ShanghaiTech at MRP 2019: Sequence-to-Graph Transduction with Second-Order Edge Inference for Cross-Framework Meaning Representation Parsing

Xinyu Wang, Yixian Liu, Zixia Jia, Chengyue Jiang, Kewei Tu
School of Information Science and Technology,
ShanghaiTech University, Shanghai, China
{wangxy1,liuyx,jiazx,jiangchy,tukw}@shanghaitech.edu.cn

Abstract

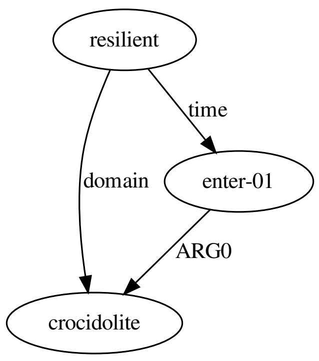
This paper presents the system used in our submission to the *CoNLL 2019 shared task: Cross-Framework Meaning Representation Parsing*. Our system is a graph-based parser which combines an extended pointer-generator network that generates nodes and a second-order mean field variational inference module that predicts edges. Our system achieved 1^{st} and 2^{nd} place for the DM and PSD frameworks respectively on the in-framework ranks and achieved 3^{rd} place for the DM framework on the cross-framework ranks.

1 Introduction

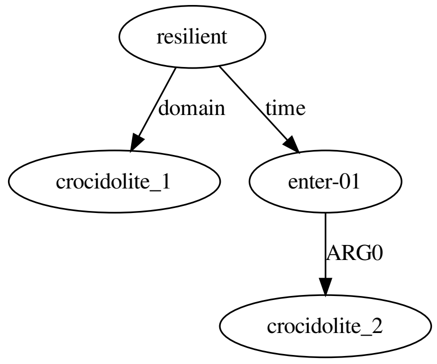
The goal of the *Cross-Framework Meaning Representation Parsing* (MRP 2019, Oepen et al. (2019)) is learning to parse text to multiple formats of meaning representation with a uniform parsing system. The task combines five different frameworks of graph-based meaning representation. DELPH-IN MRS Bi-Lexical Dependencies (DM) (Ivanova et al., 2012) and Prague Semantic Dependencies (PSD) (Hajič et al., 2012; Miyao et al., 2014) first appeared in SemEval 2014 and 2015 shared task Semantic Dependency Parsing (SDP) (Oepen et al., 2014, 2015). Elementary Dependency Structures (EDS) (Oepen and Lønning, 2006) is the origin of DM Bi-Lexical Dependencies, which encodes English Resource Semantics (Flickinger et al., 2016) in a variable-free semantic dependency graph. Universal Conceptual Cognitive Annotation (UCCA) (Abend and Rappoport, 2013) targets a level of semantic granularity that abstracts away from syntactic paraphrases. Abstract Meaning Representation (AMR) (Banarescu et al., 2013) targets to abstract away from syntactic representations, which means that sentences have similar meaning should be assigned the same AMR graph. One of the main differences between these frameworks is their level of abstraction from the sentence. SDP is a bi-lexical dependency graph, where graph nodes correspond to tokens in the sentence. EDS and UCCA are general forms of anchored semantic graphs, in which the nodes are anchored to arbitrary spans of the sentence and the spans can have overlaps. AMR is an unanchored graph, which does not consider the correspondence between nodes and the sentence tokens. The shared task also provides a cross-framework metric which evaluates the similarity of graph components in all frameworks.

Previous work mostly focused on developing parsers that support only one or two frameworks while few work has explored cross-framework semantic parsing. Peng et al. (2017), Stanovsky and Dagan (2018) and Kurita and Søgaard (2019) proposed methods learning jointly on the three frameworks of SDP and Peng et al. (2018) further proposed to learn from different corpora. Hershcovich et al. (2018) converted UCCA, AMR, DM and UD (Universal Dependencies) into a unified DAG format and proposed a transition-based method for UCCA parsing.

In this paper, we present our system for MRP 2019. Our system is a graph-based method which combines an extended pointer-generator network introduced by Zhang et al. (2019) to generate nodes for EDS, UCCA and AMR graphs and a second-order mean field variational inference module introduced by Wang et al. (2019) to predict edges for all the frameworks. According to the official results, our system gets 94.88 F1 score in the cross-framework metric for DM, which is the 3^{rd} place in the ranking. For in-framework metrics, our system gets 92.98 and 81.61 labeled F1 score for DM and PSD respectively, which are ranked 1^{st} and 2^{nd} in the ranking.

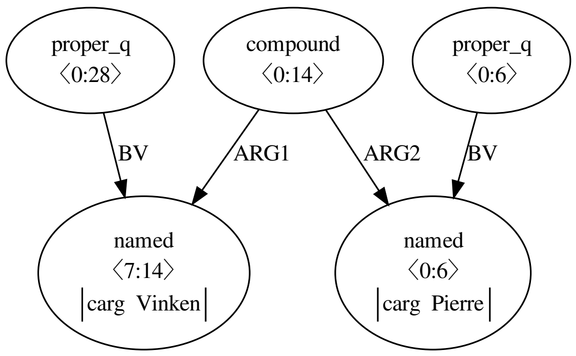


(a) Before conversion.

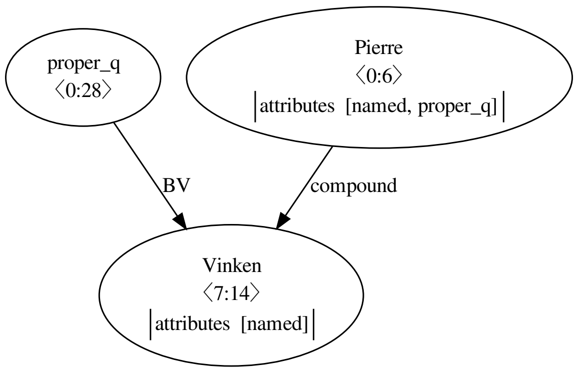


(b) After conversion.

