IHS-RD-Belarus at SemEval-2016 Task 9: Transition-based Chinese Semantic Dependency Parsing with Online Reordering and Bootstrapping.

Artsiom Artsymenia
IHS Inc. / IHS Global Belarus
Minsk, Belarus

Palina Dounar
IHS Inc. / IHS Global Belarus
Minsk, Belarus

Maria Yermakovich
IHS Inc. / IHS Global Belarus
Minsk, Belarus

{Artsiom.Artsymenia,Polina.Dovnar,Maria.Yermakovich}@ihs.com

Abstract

This paper is a description of our system developed for SemEval-2016 Task 9: Chinese Semantic Dependency Parsing. We have built a transition-based dependency parser with online reordering, which is not limited to a tree structure and can produce 99.7% of the necessary dependencies while maintaining linear algorithm complexity. To improve parsing quality we used additional techniques such as pre- and post-processing of the dependency graph, bootstrapping and a rich feature set with additional semantic features.

1 Introduction

Dependency parsing is one of the core tasks in natural language processing, as it provides useful information for other NLP tasks. Traditional syntactic parsing usually represents a sentence as a tree-shape structure and this restriction is essential for most of efficient algorithms developed in the past years. Semantic dependency parsing, on the other hand, deals with acyclic graphs, where words may have multiple incoming dependencies. It significantly complicates the task and requires development of the new algorithms or special adoption of the old ones.

There are two main approaches to dependency parsing (Nivre and McDonald, 2008). The first one is a graph-based approach, for example, spanning tree algorithms (McDonald et al., 2005), where the goal is to find the highest scoring tree from a complete graph of dependencies between words in the sentence. The second approach is transition-based, which, instead of searching for global optimum, greedily finds local optimum with a chain of actions that lead to a parsing tree. The main advantage of the transition-based parsers is that they are in general faster than graph-based ones because of the linear complexity of the algorithm (Nivre, 2003).

As we move from syntactic parsing to semantic parsing, and instead of projective trees have to deal with acyclic graphs, exact inference becomes NP-hard, so some strong independence assumptions or heuristics are needed. A lot of modifications have been proposed for transition-based parsers to support non-projective structures. Nivre and Nilsson (2005) proposed a pseudo-projective parsing technique which consists in modifying the input into a projective dependency tree with extended labels, performing the projective parsing and then applying an approximate back transformation. Attardi (2006) introduced additional actions that add dependencies between the roots of non-adjacent subtrees. Both techniques maintain linear algorithm complexity at the expense of incomplete coverage of all possible dependency trees. Complete coverage of all non-projective trees is achieved by Nivre (2009) with a technique called *on-line reordering* but it increases the worst-case complexity from linear to quadratic.

2 System Description

The core of our system is a transition-based dependency parser with on-line reordering in style of Titov et al, (2009). It continues the tradition of At-

Proceedings of SemEval-2016, pages 1207–1211,
San Diego, California, June 16-17, 2016.

tardi (2006) by extending the action-set of the model, but adds only one new action. While this model is not enough to parse any arbitrary structure, it still can successfully reproduce 99.72% of the dependencies in the "sdpv2" corpus and 99.78% of the dependencies in the "text" corpus.

2.1 Parsing Algorithm

The state of the parser is defined by the current stack *S*, the queue *I* of remaining input sentence words, and partial dependency graph constructed by previous actions. The parser starts with artificial *TOP* node in *S* and all input words in *I*. The processing terminates when a state with an empty queue is reached. Parser can change its state with one of the five possible actions:

- Action *Left-Arc* adds a dependency arc from the first word in queue to the word on top of the stack
- Action *Right-Arc* adds a dependency arc from the word on top of the stack to the first word in queue
- Action *Reduce* removes the word from the top of the stack
- Action *Shift* moves the first word from the queue to the stack.
- Action *Swap* swaps two words at the top of the stack.

