SUDA–Alibaba at MRP 2019: Graph-Based Models with BERT

Yue Zhang[1], Wei Jiang[2], Qingrong Xia[2], Junjie Cao[1], Rui Wang[1], Zhenghua Li[2]*, Min Zhang[2]
[1] Alibaba Group, China
[2] School of Computer Science and Technology, Soochow University, China
{shiyu.zy, junjie.junjiecao, masi.wr}@alibaba-inc.com
wjiang0501@stu.suda.edu.cn, kirosummer.nlp@gmail.com
{zhli13, minzhang}@suda.edu.cn

Abstract

In this paper, we describe our participating systems in the shared task on Cross-Framework Meaning Representation Parsing (MRP) at the 2019 Conference for Computational Language Learning (CoNLL). The task includes five frameworks for graph-based meaning representations, i.e., DM, PSD, EDS, UCCA, and AMR. One common characteristic of our systems is that we employ graph-based methods instead of transition-based methods when predicting edges between nodes. For SDP, we jointly perform edge prediction, frame tagging, and POS tagging via multi-task learning (MTL). For UCCA, we also jointly model a constituent tree parsing and a remote edge recovery task. For both EDS and AMR, we produce nodes first and edges second in a pipeline fashion. External resources like BERT are found helpful for all frameworks except AMR. Our final submission ranks the third on the overall MRP evaluation metric, the first on EDS and the second on UCCA.

1 Introduction

Cross-Framework Meaning Representation Parsing (MRP) at CoNLL 2019 contains five different graph-based semantic representations, including DM, PSD, EDS, UCCA and AMR. The shared task releases training and testing data for all five frameworks. For different frameworks, organizers design different evaluation criteria and provide standard evaluation scripts. Details about the five semantic formalisms and evaluation criteria are given in the MRP shared task homepage[1] and the overview paper (Oepen et al., 2019). In the following, we give a brief introduction of each framework and followed by our corresponding approaches.

Semantic Dependency Parsing (SDP) aims to parse the predicate-argument relationships for all words in the input sentence, leading to bilexical semantic dependency graphs (Oepen et al., 2014, 2015, 2016). This shared task focuses on two different formal types of SDP representations, i.e., DELPH-IN MRS Bi-Lexical Dependencies (abbr. as DM, Ivanova et al., 2012) and Prague Semantic Dependencies (abbr. as PSD, Hajič et al., 2012; Miyao et al., 2014). They are both classified as Flavor 0 representations in the sense that every node in the graph must anchor to one and only one token unit, and vice verse. Compared with syntactic dependency trees, some nodes in an SDP graph may have no incoming edges and some may have multiple ones. Borrowing the idea of Dozat and Manning (2018), we encode the input word sequence with BiLSTMs and predict the edges and labels between words with two MLPs. We also predict the POS tag and frame of each word jointly under the MTL framework.

Universal Conceptual Cognitive Annotation (UCCA) is a multi-layer linguistic framework (Flavor 1) firstly proposed by Abend and Rappoport (2013). In UCCA graphs, input words are leaf (or terminal) nodes. One non-terminal node governs one or more nodes, which may be discontinuous; and one node can have multiple governing (parent) nodes through multiple edges, consisting of a single primary edge and other remote edges. Relationships between nodes are given by edge labels. The primary edges form a tree structure, whereas the remote edges introduce reentrancy, forming directed acyclic graphs (DAGs).[2] We directly adopt the previous graph-based UCCA parser proposed by Jiang et al. (2019), treating UCCA graph parsing as constituent parsing and remote edge recovery under the MTL framework.

*Corresponding author
[1] http://mrp.nlpl.eu/index.php?page=1

[2] The full UCCA scheme also has implicit nodes and linkage relations, which are excluded in the shared task.

Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 CoNLL, pages 149–157

https://doi.org/10.18653/v1/K19-2014

Elementary Dependency Structure (EDS) is a graph-structured semantic representation formalism (Flavor 1) proposed by Oepen and Lønning (2006). Buys and Blunsom (2017) introduce a neural encoder-decoder transition-based model to obtain the EDS graph. They use external knowledge to generate nodes[3]. Chen et al. (2018) introduce a novel SHRG (Synchronous Hyperedge Replacement Grammar) extraction algorithm which requires a syntactic tree and alignments between conceptual edges and surface strings. Such alignment information is not provided in the shared task and seems difficult for us to induce due to time limitation. Therefore, we divide the EDS task into two-stage task: node prediction and edge prediction, and treat both as sequence labeling. To tackle with the explicit, many-to-many relationship between nodes and sub-strings of the underlying sentence (via anchoring), we introduce a similar method used in dependency SRL (semantic role labeling) to produce nodes. For the edge prediction, the widely-used Biaffine model is used.