Figure 1: An example of converting AMR graphs into tree structures. This is a sub-graph of sentence *#20003002*.



(a) Before reduction.


proper_q ⟨0:28⟩
Pierre ⟨0:6⟩ |attributes [named, proper_q]|
BV
compound
Vinken ⟨7:14⟩ |attributes [named]|

(b) After reduction.

Figure 2: An example of EDS reduction. This is a sub-graph of sentence *#20001001*.

2 Data Processing

In this section, we introduce our data pre-processing and post-processing in our system for all the frameworks. We use sentence tokenizations, POS tags and lemmas from the official companion data and named entity tags extracted by Illinois Named Entity Tagger (Ratinov and Roth, 2009) in the official 'white-list'. We follow Zhang et al. (2019) to convert each EDS, UCCA, and AMR graph to a tree through duplicating the nodes that have multiple edge entrances, An example is shown in Fig. 1. The node sequences for EDS, UCCA and AMR are decided by depth-first search that starts from the root node and sorts neighbouring nodes in alphanumerical order.

2.1 AMR Data Processing

Our data processing follows Zhang et al. (2019). In pre-processing, we remove the senses, wiki links and polarity attributes in AMR nodes, and replace the sub-graphs of special named entities, such as names, places, time, with anonymized words. The corresponding phrases in the sentences are also anonymized. A mapping from NER tags to these entities is built to process the test data.

In post-processing, we generate the AMR sub-graphs from the anonymized words. Then we assign the senses, wiki links and polarity attributes with the method in Zhang et al. (2019).

2.2 EDS and UCCA Data Processing

In pre-processing we first clean the companion data to make sure the tokens in the companion data is consistent with those in the MRP input. We suppose anchors are continuous for each node, so we replace the anchors with the corresponding start and end token indices.

In EDS graphs, there are a lot of nodes without a direct mapping to individual surface tokens, which we call *type* 1 nodes. We call nodes with corresponding surface tokens *type* 2 nodes. We reduce *type* 1 nodes in two ways:

- If a node a of *type* 1 is connected to only one node b which is of *type* 2 and has the same anchor as a, we reduce node a into node b as a special *attribute* for the node.

- If a node a of *type* 1 is connected to exactly two nodes b and c which are of *type* 2 and have a combined anchor range that matches the anchor of a. We reduce node a as an edge connecting b and c with the same label. The edge direction is decided by the labels of the edges connecting a to b and c. For example, if node a has two child nodes b and c, edge (a, c) has label $ARG2$ and edge (a, b) has label $ARG1$, then node a will be reduced to directed edge (b, c) with the label of node a.

An example of the reduction is shown in Fig. 2. This method reduces 4 nodes on average for each graph. We also look at nodes whose node label corresponds to a multi-word in the sentence For example, '_such+as' in an EDS graph corresponds to 'such as' in the sentence. In such case, if the phrase has a probability over 0.5 that maps to a single node, then all words in this phrase will be combined to a single token in the sentence.

In the post-processing, we recover reduced nodes by reversing the reduction precedure according to the node attributes and edge labels.

For UCCA, we label implicit nodes with special labels n_i, where i is the index that the implicit node appears in the node sequence.

3 System Description

In this section, we describe our model for the task. We first predict the nodes of the parse graph. For DM and PSD, there is a one-to-one mapping between sentence tokens and graph nodes. For EDS, UCCA and AMR, we apply an extended pointer-generator network (Zhang et al., 2019) for node prediction. Given predicted nodes, we then adopt the method of second-order mean field variational inference (Wang et al., 2019) for edge prediction. Figure 3 illustrates our system architecture.

3.1 Word Representation

Previous work found that various word representation could help improve parser performance. Many state-of-the-art parsers use POS tags and pre-trained GloVe (Pennington et al., 2014) embeddings as a part of the word representation. Dozat and Manning (2018) find that character-based LSTM and lemma embeddings can further improve the performance of semantic dependency parser. Zhang et al. (2019) use BERT (Devlin et al., 2019) embeddings for each token to improve the performance of AMR parsing. In our system, we find that predicted named entity tags are helpful as well. The word representation o_i in our system is:

$$\mathbf{o}_i = [\mathbf{o}_i^{w}; \mathbf{o}_i^{\text{pos}}; \mathbf{o}_i^{\text{lemmas}}; \mathbf{o}_i^{pw}; \mathbf{o}_i^{bw}; \mathbf{o}_i^{\text{char}}; \mathbf{o}_i^{\text{ne}}]$$

where $\mathbf{o}_i^{w}$ is word embedding with random initialization, $\mathbf{o}_i^{pw}$ is pre-trained GloVe embedding and $\mathbf{o}_i^{bw}$ are BERT embedding through average pooling over subwords. $\mathbf{o}_i^{\text{pos}}$, $\mathbf{o}_i^{\text{lemmas}}$, $\mathbf{o}_i^{\text{char}}$, $\mathbf{o}_i^{\text{ne}}$ are XPOS, lemmas, character and NER embedding respectively. XPOS and lemmas are extracted from the official companion data.