With additional limitation on the number of times *Swap* operation can be performed for each node, parser with this action set has a linear time complexity.

To convert gold dependencies into gold actions, the following algorithm is used for each state of the parser:

- If stack is empty, perform *Shift* action.
- If the word on the top of the stack has no incoming or outcoming dependencies or the rightmost dependency is located to the left of the first word in the queue, perform *Reduce* action.
- If the rightmost dependency of the word on top of the stack is equal to the first word in queue, but this dependency is already present in the partial dependency graph, perform *Reduce* action.
- If there is a dependency arc between the top of the stack and the first word in queue and it is not present in the partial dependency graph, perform *Left-Arc* or *Right-Arc* action according to the direction of dependency.
- If the size of the stack is less than two, or two top words in the stack in the same order were already swapped, perform *Shift* action.
- If there is a dependency arc between the second word in stack and the first word in the queue, and it is not present in the partial dependency graph, perform *Swap* action
- Otherwise, perform *Shift* action.

After that, the obtained gold actions are used to train a log-linear classifier with the following set of features:

- Words in stack and queue: W(S0), W(S1), W(S2), W(I0), W(I1), W(I2)
- POS tags: T(S0), T(S1), T(S2), T(I0), T(I1), T(I2)
- Words of the last left child(LC), last right child(RC) and last parent(P) in the partial dependency graph for words in stack and queue: W(LC_S0), W(RC_S0), W(P_S0), W(LC_S1), W(RC_S1), W(P_S1), W(LC_I0), W(P_I0)
- POS tags of LC, RC and P: T(LC_S0), T(RC_S0), T(P_S0), T(LC_S1), T(RC_S1), T(P_S1), T(LC_I0), T(P_I0)
- The number of left children, right children and parents: N(LC_S0), N(RC_S0), N(P_S0), N(LC_I0), N(P_I0)
- Previous action: PA
- Information about existing arcs in partial dependency graph: A(S0,I0), A(S1,I0)
- Distance between words: D(S0,I0), D(S1,S0)

- Various combinations of the above features, for example: W(I0)+T(I0), W(S0)+W(I0), T(S3)+T(S2)+T(S1)+T(I0)+T(I1) and other.

One more log-linear classifier is used to set a semantic label to the dependency. The feature set is similar to the one described above, but instead of words in the stack and the queue this model uses a parent and a child of the dependency arc in question.

2.2 Bootstrapping

Transition-based parsers use history to predict the next action. In our system the words in the stack, queue and partial dependency graph are used as features of the log-linear classifier. While it is the source of useful information (when past actions are correct), it is also the source of errors (when past actions are incorrect). If statistical model is trained only on gold parses, it has little possibility to recover from mistakes, because it will have to predict next decisions from a state that was never encountered during training. To reduce the gap between gold and real-life parser configurations, Choi and Palmer (2011) proposed to use bootstrapping on automatic parses. In this approach a model trained on gold configurations is used to parse training corpus and generate new training instances, where the current configuration is achieved by automatic parsing.

In our system we split training corpus into 10 parts and parse each part with a model trained with other 9 parts to be as close to real-life as possible. After that a new model is trained with both gold training instances and bootstrapped training instances.

2.3 Semantic Features

To analyze the influence of different semantic dependencies on each other, we used some additional features produced from the output of IHS Goldfire Question-Answering system (Todhunter et al., 2013). This system has its own Semantic Processor, which performs complete linguistic analysis of text documents, such as lexical, part-of-speech, syntactic, and semantic analysis and other. The Q-A system is built on top of the semantic labeling of words with basic knowledge types (e.g., objects/classes of objects, cause-effect relations, whole-part relations etc.). A matching procedure makes use of the aforementioned types of semantic labels to determine the exact answers to the questions and present them to the user in the form of the fragments of sentences or a newly synthesized phrase in the natural language.