Abstract meaning representation (AMR), proposed by Banarescu et al. (2013), is a broad-coverage sentence-level semantic formalism (Flavor 2) to encode the meaning of natural language sentences. AMR can be regarded as a rooted labeled directed acyclic graph. Nodes in AMR graphs represent concepts, and labeled directed edges are relations between the concepts. Due to the time limitation and the complexity of the AMR parsing problem, we directly employ the state-of-the-art parser of Lyu and Titov (2018), which treats AMR parsing as a graph prediction problem.

Methodology Summarization. Our participating systems can be characterized in the following aspects:

- **Graph-Based.** All our methods for the five frameworks belong to graph-based methods in the sense that we directly predict edges among nodes, instead of using a transition system. In particular, the constituent parser for UCCA is also graph-based.
- **Joint Model.** We simplify our architecture and use less training steps by jointly modeling subtasks whenever it is possible. This is achieved by sharing the encoder component under the MTL framework and it is adopted by the DM, PSD, and UCCA models. For both EDS and AMR, we first produce nodes and then predict edges in a pipeline architecture. We have not attempted to jointly solve multiple semantic frameworks via MTL yet.
- **BERT.** We observe that using BERT as our extra inputs is effective for all the models, except AMR. It is also interesting that BERT-large does not produce more improvements over BERT-base based our preliminary experiments.

Our final submission ranks the third on the overall evaluation metric, the first on EDS and the second on UCCA. In the following, We introduce our methods in detail in Section 2, and present the experimental results in Section 3, and finally conclude our paper in Section 4.

2 Methods

2.1 SDP

We construct our SDP parser based on the ideas of Dozat and Manning (2017) and Dozat and Manning (2018). Note that lemmas, POS tags and frames are also included in the MRP evaluation metrics, so our method is a bit different from Dozat and Manning (2018).

Edge Prediction. Our basic edge prediction model is similar to the Dozat and Manning (2017) and Dozat and Manning (2018). The input words are first mapped into a dense vector composed by pretrained word embeddings and character-level features.

$$\mathbf{x}_i = \mathbf{e}_i^{word} \oplus \mathbf{e}_i^{char}$$

where $\mathbf{e}_i^{char}$ is extracted by the bidirectional character-level LSTM (Lample et al., 2016). They are then fed into a multilayer bidirectional word-level LSTM to get contextualized representations. Finally, two modules are applied to predict edges. One is to predict whether or not a directed edge exists between two words (keeping the edges between pairs of words with positive scores); and the other is to predict the most probable label for each potential edge (choosing the label with maximum score). Each of them has two seperate MLPs for head and dependent representations and a biaffine layer for scoring. The training loss is the sum of sigmoid cross-entropy loss for edges and softmax

[3]Their knowledge sources ERG 1214 (http://svn.delph-in.net/erg/tags/1214) are not in the white list of this shared task.

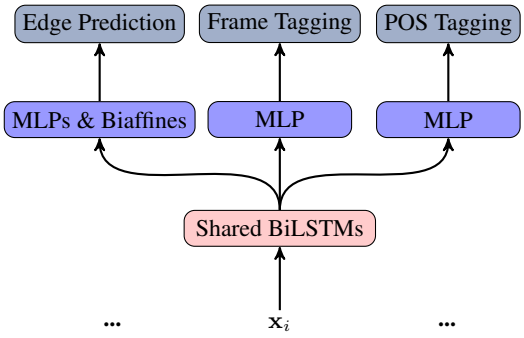

Figure 1: The framework of our SDP Parser.

