Line Graph Enhanced AMR-to-Text Generation with Mix-Order Graph Attention Networks

Yanbin Zhao, Lu Chen, Zhi Chen, Ruisheng Cao, Su Zhu, Kai Yu*
MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
SpeechLab, Department of Computer Science and Engineering
Shanghai Jiao Tong University, Shanghai, China
{zhaoyb, chenlusz, zhenchi713}@sjtu.edu.cn
{211314, paul2204, kai.yu}@sjtu.edu.cn

Abstract

Efficient structure encoding for graphs with labeled edges is an important yet challenging point in many graph-based models. This work focuses on AMR-to-text generation – A graph-to-sequence task aiming to recover natural language from Abstract Meaning Representations (AMR). Existing graph-to-sequence approaches generally utilize graph neural networks as their encoders, which have two limitations: 1) The message propagation process in AMR graphs is only guided by the first-order adjacency information. 2) The relationships between labeled edges are not fully considered. In this work, we propose a novel graph encoding framework which can effectively explore the edge relations. We also adopt graph attention networks with higher-order neighborhood information to encode the rich structure in AMR graphs. Experiment results show that our approach obtains new state-of-the-art performance on English AMR benchmark datasets. The ablation analyses also demonstrate that both edge relations and higher-order information are beneficial to graph-to-sequence modeling.

1 Introduction

Abstract Meaning Representation (Banarescu et al., 2013) is a sentence-level semantic representation formalized by a rooted directed graph, where nodes are concepts and edges are semantic relations. Since AMR is a highly structured meaning representation, it can promote many semantic related tasks such as machine translation (Song et al., 2019) and summarization (Liao et al., 2018). However, the usage of AMR graphs can be challenging, since it is non-trivial to completely capture the rich structural information in the graph-based data, especially when the graph has labeled edges.

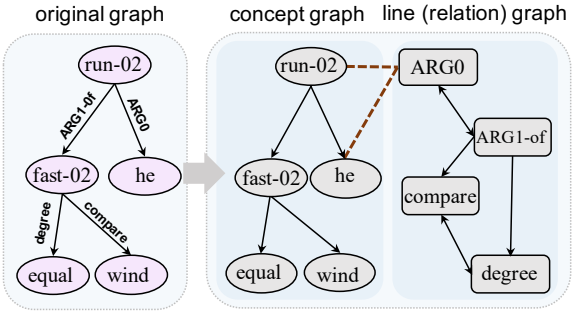

Figure 1: An AMR graph (left) for sentence "He runs as fast as the wind." and its concept graph and relation graph (line graph). Two graphs are aligned with each other based on the node-edge relations in the original graph.

Generation from AMR aims to translate the AMR semantics into the surface form (natural language). It is a basic Graph-to-sequence task that directly takes AMR as input. Figure 1 (left) gives a standard AMR graph and its corresponding surface form. Early works utilize sequence-to-sequence framework by linearizing the entire graph (Konstas et al., 2017; Cao and Clark, 2019). Such representation may lose useful structural information. In recent studies, graph neural networks (GNNs) have been in a dominant position on this task and achieved state-of-the-art performance (Beck et al., 2018; Song et al., 2018; Guo et al., 2019; Damonte and Cohen, 2019). However, In these GNN-based models, the representation of each concept node is only updated by the aggregated information from its neighbors, which leads to two limitations: 1) The interaction between indirectly connected nodes heavily relies on the number of stacked layers. When the graph size becomes larger, the dependencies between distant AMR concepts cannot be fully explored. 2) They only focus on modeling the relations between concepts while ignoring edge relations and their structures. Zhu et al. (2019)

*Kai Yu is the corresponding author.

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 732–741
July 5 - 10, 2020.

and Cai and Lam (2019) use Transformer to model arbitrary concept pairs no matter whether directly connected or not, but they still ignore the topological structures of the edges in the entire AMR graph.

To address the above limitations, we propose a novel graph-to-sequence model based on graph attention networks (Velickovic et al., 2018). We transform the edge labels into relation nodes and construct a new graph that directly reflects the edge relations. In graph theory, such a graph is called a Line Graph (Harary and Norman, 1960). As illustrated in Figure 1, we thus separate the original AMR graph into two sub-graphs without labeled edges – concept graph and relation graph. The two graphs describe the dependencies of AMR concepts and edges respectively, which is helpful in modeling these relationships (especially for edges). Our model takes these sub-graphs as inputs, and the communications between the two graphs are based on the attention mechanism. Furthermore, for both graphs, we mix the higher-order neighborhood information into the corresponding graph encoders in order to model the relationships between indirectly connected nodes.

Empirical study on two English benchmark datasets shows that our model reaches state-of-the-art performance with 30.58 and 32.46 BLEU scores on LDC2015E86 and LDC2017T10, respectively. In summary, our contributions include:

- We propose a novel graph-to-sequence model, which firstly uses the line graph to model the relationships between AMR edges.
- We integrate higher-order neighborhood information into graph encoders to model the relationships between indirectly connected nodes.
- We demonstrate that both higher-order neighborhood information and edge relations are important to graph-to-sequence modeling.

2 Mix-Order Graph Attention Networks

In this section, we first introduce graph attention networks (GATs) and their mix-order extensions, which are the basis of our proposed model.

