Incorporating EDS Graph for AMR Parsing

Ziyi SHOU and **Fangzhen LIN**
Department of Computer Science
The Hong Kong University of Science and Technology
{zshou,flin}@cse.ust.hk

Abstract

AMR (Abstract Meaning Representation) and EDS (Elementary Dependency Structures) are two popular meaning representations in NLP/NLU. AMR is more abstract and conceptual, while EDS is more low level, closer to the lexical structures of the given sentences. It is thus not surprising that EDS parsing is easier than AMR parsing. In this work, we consider using information from EDS parsing to help improve the performance of AMR parsing. We adopt a transition-based parser and propose to add EDS graphs as additional semantic features using a graph encoder composed of LSTM layer and GCN layer. Our experimental results show that the additional information from EDS parsing indeed gives a boost to the performance of the base AMR parser used in our experiments.

1 Introduction

Semantic parsing has long been considered a difficult task and an important step to natural language understanding. A number of meaning representation formalisms have been proposed. Well-known ones include EDS (Elementary Dependency Structures; Oepen and Lønning, 2006), UCCA (Universal Conceptual Cognitive Annotation; Abend and Rappoport, 2013), and AMR (Abstract Meaning Representation; Banarescu et al., 2013). Among them, AMR is more abstract from surface tokens and tries to capture the meaning of a sentence using concepts that may not appear in the sentence. If one views an AMR encoding as a graph, the AMR graph is always composed of fewer nodes than other meaning representations and some nodes in the AMR graph cannot be anchored to tokens or strings of tokens in the sentence. But EDS tries to build a meaning representation using lexical terms that are presented in the sentence, and nodes in their parse trees are anchored. In comparison, AMR has a much more fine-grained classification for the named entities, total of 124 entity types (Lin and Xue, 2019). Thus not surprisingly, AMR parsers do not perform as well as the ones for EDS. Currently the parsing accuracies for AMR are in low 80s, while they can be high 90s for EDS. In this paper, we propose to use EDS improve the performance of the AMR parser.

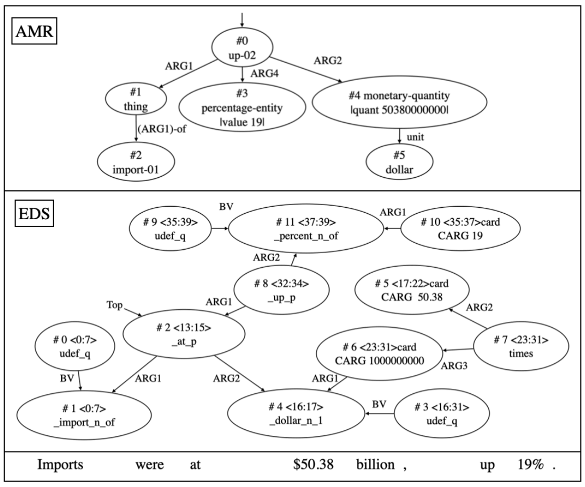

Figure 1: AMR and EDS graph for "Imports were at $50.38 billion, up 19%.", #20011008 sentence from the WSJ Corpus, Penn Treebank (Marcus et al., 1993). Take node #3 in AMR as an example. "percentage-entity" is the node label, "value" is the property of this node, and "19" is the specific value. For node #10 in EDS, "<35:37>" indicates the span of the corresponding surface string; "card" is the node label, "CARG" which means "constant argument" is the property, and "19" is the value.

To see how information from EDS parsing can be of use to AMR, consider the following sentence "Imports were at $50.38 billion, up 19%." from the Wall Street Journal Corpus, Penn Treebank (Marcus et al., 1993). Its graph encodings in AMR and EDS are shown in Figure 1. We mentioned that AMR is more abstract. This can be seen in the example as the graph for AMR is a lot smaller, and the nodes are labeled with conceptual entities. Nevertheless, EDS and AMR edges are labeled using the same semantic roles (e.g., ARG1, ARG2), indicating the relationship between a predicate and its arguments (Lin and Xue, 2019). In this example, there are some correspondences between their nodes. For example, the AMR nodes *"percentage-entity"*, *"dollar"*, and *"import-01"* correspond to the EDS nodes *"_percen_n_of"*, *"_dollar_n_1"*, and *"_import_n_of"*, respectively. In our task, the most important feature of EDS is anchoring. From the EDS graph, each node has a corresponding span of text. Conversely, we can find all related EDS nodes for each token based on the indexes. This suggests that EDS parsing may serve as an intermediate to AMR parsing, which motivated this work.