For example, one of such semantic labels, originally extracted by Goldfire system from one of the sentences in the training data, looked this way:

我六点钟出门，以便赶上火车

I left home at six, (in order) to catch the train.

出门 (left) *–Effect→* 赶上火车 (catch the train)

In order to convert semantic labels into semantic dependencies, we extracted the main word of the answer and corrected differences in word segmentation. After transformations the example above is represented as the following semantic dependency:

6 赶上 VV 3(出门) Effect

Although the semantic labels, extracted by Goldfire, are different from the ones that should be extracted in SemEval-2016 Task 9 (in the example above the label is supposed to be *ePurp*), we expected them to provide additional information in cases when the context of words in stack and queue is not sufficient.

2.4 Dependencies Pre-processing and Post-processing

Error analysis of the base model showed that incorrect extraction of the *eCoo* dependency is one of the most frequent mistakes. It turned out that *eCoo* connections may have different direction: from left to right and from right to left. Further analysis revealed that in most cases the direction correlates with the position of the parent node of *eCoo* chain: if the parent is to the left, the direction is more likely to be from left to right, and if the parent is to the right, the direction is more likely to be from right to left.

The intuition is that the parser is confused with the different direction of the *eCoo* connections. That is why we converted all *eCoo* dependencies into right-to-left direction in the training corpus, and added a post-processing procedure which converts *eCoo* connections in the output dependency graph into left-to-right direction if the parent of the top node of *eCoo* chain is to the left.

model	LP	LR	LF	NLF
Base	61.61	60.91	61.26	49.16
PP	**62.79**	62.07	**62.43**	49.89
PP_BS	62.25	62.32	62.29	47.12
PP_SEM	62.68	62.13	62.40	**52.71**
PP_BS_SEM	62.37	**62.47**	62.42	46.88

Table 1: evaluation on sdpv2 (labeled)

model	UP	UR	UF	NUF
Base	78.20	77.30	77.75	62.77
PP	**79.87**	78.96	**79.41**	63.64
PP_BS	78.99	79.07	79.03	61.99
PP_SEM	79.55	78.85	79.20	**64.27**
PP_BS_SEM	79.33	**79.45**	79.39	60.42

Table 2: evaluation on sdpv2 (unlabeled)

model	LP	LR	LF	NLF
Base	66.79	65.74	66.26	51.48
PP	67.23	66.18	66.70	51.79
PP_BS	67.32	66.87	67.09	49.04
PP_SEM	68.42	67.55	67.98	**53.45**
PP_BS_SEM	**68.46**	**68.28**	**68.37**	50.94

Table 3: evaluation on text (labeled)

model	UP	UR	UF	NUF
Base	81.08	79.82	80.44	64.56
PP	81.67	80.40	81.03	64.66
PP_BS	81.44	80.90	81.17	63.18
PP_SEM	**82.60**	81.56	82.08	**65.77**
PP_BS_SEM	82.46	**82.25**	**82.35**	65.60

Table 4: evaluation on text (unlabeled)

model	LP	LR	LF	NLF
Our system	**58.78**	**59.33**	**59.06**	40.84
a1	55.52	55.85	55.69	**49.23**
a2	55.65	56.04	55.84	47.80
lbpg	55.64	58.89	57.22	45.57
lbpgs	58.38	57.25	57.81	41.56
lbpg75	57.88	57.67	57.78	48.89

Table 5: SemEval results on sdpv2 (labeled)

model	UP	UR	UF	NUF
Our system	**77.28**	**78.01**	**77.64**	60.20
a1	73.51	73.94	73.72	**60.71**
a2	73.79	74.30	74.04	59.69
lbpg	72.87	77.11	74.93	58.03
lbpgs	76.28	74.81	75.54	54.34
lbpg75	75.55	75.26	75.40	58.28