2.1 Graph Attention Networks

GAT is a special type of networks that operates on graph-structured data with attention mechanisms. Given a graph $G = (V, E)$, where V and E are the set of nodes x_i and the set of edges (e_{ij}, ℓ_e)[1], respectively. $\mathcal{N}(x_i)$ denote the nodes which are directly connected by x_i. $\mathcal{N}_+(x_i)$ is the set including x_i and all its direct neighbors. we have $\mathcal{N}_+(x_i) = \mathcal{N}(x_i) \cup \{x_i\}$.

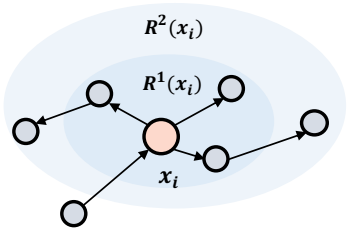

Figure 2: Neighborhood information in different orders.

Each node x_i in the graph has an initial feature $\mathbf{h}_i^0 \in \mathbb{R}^d$, where d is the feature dimension. The representation of each node is iteratively updated by the graph attention operation. At the l-th step, each node x_i aggregates context information by attending over its neighbors and itself. The updated representation $\mathbf{h}_i^l$ is calculated by the weighted average of the connected nodes:

$$\mathbf{h}_i^l = \sigma\left(\sum_{x_j \in \mathcal{N}_+(x_i)} \alpha_{ij} \mathbf{h}_j^{l-1} \mathbf{W}^l\right), \quad (1)$$

where attention coefficient α_{ij} is calculated as:

$$\alpha_{ij} = \text{softmax}_j\left(\mathbf{h}_i^{l-1}\mathbf{W}_{t1}^l\right)\left(\mathbf{h}_j^{l-1}\mathbf{W}_{t2}^l\right)^T \quad (2)$$

where σ is a nonlinear activation function, e.g. ReLU. $\mathbf{W}^l$, $\mathbf{W}_{t1}^l$ and $\mathbf{W}_{t2}^l \in \mathbb{R}^{d \times d}$ are learnable parameters for projections. After L steps, each node will finally have a context-aware representation $\mathbf{h}_i^L$. In order to achieve a stable training process, we also employ a residual connection followed by layer normalization between two graph attention layers.

2.2 Mixing Higher Order Information

The relations between indirectly connected nodes are ignored in a traditional graph attention layer. Mix-Order GAT, however, can explore these relationships in a single-step operation by mixing the higher-order neighborhood information. We first give some notations before describing the details of the Mix-Order GAT. We use $\mathbf{R}^K = \{\mathcal{R}^1, ...\mathcal{R}^K\}$ to represent neighborhood information from order

[1] ℓ_e is the edge label which are not considered in the GAT layer

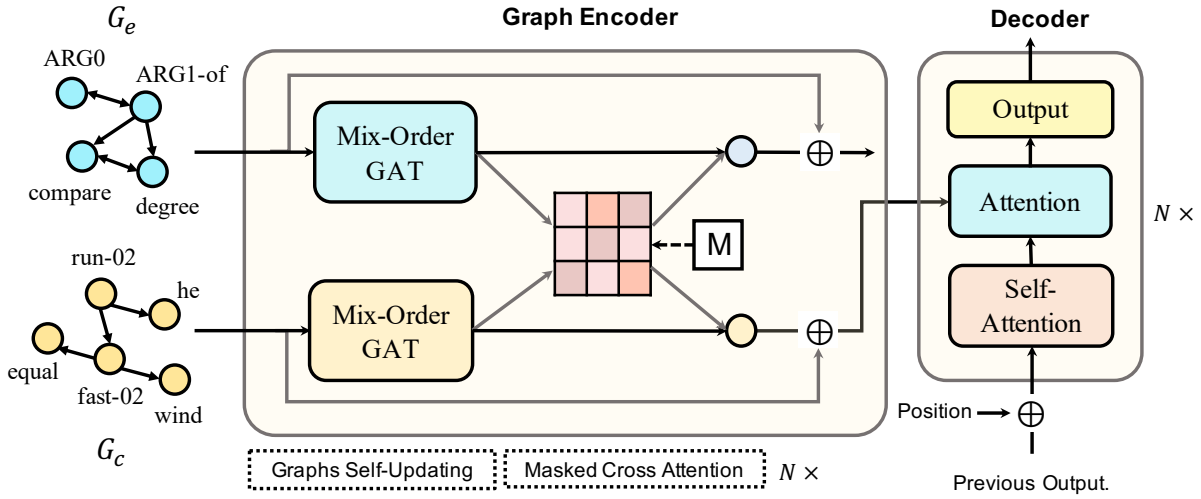

Figure 3: An overview of our proposed model

1 to order K. $\mathcal{R}^k(x_i)$ denotes the k-th order neighborhood, which means all nodes in $\mathcal{R}^k(x_i)$ are reachable for x_i within k hops ($k \geq 1$). $\mathcal{R}^1(x_i) = \mathcal{N}_+(x_i)$, and as illustrated in Figure 2, we can have:

$$\mathcal{R}^k(x_i) = \bigcup_{x_j \in \mathcal{R}^{k-1}(x_i)} \mathcal{N}_+(x_j). \tag{3}$$