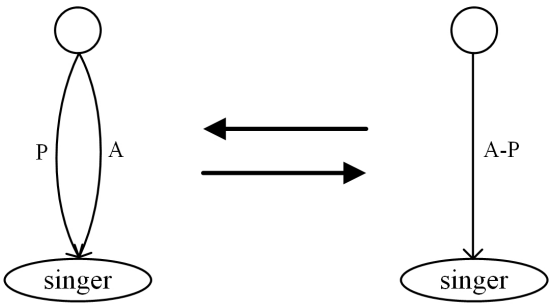

Figure 2: An example of newest version of UCCA.

cross-entropy loss for labels.

$$\ell^{'} = \ell^{label} + \ell^{edge}$$

Lexical Taggers. This SDP task is more difficult than the ealier 2014 and 2015 SDP tasks (Oepen et al., 2014, 2015), since the gold tokenization result, lemmas, and POS tags are not available in the parser input data and the predictions are parts of the MRP evaluation metrics. We use automatic tokenization result and lemmas provided by the datasets; while for POS and frames, we train the taggers with the edge predicter simultaneously under the multi-task learning framework. Figure 1 shows the framework of our SDP parser. The final training loss is :

$$\ell^{sdp} = \ell^{'} + \ell^{frame} + \ell^{pos}$$

where ℓ^{frame} and ℓ^{pos} are both softmax cross-entropy losses.

2.2 UCCA

We directly follow Jiang et al. (2019)'s graph-based UCCA parser. The key idea is to convert a UCCA semantic graph into a constituent tree, and mark remote edges and discontinuous nodes with extra labels for later recovery.

Graph-to-tree Conversion. In the new version of UCCA, one non-terminal node is allowed to point to another by more than one primary edges, e.g., the word "singer" represents a "process" and a "participant" in a semantic scene at the same time (Figure 2 shows the example). Therefore, we keep only one of the edges and concatenate all their tags in the alphabetical order. During the recovery step, the edge with a mixed label is splitted according to the label's length.

Then for the edges that point to the same node, we delete all remote edges and concatenate an extra "remote" to the label of the only primary edge. To handle discontinuous node, we trace bottom-up from a discontinuous leaf node until we find the specific node whose parent is the lowest common ancestor (LCA) of the discontinuous node and leaf node. Finally we move the edge to make the specific node become the child of the discontinuous, with "-ancestor" added behind the edge label. Please refer to Jiang et al. (2019) for more conversion details.

Constituent Parsing. We directly adopt the minimal span-based parser of Stern et al. (2017). Given an input sentence $X = \{x_0, x_1, \cdots, x_n\}$, each word x_i is mapped into a dense vector $\mathbf{x}_i$ and fed into bidirectional LSTM layers. The top-layer output of each position are used to represent the span 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. Finally, a parse tree is derived by a greedy top-down searching. In particular, We start from the span of the whole sentence, assigning it the label with maximum score and choosing the best split point where the sum of two sub-spans' splitting scores are maximum. Then we repeat this process for the left and right sub-spans until the span can no longer be split.

Remote Edge Recovery. To recover remote edges, two seperate MLPs and biaffine operations are applied on remote nodes and other candidate parent nodes representations. They share the same inputs and LSTM encoder with the constituent parser under the MTL framework. The parsing loss and the cross-entropy losses of all remote and non-terminal node pairs are added in the MTL framework.

2.3 EDS

This subsection describes our models on EDS task, which are simple but effective. Since many external resources cannot be used to generate nodes in EDS graph in this shared task, we convert the main task into two sequence labeling subtasks: node prediction and edge prediction.

Node prediction. For each input sentence $X = \{x_0, x_1, \cdots, x_n\}$, our model needs to predict nodes in EDS graph $N = \{n_0, n_1, \cdots, n_m\}$. For each node, it contains such information: id, anchors, labels, properties and values. Taking one node as example, anchors mean the span indicators of characters in the input sentence, like "$< a, b >$". It means this node strides across the sub string from character index a to b in the input sentence string. We can convert anchors from character index provided by the data to word index[4].

From the definition of EDS graph, there is a many-to-many relationship between words and nodes. Lots of nodes stride across more than one word of the underlying sentence according to their anchors. To simplify the alignment between words and nodes, we divide nodes in graphs into two types due to their characteristic.

The first type is those nodes whose labels begin with "_" or properties are not null, e.g., "_fund_n_1". We call them the *original node*, labels of which usually consist of three parts: lemma, coarse part-of-speech(POS) tag and sense according to the role the word plays in the sentence. The second type is the *append node*, e.g., "udf_q". Those nodes do not contain explicit sentence text and many of them stride across several words, which makes us difficult to obtain their anchors.