Table 6: SemEval results on sdpv2 (unlabeled)

model	LP	LR	LF	NLF
Our system	**68.71**	**68.46**	**68.59**	50.57
a1	65.36	64.98	65.17	54.70
a2	65.37	64.92	65.15	54.62
lbpg	63.34	67.89	65.54	51.75
lbpgs	67.35	65.11	66.21	47.79
lbpg75	66.43	66.33	66.38	**57.51**

Table 7: SemEval results on text (labeled)

model	UP	UR	UF	NUF
Our system	**82.56**	**82.26**	**82.41**	64.58
a1	79.06	78.60	78.83	**65.71**
a2	78.89	78.35	78.62	64.93
lbpg	76.73	82.24	79.39	63.21
lbpgs	81.22	78.52	79.85	55.51
lbpg75	79.97	79.85	79.91	63.87

Table 8: SemEval results on text (unlabeled)

3 Experiments

In order to evaluate the influence of the methods described above on parsing quality, we have built five different systems and tested them on the development set. The labeled and unlabeled results for 2 types of corpus ("sdpv2" and "text") are presented in tables from 1 to 4. LP, LR and LF are labeled precision, recall and F1-score for predicted dependencies (parent-child-dependency label triples). NLF is F1-score for non-local dependencies. UP, UR, UF and NUF are unlabeled counterparts. The b*ase* system is built with the algorithm described in 2.1. BS is bootstrapping (2.2), Sem – Semantic Features (2.3) and PP is pre- and post-processing (2.4). Pre- and post-processing procedures proved to be the most useful, especially on the "sdpv2" corpus. Bootstrapping improves recall, but harms precision and non-local dependencies. Semantic Features, on the other hand, help to extract non-local dependencies. The PP_BS_SEM system, with all features included, performed better than all other on the "text" corpus and has comparable results on the "sdpv2" corpus. That is why the output of this system was submitted to SemEval-2016 Task 9. The results of our system, compared to other submissions, are represented in tables 5-8. Our submitted system has shown the highest precision, recall and F1-score values for all dependencies, but did not perform well on non-local labeled dependencies, which may be an effect of bootstrapping or shortcomings of the selected model in general.

4 Conclusion

We have built a transition-based semantic dependency parser with online reordering, bootstrapping, additional semantic features and graph pre- and post-processing that achieved the best results in SemEval-2016 Task 9. All features proved to improve the overall performance, but some details may need further improvement. Bootstrapping requires a better balance to avoid its bad influence on precision and non-local dependencies. Also some additional restrictions to the base algorithm may be needed, because now it is allowed for some words to have no parent at all, and while it may be normal for semantic dependency parsing in general, it is not the thing for the task at hand. We leave these details for a future work.

References

Giuseppe Attardi. 2006. *Experiments with a multilanguage non-projective dependency parser*. In Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL), pages 166–170.

Jinho D. Choi and Martha Palmer. 2011. *Getting the most out of transition-based dependency parsing*. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 687–692.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. *Non-projective dependency parsing using spanning tree algorithms*. In Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP'05), pages 523–530.

Joakim Nivre. 2003. *An Efficient Algorithm for Projective Dependency Parsing*. In Proceedings of the 8th International Workshop on Parsing Technologies, IWPT'03, pages 149–160.

Joakim Nivre and Jens Nilsson. 2005. *Pseudo-projective dependency parsing*. In Proc. of ACL, 99–106.

Joakim Nivre and Ryan McDonald. 2008. *Integrating graph-based and transition-based dependency par ers.* In Proceedings of ACL, pages 950–958.

Joakim Nivre. 2009. *Non-projective dependency parsing in expected linear time*. In Proceedings of ACL-IJCNLP.

Ivan Titov and James Henderson. 2007. *A latent variable model for generative dependency parsing*. In Proceedings of the 10th International Conference on Parsing Technologies (IWPT), pages 144–155.

Todhunter, J., Sovpel, I., and Pastanohau, D. 2013. *System and method for automatic semantic labeling of natural language texts*. U.S. Patent 8 583 422.