The Berkeley’s parser works reasonably fine with minimum assumption on the linguistics, and the claim is largely confirmed with a different treebank with different annotation. Multi-headed constructions are quite challenging to handle, though we apply several heuristic rules besides using the dependency parser to restore the head information. In addition, we also utilize another dependency parser specialized in Chinese parsing for comparison.

The rest of the paper will be organized as follows: Section 2 will give some background knowledge of treebanks and parsers; Section 3 describes the structure of our parsing system; Section 4 describes and analyzes experiment results; and section 5 concludes the paper.

2 Related Work

The dataset used in the task 5 of the CIPS-ParsEval-2009 evaluation is converted from the Tsinghua Chinese Treebank. Unlike many other treebanks, the input to the syntactic tree parsing is based on event description units, instead of complete sentences. On the assumption that the boundaries and relations between these event description units are determined separately, task 5 aims to identify the local fine-grained syntactic structures. The training set contains over 33K event description units (not count the trivial ones with only one token) and 355K tokens, giving us an average length of 11 tokens per input sequence. The dataset is a mixture of several genres, including roughly 36% newspaper texts, 56% encyclopedic texts, and 8% novel texts.

The annotation in the dataset is different to the other frequently used Chinese treebank (i.e. CPTB) from many perspectives. Apart from the differences in Part-of-Speech (POS) tags and phrase tagsets (i.e. the constituent names), the dataset in this task also contains head indices, making head identification of each constituent an important subtask.

For the mainstream statistical (syntactic) parsers, there are two categories in general, namely constituent parsers and dependency parsers. For both of them, we use open source software's, the Berkeley parser[1] (Petrov and Klein, 2007) for the former, and we choose the MSTParser[2] (McDonald et al., 2005) for the latter.

The Berkeley parser is purely data-driven and it does not have linguistic assumptions. This allows us to retrain the model with minimum efforts. Some internal comparison has shown that it outperforms the modified Dan Bikel's parser[3], because the latter system heavily depends on the linguistic or annotation assumptions from the Penn-style treebanks.

The MSTParser is a graph-based approach. The best parse tree is acquired by searching for a spanning tree which maximizes the score on either a partially or a fully connected graph with all words in the sentence as nodes (Eisner, 1996; McDonald et al., 2005).

As for a complement, we also consider a dependency parser specialized in processing Chinese, Deparser[4], which is developed by the Information Retrieval Research Center of Harbin Institute of Technology (Ma et al., 2004). It can be used for various NLP tasks, ranging from the basic word segmentation, POS tagging to the advanced topics like parsing, reference resolution and so on. In this work, we mainly focus on its parsing functions, which can offer us the syntactic dependency relations of a given sentence or paragraph. Among all the functions, word segmentation and POS tagging are the foundations of other advanced functions that must utilize the intermediate results of word segmentation and POS tagging. The intermediate results are expressed in XML form, which can be viewed as a DOM tree. Also the users could make use of segmentation and POS tagging results from other NLP toolkit like the Stanford Parser[5] and adjust it to the specified format of Deparser, then you could also get the dependency relations. In our experiments, we extract the word segmentation and POS tagging information from the test corpus and then utilize the Parsing function (i.e. Deparser) to obtain the dependency relations.

3 System Description

The workflow of our system can be mainly divided into three steps: 1) constituent and dependency tree extraction from the data with TCT treebank annotation; 2) training/testing the constituent parser and the dependency parser separately; and 3) integration of the results back to the TCT annotation. The following three subsections will elaborate on constituent tree parsing, dependency tree conversion, and the integration of the results respectively.

[1] http://code.google.com/p/berkeleyparser/
[2] http://sourceforge.net/projects/mstparser/
[3] http://www.cis.upenn.edu/~dbikel/download.html
[4] http://ir.hit.edu.cn/demo/ltp/
[5] http://nlp.stanford.edu/software/lex-parser.shtml

3.1 Constituent Tree Parsing

The TCT annotation in CIPS-ParsEval-2009 uses constituent trees together with head indices. It is possible to consider the phrase category and head information as a whole, and re-encode the head daughter indices in a way that can be integrated into data-driven constituent parsers.