We align the nodes to input words, so that we predict labels and anchors based on the input word sequence:

For ***original nodes***, we use the lemma of each word provided by organizers and we take POS tag and sense as a joint label and predict them with sequence labeling, like the POS tagger. To generate training data, based on our statistics and analysis, we tag words in the following ways: 1) in most cases, original nodes are aligned to words one by one and these words are tagged with their "POS_sense"[5]; 2) for those compound nodes striding across several words (e.g., "_such+as_p" means it combines "such" and "as" two words), we tag the first word with the true label and other words tagged with "A"; and 3) we tag the input words disappearing in EDS graph with "O". Note that, when one word participates in different *original nodes*, we will concatenate all labels of one word together with separator ":".

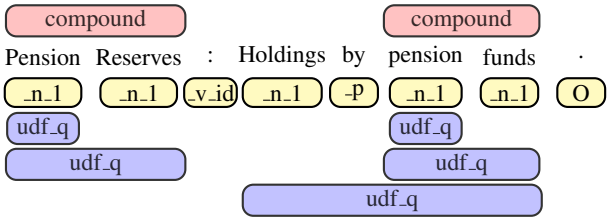

Figure 3: An example for node prediction in the EDS graph. We use nodes with different color shadows to represent different kinds of EDS nodes. Nodes with yellow shadows under the input sentence are *original nodes*. The number of *original nodes* is equal to the length of sentence. Nodes with pink shadows above the sentence are non-terminal *append nodes* and purple nodes below the sentence are leaf nodes. In the example sentence, there are two non-terminal nodes and five leaf nodes. Nodes over across several words means their anchors.

For ***append node***, we divide it into two types, leaf nodes and non-terminal nodes[6]. The difference between leaf nodes and non-terminal nodes is that whether they are pointed by other nodes. Both leaf nodes and non-terminal nodes may overlap several words and one input word may participate in different *append nodes* as Figure3 shows. Therefore, we take a similar way like the prediction of SRL predicate and argument[7]. We firstly predict the begin index of each *append node*, like predicate identification in the SRL task, and then, we predict their end index and their labels, like argument prediction. Given the beginning, we tag the end words of nodes with their labels and concatenate labels with separator ":" if more than one node have same anchors. This allows us to solve

[4]Index conversion makes us align nodes to words in the input sentence, so that we can simplify our task. And due to the evaluation standard of this shared task, we do not consider punctuation and we can take punctuation as common words if the task needs.

[5]Sometimes, the label only contains POS information.

[6]We take this division originally for edge prediction and task simplification. In EDS graph, leaf nodes should participate in only one edge, but our edge prediction model achieves great performance on all nodes, so we do not distinguish them in the edge prediction.

[7]We did not apply the same sequence labeling method for the *original node*, since it is difficult to recover when continuous words participate in more than nodes.

Detailed errors	Modification
company abbreviation, like Corp., Co., Inc	corporation, company and inc
address forms, like Mr., Mrs., Dr.	corresponding full name
numbers in English	Arabic numerals
country name abbreviation	delete "." in string
some symbol like "%", "#", "$", "&" and ":"	English String

Table 1: Effective post-processing for labels and values in EDS node.

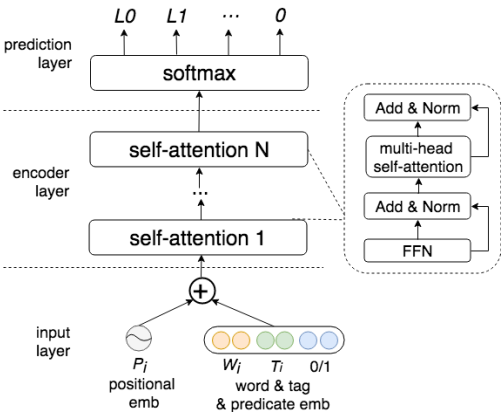

Figure 4: Architecture of our multi-layers self-attention-based model. The dotted box on the right is the detailed composition of the self-attention block.

the problem of the overlap and multi-participation of the *append nodes* elegantly.