To incorporate EDS parsing into an AMR parser, we propose an EDS encoder composed of LSTM networks that capture the contextual information and a Graph Convolutional Network (GCN, Kipf and Welling, 2017) that extracts the structure knowledge. We feed EDS into our proposed encoder and produce token-level features. These EDS token-level features are concatenated to word embedding of tokens and participate in the AMR parsing process. To demonstrate the effectiveness of our approach, we use the AMR dataset from MRP 2019 (Oepen et al., 2019) and take as our baseline model the HIT-SCIR (Che et al., 2019), which was the best overall system at MRP 2019 and the 2nd best for AMR. Our experimental results show that our EDS-enhanced parsers clearly outperform the baseline model. In fact, some of our new models beat the best score of the official submitted AMR parsers in this benchmark. We also observed that the biggest improvements happened to be on those test data that are least similar to the training data.

The rest of this paper is organized as follows: Second 2 gives a brief overview on AMR parsers; Section 3 is concerned with the baseline system we adopt and our EDS-enhanced model. We present experimental settings and experimental results in Section 4 and conclude in Section 5.

2 Related Work

We classify AMR parsing systems into grammar-based, graph-based, and transition-based ones. The grammar-based ones generate AMR graphs directly from grammar trees. Several early AMR parsing systems were of this type. For example, Artzi et al. (2015) used combinatory categorial grammar (CCG) parsing to construct AMR, while Peng et al. (2015) made use of synchronous Hyperedge Replacement Grammar (SHRG). Generally speaking, grammar-based ones suffer from information loss during the processes of both grammar tree generation and AMR conversion. They predated the current deep learning approaches.

Modern AMR parsers use deep learning methods. Depending on how the eventual AMR graphs are generated, we can divide them into graph-based and transition-based. Both approaches are popular and their performances are competitive. Briefly, a graph-based system splits AMR parsing into two tasks, concept identification and edge prediction, and then combines them to generate a final AMR graph. The idea seems to appear first in Flanigan et al. (2014), and is used in Lyu and Titov (2018); Zhang et al. (2019a); Cai and Lam (2020); Zhou et al. (2020). A transition-based system, however, uses a sequence of transition actions to construct the graph incrementally. We can include the systems in Wang et al. (2015); Ballesteros and Al-Onaizan (2017); Naseem et al. (2019); Che et al. (2019); Astudillo et al. (2020) in this category.

As we mentioned, our work is about incorporating EDS information into AMR parsing. We note that Brandt et al. (2016) considered adding preposition semantic role labeling to an AMR parser but found that the extra information did not seem to help. Hershcovich and Arviv (2019) used a multi-task learning model but found multi-task TUPA consistently falls behind the single-task one for AMR. Arviv et al. (2020) used multi-task learning on EDS and UCCA parsing, however, EDS didn't bring any benefits to UCCA parsing. Adding extra semantic information like EDS is not easy. It matters how EDS graphs are encoded and incorporated into AMR parsing. We conduct our work with the AMR dataset from MRP 2019, and pick one of the best performing systems there, HIT-SCIR (Che et al., 2019), as our baseline model. Our experimental results show that adding EDS information can indeed give a significant boost to the baseline model. We believe our method is general and can be applied to other AMR parsing systems.

3 Model

3.1 Baseline: A Transition-based Parser

Our baseline model is a transition-based system HIT-SCIR (Che et al., 2019). However, in our experiments, we use BERT-base instead of BERT-large for word embeddings (Devlin et al., 2019) due to our constraints on computing resources. Nevertheless, when the BERT-base baseline model is enhanced with EDS information, it still outperforms the best AMR parser at MRP 2019.

Task Formalization The main task of a transition-based model is to generate a sequence of actions to construct an AMR graph. The sequence of actions is predicted one at a time, and the graph is also constructed incrementally.

A state in HIT-SCIR is a tuple (S, L, B, E, V), where S is a stack holding processed words, L is a list holding tokens popped out of S that will be pushed back in the future, and B is a buffer holding tokens waiting to be processed. E is the sets of labeled dependency edges and V is a set of graph nodes include concept nodes and surface tokens. The initial state of AMR parser was $([0], [\ ], [w_1, \dots, w_n], [\ ], V)$, where V only contains surface tokens. During parsing, each token should be parsed individually, and AMR nodes and edges would be generated through the selection of actions. The final state should be $([0], [\ ], [\ ], E, V')$, where list L and buffer B is empty.

