Towards Non-projective High-Order Dependency Parser

Wenjing Fang and **Kenny Q. Zhu** and **Yizhong Wang** and **Jia Tan**
Shanghai Jiao Tong University
{littlebeanfang,kzhu}@sjtu.edu.cn and {eastonwyz,tjtanjia.tan}@gmail.com

Abstract

This paper demonstrates a novel high-order dependency parsing framework that targets non-projective languages. It imitates how a human parses sentences in an intuitive way. At every step of the parse, it determines which word is the easiest to process among all the remaining words, identifies its head word and then folds it under the head word. This greedy framework achieves competitive accuracy on WSJ evaluation set and shows additional advantage on the non-projective corpus. Further, this work is flexible enough to be augmented with other parsing techniques. [1]

1 Introduction

Dependency parse trees, as the most commonly used syntax representation, is a preliminary part in many Natural Language Processing(NLP) tasks. Existing data-driven dependency parsers are divided into two classes, graph-based and transition-based. As typical graph-based parsers, MSTParser and its variants (McDonald et al., 2005) presently enjoy high accuracy at some cost of parsing time. However, such exact inference approach limits the range of features that can be extracted (McDonald and Nivre, 2007). MaltParser (Nivre, 2003), which is the most representative of transition-based parsers, carries out a sequence of greedy actions determined by a classifier trained from parsing sequences. Transition-based parsing is done incrementally by processing smaller word spans into subtrees first before combining smaller subtrees into bigger ones. Consequently, MaltParser has not met much success with non-projective parsing.[2]

Studies in psycholinguistics revealed how humans comprehend a sentence. Humans tend to perform a rapid and shallow recognition of major phrases, which guide the understanding process from the easiest relations to the more difficult ones (Townsend and Bever, 2001). By folding modifiers under their head words, we can gradually grasp the sentence structure and incorporate the already built structures for later parse. There is an earlier attempt inspired by the same intuition (Goldberg and Elhadad, 2010), whose framework is an adaptation of transition-based parser. However, it inherits the same problem as MaltParser in which candidate heads are all locally determined and can only deal with projective parsing.

This paper builds a parsing framework that follows the above intuition. It has two key components. The *sequence predictor* generates a permutation of words in the input sentence, which indicates the processing order, from the easy to the hard. The *head mapper* takes each word from the sequence and maps the head for each word in that order. Our current implementation generates transition-based processing sequence to guide a greedy high order graph-based decoder. It outperforms the easy-first parser in that it achieves similar accuracies with projective parsing (Kong and Smith, 2014), but can also deal with non-projective cases. In this paper, we use the idea of *parsing sequence* to bridge the gap between transition-based and graph-based methods under one framework.

[1] Kenny Q. Zhu is the corresponding author and is partially supported by NSFC Grant No. 61373031.

[2] The later proposal of SWAP action ameliorates some of this problem. But training a classifier for this action is hard due to limited resources of training data.

Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations, pages 248–252, Osaka, Japan, December 11-17 2016.

2 Framework

The general architecture of the parser is shown in Figure 1 and is divided into training phase and parsing phase.

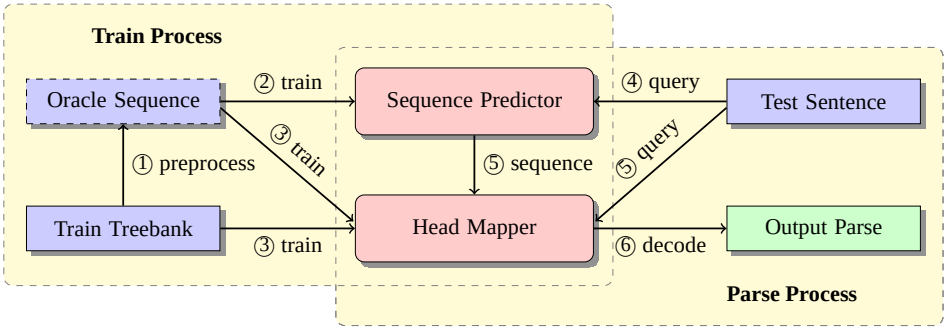

Figure 1: Sequence Based Parser Framework

Training: The preprocessing step generates oracle sequences from the gold standard parse trees. Only the word forms and the POS tags in these parse trees are used. Here, we assume that a child node is easier to process than its parent node and it is supposed to be attached before its parent. [3] We then train respectively a graph-based head mapper (a.k.a. decoder) from the gold sequences and the gold parses, and a sequence predictor from the gold sequences.

Parsing: Given an input sentence, the sequence predictor outputs a feasible decoding sequence, which is a permutation of the words in the input. For each word in this sequence, the head mapper returns its best head word according to a scoring function while employing a cycle detection mechanism. The process continues until all words in the sentence have found their heads. The procedure guarantees to produce a tree structure eventually.

In the current implementation, we generate the decoding sequence by *stackproj* algorithm (Nivre, 2009) in MaltParser and scorer-based greedy head mapper.