We apply a multi-head self-attention model (Tan et al., 2018) for our node prediction, which has been proved effective in SRL task. Details about the attention model please refer to Tan et al. (2018) and Vaswani et al. (2017). For *original nodes* and beginning of append nodes prediction, the input consists of embeddings of words and POS-tags provided by organizers; while for end of *append nodes* prediction, the input contains embedding of the beginning indicator in addition. Then we use simple softmax function to get the index or labels whose scores are the highest.

Edge prediction. Compared with the node prediction, the edge prediction model is more straightforward, which builds links between nodes and generates the final EDS graph.

Labels in edges are used to tag the relation between two nodes, like "ARG1", "BV". If there is no edges between two nodes, we use the relation "O". Note that, we add one pseudo node like the "ROOT" node in the dependency parsing, so that we can get the top node of the graph (which is pointed by the "ROOT" node).

For the edge prediction model, a multi-layered BiLSTM is firstly used to encode the original input sentence, so that we get the representation of each word. Then we represent each node according to its anchors (begin index and end index) as formula in 2.2 shows, so that each node contains information of all words in the anchors. Lastly, we compute the score of each candidate edge relation between two nodes by the biaffine mechanism, and get label whose score is the highest.

Post-processing. We convert our predicted results into nodes according to the predicted begin index and the predicted label (including the end index and corresponding label). We sort them by the anchors of nodes to get the id of each node in the EDS graph. We index these nodes in the *ascending* order by the begin index of anchors and then in the *descending* order by the end of the anchors.

We analysis the results of our splitted development data, and correct some common lemma errors by a post-processing script (Table 1).

For senses, we fix some errors according to the ERG knowledge provided by the organizers. ERG, a external knowledge provided by the shared task, contains all legal sense for each lemma and POS tag. Given a joint string of lemma and POS tag, we judge whether our predicted sense is legal, and for the illegal sense, we use the first one in ERG to replace it.

For compound words containing "-", we find that most of them should split into several parts, but limited by our model, we cannot get their lemma, POS tag and sense. Therefore, we use one split word and the predicted POS tag and sense to replace the originally predicted one after we verify sense legality[8].

[8]In dev data, we find that, such split does not work well, but we think, without the replacement, the node is unlikely to be correct.

2.4 AMR

Abstract meaning representation (AMR) (Banarescu et al., 2013) is a semantic formalism to encode the meaning of natural language sentences, which is a broad-coverage sentence-level semantic representation. AMR can be regarded as a rooted labeled directed acyclic graph. We directly follow Lyu and Titov (2018)'s joint modeling of alignments, concepts and relations. In the following, we describe the details of our AMR parser and the modifications we make due to the constrains of the white list of MRP. In general, the training process employs a probability model composed of concept identification, relation identification and alignment, while the testing process only consists of the former two.

Notations. Given a sentence $\mathbf{s} = w_1, w_2, ..., w_n$, where n is the sentence length. Its concepts are defined as $\mathbf{c} = (c_1, c_2, ..., c_m)$, where m is the number of concepts. A relation between c_i and c_j is denoted as $r_ij \in \mathcal{R}$, where $\mathcal{R}$ is the set of all relations. If there is no relation between c_i and c_j, we give them a *NULL* label. We employ $\mathbf{a} = a_1, a_2, ..., a_m$ to denote the concepts, where $a_i \in 1, 2, ..., n$ is the index of a word aligned to c_i. We use $\mathbf{h}_k (k \in 1, 2, ..., l)$ to denote the hidden states of BiLSTM encoders of our model components, where l is the number of the BiLSTM layers.

Concept Identification Model. The concept identification model chooses a concept c conditioned on the aligned word k based on the BiLSTM state $\mathbf{h}_k$, which is defined as $P_\theta(c|\mathbf{h}_k, w_k)$. For more details about the re-categorization and candidate concept, please refer to Lyu and Titov (2018).

Relation Identification Model. The relation identification model is arc-factored as:

$$P_\phi(\mathcal{R}|\mathbf{a}, \mathbf{c}, \mathbf{w}) = \prod_{i,j=1}^{m} P(r_{ij}|\mathbf{h}_{a_i}, \mathbf{c}_i, \mathbf{h}_{a_j}, \mathbf{c}_j) \quad (1)$$

The model employs a log-linear module with bilinear scorer to compute the probabilities of c_i and c_j.