Oracle An action sequence bridges the input sentence and the AMR graph. So the basic requirement for the transition-based method is alignments. Given a gold AMR graph and alignments, one can convert the graph to an action sequence for model training. For each state s, HIT-SCIR decides one of the actions to apply and this is what we called *oracle* parser. To solve the problem of parsing concept nodes from surface strings, HIT-SCIR extends the basic oracle following previous work (Liu et al., 2018). The transition inventory is the following:

- MERGE is to connect the top two tokens in the buffer to a single token waiting for being converted to a concept node.

- CONFIRM$_X$ is for converting the top element of buffer to a concept node X.

- NEW$_X$ generates a new node X and pushing into the buffer.

- ENTITY$_X$ does the same thing as CONFIRM$_X$ but adding internal properties of entity X, such as *year* of a *date-entity*.

- LEFT-EDGE$_X$ and RIGHT-EDGE$_X$ add an edge with label X between w_j and w_i, where w_i is the top element of stack and w_j is the top element of buffer. But they can be performed only when the top of buffer is a concept node.

- SHIFT is performed when no dependency exists between w_j and any word in S other than w_i, which pushes all words in list and w_j into stack S. It is only allowed to perform when the top of buffer is a concept node.

- REDUCE is performed only when w_i has head and is not the head or child of any word in buffer, which pops w_i out of stack.

- PASS will be chosen when neither SHIFT or REDUCE can be performed, which moves w_i to the front of list.

- DROP pops the top of buffer when it is a token.

- FINISH pops the root node and marks the state as terminal.

Stack-LSTM HIT-SCIR follows Ballesteros and Al-Onaizan (2017) and uses Stack-LSTM to model AMR states. The output vector of this LSTM will consider the stack pointer instead of the rightmost position of the sequence.

The system models S, L, B and action history with multiple stack-LSTMs, which supports PUSH and POP operations. Parsing states from multiple stack LSTMs are fed into the action oracle classifier at once. The possibility of action under state s is calculated as

$$p(a|s) = \frac{\exp\{g_a \cdot \text{STACK LSTM}(s) + b_a\}}{\sum_{a' \in \mathcal{A}} \exp\{g_{a'} \cdot \text{STACK LSTM}(s) + b_{a'}\}}$$

where the set $\mathcal{A}$ represents the actions listed in the previous paragraph; STACK LSTM(s) encodes the state s into a vector, g_a is the embedding of action a and b_a is the bias vector for action.

In our model, items in S, L and B are the combined embedding of tokens that concatenate the original BERT word embeddings and EDS encoding for the tokens, introduced in the following section.

3.2 EDS Incorporation

In order to incorporate the EDS annotation information in the AMR parsing, we extend the EDS graph to include tokens. We feed the extended EDS graph to our proposed EDS encoder and obtain token-level EDS features. Afterward, we concatenate token-level EDS features with word embedding and input them into the transition-based model.

EDS Extension Each node in the EDS graph has an explicit many-to-many anchoring onto substrings of the input sentence. It means that corresponding related EDS nodes for each token can be found based on the nodes' span. Therefore, we add a bottom layer consisting of the input sentence tokens. In this way, the updated embedding of tokens in this layer can be extracted as EDS features for each token.

In the preprocessing, the edges labeled as *contain* are added between token nodes and original nodes if their spans of strings intersect. Figure 2 is the example of an updated EDS graph for the sentence "*Not this year.*", #20010002 from WSJ. We only care about EDS labels in our experiments. We show *contain* edges as dash lines and original edges as solid lines. In Figure 2, the bottom

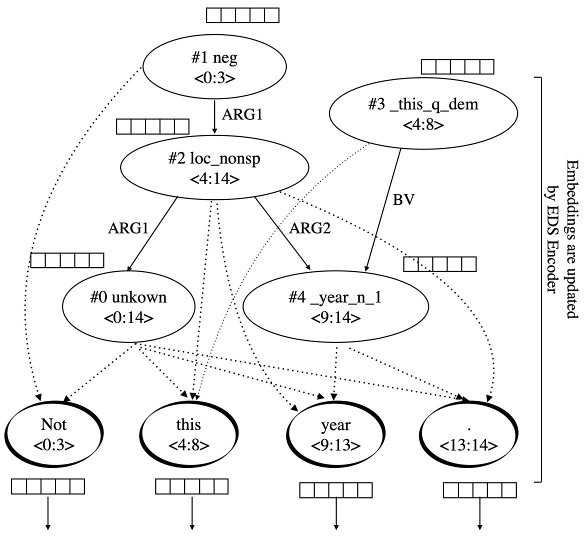

Figure 2: Example for adding *contain* edges in EDS graph. The bottom layer is the token nodes we add to the EDS graph. After EDS encoding, embeddings of token nodes are extracted and combined with original word embedding as extra semantic features.

