HLT@SUDA at SemEval-2019 Task 1: UCCA Graph Parsing as Constituent Tree Parsing

Wei Jiang, Zhenghua Li*, Yu Zhang, Min Zhang
School of Computer Science and Technology, Soochow University, China
{wjiang0501, yzhang25}@stu.suda.edu.cn, {zhli13,minzhang}@suda.edu.cn

Abstract

This paper describes a simple UCCA semantic graph parsing approach. The key idea is to convert a UCCA semantic graph into a constituent tree, in which extra labels are deliberately designed to mark remote edges and discontinuous nodes for future recovery. In this way, we can make use of existing syntactic parsing techniques. Based on the data statistics, we recover discontinuous nodes directly according to the output labels of the constituent parser and use a biaffine classification model to recover the more complex remote edges. The classification model and the constituent parser are simultaneously trained under the multi-task learning framework. We use the multilingual BERT as extra features in the open tracks. Our system ranks the first place in the six English/German closed/open tracks among seven participating systems. For the seventh cross-lingual track, where there is little training data for French, we propose a language embedding approach to utilize English and German training data, and our result ranks the second place.

1 Introduction

Universal Conceptual Cognitive Annotation (UCCA) is a multi-layer linguistic framework for semantic annotation proposed by Abend and Rappoport (2013). Figure 1 shows an example sentence and its UCCA graph. Words are represented as terminal nodes. Circles denote non-terminal nodes, and the semantic relation between two non-terminal nodes is represented by the label on the edge. One node may have multiple parents, among which one is annotated as the primary parent, marked by solid line edges, and others as remote parents, marked by dashed line edges. The primary edges form a tree structure, whereas the remote edges enable reentrancy, forming directed acyclic graphs (DAGs).[1] The second feature of UCCA is the existence of nodes with discontinuous leaves, known as discontinuity. For example, node 3 in Figure 1 is discontinuous because some terminal nodes it spans are not its descendants.

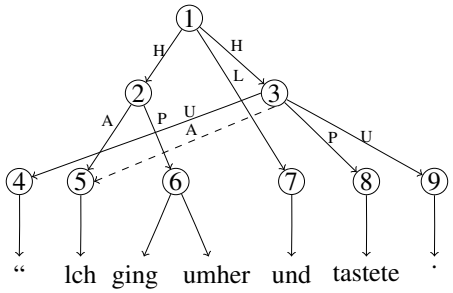

Figure 1: A UCCA graph example from the German data. The English translation is “ I went around and groped . We assign a number to each non-terminal node to facilitate illustration.

Hershcovich et al. (2017) first propose a transition-based UCCA Parser, which is used as the baseline in the closed tracks of this shared task. Based on the recent progress on transition-based parsing techniques, they propose a novel set of transition actions to handle both discontinuous and remote nodes and design useful features based on bidirectional LSTMs. Hershcovich et al. (2018) then extend their previous approach and propose to utilize the annotated data with other

*Corresponding author, hlt.suda.edu.cn/zhenghua

[1]The full UCCA scheme also has implicit and linkage relations, which are overlooked in the community so far.

Proceedings of the 13th International Workshop on Semantic Evaluation (SemEval-2019), pages 11–15
Minneapolis, Minnesota, USA, June 6–7, 2019.

semantic formalisms such as abstract meaning representation (AMR), universal dependencies (UD), and bilexical Semantic Dependencies (SDP), via multi-task learning, which is used as the baseline in the open tracks.

In this paper, we present a simple UCCA semantic graph parsing approach by treating UCCA semantic graph parsing as constituent parsing. We first convert a UCCA semantic graph into a constituent tree by removing discontinuous and remote phenomena. Extra labels encodings are deliberately designed to annotate the conversion process and to recover discontinuous and remote structures. We heuristically recover discontinuous nodes according to the output labels of the constituent parser, since most discontinuous nodes share the same pattern according to the data statistics. As for the more complex remote edges, we use a biaffine classification model for their recovery. We directly employ the graph-based constituent parser of Stern et al. (2017) and jointly train the parser and the biaffine classification model via multi-task learning (MTL). For the open tracks, we use the publicly available multilingual BERT as extra features. Our system ranks the first place in the six English/German closed/open tracks among seven participating systems. For the seventh cross-lingual track, where there is little training data for French, we propose a language embedding approach to utilize English and German training data, and our result ranks the second place.