Alignment Model. The alignment model is only used in training, and thus it only depends on the BiLSTM hidden states $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_n$ and the concept list $c_1, c_2, ..., c_m$. Given the concepts list $\mathbf{c}$, the alignment model encodes $\mathbf{c}$ with a BiLSTM encoder, which defines the state of c_i as $\mathbf{g}_i$, $i \in 1, 2, ..., n$. A globally-normalized alignment model is used, which is defined as $Q_\psi(\mathbf{a}|\mathbf{c}, \mathcal{R}, \mathbf{w})$, and the score of the alignment a_i is also computed via a bilinear scorer.

Pre-processing and Post-processing. Since the text format of MRP AMR is different from the original AMR, we need to convert the MRP AMR text to original AMR text, which is same as the input file of the parser (Lyu and Titov, 2018). We utilize Illinois Named Entity Tagger[9] (NER) to generate the NER labels for the AMR data; and we use the Part-of-Speech (PoS) tags and lemmas provided by MRP. After the parser generating the test data output, we convert the AMR form to the MRP form. For details about the pre-processing and post-processing, please refer to the original paper Lyu and Titov (2018) as well.

3 Experiments

This section describes model parameters used in our models, and the overall results of all the five tasks.

3.1 Model Parameters

In both **SDP** and **UCCA** tasks, we use 100-dimensional GloVe (Pennington et al., 2014) as pretrained embedding and random initialized 50-dimensional char embedding. The char lstm output is 100-dimensional. We also utilize the BERT embeddings extracted from the last four transformer layers. The final BERT representation is their normalized weighted sum, which is concatenated with the word embeddings. The other parameters are the same with the previous works (Dozat and Manning, 2018; Jiang et al., 2019).

In **EDS** task, external resources we use are: 1) word embeddings pre-trained with GloVe (Pennington et al., 2014) on the Gigaword corpus for Chinese; and 2) BERT [10] (Devlin et al., 2018), recently proposed effective deep contextualized word representation. We split the provided training data into train/dev/test data, and both dev and test data contain 2500 examples respectively. We evaluate our model on the split data, and before submitting the final result, we train our model on all the data and predict the provided test data as our submission.

In **AMR** task, we randomly choose the samples of the training data according to the proportion of

[9] https://cogcomp.org/page/software_view/NETagger

[10] We generate our pre-trained BERT embedding with the released model in https://github.com/google-research/bert.

	DM			PSD			UCCA			EDS			AMR		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Node prediction															
labels	89	90	89.26	83	85	83.80	#	#	#	91	91	91.20	82	81	81.53
properties	90	91	90.65	83	85	84.43	#	#	#	89	91	89.72	77	73	74.96
anchors	#	#	#	#	#	#	96	94	95.02	95	95	94.86	#	#	#
Edge prediction															
top	91	91	91.01	96	79	86.49	100	100	99.56	90	90	89.94	63	63	62.86
edges	88	90	88.69	74	75	74.41	70	65	67.74	90	90	89.66	64	60	61.78
attributes	#	#	#	#	#	#	54	33	40.80	#	#	#	#	#	#
Overall															
all	90	92	91.26	84	86	84.81	81	76	78.43	92	92	91.85	73	70	71.72

Table 2: Experiment results on the provided test data from the shared task. We divide different evaluation criteria into two types: node-related and edge related, which agrees with our task division.

each domain, and compose them as the development data, which contain 2993 samples. We have also attempted to integrate BERT representations into the basic model input, but it did not bring significant improvements. For the model parameters, we directly use the default settings of Lyu and Titov (2018).

3.2 Experiment Results

Our experiment results on the provided test data are shown in Table 2.

SDP. We randomly selected 20% sentences as our development set and the rest as our training set. After tuning on the development set, we train the parser on the whole dataset from scratch and early stop at the best epoch on the development data. The MRP F1 scores of our DM and PSD deveplopment data are 93.37 and 87.89, respectively. After utilizing BERT embeddings, the results rise to 94.06 and 88.79 respectively. As the improvements are not very significant, we will explore better ways of integrating BERT in the future. We achieve 91.26 and 84.81 F1 scores on PSD and SDP test data regarding to the MRP evaluation metrics, which rank the eighth and the ninth respectively.

UCCA. The training strategy is the same as SDP. The MRP F1 scores on our deveplopment data are 79.80 and 73.41, w/o BERT respectively. Unlike SDP, the result is significantly improved after using BERT embeddings. This is consistent with Jiang et al. (2019). We achieve 78.43 F1 score on the test set which ranks the second.