four nodes are token nodes, whose embeddings are used as EDS features. For token node v_t, the hidden state at layer k is $h^k_{v_t}$. The calculation details can be found in the following paragraph. BERT splits each token into several pieces. The system extracts the first piece as its word embedding, denoted as BERT(t). Therefore, for each token t, we get embedding from two parts, BERT embedding BERT(t) and final hidden state h_{v_t} of corresponding new added node v_t. Then the concatenation of two vectors (BERT$(t)\|h_{v_t}$) would be pushed in the buffer, waiting for the next step of processing.

EDS Encoder The emergence of neural networks has had tremendous impacts on many fields, including graph data parsing systems. GCNs (Kipf and Welling, 2017; Marcheggiani and Titov, 2017) have emerged to be the neural networks of choice for encoding graphs. Our proposed EDS encoder consists of an LSTM layer to capture context information and GCN layers to encode structural knowledge.

EDS represents the meaning of a sentence in a directed graph where nodes represent logical predicates and edges to labeled arguments. The definition of EDS is $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, L_\mathcal{V}, L_\mathcal{E}\}$ where $\mathcal{V}$ is a set of nodes (v, ℓ_v), $\mathcal{E}$ is a set of edges (v_i, v_j, ℓ_e) and $L_\mathcal{V}$, $L_\mathcal{E}$ are vocabularies for node labels and edge labels respectively.

To reinforce relations between nodes through layers, we add *self* edges (v_i, v_i) for every node in the graph and inverted edges (v_j, v_i) with label *inv_*ℓ_e for each directed edge (v_i, v_j) with label ℓ_e, including the new added *contain* edges. Therefore, $\mathcal{G}$ becomes $\{\mathcal{V}', \mathcal{E}', L_{\mathcal{V}'}, L_{\mathcal{E}'}\}$. $\mathcal{V}' = \mathcal{V} \cup \mathcal{T}$, where $\mathcal{T}$ is the set of token nodes. $\mathcal{E}' = \mathcal{E} \cup \{contain, self\} \cup \mathcal{I}$, where $\mathcal{I}$ is the set of inverted edges.

The goal of our EDS encoder is to update representation of each node considering the whole EDS graph. First, we adopt GCN to update word embedding based on their neighbors. Directed edges in the EDS graph represent the relationship of nodes, so we make the same assumption that the GCN parameters are label-specific as Marcheggiani and Titov (2017). Therefore, we calculate the hidden state of node v at k-th layer h^k_v as:

$$h^k_v = ReLU\left(\sum_{u \in \mathcal{N}(v)} W^{(k-1)}_{L(u,v)} h^{(k-1)}_u + b^{(k-1)}_{L(u,v)}\right)$$

where $\mathcal{N}(v)$ represents the neighbor nodes of v; ReLU is the rectifier linear unit activation function. However, to reduce the size of parameters and simplify the calculation, we classify edges into three kinds: *self* edge, edges in the original direction including *contain* and inverted edges. Therefore, instead of using $W_{L(u,v)}$, we define them as $W_{L(u,v)} = V_{dir(u,v)}$, where $dir(u, v)$ specifies the kind of edge.

EDS annotation in this experiment is automatically generated by EDS parser, so accepting all information from the EDS graph is risky. To solve this problem, we adopt gate schema. We calculate a scalar gate for each edge node pair in the form as:

$$g^k_{u,v} = \sigma\left(\hat{v}^k_{dir(u,v)} h^k_u + \hat{b}^k_{L(u,v)}\right)$$

where σ is the logistic sigmoid function; $\hat{v}^k_{dir(u,v)}$ and $\hat{b}^k_{L(u,v)}$ are weights and a bias for the gate. Therefore, the final formalism of the hidden state calculation is:

$$h^k_v = ReLU\Big(\sum_{u\in\mathcal{N}(v)} g^{(k-1)}_{u,v}\big(V^{(k-1)}_{dir(u,v)} h^{(k-1)}_u + b^{(k-1)}_{L(u,v)}\big)\Big).$$

GCN introduced so far learns effective representation on the structure. Still, there is the limitation in that nodes can only be updated based on their immediate neighbors on each GCN layer. Nodes far away from each other with n-order in the graph are hard to encode on GCN models. Adding an LSTM layer can compensate for this limitation. The hidden states of LSTM instead of embedding of EDS nodes are fed into GCN layers, that is, $h^0_v = \boldsymbol{s}_v$ where $\boldsymbol{s}_v$ is the final LSTM state of node v.