The K-Mix GAT integrates the neighborhood information $\mathbf{R}^K$. At the l-th update step, each x_i will interact with its reachable neighbors with different orders and calculate the attentive features independently. The representation $\mathbf{h}_i^l$ is updated by the concatenated features from different orders, i.e.

$$\begin{aligned}\mathbf{h}_i^l &= \text{MixGAT}^l(\mathbf{h}_i^{l-1}, \mathbf{R}^K) \\ &= \Big\|_{k=1}^{K} \sigma\left(\sum_{x_j \in \mathcal{R}^k(x_i)} \alpha_{ij}^k \mathbf{h}_j^{l-1} \mathbf{W}_k^l\right),\end{aligned} \tag{4}$$

where $\|$ represents concatenation, α_{ij}^k are the attention weights in the k-th order, and $\mathbf{W}_k^l \in \mathbb{R}^{d \times d/K}$ are learnable weights for projections. We will use MixGAT$(\cdot)$ to denote the Mix-Order GAT layer in the following section.

3 Method

The architecture of our method is illustrated in Figure 3. As mentioned above, we separate the AMR graph into two sub-graphs without labeled edges. Our model follows the Encoder-Decoder architecture, where the encoder takes the two sub-graphs as inputs, and the decoder generates corresponding text from the encoded information. We first give some detailed explanations about the line graph and input representation.

3.1 Line Graph & Input Representation

The line graph of a graph G is another graph $L(G)$ that represents the adjacencies between edges of G. $L(G)$ is defined as:

- Each node of $L(G)$ represents an edge of G
- Two nodes of $L(G)$ are adjacent if and only if their corresponding edges share a common node in G.

For directed graphs, the directions are maintained in the corresponding line graphs. Redundant edges between two relation nodes are removed in the line graphs. Figure 4 provides several examples.

In our model, we use the line graph to organize labeled edges and transform the original AMR graph into two sub-graphs. Given an AMR graph $G_a = (V_a, E_a)$, we separate it into concept graph $G_c = (V_c, E_c)$ and relation graph $G_e = (V_e, E_e)$, where $G_e = L(G_a)$. As for concept graph G_c, its topological structure is the same with G_a, but the edge labels are eliminated, i.e.

$$V_c = V_a; \quad E_c = \hat{E}_a, \tag{5}$$

Where $\hat{E}_a$ is the edge set without label information. Both G_c and G_e have no labeled edges, which can be efficiently encoded by Mix-Order GAT.

We use $\mathbf{R}_c^K$ and $\mathbf{R}_e^K$ to denote $1 \sim K$ orders neighborhood information of G_c and G_e. We represent each concept node $x_i \in V_c$ with an initial embedding $\mathbf{c}_i^0 \in \mathbb{R}^d$, and each relation node $y_i \in V_e$

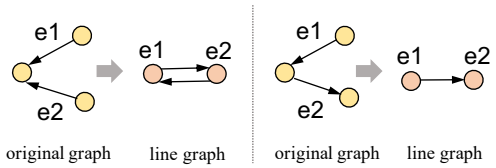

Figure 4: Examples of finding line graphs. In the left part, $e1$ and $e2$ have opposite directions, so each direction is maintained in the line graph. In the right part, $e1$ and $e2$ follow the same direction, so there is only one direction in the corresponding line graph.

with an embedding $\mathbf{e}_i^0 \in \mathbb{R}^d$. The sets of node embeddings are denoted as $\mathbf{C}^0 = \{\mathbf{c}_i^0\}_{i=1}^m$ and $\mathbf{E}^0 = \{\mathbf{e}_i^0\}_{i=1}^n$, where $m = |V_c|$ and $n = |V_e|$ denote the numbers of concept nodes and relation nodes, respectively. Thus, the inputs of our system can be formulated by $\mathbf{I} = \left\{\mathbf{C}^0, \mathbf{E}^0, \mathbf{R}_c^K, \mathbf{R}_e^K\right\}$.

3.2 Self Updating

The encoder of our system consists of N stacked graph encoding layers. As illustrated in Figure 3, each graph encoding layer has two parts: self-updating for each graph and masked cross attention. For G_c and G_e, We use $\mathbf{C}^{l-1} = \{\mathbf{c}_i^{l-1}\}_{i=1}^m$ and $\mathbf{E}^{l-1} = \{\mathbf{e}_i^{l-1}\}_{i=1}^n$ to denote the input node embeddings of the l-th encoding layer. The representations of the two graphs are updated independently by mix-order graph attention networks (MixGAT). At the l-th step (layer), we have:

$$\begin{aligned}\vec{\mathbf{C}}_{self}^l &= \text{MixGAT}_{c1}^l(\mathbf{C}^{l-1}, \mathbf{R}_c^K), \\ \vec{\mathbf{E}}_{self}^l &= \text{MixGAT}_{e1}^l(\mathbf{E}^{l-1}, \mathbf{R}_e^K).\end{aligned} \tag{6}$$