2 The Main Approach

Our key idea is to convert UCCA graphs into constituent trees by removing discontinuous and remote edges and using extra labels for their future recovery. Our idea is inspired by the pseudo non-projective dependency parsing approach propose by Nivre and Nilsson (2005).

2.1 Graph-to-Tree Conversion

Given a UCCA graph as depicted in Figure 1, we produce a constituent tree shown in Figure 2 based on our algorithm described as follows.

1) Removal of remote edges. For nodes that have multiple parent nodes, we remove all remote edges and only keep the primary edge. To facilitate future recovery, we concatenate an extra "remote" to the label of the primary edge, indicating that the corresponding node has other remote relations. We can see that the label of the child node 5 becomes "A-remote" after conversion in Figure 1 and 2.

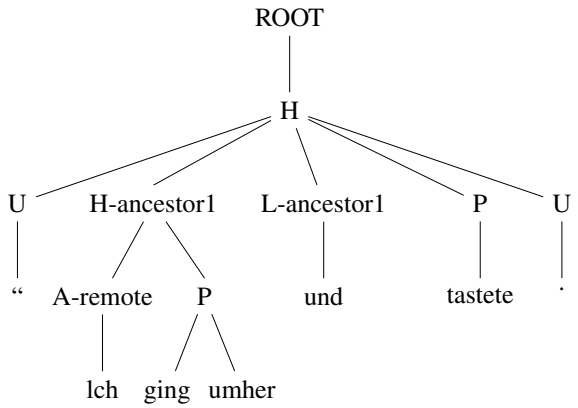

Figure 2: Constituent tree converted from UCCA gragh.

	train	dev	total	percent(%)
ancestor 1	1460	149	1609	91.3
ancestor 2	96	19	115	6.5
ancestor 3	21	0	21	1.2
discontinuous	16	2	18	1.0

Table 1: Distribution of discontinuous structures in the English-Wiki data, which is similar in the German data.

2) Handling discontinuous nodes. We call node 3 in Figure 1 a discontinuous node because the terminal nodes (also words or leaves) it spans are not continuous ("lch ging umher und" are not its descendants). Since mainstream constituent parsers cannot handle discontinuity, we try to remove discontinuous structures by moving specific edges in the following procedure.

Given a discontinuous node $A = 3$, we first process the leftmost non-descendant node $B =$ "lch". We go upwards along the edges until we find a node $C = 2$, whose father is either the lowest common ancestor (LCA) of $A = 3$ and $B =$ "lch" or another discontinuous node. We denote the father of $C = 2$ as $D = 1$.

Then we move $C = 2$ to be the child of $A = 3$, and concatenate the original edge label with an extra string (among "ancestor 1/2/3/..." and "discontinuous") for future recovery, where the number represents the number of edges between

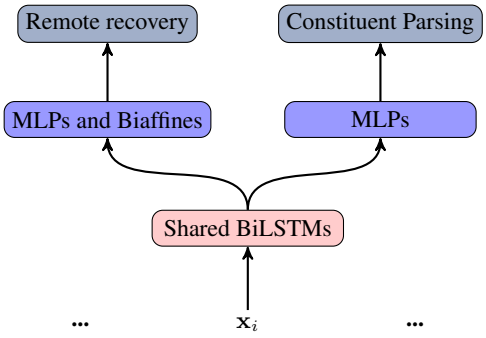

Figure 3: The framework of MTL.

the ancestor $D = 1$ and $A = 3$.

After reorganizing the graph, we then restart and perform the same operations again until there is no discontinuity.

Table 1 shows the statistics of the discontinuous structures in the English-Wiki data. We can see that D is mostly likely the LCA of A and B, and there is only one edge between D and A in more than 90% cases.

Considering the skewed distribution, we only keep "ancestor 1" after graph-to-tree conversion, and treat others as continuous structures for simplicity.

3) Pushing labels from edges into nodes. Since the labels are usually annotated in the nodes instead of edges in constituent trees, we push all labels from edges to the child nodes. We label the top node as "ROOT".