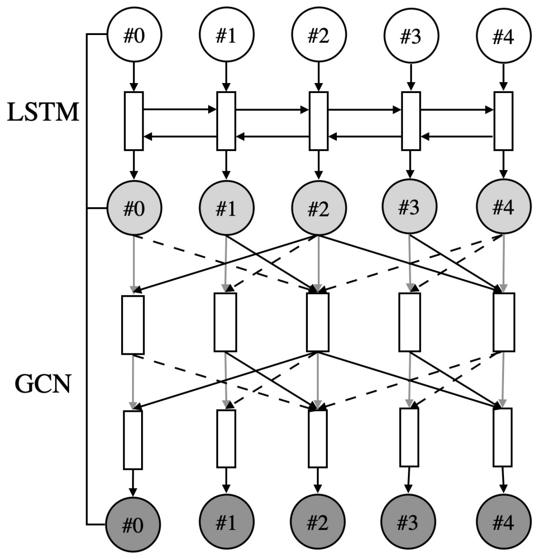

Figure 3: Example for EDS Encoder structure. Nodes embedding (circles) are sequentially fed into an LSTM layer and GCN layers. In GCN layers, solid lines are original edges in the EDS graph, dash lines represent the inverted edges and gray lines are *self* edges.

The structure of the EDS encoder is illustrated in Figure 3. The embeddings of EDS nodes (hollow circles) are first fed into an LSTM layer. After processing in the LSTM layer, contextual information is included in the light gray circles. After several GCN layers, dark gray circles that hold edges and neighbors information are the final hidden states of GCN layers.

4 Experiment

4.1 Experimental Setup

Our experiments were done using the toolkit AllenNLP (Gardner et al., 2018).

EDS Parser In this study, we adopted open-source distribution LOGON (Lønning and Oepen, 2006) to generate EDS annotations. LOGON [1] package contains ERG parsers and the ERG-to-EDS converter. Compared to the purely data-driven parsers, general-purpose grammatical knowledge encoded in the ERG aids EDS parsing (Oepen and Flickinger, 2019). We applied ERG release 1214 and use LOGON in one-best mode. However, LOGON failed to parse part of sentences due to limitation of search tree or other reasons (about 15% of data), so we used the EDS model of Che et al. (2019) to parse those sentences.

Baseline Model As we mentioned, we adopt HIT-SCIR as our baseline model. However, we use the smaller pre-trained model, BERT-base, for word embeddings due to GPU limitation. For alignments, the baseline model uses an enhanced rule-based aligner TAMR (Liu et al., 2018) to generate transition actions for AMR graph. More details on hyper-parameters can be found Table 3 in appendix. Our experiments were done using GeForce RTX 2080 Ti GPU. During model training, each epoch took about 4 hours on one GPU.

Dataset We use the dataset from MRP 2019 so that we can compare our models with the officially submitted models there. The shared task has constraints on which additional data or pretrained models can be used for reasons of comparability and fairness. Our models meet the requirements as both our baseline model (HIT-SCIR) and the EDS parser that we use to generate EDS graphs for use by the baseline model satisfy them.

There are 56,240 sentences in MRP 2019 AMR training set. The test set contains 1,998 sentences, and among them are 100 randomly selected sentences from the novel *The Little Prince*. MRP 2019 provided results for AMR parsing models on both

[1] http://wiki.delph-in.net/LogonProcessing

the entire test set (called All Data) and the subset of the setences from *The Little Prince* (called Lpps). The reason for the special interest on the latter was that the sentences from the novel are presumably least similar to the training data, which are mostly from the WSJ-corpus. Indeed, most models at MRP 2019 have lower scores on Lpps than on All Data. We will have more to say about this in the next section on our experimental results.

Metrics MRP 2019 used two metrics to evaluate the models: the standard SMATCH scorer (Cai and Knight, 2013) included in the open-source mtool software (the Swiss Army Knife of Meaning Representation) [2], and an MRP 2019 specific scorer that is similar to SMATCH but can compare two meaning representation graphs (the ground truth and the model output) according to certain fine-grained attributes such as edges, node labels and so on. We'll mainly use the SMATCH metric but also give MRP metric for the Lpps test set. We refer the reader to Oepen et al. (2019) for more details on MRP 2019 datasets and metrics.