EDS. For nodes prediction, our models cannot assign the labels well compared to the anchors as shown in Table 2, and properties are even worse. This trend is consistent with our model design and the evaluation strategy, since they predict anchors first and the labels second. Another important reason is that provided lemmas of the words are not always correct for EDS task.

For edge prediction, on our split test/dev data, edge model can achieve much better performance based on the gold nodes than predicted nodes, up to ca. 98%. Based on such high performance, we have not considered constraints like leaf node cannot be pointed by other nodes. Edge prediction performance in Table 2 is not optimal mainlly due to the error propagation from node prediction.

External knowledge including BERT and pretrained word embedding are effective; and postprocessing listed in can achieve about 1% improvement on our split dev/test data. Another interesting observation during our experiments is that the complete match score is much higher than the normal dependency parsing (ca. 75% vs. 30%), although the corresponding LAS can be as high as ca. 92%.

Finally, we obtain 91.85 F1 score on the test data, ranks the first.

AMR. We choose the best model that tuned on the development data to generate AMR graphs for the test data, which achieves 69.9 F1 smatch score on development data. Table 2 shows the results of the AMR test data regarding to the MRP evaluation metrics. We achieve 71.72 F1 score on the test data and it ranks the fifth.

Our overall result on all five tasks ranks third, and our results ranks first on EDS and second on UCCA.

4 Conclusions and Future Work

We participate in the shared task on Cross-Framework Meaning Representation Parsing (MRP) at CoNLL-2019. The shared task combines five frameworks for graph-based meaning

representation, including DM, PSD, EDS, UCCA, and AMR. Considering the common characteristics of the five semantic formalisms, we treat them as two-stage processing using graph-based methods: node prediction (no need for DM and PSD) and edge prediction. For different graphs, we generate nodes and edges in a joint way or in a pipeline way. BERT is also employed to boost the performance (except AMR). Our system ranked the third on the overall evaluation metrics, the first on EDS and the second on UCCA. For the future work, we plan to jointly handle multiple semantic frameworks (e.g., DM, PSD, and UCCA) at the same time via MTL, in order to facilitate mutual benefits and interactions, and make better use of the non-overlapping training data. Moreover, model ensemble may also further enhance the performance.

References

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

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1215–1226, Vancouver, Canada. Association for Computational Linguistics.

Yufei Chen, Weiwei Sun, and Xiaojun Wan. 2018. Accurate SHRG-based semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 408–418, Melbourne, Australia. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

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

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Jan Hajič, Eva Hajičová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fučíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Šindlerová, Jan Štěpánek, Josef Toman, Zdeňka Urešová, and Zdeněk Žabokrtský. 2012. Announcing prague czech-english dependency treebank 2.0. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*.

Angelina Ivanova, Stephan Oepen, Lilja Øvrelid, and Dan Flickinger. 2012. Who did what to whom? a contrastive study of syntacto-semantic dependencies. In *Proceedings of the Sixth Linguistic Annotation Workshop*.

Wei Jiang, Zhenghua Li, Yu Zhang, and Min Zhang. 2019. Hlt@suda at semeval-2019 task 1: Ucca graph parsing as constituent tree parsing. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 11–15.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Chunchuan Lyu and Ivan Titov. 2018. Amr parsing as graph prediction with latent alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 397–407.

Yusuke Miyao, Stephan Oepen, and Daniel Zeman. 2014. In-house: An ensemble of pre-existing off-the-shelf parsers. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*.

Stephan Oepen, Omri Abend, Jan Hajič, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, Jayeol Chun, Milan Straka, and Zdeňka Urešová. 2019. MRP 2019: Cross-framework Meaning Representation Parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 1 – 27, Hong Kong, China.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Zdenka Uresova. 2016. Towards comparability of linguistic graph banks for semantic parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, and Zdeňka Urešová. 2015. SemEval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Yi Zhang. 2014. SemEval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*.

Stephan Oepen and Jan Tore Lønning. 2006. Discriminant-based MRS banking. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy. European Language Resources Association (ELRA).

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543.

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

Zhixing Tan, Mingxuan Wang, Jun Xie, Yidong Chen, and Xiaodong Shi. 2018. Deep semantic role labeling with self-attention. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.