3.2 Node Prediction

We use extended pointer-generator network (Zhang et al., 2019) for nodes prediction. Given a sentence with n words $\mathbf{w} = [w_1, w_2, ..., w_n]$, we predict a list of nodes $\mathbf{u} = [u_1, u_2, ..., u_m]$ sequentially and assign their corresponding indices $\mathbf{idx} = [idx_1, idx_2, ..., idx_m]$. The indices $\mathbf{idx}$ are used to track whether a copy of a previous generated nodes or a newly generated node.

$$P(\mathbf{u}) = \prod_{i=1}^{m} P(u_i \mid u_{<i}, idx_{<i}, \mathbf{w})$$

To encode the input sentence, we use a multi-layer BiLSTM fed with embeddings of the words:

$$R = \text{BiLSTM}(O) \qquad (1)$$

where O represents $[\mathbf{o}_1, \ldots, \mathbf{o}_n]$, $\mathbf{o}_i$ is the concatenation different types of embeddings for w_i, and $R = [\mathbf{r}_1, \ldots, \mathbf{r}_n]$ represents the output from the BiLSTM.

For the decoder, at each time step t, we use an l-layer LSTM for generating hidden states z_t^l sequentially:

$$\mathbf{z}_t^l = f^l(\mathbf{z}_t^{l-1}, \mathbf{z}_{t-1}^l)$$

where f^l is the l-th layer of LSTM, $\mathbf{z}_0^l$ is the last hidden state r_n in Eq. 1. $\mathbf{z}_t^0$ is the concatenation of the label embedding of node u_{t-1} and attentional vector $\widetilde{\mathbf{z}}_{t-1}$. $\widetilde{\mathbf{z}}_t$ is defined by:

$$\begin{aligned}
\mathbf{e}_{\text{src}}^t &= \mathbf{W}_{satt}^{\top}\text{tanh}(\mathbf{W}_{\text{src}}R + \mathbf{U}_{\text{src}}\mathbf{z}_t^l + \mathbf{b}_{\text{src}}) && (2)\\
\mathbf{a}_{\text{src}}^t &= \text{softmax}(\mathbf{e}_{\text{src}}^t) && (3)\\
\mathbf{c}_t &= \sum_i^n \mathbf{a}_{\text{src,i}}^t \mathbf{r}_i \\
\widetilde{\mathbf{z}}_t &= \text{tanh}(\mathbf{W}_c[\mathbf{c}_t; \mathbf{z}_t^l] + \mathbf{b}_c) && (4)
\end{aligned}$$

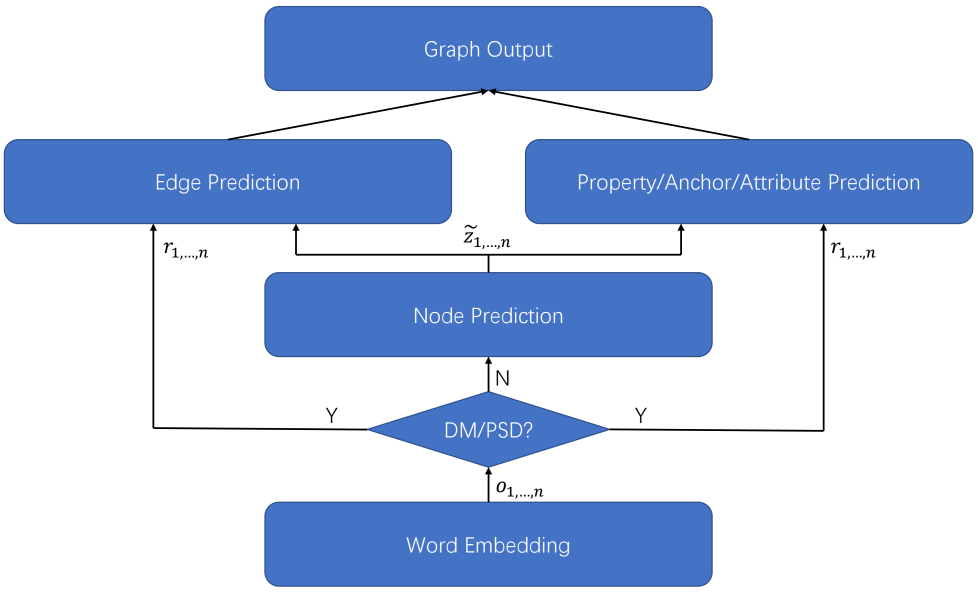


Figure 3: Illustration of our system architecture.

Where $\mathbf{a}_{\text{src}}^{t}$ is the source attention distribution, and $\mathbf{c}_t$ is contextual vector of encoder hidden layers, $\mathbf{W}_{satt}$, $\mathbf{W}_{\text{src}}$, $\mathbf{U}_{\text{src}}$, $\mathbf{b}_{\text{src}}$, $\mathbf{W}_c$, $\mathbf{b}_c$ are learnable parameters. The vocabulary distribution is given by:

$$P_{\text{vocab}} = \text{softmax}(\mathbf{W}_{\text{vocab}}\widetilde{\mathbf{z}}_t + \mathbf{b}_{\text{vocab}}) \qquad (5)$$

where $\mathbf{W}_{\text{vocab}}$ and $\mathbf{b}_{\text{vocab}}$ are learnable parameters. The target attention distribution is defined similarly as Eq. 2 and 3:

$$\begin{aligned}\mathbf{e}_{\text{tgt}}^{t} &= \mathbf{W}_{tatt}^{\top}\tanh(\mathbf{W}_{\text{tgt}}\widetilde{\mathbf{z}}_{1:t-1} + \mathbf{U}_{\text{tgt}}\widetilde{\mathbf{z}}_t + \mathbf{b}_{\text{tgt}}),\\ \mathbf{a}_{\text{tgt}}^{t} &= \text{softmax}(\mathbf{e}_{\text{tgt}}^{t}),\end{aligned}$$

where $\mathbf{W}_{tatt}^{\top}$, $\mathbf{W}_{\text{tgt}}$, $\mathbf{U}_{\text{tgt}}$, $\mathbf{b}_{\text{tgt}}$ are learnable parameters. Finally, at each time step, we need to decide which action should be taken. Possible actions include copying an existing node from previous nodes and generating a new node whose label is either from the vocabulary or a word from the source sentence. The corresponding probability of these three actions are p_{tgt}, p_{gen} and p_{src}:

$$[p_{\text{tgt}}, p_{\text{gen}}, p_{\text{src}}] = \text{softmax}(\mathbf{W}_{\text{action}}\widetilde{\mathbf{z}}_t + \mathbf{b}_{\text{action}})$$

where $p_{\text{tgt}} + p_{\text{gen}} + p_{\text{src}} = 1$.