4.2 Results

Results on Different Structures Our SMATCH experimental results are summarized in Table 1. To see the effects of different encodings of EDS graphs, we tried five EDS-enhanced systems. Among the five, three use only GCNs ([G1],[G2],[G3]), from single layer to three layers, and two with a single BiLSTM layer plus one or two GCN layers ([LG1],[LG2]). As can be seen from Table 1, LG1 achieves the highest F1 score, outperforming Amazon (Cao et al., 2019), the best overall AMR parser at MRP 2019.

We note that all our five EDS-enhanced systems perform better than HIT-SCIR, our reference baseline model. Interestingly, the number of GCN layers matters and it's not necessarily the more the better. The reason for GCN performance degradation in our work is possible to be over-smoothing, which was discussed in previous work (Li et al., 2018).

Among the three with GCN layers only ([G1],[G2],[G3]), the best is G2. When BiLSTM is added, one layer of GCN ([LG1]) is better than two ([LG2]). It's possible there is some theoretical explanation for this but we suspect it also has something to do with the dataset. For example, for All Data, the F1 score of BiLSTM plus one layer of GCN is the same as the one with two layers.

Results on Lpps Some interesting observations can be made on the Lpps test set. As we mentioned, this test set contains 100 random sentences from the book *The Little Prince*. These sentences seem to be quite different from those in the training set given at MRP 2019. So not surprisingly, most models have poorer performance on this test set except for Saarland (Donatelli et al., 2019) which somehow performs better in this test set than the All Data set. As for HIT-SCIR, its performance on Lpps is a lot worse than that on All Data. What is worthwhile noting is that our models, which are basically HIT-SCIR enhanced with EDS in various ways, boost its performance on Lpps significantly. Our best model is even better than Saarland on this test set. Compared to HIT-SCIT, it increases its F1 scores by 4.6% (from .680 to .726) on Lpps while only 1.1% on All Data (from .725 to .736). So the extra EDS information really pays off on this test set.

SMATCH is a general tool for computing the overall differences between two answers. To give a more fine-grained comparison between two meaning representations, MRP has its own scorer to compute what are called the Tops, Labels, Properties, Edges, and All scores, where the All scores are close to the SMATCH score. Table 2 gives MRP F1 scores for the Lpps test set for our baseline model HIT-SCIR and our EDS-enhanced models. Again we see that our models improve the performance of the baseline model significantly in all subtasks, especially in Labels and Properties. For F1 score on Tops, Labels, Properties, the model LG1 performs best, 5%, 6% and 9% improvement respectively. Whereas, G2 performs best in Edge F1 score, about 3% improvement.

Our model can handle these "out of domain" cases better because we have accurate EDS parsing for them. Consider the verb "look" in the sentence "I shall look as if I were suffering." In the baseline model, the predicted AMR node for it is "look-01", which is wrong. Our model with the extra information from EDS correctly labels it "look-02". The possible reason why the baseline model selects "look-01" is that "look-01" appears nearly twice as often as "look-02" in training data: the former 198 times and the latter 103. However, the EDS subgraph for the phrase "sb. look as if " is node(pron)-edge(arg1)-node(look-v)-

[2] https://github.com/cfmrp/mtool

System	Precision		Recall		F1	
	All Data	Lpps	All Data	Lpps	All Data	Lpps
Amazon (Cao et al., 2019)	.75	.70	.71	.71	.730	.704
Saarland (Donatelli et al., 2019)	.70	.73	.63	.71	.661	.722
SJTU-NICT (Li et al., 2019)	.75	.71	.68	.69	.714	.696
Suda-Alibaba (Zhang et al., 2019b)	.73	.66	.70	.69	.713	.674
HIT-SCIR (Che et al., 2019)	.77	.71	.69	.65	.725	.680
+EDS (GCNs, K=1)[G1]	.787	.738	.683	.671	.731	.703
+EDS (GCNs, K=2)[G2]	.780	.763	.691	.689	.733	.724
+EDS (GCNs, K=3)[G3]	.783	.752	.689	.674	.733	.711
+EDS (BiLSTM+GCNs, K=1)[LG1]	.785	.770	.692	.687	**.736**	**.726**
+EDS (BiLSTM+GCNs, K=2)[LG2]	.785	.774	.690	.678	.735	.723

Table 1: SMATCH scores on the evaluation data, ”All Data” means results on all evaluation data and ”Lpps” is results on 100 sentences of *The Little Prince*. In this table, the top part is the official results for four systems achieving competitive results in MRP 2019; the middle part is the results for our baseline model; the bottom part ”+EDS” are our proposed EDS-enhanced models. For example, ”GCNs, K=1” means EDS graph encoder consists of one GCN layer, denoted as [G1]; ”BiLSTM+GCN, K=2” represents EDS graph encoder is composed of one BiLSTM layer and two GCN layers, denoted as [LG2].