2.2 Constituent Parsing

We directly adopt the minimal span-based parser of Stern et al. (2017). Given an input sentence $\mathbf{s} = w_1...w_n$, each word w_i is mapped into a dense vector $\mathbf{x}_i$ via lookup operations.

$$\mathbf{x}_i = \mathbf{e}_{w_i} \oplus \mathbf{e}_{t_i} \oplus ...$$

where $\mathbf{e}_{w_i}$ is the word embedding and $\mathbf{e}_{t_i}$ is the part-of-speech tag embedding. To make use of other auto-generated linguistic features, provided with the datasets, we also include the embeddings of the named entity tags and the dependency labels, but find limited performance gains.

Then, the parser employs two cascaded bidirectional LSTM layers as the encoder, and use the top-layer outputs as the word representations.

Afterwards, the parser represents each span $w_i...w_j$ as

$$\mathbf{r}_{i,j} = (\mathbf{f}_j - \mathbf{f}_i) \oplus (\mathbf{b}_i - \mathbf{b}_j)$$

where $\mathbf{f}_i$ and $\mathbf{b}_i$ are the output vectors of the top-layer forward and backward LSTMs.

The span representations are then fed into MLPs to compute the scores of span splitting and labeling. For inference, the parser performs greedy top-down searching to build a parse tree.

2.3 Remote Edge Recovery

We borrow the idea of the state-of-the-art biaffine dependency parsing (Dozat and Manning, 2017) and build our remote edge recovery model. The model shares the same inputs and LSTM encoder as the constituent parser under the MTL framework (Collobert and Weston, 2008). For each remote node, marked by "-remote" in the constituent tree, we consider all other non-terminal nodes as its candidate remote parents. Given a remote node A and another non-terminal node B, we first represent them as the span representations. $\mathbf{r}_{i,j}$ and $\mathbf{r}_{i',j'}$, where i, i', j, j' are the start and end word indices governed by the two nodes. Please kindly note that B may be a discontinuous node.

Following Dozat and Manning (2017), we apply two separate MLPs to the remote and candidate parent nodes respectively, producing $\mathbf{r}_{i,j}^{\texttt{child}}$ and $\mathbf{r}_{i',j'}^{\texttt{parent}}$.

Finally, we compute a labeling score vector via a biaffine operation.

$$\mathbf{s}(A \leftarrow B) = \begin{bmatrix} \mathbf{r}_{i,j}^{\texttt{child}} \\ 1 \end{bmatrix}^{\mathrm{T}} \mathbf{W} \mathbf{r}_{i',j'}^{\texttt{parent}} \quad (1)$$

where the dimension of the labeling score vector is the number of the label set, including a "NOT-PARENT" label.

Training loss. We accumulate the standard cross-entropy losses of all remote and non-terminal node pairs. The parsing loss and the remote edge classification loss are added in the MTL framework.

2.4 Use of BERT

For the open tracks, we use the contextualized word representations produced by BERT (Devlin et al., 2018) as extra input features.[2] Following previous works, we use the weighted summation of the last four transformer layers and then multiply a task-specific weight parameter following (Peters et al., 2018).

[2]We use the multilingual cased BERT from https://github.com/google-research/bert.

3 Cross-lingual Parsing

Because of little training data for French, we borrow the treebank embedding approach of Stymne et al. (2018) for exploiting multiple heterogeneous treebanks for the same language, and propose a language embedding approach to utilize English and German training data. The training datasets of the three languages are merged to train a single UCCA parsing model. The only modification is to concatenate each word position with an extra language embedding (of dimension 50), i.e. $\mathbf{x}_i \oplus \mathbf{e}_{lang=en/de/fr}$ to indicate which language this training sentence comes from. In this way, we expect the model can fully utilize all training data since most parameters are shared except the three language embedding vectors, and learn the language differences as well.

4 Experiments

Except BERT, all the data we use, including the linguistic features and word embeddings, are provided by the shared task organizer (Hershcovich et al., 2019). We also adopt the averaged F1 score as the main evaluation metrics returned by the official evaluation scripts (Hershcovich et al., 2019).

We train each model for at most 100 iterations, and early stop training if the peak performance does not increase in 10 consecutive iterations.