At time step t, if u_t is a copy of an existing nodes, then the probability $P^{(\text{node})}(u_t)$ and the index idx_t is defined by:

$$P^{(\text{node})}(u_t) = p_{\text{tgt}} \sum_{i:u_i=u_t} \mathbf{a}_{\text{tgt}}^{t}[i]$$
$$idx_t = idx_j$$

where idx_j is the copied node index. If u_t is a new node:

$$P^{(\text{node})}(u_t) = p_{\text{gen}}P_{\text{vocab}}(u_t) + p_{\text{src}} \sum_{i:w_i=u_t} \mathbf{a}_{\text{src}}^{t}[i]$$
$$idx_t = t$$

3.3 Edge Prediction

We adopt the method presented in Wang et al. (2019) for edge prediction, which is based on second-order scoring and inference. Suppose that we have a sequence of vector representations of the predicted nodes $[\mathbf{r}'_1, \ldots, \mathbf{r}'_m]$, which can be the BiLSTM output $\mathbf{r}_i$ in Eq. 1 in the cases of DM and PSD, or the extended pointer-generator network output $\widetilde{\mathbf{z}}_i$ in Eq. 4 in the cases of EDS, UCCA and AMR. The edge prediction module is shown in Fig. 4.

To score first-order and second-order parts (i.e., edges and edge-pairs) in both edge-prediction and label-prediction, we apply the Biaffine function (Dozat and Manning, 2017, 2018) and Trilinear function (Wang et al., 2019) fed with node representations.

$$\begin{aligned}\text{Biaff}(\mathbf{v}_1, \mathbf{v}_2) &:= \mathbf{v}_1^{\top}\mathbf{U}\mathbf{v}_2 + \mathbf{b}\\ \mathbf{g}_i &:= \mathbf{U}_i\mathbf{v}_i \qquad i \in [1, 2, 3]\\ \text{Trilin}(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3) &:= \sum_{i=1}^{d} \mathbf{g}_{1i} \circ \mathbf{g}_{2i} \circ \mathbf{g}_{3i} \qquad (6)\end{aligned}$$

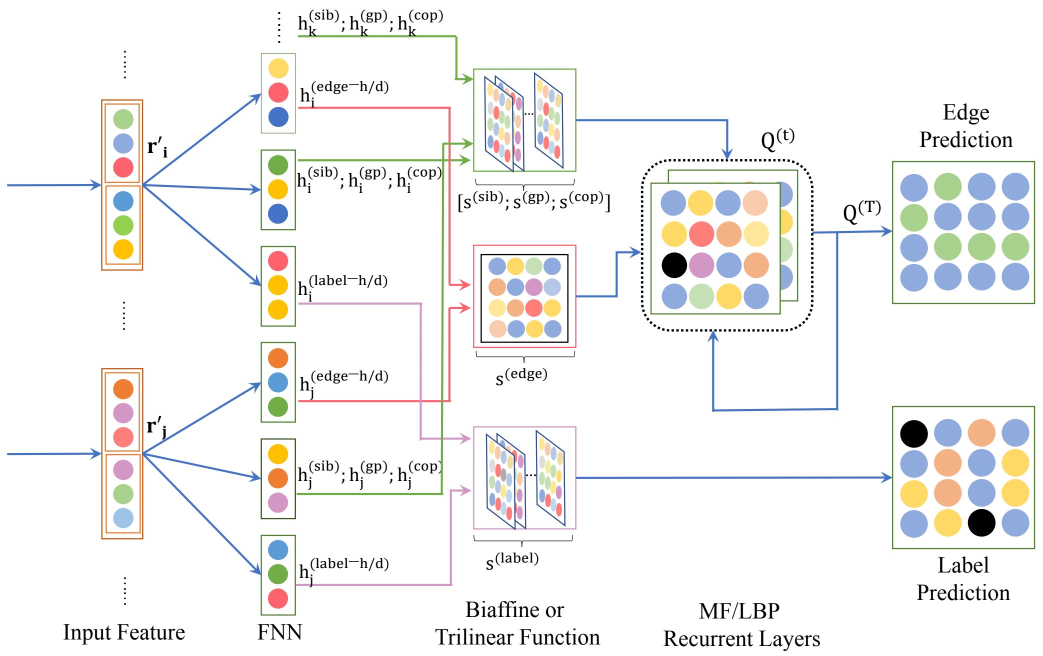


Figure 4: The structure of our edge prediction module. The figure is from Wang et al. (2019) with minor modifications.