Where $\vec{\mathbf{C}}_{self}^l$ and $\vec{\mathbf{E}}_{self}^l$ are updated representations according to the mix-order neighborhood information $\mathbf{R}_c^K$ and $\mathbf{R}_e^K$. One thing should be noticed is that both G_c and G_e are directed graphs. This implies that the information propagation in the graph is in a top-down manner, following the pre-specified direction. However, unidirectional propagation loses the structural information in the reversed direction. To build communication in both directions, we employ Dual Graph (Ribeiro et al., 2019). Dual graph has the same node representations but reversed edge directions compared to the original graph. For example, if edge A→B is in the original graph, it turns to B→A in the corresponding dual graph. Since dual graphs have the same node representations, we only need to change the neighborhood information. Denote $\widetilde{G}_c$ and $\widetilde{G}_e$ as the dual graph of G_c and G_e. $\widetilde{\mathbf{R}}_c^K$ and $\widetilde{\mathbf{R}}_e^K$ are the corresponding neighborhood information. We have:

$$\begin{aligned}\overleftarrow{\mathbf{C}}_{self}^l &= \text{MixGAT}_{c2}^l(\mathbf{C}^{l-1}, \widetilde{\mathbf{R}}_c^K), \\ \overleftarrow{\mathbf{E}}_{self}^l &= \text{MixGAT}_{e2}^l(\mathbf{E}^{l-1}, \widetilde{\mathbf{R}}_e^K).\end{aligned} \tag{7}$$

Since we have updated the node embeddings in two directions, the final representations of the independent graph updating process are the combination of the bi-directional embeddings, i.e.

$$\begin{aligned}\mathbf{C}_{self}^l &= \left[\vec{\mathbf{C}}_{self}^l; \overleftarrow{\mathbf{C}}_{self}^l\right] \mathbf{W}_{c1}^l, \\ \mathbf{E}_{self}^l &= \left[\vec{\mathbf{E}}_{self}^l; \overleftarrow{\mathbf{E}}_{self}^l\right] \mathbf{W}_{e1}^l,\end{aligned} \tag{8}$$

where $\mathbf{W}_{c1}^l$ and $\mathbf{W}_{e1}^1 \in R^{2d \times d}$ are trainable matrix for projections. $\mathbf{C}_{self}^l \in \mathbb{R}^{m \times d}$ and $\mathbf{E}_{self}^l \in \mathbb{R}^{n \times d}$ are results of the self-updating process.

3.3 Masked Cross Attention

Self updating for G_c and G_e can model the relationships of AMR concepts and edge respectively. However, it is also necessary to explore the dependencies *between* concept nodes and relation nodes. As a result, the cross-graph communication between G_c and G_e is very important. From the structure of the original AMR graph, we can easily build alignment between G_c and G_e. A relation node y_i is directly aligned to a concept node x_i if x_i is the start-point/end-point of the edge corresponding to y_i. As illustrated in Figure 1, ARG0 is the edge between run-02 and he. As a result, node ARG0 in G_e is directly connect to run-02 and he in G_c.

We apply the attention mechanism to complete the interaction between the two graphs, and use $\mathbf{M} \in \mathbb{R}^{n \times m}$ to mask the attention weights of unaligned pairs between G_c and G_e. For element m_{ij} in $\mathbf{M}$, we let $m_{ij} = 0$ if $y_i \in V_e$ is aligned to $x_j \in V_c$, otherwise $m_{ij} = -\infty$. The masked cross attention is employed between the representation sets $\mathbf{E}_{self}^l$ and $\mathbf{C}_{self}^l$, and the matrix of attention weights $\mathbf{A}_l$ can be calculated as:

$$\mathbf{A}_l = \left(\mathbf{E}_{self}^l \mathbf{W}_{a1}^l\right)\left(\mathbf{C}_{self}^l \mathbf{W}_{a2}^l\right)^T + \mathbf{M}, \tag{9}$$

where $\mathbf{W}_{a1}^l$ and $\mathbf{W}_{a2}^l \in \mathbb{R}^{d \times d}$ are learnable projection matrixes. The weight scores of unaligned pairs are set to $-\infty$ according to $\mathbf{M}$. For nodes in $\mathbf{E}_{self}^l$, the relevant representation from $\mathbf{C}_{self}^l$ is identified using $\mathbf{A}_l$ as:

$$\mathbf{E}_{cross}^l = \text{softmax}\left(\mathbf{A}_l\right) \mathbf{C}_{self}^l, \tag{10}$$

where $\mathbf{E}^l_{cross} \in \mathbb{R}^{n \times d}$ is the masked weighted summation of $\mathbf{C}^l_{self}$. The same calculation is performed for nodes in $\mathbf{C}^l_{self}$ as:

$$\mathbf{C}^l_{cross} = \text{softmax}(\mathbf{A}^T_l)\mathbf{E}^l_{self}. \quad (11)$$

The final outputs of a graph encoding layer are the combination of the original embeddings and the context representations from another graph. We also employ the outputs from previous layer as residual inputs, i.e.

$$\begin{aligned}\mathbf{C}^l &= \text{FFN}\left(\left[\mathbf{C}^l_{self}; \mathbf{C}^l_{cross}\right]\mathbf{W}^l_{c2} + \mathbf{C}^{l-1}\right), \\ \mathbf{E}^l &= \text{FFN}\left(\left[\mathbf{E}^l_{self}; \mathbf{E}^l_{cross}\right]\mathbf{W}^l_{e2} + \mathbf{E}^{l-1}\right),\end{aligned} \quad (12)$$

where FFN is a feed-forward network consists of two linear transformations. After N-stacked graph encoding layers, The two graphs G_c and G_e are finally encoded as $\mathbf{C}^N$ and $\mathbf{E}^N$.