It should be noted that the annotation in TCT are very different from other Chinese treebanks, e.g. CPTB. The difference is evident on many levels of annotation. Needless to mention are the differences on segmentation and POS tagsets. On the syntactic level, while most other treebanks provide complete tree annotation for sentences, the data provided in CIPS-ParsEval-2009 separate two levels of annotation, i.e. the detection of "event description units" (clauses), and syntactic tree annotation within each event description units. Therefore, the average length of the parser inputs is much shorter than usual.

Several preprocessing steps are taken to prepare the training data for two alternative constituent tree parsing models. For the first model, we remove the head information from the trees, and only use the constituent parser to produce the phrase boundaries and category. In the second model, we convert the head indexing digits to the following four types:

- -l: if the head is the left-most daughter of the constituent;
- -r: if the head is the right-most daughter of the constituent;
- -m: if the constituent has a single head that is neither on the left-most or right-most position;
- -c: if the constituent has more than one heads;

This information is combined with the phrase categories, so that e.g. a nominal phrase with its head daughter on the right-most position will be marked as "np-r". Obviously, the label '-l' and '-r' can be easily reverted to digit based head indices. For constituents marked with '-m' and have more than three daughters, further information is needed to locate the correct head. The same applies to '-c' that normally occurs with coordination phenomena. In the dataset, there are also phrases marked with no head "xp-", which will be copied as it is in our conversion.

3.2 Dependency Tree Conversion

The conversion to the dependency tree is not straightforward, since it has a different view of the depth of a tree from the constituent structure. Furthermore, the dependency tree does not contain any other nodes than the words themselves (i.e. no "constituent" nodes).

In general, the conversion algorithm works recursively to acquire the lexical head of each constituent or word token, and link them to their corresponding dependent heads. Following the different constituent markers we have in the previous section (e.g. "-l", "-r", etc.), we deal with them separately.

The normal cases, which have single heads, can be converted directly by linking all the non-headed components to the head. For instance,

```
 (np-2 X Y Z).
```

can be converted to,

```
Z → X;
Z → Y.
```

This includes both the single-headed cases and the xp-m cases mentioned in the previous section.

A more complex case is the xp-c case (e.g. "np-024", meaning a nominal phrase whose daughters whose indices are 0, 2, and 4 are heads), where the TCT annotation scheme allows multiple heads. There are two alternative ways to convert this structure into a dependency structure. Suppose we have the following constituent and we want to convert it into a dependency tree,

```
 (np-024 X , Y and Z)
```
[6].

One option is to link all the components (i.e. daughters from the tree view) to the first head, which is X here. The resulting dependency tree will be,

```
X → ,;
```

[6] For parsing the multiple-head information, we use a left greedy algorithm, which takes the head as "left" as possible. For instance, (np-124) would be parsed as 1, 2, and 4 instead of 1 and 24.

```
 X → Y;
 X → and;
 X → Z.
```

The alternative way is to binarize the tree first and then convert it recursively by taking the first head as the head of each constituent. After the binarization, the previous example looks like,

```
  (np-0 X , (np-0 Y and Z)).
```

Then the conversion is the same as the normal cases. The resulting tree will be,

```
 X → ,;
 X → Y;
 Y → and;
 Y → Z.
```

In practice, we use the second conversion approach, since the first one fails to emphasize on the components of the coordination (which occupies most of the xp-c cases), but treat *all* the components equally, including the punctuations.

There is one case left, which is xp- without any head information. This occurs frequently when a long sentence consists of several shorter clauses. To be different from the coordination, we simply link all the clauses to the first one as follows,

```
  (dj- X , Y , Z)
→
 X → ,;
 X → Y;
 X → ,;
 X → Z.
```

Notice that the conversion here is *not fully* reversible. For instance, (np-02 X , Y) and (np-0 X , Y) will be converted into the same dependency tree, "X→,; X→Y". Consequently, in the final stage, we apply some heuristics to restore some of the information loss.

Besides the dependency structure conversion, we also need to consider the labels being given to the dependencies. Enlightened by Hall (2008)'s conversion algorithm, we not only annotate the dependency label with constituent names, but more, in order to preserve the information as much as possible. For example, if we have a constituent tree as follows,

```
  (np-0 (np-1 X Y) Z).
```

The resulting dependency tree (after applying the algorithm mentioned above) will be,

```
 Y → X;
 Y → Z.
```

If we simply annotate the dependency label with the constituent names, which is "np", the resulting dependency structure will be the same as the tree converted from another constituent tree as follows,

```
  (np-1 X (np-0 Y Z)).
```

Thus, we need a richer annotation on each dependency label, which contains two elements,

- All the constituent names from the head word up to the last constituent where the two words share the same ancestor on the tree;
- The index of the shared ancestor on the corresponding constituent name path.