where $\mathbf{U}_i$ is a $(d \times d)$-dimensional tensor, where d is hidden size and $\circ$ represents element-wise product. We consider three types of second-order parts: siblings (sib), co-parents (cop) and grandparents (gp) (Martins and Almeida, 2014). For a specific first-order and second-order *part*, we use single-layer FNNs to compute a *head* representation and a *dependent* representation for each word, as well as a *head_dep* representation which is used for grandparent parts:

$$\text{part} \in \{\text{edge}, \text{label}, \text{sib}, \text{cop}, \text{gp}\}$$
$$\mathbf{h}_i^{(\text{part-head})} = \text{FNN}^{(\text{part-head})}(\mathbf{r}'_i)$$
$$\mathbf{h}_i^{(\text{part-dep})} = \text{FNN}^{(\text{part-dep})}(\mathbf{r}'_i)$$
$$\mathbf{h}_i^{(\text{gp-head\_dep})} = \text{FNN}^{(\text{gp-head\_dep})}(\mathbf{r}'_i)$$

We then compute the part scores as follows:

$$s_{ij}^{(\text{edge})} = \text{Biaff}^{(\text{edge})}(\mathbf{h}_i^{(\text{edge-dep})}, \mathbf{h}_j^{(\text{edge-head})}) \quad (7)$$
$$\mathbf{s}_{ij}^{(\text{label})} = \text{Biaff}^{(\text{label})}(\mathbf{h}_i^{(\text{label-dep})}, \mathbf{h}_j^{(\text{label-head})}) \quad (8)$$
$$s_{ij,ik}^{(sib)} \equiv s_{ik,ij}^{(sib)} = \text{Trilin}^{(\text{sib})}(\mathbf{h}_i^{(\text{head})}, \mathbf{h}_j^{(\text{dep})}, \mathbf{h}_k^{(\text{dep})}) \quad (9)$$
$$s_{ij,kj}^{(cop)} \equiv s_{kj,ij}^{(cop)} = \text{Trilin}^{(\text{cop})}(\mathbf{h}_i^{(\text{head})}, \mathbf{h}_j^{(\text{dep})}, \mathbf{h}_k^{(\text{head})}) \quad (10)$$
$$s_{ij,jk}^{(gp)} = \text{Trilin}^{(\text{gp})}(\mathbf{h}_i^{(\text{head})}, \mathbf{h}_j^{(\text{head\_dep})}, \mathbf{h}_k^{(\text{dep})}) \quad (11)$$

In Eq. 7,8, the tensor $\mathbf{U}$ in the biaffine function is $(d \times 1 \times d)$-dimensional and $(d \times c)$-dimensional, where c is the number of labels. We require $j < k$ in Eq. 9 and $i < k$ in Eq. 10.

In the label-prediction module, $\mathbf{s}_{i,j}^{(\text{label})}$ is fed into a softmax layer that outputs the probability of each label for edge (i, j). In the edge-prediction module, we can view computing the edge probabilities as doing posterior inference on a Conditional Random Field (CRF). Each Boolean variable X_{ij} in the CRF indicates whether the directed edge (i, j) exists. We use Eq. 7 to define our unary potential ψ_u representing scores of an edge and Eqs. (9-11) to define our binary potential ψ_p. We define a unary potential $\phi_u(X_{ij})$ for each variable X_{ij}.

$$\phi_u(X_{ij}) = \begin{cases} \exp(s_{ij}^{(\text{edge})}) & X_{ij} = 1 \\ 1 & X_{ij} = 0 \end{cases}$$

For each pair of edges (i, j) and (k, l) that form a second-order part of a specific *type*, we define a binary potential $\phi_p(X_{ij}, X_{kl})$.

$$\phi_p(X_{ij}, X_{kl}) = \begin{cases} \exp(s_{ij,kl}^{(type)}) & X_{ij} = X_{kl} = 1 \\ 1 & \text{Otherwise} \end{cases}$$

Exact inference on this CRF is intractable. We use mean field variational inference to approximate a true posterior distribution with a factorized variational distribution and tries to iteratively minimize their KL divergence. We can derive the following iterative update equations of distribution

$Q_{ij}(X_{ij})$ for each edge (i, j).

$$\begin{aligned}\mathcal{F}_{ij}^{(t-1)} = \sum_{k \neq i,j} Q_{ik}^{(t-1)}(1) s_{ij,ik}^{(sib)} + Q_{kj}^{(t-1)}(1) s_{ij,kj}^{(cop)} \\ + Q_{jk}^{(t-1)}(1) s_{ij,jk}^{(gp)} + Q_{ki}^{(t-1)}(1) s_{ki,ij}^{(gp)} \\ Q_{ij}^{(t)}(0) \propto 1 \\ Q_{ij}^{(t)}(1) \propto \exp\{s_{ij}^{(\text{edge})} + \mathcal{F}_{ij}^{(t-1)}\}\end{aligned} \tag{12}$$

The initial distribution $Q_{ij}^{(0)}(X_{ij})$ is set by normalizing the unary potential $\phi_u(X_{ij})$. We iteratively update the distributions for T steps and then output $Q_{ij}^{(T)}(X_{ij})$, where T is a hyperparameter. We can then predict the parse graph by including every edge $y_{ij}^{(\text{edge})}$ such that $Q_{ij}^{(T)}(1) > 0.5$. The edge labels $y_{ij}^{(\text{label})}$ are predicted by maximizing the label probabilities computed by the label-prediction module.

$$\begin{aligned}P(y_{ij}^{(\text{edge})}|\mathbf{w}) = \text{softmax}(Q_{ij}^{(T)}(X_{ij})) \\ P(y_{ij}^{(\text{label})}|\mathbf{w}) = \text{softmax}(\mathbf{s}_{ij}^{(\text{label})})\end{aligned}$$

Note that the iterative updates in mean-field variational inference can be seen as a recurrent neural network that is parameterized by the potential functions. Therefore, the whole edge prediction module can be seen as an end-to-end neural network.

3.4 Other Predictions

The shared task also requires prediction of component pieces such as top nodes, node properties, node anchoring and edge attributes. In this section, we present our approaches to predicting these components.