System	Tops			Labels			Properties			Edges			All		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
HIT-SCIR	.81	.81	.81	.78	.74	.76	.51	.57	.54	.66	.56	.61	.722	.660	.689
+EDS [G1]	.84	.84	.84	.81	.77	.79	.77	.48	.59	.65	.57	.61	.746	.678	.710
+EDS [G2]	.82	.84	.83	.83	.78	.81	.76	.52	.62	.69	.59	**.64**	.771	.697	.732
+EDS [G3]	.83	.84	.83	.81	.77	.79	.73	.50	.59	.68	.58	.63	.761	.682	.720
+EDS [LG1]	.86	.86	**.86**	.85	.79	**.82**	.80	.52	**.63**	.68	.58	.62	.779	.695	**.734**
+EDS [LG2]	.83	.85	.84	.85	.78	.81	.84	.48	.61	.69	.58	.63	.783	.686	.732

Table 2: MRP scores of Lpps AMR sub tasks. In this table, HIT-SCIR is our baseline model (Che et al., 2019). ”+EDS(G1)” to ”+EDS(LG2)” correspond to the EDS-enhanced model ”+EDS(GCNs, K=1)” to ”+EDS(BiLSTM+GCNs, K=2)” in Table 1 according to the indexes.

edge(arg1)-node(as+if), which suggests ”look-02”. EDS graphs often encode multiple tokens as one node, and this helps predict edges more accurately. An example is ”_blow_v_away” in ”The wind blows them away.” The baseline model predicts the edge label between ”blow” and ”away” as ”:ARG2”, whereas the gold answer is ”:direction”, which can be predicted by EDS-enhanced model. Here we believe the EDS subgraph ”node(_blow_v_away)-edge(contain)-node(away)” affects the final result.

Results on gold EDS annotations Finally, we notice that MRP provided gold EDS annotation for Lpps test data. We tried our models using these gold EDS annotations on the Lpps test set, and observed that this actually resulted in a minor reduction in F1 scores, around 0.0001 worth than the model using silver EDS annotations. The reason the gold label actually performed a bit worse is because the model was trained using the actual EDS parsing results, so seems to ”adapted” to the bias of the EDS parser used.

As a footnote, we remark here that we are aware of the new results on AMR parsers at MRP 2020 that were released in late November 2020. This work was done prior to MRP 2020. While MRP 2020 also used Lpps as a test set, the results there and our results here are not directly comparable as they were done using different training sets. Furthermore, the main purpose of this paper is about incorporating EDS graphs into AMR parsing, so the comparison with the baseline model is more meaningful.

5 Conclusions

In this study, we incorporate the EDS, a meaning representation that is more accessible than AMR, to improve the performance of AMR parsing. To encode EDS graphs for AMR parsing, we used both LSTM and GCN layers. As a case study, we enhanced a transition-based AMR parser with EDS graphs, and showed that on the AMR benchmarks that the baseline model already performs well, our EDS-enhanced parsers can further improve its performance. The improvements are especially noticeable on the Lpps (*The Little Prince*) test set where the baseline parser performs poorly and lags behind other AMR parsers at MRP 2019. In fact, on the Lpps test set, our EDS enhanced parsers outperform even the best one submitted there.

We can also see some other implications of this work. For us, the ultimate goal of semantic parsing is to use it in downstream tasks such as question answering, reasoning, and knowledge extraction from texts. Given that almost all meaning representations are graph-based, we believe our encoding of EDS graphs with LSTM and GCN layers can be applied in these downstream tasks. We are currently exploring this as a future work. Another insight from this work is about possible connections among different meaning representations. We have demonstrated the usefulness of EDS graphs for AMR parsing. It is likely they can also be useful for other frameworks, even vice versa. More generally, whether there is a universal semantic parser that can take advantages of information from each framework is an interesting question worth investigating.

References

Omri Abend and Ari Rappoport. 2013. UCCA: A semantics-based grammatical annotation scheme. In *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013) – Long Papers*, pages 1–12, Potsdam, Germany. Association for Computational Linguistics.

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage ccg semantic parsing with amr. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710.

Ofir Arviv, Ruixiang Cui, and Daniel Hershcovich. 2020. Huji-ku at mrp 2020: Two transition-based neural parsers. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 73–82.

Ramón Fernandez Astudillo, Miguel Ballesteros, Tahira Naseem, Austin Blodgett, and Radu Florian. 2020. Transition-based parsing with stack-transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 1001–1007.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. Amr parsing using stack-lstms. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1269–1275.