3 System Architecture

In the following, we present the preliminary investigation on the two key components of the our parser: head mapper and sequence predictor .

3.1 Head Mapper

Figure 2 shows the decoding process of the head mapper for a non-projective example sentence (McDonald et al., 2005): "*John saw a dog yesterday which was a Yorkshire Terrier*". A head mapper takes the lexical information of a sentence and a permutated sequence of words in that sentence as inputs. Suppose the sequence is:
$John_1 \rightarrow a_3 \rightarrow dog_4 \rightarrow yesterday_5 \rightarrow Yorkshire_9 \rightarrow a_8 \rightarrow was_7 \rightarrow which_6 \rightarrow Terrier_{10} \rightarrow saw_2$.

The subscript stands for the position of the word in original sentence. At step one, we look for the head of *John*. At this point, all other words are potential candidate heads. In order to measure the probabilities of these candidate arcs, we introduce a scorer, which is the key idea of graph-based parsers. By comparing the scores printed on every black arc in Figure 2, the red arc was eventually selected, i.e. saw is made the head of $John$. The process continues for the word a, etc.

In practice, we ensure that there are no cycles of nodes generated during parsing, so that the final output is a dependency tree structure starting from the ROOT node[4]. We also build a parse agenda to

[3] By this rule, multiple gold sequences can be generated from one dependency tree. In this paper, when a parent node has multiple children, we generate the sequence by a left-to-right order.

[4] A manually introduced node in dependency parsing task, it is the root of a dependency tree.

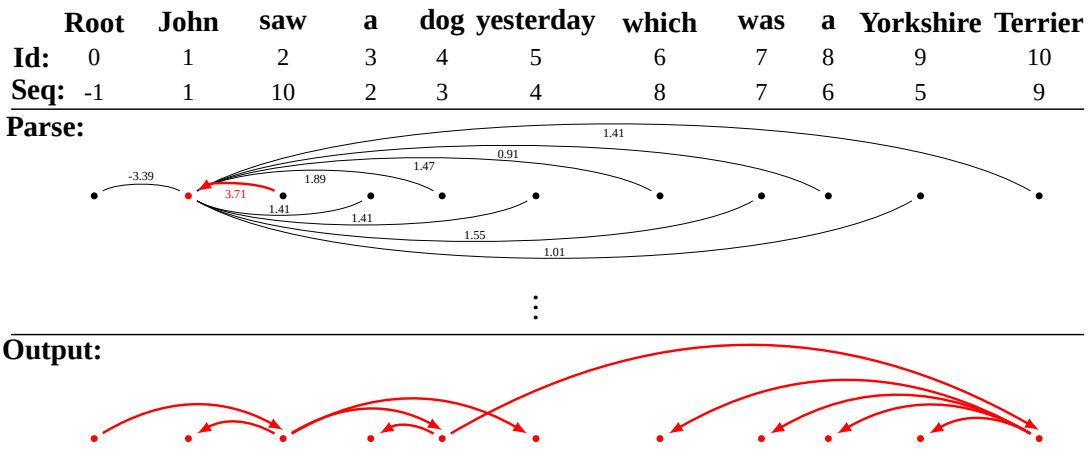

Figure 2: Example Parse of head mapper

record the existing arcs, which provides the high order information for our scorer. For example, after adding arc:$saw \rightarrow John$, all the attachment on these two nodes will take this arc into consideration.

We introduce a linear arc scorer to measure the score of a directed arc. The sum of all arc scores gives the final score of the whole parse tree. We currently use the typical high-dimensional binary features, including second order features (McDonald and Pereira, 2006). Because of the deterministic decoding in our framework, we can make use of existing arcs to guide later head mapping. This kind of decoder gives us the flexibility of applying any high order features explored by previous works (Carreras, 2007; Koo and Collins, 2010; Ma and Zhao, 2012).

The arc scorer is trained by the iterative online training framework MIRA (Margin Infused Relaxed Algorithm) (Crammer and Singer, 2003). In each iteration, we update the feature weights based on one sentence. The decoder gives a greedy parse according to current feature weights. By scoring the gold dependency tree and the current parse, along with the number of incorrect arcs in the current parse, MIRA keeps updating the weights until it eventually converges to an optimal scorer. The learning algorithm typically terminates after a few iterations.

3.2 Sequence Predictor

The intuition of sequence predictor is to rank words according to the ease of head word attaching. Words that are easy to handle can be processed earlier without high order features. To decide whether process a word immediately, we imitate the action classifier in MaltParser.

In fact, we can understand the action classifier in a different way that it can reflect the relative priority between the top two words on processing stack. We translate the actiones as:

- LA - process the word on the top of stack;
- RA - process the second word in stack;
- SH - postpone the process of both the two words on top of stack.

In this way, a word sequence can be inferred rather straightforwardly from the action sequence.