Top Nodes

We add an extra *ROOT* node for each sentence to determine the top node through edge prediction for DM and PSD. For the other frameworks, we use the first predicted node as the top node.

Node Properties

Node properties vary among different frameworks. For DM and PSD, we need to predict the POS and frame for each node. As DM and PSD are bilexical semantic graphs, we directly use the prediction of XPOS from the official companion data. We use a single layer MLP fed with word features obtained in Eq. 1 for frame prediction. For EDS, the properties only contain 'carg' and the corresponding values are related to the surface string. For example, the EDS sub-graph in Fig. 2 contains a node with label 'named' which has property 'carg' with a corresponding value 'Pierre'. The anchor of this node matches the token 'Pierre' in the sentence. We found that nodes with properties have limited types of node labels. Therefore, we exchange node labels and values for EDS nodes containing properties during training. We combine the node *attributes* and value predictions described in Section 2.2 together as a multi-label prediction task. We use a single layer MLP to predict node labels specially for nodes with properties. For each property value, we regard it as a node label and use the extended pointer-generator network described in Section 3.2 to predict it. Therefore, the probability of node property prediction is:

$$P_{prop} = \text{softmax}(\mathbf{W}_{\text{prop}}\tilde{\mathbf{r}}_t' + \mathbf{b}_{\text{prop}}) \tag{13}$$

Node Anchoring

As DM and PSD contain only token level dependencies, we can decide a node anchor by the corresponding token. For the other frameworks, we use two biaffine functions to predict the 'start token' and 'end token' for each node and the final anchor range is decided by the start position of 'start token' and the end position of 'end token'. The biaffine function is fed by word features from the encoder RNN and node features from decoder RNN.

$$\begin{aligned}s_{ij}^{(\text{start/end})} = \text{Biaff}^{(\text{start/end})}(\mathbf{r}_i, \tilde{\mathbf{z}}_j) \\ P_{\text{start/end},j} = \text{softmax}([s_{1j}, s_{2j}, \ldots, s_{nj}])\end{aligned} \tag{14}$$

where i ranges from 1 to n and j ranges from 1 to m.

Edge Attributes

Only UCCA requires prediction of edge attributes, which are the 'remote' attributes of edges. We create new edge labels by combining the original edge labels and edge attributes. In this way, edge attribute prediction is done by edge label prediction.

3.5 Learning

Given a gold graph $y^\star$, we use the cross entropy loss as learning objective:

$$\begin{aligned}\mathcal{L}^{(\text{edge})}(\theta) = -\sum_{i,j} \log(P_\theta(y_{ij}^{\star(\text{edge})}|\mathbf{w})) \\ \mathcal{L}^{(\text{label})}(\theta) = -\sum_{i,j} \mathbb{1}(y_{ij}^{\star(\text{edge})}) \log(P_\theta(y_{ij}^{\star(\text{label})}|\mathbf{w}))\end{aligned}$$

	DM	PSD	EDS	UCCA	AMR
Ours-all	94.88	89.49	86.90	-	63.59
Best-all	**95.50**	**91.28**	**94.47**	**81.67**	**73.38**
Ours-lpps	94.28	85.22	87.49	-	66.82
Best-lpps	**94.96**	**88.46**	**92.82**	**82.61**	**73.11**

Table 1: Comparison of cross-framework F1 scores achieved by our system and best scores of other teams for each metric. *all* represents the F1 score over the full test set for each framework. *lpps* represents a 100-sentence sample from the little prince containing graphs over all the frameworks.

$$\mathcal{L}^{(\text{prop})}(\theta) = -\sum_{i,k} \log(P_\theta(y_{ik}^{\star(\text{prop})}|\mathbf{w}))$$

$$\mathcal{L}^{(\text{anchor})}(\theta) = -\sum_{i} \sum_{j\in\{\text{start,end}\}} (\log(P_\theta(y_i^{\star(\text{j})}|\mathbf{w}))$$

where θ is all the parameters of the model, $\mathbb{1}(\mathcal{X})$ is an indicator function of whether $\mathcal{X}$ exists in the graph, i, j range over all the nodes and k ranges over all possible *attributes* in the graph. The total loss is defined by:

$$\begin{aligned}\mathcal{L} =& \lambda_1\mathcal{L}^{(\text{edge})} + \lambda_2\mathcal{L}^{(\text{label})} + \mathbb{1}(y^{\star(\text{prop})})\lambda_3\mathcal{L}^{(\text{prop})} \\ &+ \mathbb{1}(y^{\star(\text{anchor})})\lambda_4\mathcal{L}^{(\text{anchor})}\end{aligned}$$

where $\lambda_{1,\dots,4}$ are hyperparameters. For DM and PSD, we tuned on λ_1, λ_2 and λ_3. For other frameworks, we set all of them to be 1.

4 Experiments and Results

4.1 Training

For DM, PSD and EDS, we used the same dataset split as previous approaches (Martins and Almeida, 2014; Du et al., 2015) with 33,964 sentence in the training set and 1,692 sentences in the development set. For each of the other frameworks, we randomly chose 5% to 10% of the training set as the development set. We additionally removed graphs with more than 60 nodes (or with input sentences longer than 60 words for DM and PSD). We trained our model for each framework separately and used Adam (Kingma and Ba, 2015) to optimize our system, annealing the learning rate by 0.5 for 10,000 steps. We trained the model for 100,000 iterations with a batch size of 6,000 tokens and terminated with 10,000 iterations without improvement on the development set.