3.4 Decoder

The decoder of our system is similar to the Transformer decoder. At each generation step, the representation of the output token is updated by multiple rounds of attention with the previously-generated tokens and the encoder outputs. Note that the outputs of our graph encoder have two parts: concept representations $\mathbf{C}^N$ and the relation representations $\mathbf{E}^N$. For generation, concept information is more important, since the concept graph directly contains the natural words. With the multi-step cross attention, $\mathbf{C}^N$ also caries abundant relation information. For simplicity, we only use $\mathbf{C}^N$ as the encoder output on the decoder side[2].

To address the data sparsity issue in sequence generation, we employ the Byte Pair Encoding (BPE) (Sennrich et al., 2016) following the settings of Zhu et al. (2019). We split the word nodes in AMR graphs and reference sentences into subwords, and the decoder vocabulary is shared with the encoder for concept graphs.

4 Experiments

4.1 Settings

Data and preprocessing We conduct our experiments with two benchmark datasets: LDC2015E85 and LDC2017T10. The two datasets contain 16833 and 36521 training samples, and they use a common development set with 1368 samples and a common test set with 1371 samples. We segment natural words in both AMR graphs and references into sub-words. As a result, a word node in AMR graphs may be divided into several sub-word nodes. We use a special edge `subword` to link the corresponding sub-word nodes. Then, for each AMR graph, we find its corresponding line graph and generate G_c and G_e respectively.

Training details For model parameters, the number of graph encoding layers is fixed to 6, and the representation dimension d is set to 512. We set the graph neighborhood order $K = 1, 2$ and 4 for both G_c and G_e. The Transformer decoder is based on Open-NMT (Klein et al., 2018), with 6 layers, 512 dimensions and 8 heads. We use Adam (Kingma and Ba, 2015) as our optimizer and $\beta = (0.9, 0.98)$. The learning rate is varied over the course of training, similar with Vaswani et al. (2017):

$$lr = \gamma d^{-0.5} \cdot \min(t^{-0.5}, t * w^{-1.5}), \quad (13)$$

where t denotes the accumulative training steps, and w indicates the warmup steps. We use $w = 16000$ and the coefficient γ is set to 0.75. As for batch size, we use 80 for LDC2015E86 and 120 for LDC2017T10.[3]

4.2 Results

We compare our system with several baselines, including traditional sequence-to-sequence models, several graph-to-sequence models with multiple graph encoders, and transformer-based models. All models are trained on the single dataset without ensemble or additional unlabeled data. For performance evaluation, we use BLEU (Papineni et al., 2002) as our major metric. We also use Meteor (Banerjee and Lavie, 2005), which considers the synonyms between predicted sentences and references.

The experimental results on the test sets of LDC2015E86 and LDC2017T10 are reported in Table 1. As we can see, Sequence-based models perform the worst, since they lose useful structural information in graphs. Graph-based models get better results with varied graph encoders to capture the structural information in graphs.

[2] We also implement a version which considers both $\mathbf{C}^N$ and $\mathbf{E}^N$, and achieve similar results

[3] Our code is available at https://github.com/ybz79/AMR2text

Models	LDC2015E86		LDC2017T10	
	BLEU	Meteor	BLEU	Meteor
Sequence-Based Model				
Seq2Seq (Konstas et al., 2017)	22.00	–	–	–
Syntax+S2S (Cao and Clark, 2019)	23.50	–	26.80	–
Graph-Based Model				
Graph LSTM (Song et al., 2018)	23.30	–	–	–
GCNSEQ (Damonte and Cohen, 2019)	24.40	23.60	24.54	24.07
Dual Graph (Ribeiro et al., 2019)	24.32	30.53	27.87	33.21
DCGCN (Guo et al., 2019)	25.70	31.50	27.60	34.00
Transformer-Based Model				
Transformer (Zhu et al., 2019)	25.50	33.16	27.43	34.62
Graph Transformer (Cai and Lam, 2019)	27.40	32.90	29.80	35.10
Structural Transformer (SA) (Zhu et al., 2019)	29.66	35.45	31.54	36.02
Our Approach				
Line Graph + MixGAT, $K = 1$	28.64	34.51	29.96	35.15
Line Graph + MixGAT, $K = 2$	29.62	35.38	31.06	36.13
Line Graph + MixGAT, $K = 4$	**30.58**	**35.81**	**32.46**	**36.78**

Table 1: Main results of our approaches and several baselines on the test sets of LDC2015E86 and LDC2017T10

Transformer-based models reach previous state-of-the-art with structure-aware self-attention approach to better modeling the relations between indirectly connected concepts. Comparing to previous studies, our approach with $K = 4$ order neighborhood information reaches the best BLEU scores, improving over the state-of-the-art model (Zhu et al., 2019) by 0.92 on both datasets. Similar phenomena can be found on the additional metrics of Meteor.

5 Analysis

As mentioned above, our system has two critical points: higher-order graph neighborhood information and relationships between AMR edges. To verify the effectiveness of these two settings, we conduct a series of ablation tests based on different characteristics of graphs.