4 Demo

We build this sequence-based non-projective dependency parser and the part of the work is licensed under the GNU General Public License.

We evaluate our demo systsem on the WSJ test set under english[5] and five non-projective treebanks in different languages.[6] [7]

[5] the training set is sections 2-21 of WSJ corpus and test set is sections 00-01

[6] http://ilk.uvt.nl/conll/post_task_data.html

[7] http://www.nltk.org/nltk_data/

Table 1 shows the results of our system(nonproj), MaltParser and MSTParser. Generally, we outperform MaltParser in non-projective treebanks, which indicates that our framework tolerates free word order better. Our accuracy is not as good as MSTParser, because of the greedy decoding strategy. Nevertheless, this strategy gives rise to improvement in parsing time and flexibility in defining high order features than MSTParser.

Table 1: End-to-end accuracies on 8 languages

Language	nonproj	MSTParser	MaltParser
basque	77.45%	81.81%	74.88%
dutch	81.43%	85.66%	77.28%
danish	86.84%	89.39%	85.65%
portuguese	86.93%	88.63%	85.97%
slovene	78.26%	80.16%	76.09%
WSJ	89.50%	90.64%	90.23%

Further, we compare the accuracies of the non-projective arcs in the test data in Table 2. The system produces reasonable accuracies and outperforms MaltParser and MSTParser on parsing non-projective arcs.

Table 2: Accuracy of non-projective arcs in 5 languages

	basque			dutch			danish			portuguese			slovene		
parser	**correct**	**total**	**accuracy**	**correct**	**total**	**accuracy**	**correct**	**total**	**accuracy**	**correct**	**total**	**accuracy**	**correct**	**total**	**accuracy**
nonproj	225	569	0.395431	339	529	0.640832	79	121	0.652893	104	191	0.544503	101	263	0.3840304
MaltParser	200	569	0.351494	300	529	0.567108	58	121	0.479339	103	191	0.539267	98	263	0.3726236
MSTParesr	204	569	0.358524	204	529	0.385633	63	121	0.520661	90	191	0.471204	109	263	0.4144487

Given a CoNLL formatted training data and test data, our demo can parse out the dependency tree. Figure 1 is the snapshot of the demo showing the parsing result on the multilingual corpus.

Figure 3: Example Parse of head mapper

5 Conclusion

We develop a novel sequence-based dependency parsing framework. It shows promising results despite of an unoptimized implementation. The key idea is that a good parsing sequence can be predetermined and can contribute to good parsing accuracy and substantial speedup. Although only a few simple approaches are attempted to train the sequence predictor, the framework allows the integration of better and

more advanced models, which may lead to results closer to an upper bound 93.59% [8] for the WSJ test set.

Even though the current classifier based sequence predictor produces better results among our preliminary attempts, the parsing accuracy is limited by the rather localized or even incorrect sequence order produced. More importantly, we discovered that the parsing accuracy is very sensitive to the quality of parsing sequence. Future work can be focused on developing better sequence predictors that outperform this classifier based method.

Graph-based methods spend most of the time extracting features. Some work attempted to save time by displaying arc filter (Bergsma and Cherry, 2010; Rush and Petrov, 2012). We can incorporate some of these techniques to speed up the parsing. Furthermore, Beam search works well in a left-to-right head attaching. We can also adapt beam search to our framework so as to relax its strictly greedy nature.

References

[Bergsma and Cherry2010] Shane Bergsma and Colin Cherry. 2010. Fast and accurate arc filtering for dependency parsing. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 53–61.

[Carreras2007] Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *EMNLP-CoNLL*, pages 957–961.

[Crammer and Singer2003] Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991.

[Goldberg and Elhadad2010] Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *NAACL HLT*, pages 742–750.

[Kong and Smith2014] Lingpeng Kong and Noah A Smith. 2014. An empirical comparison of parsing methods for stanford dependencies. *arXiv preprint arXiv:1404.4314*.

[Koo and Collins2010] Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *ACL*, pages 1–11.

[Ma and Zhao2012] Xuezhe Ma and Hai Zhao. 2012. Fourth-order dependency parsing. In *COLING (Posters)*, pages 785–796. Citeseer.

[McDonald and Nivre2007] Ryan T McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *EMNLP-CoNLL*, pages 122–131.

[McDonald and Pereira2006] Ryan T McDonald and Fernando CN Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL*.

[McDonald et al.2005] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *HLT/EMNLP*, pages 523–530.

[Nivre2003] Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *IWPT*.

[Nivre2009] Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *ACL-IJCNLP: Volume 1-Volume 1*, pages 351–359.

[Rush and Petrov2012] Alexander M Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *NAACL HLT*, pages 498–507.

[Townsend and Bever2001] David J Townsend and Thomas G Bever. 2001. *Sentence comprehension: The integration of habits and rules*, volume 1950. MIT Press.

[8] Take the sequence inferred the oracle actions from MaltParser as both training sequence and parsing sequence and define only first and second order features in MSTParser for head mapper to get this raw bound