4.2 Main Results

Due to an unexpected bug in UCCA anchor prediction, we failed to submit our UCCA prediction. Our results are still competitive to those of the other teams and we get the 3rd place for the DM framework in the official metrics. The main result is shown in Table 1. Our system performs well on the DM framework with an F1 score only 0.4 percent F1 below the best score on DM. Note that our system does not learn to predict node labels for DM and PSD and simply uses lemmas from the companion data as node labels. We find that compared to gold lemmas from the original SDP dataset, lemmas from the companion data have only 71.4% accuracy. We believe that it is the main reason for the F1 score gap between our system and the best one on DM and PSD. A detailed comparison between each component will be discussed in Section 4.3. For PSD, EDS and AMR graph, our system ranks 6th, 5th and 7th among 13 teams.

4.3 Analysis

DM and PSD

Table 2 and 3 show detailed comparison for each evaluation component for DM and PSD. For DM, our system outperforms systems of the other teams on tops, properties and edges prediction and is competitive on anchors. For PSD, our system is also competitive on all the components except labels. There is a large gap in the performance of node label prediction between our system and the best one on both DM and PSD, we believe adding an MLP layer for label prediction would diminish this gap.

Table 4 shows the performance comparison on in-framework metrics for DM and PSD. For DM, our system outperforms the best of the other systems by 0.5 and 0.8 F1 scores on *all* and *lpps* test sets. For PSD, our system outperforms the best of the other systems by 0.4 F1 score for *lpps* and only 0.05 F1 score below the best score for *all*.

AMR

For AMR graph prediction, our node prediction module is based on Zhang et al. (2019), but our edge prediction module is based on the second-order method of Wang et al. (2019). To verify the effectiveness of second-order edge prediction, we compare the performances on the development set of our model and Zhang et al. (2019). The result is shown in Table 5. The result shows that our second-order edge prediction is useful not only on the SDP frameworks but also on the AMR framework.

	tops	labels	properties	anchors	edges	average
Ours-all	**93.68**	90.51	**95.16**	98.38	**92.32**	94.32
Best-all	93.23	**96.34**	94.93	**98.74**	92.08	**94.76**
Ours-lpps	**99.00**	87.26	**94.53**	**99.36**	**93.92**	94.03
Best-lpps	96.48	**94.82**	94.36	99.04	93.28	**94.64**

Table 2: Comparison of cross-framework F1 scores achieved by our system and best scores of the other teams for each evaluation component on DM. *average* is the micro-average among all components.

	tops	labels	properties	anchors	edges	average
Ours-all	95.68	84.79	91.83	97.66	**79.50**	88.77
Best-all	**95.83**	**94.68**	**92.38**	**98.35**	79.44	**90.76**
Ours-lpps	96.00	76.72	84.73	97.61	**79.80**	85.22
Best-lpps	**96.40**	**92.04**	**86.00**	**98.46**	79.18	**88.40**

Table 3: Comparison of cross-framework F1 scores achieved by our system and best scores of the other teams for each evaluation component on PSD.

	DM		PSD		Avg	
	all	lpps	all	lpps	all	lpps
Ours	**92.98**	**94.46**	81.61	**81.91**	**87.30**	**88.19**
Best	92.52	93.68	**81.66**	81.47	87.09	87.58

Table 4: Comparison of in-framework labeled F1 scores by our system and best scores over the other teams. Note that the *Best* scores are not only from a single system.

Model	Smatch
Zhang et al. (2019)	69.1
Ours	69.3

Table 5: Smatch F1 score on AMR development set. We compare the results without post-processing.

Set	MRP	Smatch
test	63.59	63.08
dev	72.03	71.55

Table 6: MRP and Smatch score on the development set and the test set.

From the official results on the test sets, we find it surprising that there is a huge gap between the test and development results on both the MRP and the Smatch (Cai and Knight, 2013) scores, as shown in Table 6. In future work, we will figure out the reason behind this problem.

EDS

For EDS, our parser ranks 5th. There are multiple details of our parser that can be improved. For example, our anchor prediction module described in Eq. 14 (ranking 4th in the task) may occasionally predict an end anchor positioned before a start anchor, which would be rejected by the evaluation system. This can be fixed by adding constraints.

UCCA

For UCCA, we failed to submit the result because of the same reversed start-end anchor predictions, which prevents us from obtaining an MRP score.

4.4 Ablation Study

BERT with Other Embeddings

We use BERT (Devlin et al., 2019) embedding in our model. We compared the performance of DM in the original SDP dataset with different subtoken pooling methods, and we also explored whether combining other embeddings such as pre-trained word embedding Glove (Pennington et al., 2014) and contextual embedding ELMo (Peters et al., 2018) will further improve the performance. The detailed results are shown in table 7. We found that Glove, lemma and character embeddings are helpful for DM and fine-tuning on the training set slightly improves the performance. ELMo embedding is also helpful but cannot outperform BERT embedding. However, the performance dropped when ELMo embedding and BERT embedding are combined. We speculate that the drop is caused by the conflict between the two types of contextual information. For subtoken pooling, we compared the performance of using first subtoken pooling and average pooling as token embedding. We found that average pooling is slightly better than

	LF1
Baseline	93.41
Base-fixed	94.17
Base-tuned	94.22
Base-fixed + Glove	94.45
Base-tuned + Glove	94.48
Large-fixed + Glove	94.62
Large-tuned + Glove	94.64
Large-fixed + Glove + Lemma	95.10
Large-fixed + Glove + Lemma + Char	95.22
ELMo + Large-fixed + Glove + Lemma	94.78
ELMo + Glove + Lemma + Char	95.06
BERT-First	95.22
BERT-Avg	95.28
BERT-Avg + dep-tree	95.30

Table 7: Comparing Labeled F1 scores of models with different types of embedding combinations on the development set of the gold DM dataset. *Baseline* represents the parser of Wang et al. (2019). *Base* represents the pre-trained BERT-Base uncased model and *Large* represents the pre-trained BERT-Large uncased model. *fixed* and *tuned* represents whether to fine-tune the BERT model. *BERT* in the last block represents the last embedding combination (Large-fixed + Glove + Lemma + Char) in the first block. *First* represents first subtoken pooling, *Avg* represents average pooling over subtokens. *dep-tree* represents adding dependency information into embeddings. For each case, we report the highest Labeled F1 score on the development set in our experiments.