Orders	LDC2015E86	LDC2017T10
K=1	24.91%	31.03%
K=2	33.93%	40.71%
K=4	41.67%	48.30%

Table 2: The connectivity of the concept graphs under different orders.

5.1 Ablation Study on Neighborhood information

Higher order neighborhood information includes the relationships between indirectly connected nodes. Table 2 shows the connectivity of the concept graphs under different orders. When $K = 1$, each node can reach 24.91% of the other nodes directly in the graph (LDC2015E86), and it grows to 41.67% when $K = 4$.

As suggested in Table 1, if graph nodes only interact with their direct neighbors ($K = 1$), it performs worse than previous Transformer-based models. However, significant improvement can be observed when we integrate higher-order neighborhood information. As K grows form 1 to 4, the BLEU score increases 1.94 and 2.50 on LDC2015E86 and LDC2017T10, respectively.

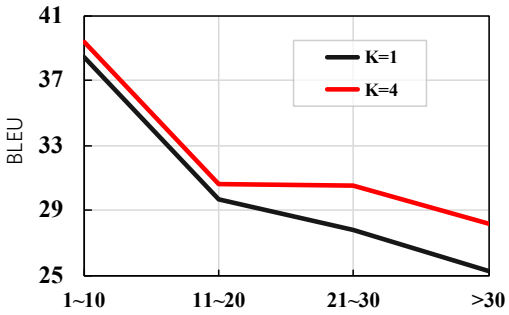

Figure 5: BLEU variation between models with different orders K with respect to AMR graph size.

As mentioned above, if only consider the first-order neighborhood, the dependencies between distant AMR concepts cannot be fully explored when the graph size becomes larger. To verify this hypothesis, we split the test set into different parts according to the AMR graph size (i.e. number of concepts). We evaluate our models with order

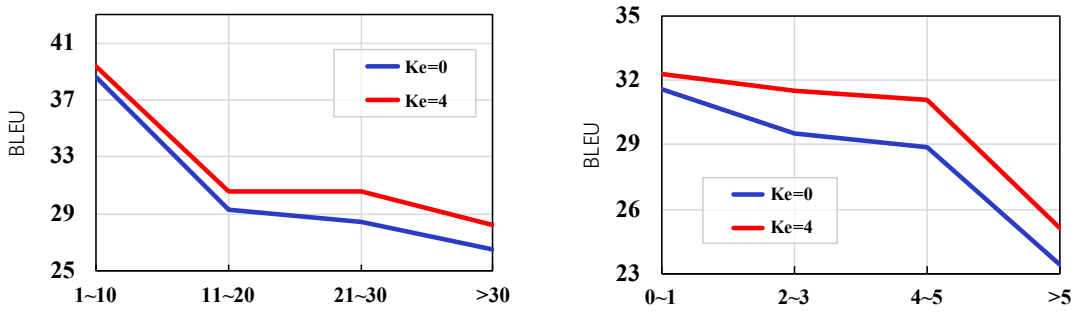

Figure 6: BLEU variation between models with different K_e with respect to size of AMR graph and (left) and reentrancy numbers (right).

$K = 4$ and $K = 1$ on different partitions. All models are trained on LDC2015E86 set. Figure 5 shows the result. The model with $K = 4$ significantly outperforms the one with $K = 1$. Furthermore, we can find that the performance gap between the two models increases when the graph gets bigger. As a result, higher-order neighborhood information does play an important role in graph-to-sequence generation, especially for larger AMR graphs.

5.2 Ablation Study on Relationships of Labeled Edges

We are the first one to consider the relationships between labeled edges in AMR graph by integrating the line graph (relation graph) G_e in our system. This section will deeply analyze the effectiveness of this contribution. In previous settings, the graph neighborhood order K is the same for both G_c and G_e. To conduct the ablation test, we fix the neighborhood order K_c for G_c and vary the order K_e for relation graph G_e. We set $K_e = 0, 1$ and 4, where $K_e = 0$ indicates that the relation nodes in G_e can only interact with itself. This means the dependencies between AMR edges are completely ignored, and the edge information is simply combined with the corresponding concepts. We report the results on both test sets in Table 3.

	LDC2015E86		LDC2017T10	
Models	BLEU	Meteor	BLEU	Meteor
$K_c = 4, K_e = 0$	28.89*	35.00	31.08*	36.11
$K_c = 4, K_e = 1$	29.41*	35.29	31.35*	36.18
$K_c = 4, K_e = 4$	**30.58**	**35.81**	**32.46**	**36.78**

Table 3: Results of models with varied neighborhood orders of relation graph G_e. BLEU scores significantly different from the best model is marked with * ($p < 0.01$), tested by bootstrap resampling (Koehn, 2004).

If we ignore the dependencies between AMR edges (K_e = 0), there is a significant performance degradation: 1.69 and 1.38 BLEU score decline on LDC2015E86 and LDC2017T10 respectively. The performance gets better when $K_e > 0$, which means the edge relations do bring benefits to the graph encoding and sequence generation. When $K_e = 4$, the edge relations are fully explored in varied neighborhood orders, and it reaches the best performance on both datasets. Performance test on different partitions of AMR graph size (Figure 6, left) also suggests that relationships of edges are helpful when the graph becomes larger.