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.

Lauritz Brandt, David Grimm, Mengfei Zhou, and Yannick Versley. 2016. Icl-hd at semeval-2016 task 8: Meaning representation parsing-augmenting amr parsing with a preposition semantic role labeling neural network. In *Proceedings of the 10th international workshop on semantic evaluation (semeval-2016)*, pages 1160–1166.

Deng Cai and Wai Lam. 2020. AMR parsing via graph-sequence iterative inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1290–1301, Online. 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.

Jie Cao, Yi Zhang, Adel Youssef, and Vivek Srikumar. 2019. Amazon at mrp 2019: Parsing meaning representations with lexical and phrasal anchoring. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 138–148.

Wanxiang Che, Longxu Dou, Yang Xu, Yuxuan Wang, Yijia Liu, and Ting Liu. 2019. Hit-scir at mrp 2019: A unified pipeline for meaning representation parsing via efficient training and effective encoding. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 76–85.

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.

Lucia Donatelli, Meaghan Fowlie, Jonas Groschwitz, Alexander Koller, Matthias Lindemann, Mario Mina, and Pia Weißenhorn. 2019. Saarland at mrp 2019: Compositional parsing across all graphbanks. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 66–75. Association for Computational Linguistics (ACL).

Jeffrey Flanigan, Sam Thomson, Jaime G Carbonell, Chris Dyer, and Noah A Smith. 2014. A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6.

Daniel Hershcovich and Ofir Arviv. 2019. Tupa at mrp 2019: A multi-task baseline system. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 28–39.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Zuchao Li, Hai Zhao, Zhuosheng Zhang, Rui Wang, Masao Utiyama, and Eiichiro Sumita. 2019. Sjtu-nict at mrp 2019: Multi-task learning for end-to-end uniform semantic graph parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 45–54.

Zi Lin and Nianwen Xue. 2019. Parsing meaning representations: is easier always better? In *Proceedings of the First International Workshop on Designing Meaning Representations*, pages 34–43.

Yijia Liu, Wanxiang Che, Bo Zheng, Bing Qin, and Ting Liu. 2018. An amr aligner tuned by transition-based parser. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2422–2430.

Jan Tore Lønning and Stephan Oepen. 2006. Re-usable tools for precision machine translation. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 53–56, Sydney, Australia. Association for Computational Linguistics.

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.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1506–1515.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Tahira Naseem, Abhishek Shah, Hui Wan, Radu Florian, Salim Roukos, and Miguel Ballesteros. 2019. Rewarding smatch: Transition-based amr parsing with reinforcement learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4586–4592.

Stephan Oepen, Omri Abend, Jan Hajic, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, Jayeol Chun, Milan Straka, and Zdenka Uresova. 2019. MRP 2019: Cross-framework meaning representation parsing. In *CoNLL Shared Task*, pages 1–27.

Stephan Oepen and Dan Flickinger. 2019. The erg at mrp 2019: Radically compositional semantic dependencies. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 40–44.

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).

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A synchronous hyperedge replacement grammar based approach for amr parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 32–41.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015. A transition-based algorithm for amr parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–375.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019a. Amr parsing as sequence-to-graph transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 80–94.

Yue Zhang, Wei Jiang, Qingrong Xia, Junjie Cao, Rui Wang, Zhenghua Li, and Min Zhang. 2019b. Suda-alibaba at mrp 2019: Graph-based models with bert. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 149–157.

Qiji Zhou, Yue Zhang, Donghong Ji, and Hao Tang. 2020. AMR parsing with latent structural information. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4306–4319, Online. Association for Computational Linguistics.

Table 3 contains the values of hyper-parameters.

HYPER PARAMETER	VALUE
Word Embeddings	
source	BERT
dim	768
Model Parameters	
Action dim	128
Entity dim	64
Relation dim	64
Hidden dim	256
Dropout	0.2
Layer dropout	0.2
Recurrent dropout	0.2
EDS nodes embedding	64
EDS LSTM hidden dim	64
EDS GCN hidden dim	128
Trainer Parameters	
Learning rate scheduler	slanted triangular
Gradual unfreezing	True
Cut Frac	0.1
Ratio	32
Base learning rate	1×10^{-3}
BERT learning rate	5×10^{-5}
Batch size	6
Epoch	20
Gradient clipping	5
Gradient norm	5
Optimizer	Adam
β_1, β_2	0.9,0.999

Table 3: Hyper-parameters settings