For the previous example, of the first dependency label Y → X, the first element is "np", which is the constituent name path from Y to the constituent where Y and X have the same ancestor; for the second dependency label Y → Z, the first element is "np, np", since Y and Z meet at the outer "np" (i.e. the grandparent node from the tree view). The second element for these two dependency labels will be "1" and "2" respectively, which are the indices of the shared ancestor constituent on the corresponding constituent paths. The final result is,

```
 Y → X (np|1);
 Y → Z (np,np|2).
```

After applying all these steps mentioned above, an example of a complete conversion is as follows,

```
 [dj-1 春节/t  [vp-0 是/vC  [np-2 [dj-1 [np-1 中国/nS  人民/n  ]
 [vp-1 十分/dD  重视/v  ] ] 的/uJDE  [np-1 传统/a  节日/n  ] ] ] ]
→
```

```
ROOT → 是 (NULL|0);
是 → 春节 (vp,dj|2);
是 → 节日 (vp|1);
节日 → 传统 (np|1);
节日 → 的 (np,np|2);
节日 → 重视 (np,np|2);
重视 → 十分 (vp|1);
重视 → 人民 (vp,dj|2);
人民 → 中国 (np|1).
```

And the resulting representation can be passed to any general dependency parsers, which are the MSTParser and the Deparser in this work.

3.3 Integration of Constituent and Dependency Trees

After obtaining both the constituent tree and the dependency tree (for the same sentence) from the parsers, we could then combine them and convert the result back to the original TCT representation. The result is mainly based on constituent parser's result, which is almost the same structure plus the partial head information.

For those cases with constituent names, xp-l or xp-r, we simply take the leftmost component or the rightmost component as the heads.

For the xp-m cases, we use the dependency parser's result as a hint to obtain the head. If there are only three components in that constituent, we take the second component (the middle one) as the head; if there are more than three components, we use the left-most head given by the dependency parser as the head, but it cannot be the left-most daughter of the constituent (in fact, "-m" stands for "head in middle").

For the xp-c cases, we again use the dependency parser's result as a hint. The tricky part here is that sometimes the dependency parser cannot discover all the heads correctly. Therefore, we apply heuristics to fix the errors. If the number of the components is an odd number, we add all the components at the odd positions to the head list, when the dependency parser misses some of them; if the number of the components is an even number, we do the same[7], except for the second last component. These two rules are aiming to capture coordination like "X , Y and Z" and "X , Y , and Z".

For the xj- cases, we simply keep them as they are.

4 Experiments

For the Berkeley parser and the Deparser, we use the default settings. For the MSTParser, based on our experience in participating in the CoNLL 2009 shared task (Zhang et al., 2009), we use the second order setting of the parser, which includes features over pairs of adjacent edges as well as features over single edges in the graph, and other settings are default.

4.1 Results

In all, we have submitted six runs for task 5 of the evaluation. Four of them take part in the open challenge, the other two are for the closed challenge. We assume that if we use other training data than those given by the challenge, additional resources, or heuristic rules, that submission will be counted as open; otherwise, it is closed.

We have three main modules/parameters to tune in our system: to use the MSTParser or the Deparser; to use heuristics to recover the head information or not; and whether to convert the dependency structure back to TCT annotation solely when the constituent parser fails. Consequently, we have six variants of the system, which form our six submissions as follows,

- Use MSTParser; use heuristics; and convert dependency back to TCT if constituent parser fails (open)

[7] Odd positions are counted from the left to the right, starting with 1 instead of 0.