We also study the effectiveness of edge relations when handling reentrancies. Reentrancies are the nodes with multiple parents. Such structures are identified as very difficult aspects in AMR graph (Damonte and Cohen, 2019). We think the relation graph G_e is helpful in exploring different dependencies with the same concept, which can bring benefits to those graphs containing more reentrancies. To test this hypothesis, we also split the test set into different parts according to their numbers of reentrancies and evaluate our models with $K_e = 4$ and $K_e = 0$ on different partitions. As shown in Figure 6 (right), the gap becomes wide when the number of reentrancies grows to 5. Also, compare to the graph size, edge relations are more important in handling graphs with reentrancies.

5.3 Case Study

To gain insight into the model performance. Table 4 provides a few examples. The reentrancies in the AMR graphs is marked with bold type.

In Example (a), two different nodes have same concept – `compete`, but they have different forms in the corresponding natural language. According to the references, one is for "competitors" and the other is for "competition". Our model with $K_e = 0$ fails to distinguish the difference and generate two

```
(f / feel-02
    :ARG0 (h / he)
    :ARG1 (p / person
        :quant (m / more)
        :ARG0-of (c / compete-01)
        :ARG1-of (n / new-01)
        :source (c2 / country
            :poss (w / we)))
        :ARG0-of (p2 / participate-01
            :ARG1 (c3 / compete-01
                :mod (t / this)))))
```

Reference: he felt that , there were more new competitors from our country participating in this competition .

$K_e = 0$: he feels more competition from our country who participate in this competition .
$K_e = 4$: he feels that more new competitors from our country who participate in this competition .

(a)

```
(c / contrast-01
    :ARG1 (w / want-01
        :ARG0 (t / they)
        :ARG1 (m / money))
    :ARG2 (w2 / want-01 :polarity -
        :ARG0 (t / they)
        :ARG1 (f / face)))
```

Reference: they want money , not the face

$K_e = 0$: they want money but do n't want to face .
$K_e = 4$: they want the money , not the face .

(b)

```
(p / possible-01
    :ARG1 (h / help-01
        :ARG0 (p2 / person)
        :ARG1 (y / you))
    :condition (t / tell-01
        :ARG0 (y / you)
:ARG2 (p2 / person)))
```

Reference: if you tell people they can help you,
GCNSEQ: if you tell them , you can help you !
ST-Transformer: if you tell people , people can help you !

Ours ($K_e = 4$): people can help you if you tell them ...

(c)

Table 4: Examples comparison between (a) Our approach with different K_e. (b) Our approach and several baselines.

"competition" in the output. However, model with $K_e = 4$ successfully recover word "competitors" from the context of the AMR graph.

In Example (b), the concept `they` has two parents with the same concept – `want`. Though our model with $K_e = 0$ successfully finds `they` is the subject of the both two `want`, it fails to recognize the parallel relationship between the objects `money` and `face` and regard `face` as a verb. In the contrast, our model with $K_e = 4$ perfectly finds the parallel structure in the AMR graph and reconstructs the correct sentence.

In Example (c), we compare our best model with two baselines: GCNSEQ (Damonte and Cohen, 2019) and Structural Transformer (Denote as ST-Transformer) from Zhu et al. (2019). The AMR graph in Example (b) has two reentrancies, which makes it more difficult to recover the corresponding sentence. As we can see, traditional graph-based model GCNSEQ cannot predict the correct subject of the predicate `can`. Structural-Transformer uses the correct subject, but the recovered sentence is quite disfluent because of the redundant `people`. This overgeneration problem is mainly caused by reentrancies (Beck et al., 2018). However, our model can effectively handle this problem and generates a proper sentence with correct semantics.

6 Related Work

AMR-to-text generation is a typical graph-to-sequence task. Early research employs rule-based methods to deal with this problem. Flanigan et al. (2016) use two-stage method by first split the graphs into spanning trees and use multiple tree transducers to generate natural language. Song et al. (2017) use heuristic extraction algorithm to learn graph-to-string rules. More works frame graph-to-sequence as a translation task and use either phrase-based (Ferreira et al., 2017; Pourdamghani et al., 2016) or neural-based (Konstas et al., 2017) models. These methods usually need to linearize the input graphs by means of a depth-first traversal. Cao and Clark (2019) get a better sequence-based model by leveraging additional syntactic information.

Moving to graph-to-sequence approaches, Marcheggiani and Perez-Beltrachini (2018) first show that graph neural networks can significantly improve the generation performance by explicitly encoding the structure of the graph. Since than, models with variant graph encoders have been proposed in recent years, such as graph LSTM (Song et al., 2018), gated graph neural networks (GGNN) (Beck et al., 2018) and graph convolutional neural networks (Damonte and Cohen, 2019). Guo et al. (2019) introduce dense connectivity to allow the information exchange across different of layers. Ribeiro et al. (2019) learn dual representations capturing top-down and bottom-up adjuvant view of the graph, and reach the best performance in graph-based models.