Table 2 shows the results on the dev data. We have experimented with different settings to gain insights on the contributions of different components. For the single-language models, it is clear that using pre-trained word embeddings outperforms using randomly initialized word embeddings by more than 1% F1 score on both English and German. Finetuning the pre-trained word embeddings leads to consistent yet slight performance improvement. In the open tracks, replacing word embedding with the BERT representation is also useful on English (2.8% increase) and German (1.2% increase). Concatenating pre-trained word embeddings with BERT outputs leads is also beneficial.

For the multilingual models, using randomly initialized word embeddings is better than pre-trained word embeddings, which is contradictory to the single-language results. We suspect this is due to that the pre-trained word embeddings are independently trained for different languages and would lie in different semantic spaces without proper aligning. Using the BERT outputs is tremendously helpful, boosting the F1 score by more than 10%. We do not report the results on English and German for brevity since little improvement is observed for them.

Methods	F1 score		
	Primary	Remote	Avg
Single-language models on English			
random emb	0.778	0.542	0.774
pretrained emb (no finetune)	0.790	0.494	0.785
pretrained emb	0.794	0.535	**0.789**
bert	0.821	0.593	0.817
pretrained emb ⊕ bert	0.825	0.603	**0.821**
official baseline (closed)	0.745	0.534	0.741
official baseline (open)	0.753	0.514	0.748
Single-language models on German			
random emb	0.817	0.549	0.811
pretrained emb (no finetune)	0.829	0.544	0.823
pretrained emb	0.831	0.536	**0.825**
bert	0.842	0.610	0.837
pretrained emb ⊕ bert	0.849	0.628	**0.844**
official baseline (closed)	0.737	0.46	0.731
official baseline (open)	0.797	0.587	0.792
Multilingual models on French			
random emb	0.688	0.343	0.681
pretrained emb	0.673	0.174	0.665
bert	0.796	0.524	**0.789**
official baseline (open)	0.523	0.016	0.514

Table 2: Results on the dev data.

5 Final Results

Table 3 lists our final results on the test data. Our system ranks the first place in six tracks (English/German closed/open) and the second place in the French open track. Note that we submitted a wrong result for the French open track during the evaluation phase by setting the wrong index of language, which leads to about 2% drop of averaged F1 score (0.752). Please refer to (Hershcovich et al., 2019) for the complete results and comparisons.

6 Conclusions

In this paper, we describe our system submitted to SemEval 2019 Task 1. We design a simple UCCA semantic graph parsing approach by making full use of the recent advance in syntactic parsing community. The key idea is to convert UCCA graphs into constituent trees. The graph recovery

Tracks	F1 score		
	Primary	Remote	Avg
English-Wiki_closed	0.779	0.522	0.774
English-Wiki_open	0.810	0.588	0.805
English-20K_closed	0.736	0.312	0.727
English-20K_open	0.777	0.392	0.767
German-20K_closed	0.838	0.592	0.832
German-20K_open	0.854	0.641	0.849
French-20K_open	0.779	0.438	0.771

Table 3: Final results on the test data in each track. Please refer to the official webpage for more detailed results due to the limited space

problem is modeled as another classification task under the MTL framework. For the cross-lingual parsing track, we design a language embedding approach to utilize the training data of resource-rich languages.

Acknowledgements

The authors would like to thank the anonymous reviewers for the helpful comments. We also thank Chen Gong for her help on speeding up the minimal span parser. This work was supported by National Natural Science Foundation of China (Grant No. 61525205, 61876116).

References

Omri Abend and Ari Rappoport. 2013. Universal Conceptual Cognitive Annotation (UCCA). In *Proc. of ACL*, pages 228–238.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proc. of ICML*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of ICLR*.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for ucca. In *Proc. of ACL*, pages 1127–1138.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018. Multitask parsing across semantic representations. In *Proc. of ACL*, pages 373–385.

Daniel Hershcovich, Zohar Aizenbud, Leshem Choshen, Elior Sulem, Ari Rappoport, and Omri Abend. 2019. Semeval 2019 task 1: Cross-lingual semantic parsing with ucca. *arXiv:1903.02953*.

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

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proc. of ACL*, pages 818–827.

Sara Stymne, Miryam de Lhoneux, Aaron Smith, and Joakim Nivre. 2018. Parser training with heterogeneous treebanks. In *Proc. of ACL*, pages 619–625.