- Use MSTParser; do not use heuristics; and convert dependency back to TCT if constituent parser fails (closed)
- Use MSTParser; use heuristics (open)
- Use MSTParser; do not use heuristics (closed)
- Use the Deparser; use heuristics (open)
- Use the Deparser; do not use heuristics (open)

The results are shown in the following tables,

Table 1 Official Results of Task 5 (open)

	Without-head match F1	Partial-head match F1	Complete-head match F1
Best	86.08	81.83	78.86
Run 1	85.23	80.05	72.79
Run 3	84.41	81.21	73.75
Run 5	84.41	81.21	73.74
Run 6	84.41	81.15	73.03
Median	84.41	81.21	73.75

Table 2 Official Results of Task 5(closed)

	Without-head match F1	Partial-head match F1	Complete-head match F1
Best	88.77	86.42	82.53
Run 2	85.23	82.35	74.27
Run 4	84.41	81.29	73.21
Median	85.23	82.35	74.27

The “Best” row shows the best results of all the submissions in the challenges; and the “Median” row shows the median number of the results for each column.

From the above table, we observe that our system performs fairly well for the Without-head match test and the Partial-head match test, but we are left behind in the Complete-head match test. Some possible reasons will be described in the following subsection.

4.2 Analysis and Discussion

By manually checking the differences between our results and the official answer, we have observed that there are mainly four types of errors in our system outputs:

- The structure of the constituent tree is wrong;
- Only one head is identified in a multi-headed constituent;
- Although the structure of constituent is correct, we label them wrong;
- The constituent tree and the labels are both correct, but the head indices are wrong.

Table 3 Percentage of the four error types in 100 wrong results randomly chosen

	A	B	C	D
Percentage	0.63	0.46	0.06	0.01

From Table 3 we can conclude that the type A and B errors largely affect our system's performance. However, due to the complexity of some linguistic phenomena (e.g. coordination), no parser does well enough to avoid the type A errors. Reflected in the official results, we are not fallen behind in the without-head match test or partial-head match test. Our system makes lots of mistakes of the type B, because 1) to handle multi-head cases is not trivial, and 2) our algorithm of combining the constituent tree and the dependency tree (for the same sentence) could not (always) restore the head information fully (though we have adopted some heuristic rules). This also explains the unsatisfied results for the complete-head match.

5 Conclusion and Future Work

In this paper, we described our participation of the Chinese parsing evaluation, CIPS-ParsEval-2009 task 5. In order to develop a system dealing with the TCT treebank annotation, which combines constituent tree structure and the head information, we integrated a constituent parsing model with two dependency parsing models. The results are reasonably good and we are quite satisfied with our first-time participation and the distant but effective collaboration between DFKI and UdS-SJTU joint research lab.

The future work lies on how to feed the pure data-driven parsing models with more fine-grained linguistic knowledge, such as handcrafted grammars, in order to further improve the performance. A closer investigation of the performance of dependency to constituent conversion based on Hall (2008) will also be done in the future.

Acknowledgments

The first author is supported by the PIRE scholarship program; the second author thanks DFKI and the Cluster of Excellence on Multimodal Computing and Interaction for their support of the work; and this work is also financially supported by the National Science Foundation of China (Grant No. 60773087) and the UDS-SJTU Joint Research Lab for Language Technology.

References

Eisner, J. 1996. *Three new probabilistic models for dependency parsing: An exploration.* In Proceedings of the 16th International Conference on Computational Linguistics (COLING-96), pages 340–345, Copenhagen, Denmark.

Ma, J., Zhang, Y., Liu, T., and Li, S. 2004. *A Statistical Dependency Parser of Chinese under Small Training Data*. In the 1st International Joint Conference of Natural Language Processing (IJCNLP-04), March 22-24, Sanya City, Hainan Island, China.

McDonald, R., Pereira, F., Ribarov, K., Hajic, J.: *Non-Projective Dependency Parsing using Spanning Tree Algorithms*. In: Proceedings of HLT-EMNLP 2005, Vancouver, Canada, pp. 523–530.

Hall, J. 2008. *Transition-Based Natural Language Parsing with Dependency and Constituency Representations*. Acta Wexionensia, No 152/2008, Computer Science, Växjö University (PhD Thesis)

Petrov, S. and Klein, D. 2007. *Improved Inference for Unlexicalized Parsing*. In HLT-NAACL 2007, April 22-27, Rochester, NY, USA.

Zhang, Y., Wang R., and Oepen, S. 2009. *Hybrid Multilingual Parsing with HPSG for SRL*. In Proceedings of the Twelfth Conference on Computational Natural Language Learning (CoNLL 2009), Boulder, CO, USA.