	DM	PSD
basic	96.01	90.80
+lemma	96.09	90.79
+ner	96.07	90.80
+lemma & ner	**96.16**	**90.88**

Table 8: F1 score averaged over the labeled F1 score and the frame F1 score on the development sets of DM and PSD. *basic* represents our model with embeddings described in 3.1 except lemma and named entity embeddings.

first pooling. For syntactic information, we encode each head word and dependency label as embeddings and concatenate them together with other embeddings. The result shows that syntactic information as embeddings is not very helpful for the task. We will try other methods utilizing syntactic information in future work.

Lemma and Named Entity Tags

Dozat and Manning (2018) found that gold lemma embedding is helpful for semantic dependency parsing. However, in section 4.2, we note that the lemmas from the official companion data have only 71.4% accuracy compared to lemmas in gold SDP data, which makes lemma embeddings less helpful for parsing. We found that one of the difference is about the lemma annotations of entities, for example, lemmas of "Pierre Vinken" are "Pierre" and "Vinken" in the companion data while the lemmas are named-entity-like tags "Pierre" and "_generic_proper_ne" in the original SDP dataset. Based on this discovery, we experimented on the influence of named entity tags on parsing performance. We used Illinois Named Entity Tagger (Ratinov and Roth, 2009) in white list to predict named entity tags and compared the performance on the development sets of DM and PSD. The result is shown in table 8. We tuned the hyperparameters for all the embedding conditions in the table, and we found that adding lemma or named entity embeddings results in a slight improvement on DM but does not help on PSD. With both lemma and named entity embeddings, there is a further improvement on both DM and PSD, which shows the named entity tags are helpful for semantic dependency parsing. As a result, we apply named entity information in parsing other frameworks.

5 Conclusion

In this paper, we present our graph-based parsing system for MRP 2019, which combines two state-of-the-art methods for sequence to graph node generation and second-order edge inference. The result shows that our system performs well on the DM and PSD frameworks and achieves the best scores on the in-framework metrics. For future work, we will improve our system to achieve better performance on all these frameworks and explore cross-framework multi-task learning. Our code for DM and PSD is available at `https://github.com/wangxinyu0922/Second_Order_SDP`.

References

Omri Abend and Ari Rappoport. 2013. Universal conceptual cognitive annotation (UCCA). In *Proceedings of the 51st Annual Meeting of the Association* *for Computational Linguistics (Volume 1: Long Papers)*, pages 228–238, Sofia, Bulgaria. Association for Computational Linguistics.

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, Sofia, Bulgaria. Association for Computational Linguistics.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752, Sofia, Bulgaria. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D Manning. 2017. Deep biaffine attention for neural dependency parsing. In *International Conference on Learning Representations*.

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)*, pages 484–490, Melbourne, Australia. Association for Computational Linguistics.

Yantao Du, Fan Zhang, Xun Zhang, Weiwei Sun, and Xiaojun Wan. 2015. Peking: Building semantic dependency graphs with a hybrid parser. In *Proceedings of the 9th international workshop on semantic evaluation (semeval 2015)*, pages 927–931.

Dan Flickinger, Emily M. Bender, and Woodley Packard. 2016. English resource semantics. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorial Abstracts*, pages 1–5, San Diego, California. Association for Computational Linguistics.

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 Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 3153–3160, Istanbul, Turkey. European Languages Resources Association (ELRA).

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018. Multitask parsing across semantic representations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 373–385, Melbourne, Australia. Association for Computational Linguistics.

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 6th Linguistic Annotation Workshop*, pages 2 – 11, Jeju, Republic of Korea.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Shuhei Kurita and Anders Søgaard. 2019. Multi-task semantic dependency parsing with policy gradient for learning easy-first strategies. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2420–2430, Florence, Italy. Association for Computational Linguistics.

André FT Martins and Mariana SC Almeida. 2014. Priberam: A turbo semantic parser with second order features. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 471–476.

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)*, pages 335–340, Dublin, Ireland. Association for Computational Linguistics.

Stephan Oepen, Omri Abend, Jan Hajič, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, and Milan Straka. 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 – 20, Hong Kong, China.

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)*, pages 915–926, Denver, Colorado. Association for Computational Linguistics.

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), pages 63–72, Dublin, Ireland. Association for Computational Linguistics.

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).

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep multitask learning for semantic dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2037–2048, Vancouver, Canada. Association for Computational Linguistics.

Hao Peng, Sam Thomson, Swabha Swayamdipta, and Noah A. Smith. 2018. Learning joint semantic parsers from disjoint data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1492–1502, New Orleans, Louisiana. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Lev Ratinov and Dan Roth. 2009. Design Challenges and Misconceptions in Named Entity Recognition. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*.

Gabriel Stanovsky and Ido Dagan. 2018. Semantics as a foreign language. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2412–2421, Brussels, Belgium. Association for Computational Linguistics.

Xinyu Wang, Jingxian Huang, and Kewei Tu. 2019. Second-order semantic dependency parsing with end-to-end neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4609–4618, Florence, Italy. Association for Computational Linguistics.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. AMR Parsing as Sequence-to-Graph Transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Florence, Italy. Association for Computational Linguistics.