Despite the great success of graph neural net-

works, they all restrict the update of node representation based on only first-order neighborhood and rely on stacked layers to model the relationships between indirectly connected nodes. To solve this problem, recent studies extend the Transformer (Vaswani et al., 2017) to encode the graph structure. Zhu et al. (2019) and Cai and Lam (2019) use relation-aware self-attention to encode structural label sequences of concept pairs, which can model arbitrary concept pairs no matter whether directly connected or not. With several mechanisms such as sub-word (Sennrich et al., 2016) and shared vocabulary, Zhu et al. (2019) achieved state-of-the-art performance on this task.

Our model follows the same spirit of exploring the relations between indirectly connected nodes, but our method is substantially different: (1) we use a graph-based method integrated with higher-order neighborhood information while keeping the explicit structure of graphs. (2) we first consider the relations between labeled edges by introducing line graphs.

7 Conclusion and Future Work

In this work, we presented a novel graph-to-sequence approach which uses line graph to model the relationships between labeled edges from the original AMR graph. The mix-order graph attention networks are found effective when handling indirectly connected nodes. The ablation studies also demonstrate that exploring edge relations brings benefits to graph-to-sequence modeling. Furthermore, our framework can be efficiently applied to other graph-to-sequence tasks such as WebNLG (Gardent et al., 2017) and syntax-based neural machine translation (Bastings et al., 2017). In future work we would like to do several experiments on other related tasks to test the versatility of our framework. Also, we plan to use large-scale unlabeled data to improve the performance further.

Acknowledgments

We thank the anonymous reviewers for their thoughtful comments. This work has been supported by the National Key Research and Development Program of China (Grant No. 2017YFB1002102) and Shanghai Jiao Tong University Scientific and Technological Innovation Funds (YG2020YQ01).

References

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 *LAW@ACL*, pages 178–186. The Association for Computer Linguistics.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: an automatic metric for MT evaluation with improved correlation with human judgments. In *IEEvaluation@ACL*, pages 65–72. Association for Computational Linguistics.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In *EMNLP*, pages 1957–1967. Association for Computational Linguistics.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *ACL (1)*, pages 273–283. Association for Computational Linguistics.

Deng Cai and Wai Lam. 2019. Graph transformer for graph-to-sequence learning. *arXiv preprint arXiv:1911.07470*.

Kris Cao and Stephen Clark. 2019. Factorising AMR generation through syntax. In *NAACL-HLT (1)*, pages 2157–2163. Association for Computational Linguistics.

Marco Damonte and Shay B. Cohen. 2019. Structural neural encoders for amr-to-text generation. *CoRR*, abs/1903.11410.

Thiago Castro Ferreira, Iacer Calixto, Sander Wubben, and Emiel Krahmer. 2017. Linguistic realisation as machine translation: Comparing different MT models for amr-to-text generation. In *INLG*, pages 1–10. Association for Computational Linguistics.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime G. Carbonell. 2016. Generation from abstract meaning representation using tree transducers. In *HLT-NAACL*, pages 731–739. The Association for Computational Linguistics.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The webnlg challenge: Generating text from RDF data. In *INLG*, pages 124–133. Association for Computational Linguistics.

Zhijiang Guo, Yan Zhang, Zhiyang Teng, and Wei Lu. 2019. Densely connected graph convolutional networks for graph-to-sequence learning. *TACL*, 7:297–312.

Frank Harary and Robert Z Norman. 1960. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, 9(2):161–168.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Vincent Nguyen, Jean Senellart, and Alexander M. Rush. 2018. Opennmt: Neural machine translation toolkit. In *AMTA (1)*, pages 177–184. Association for Machine Translation in the Americas.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *EMNLP*, pages 388–395. Association for Computational Linguistics.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: sequence-to-sequence models for parsing and generation. In *ACL (1)*, pages 146–157. Association for Computational Linguistics.

Kexin Liao, Logan Lebanoff, and Fei Liu. 2018. Abstract meaning representation for multi-document summarization. In *COLING*, pages 1178–1190. Association for Computational Linguistics.

Diego Marcheggiani and Laura Perez-Beltrachini. 2018. Deep graph convolutional encoders for structured data to text generation. In *INLG*, pages 1–9. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318. ACL.

Nima Pourdamghani, Kevin Knight, and Ulf Hermjakob. 2016. Generating english from abstract meaning representations. In *INLG*, pages 21–25. The Association for Computer Linguistics.

Leonardo F. R. Ribeiro, Claire Gardent, and Iryna Gurevych. 2019. Enhancing AMR-to-text generation with dual graph representations. In *EMNLP-IJCNLP*. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *ACL*. The Association for Computer Linguistics.

Linfeng Song, Daniel Gildea, Yue Zhang, Zhiguo Wang, and Jinsong Su. 2019. Semantic neural machine translation using AMR. *TACL*, 7:19–31.

Linfeng Song, Xiaochang Peng, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2017. Amr-to-text generation with synchronous node replacement grammar. In *ACL (2)*, pages 7–13. Association for Computational Linguistics.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for amr-to-text generation. In *ACL (1)*, pages 1616–1626. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*, pages 5998–6008.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*. OpenReview.net.

Jie Zhu, Junhui Li, Muhua Zhu, Longhua Qian, Min Zhang, and Guodong Zhou. 2019. Modeling graph structure in transformer for better AMR-to-text generation. In *EMNLP-IJCNLP*. Association for Computational Linguistics.