Generalized Shortest-Paths Encoders for AMR-to-Text Generation

Lisa Jin and **Daniel Gildea**
Department of Computer Science
University of Rochester
Rochester, NY 14627

Abstract

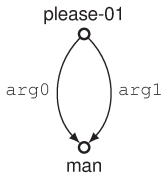
For text generation from semantic graphs, past neural models encoded input structure via gated convolutions along graph edges. Although these operations provide local context, the distance messages can travel is bounded by the number of encoder propagation steps. We adopt recent efforts of applying Transformer self-attention to graphs to allow global feature propagation. Instead of feeding shortest paths to the vertex self-attention module, we train a model to learn them using generalized shortest-paths algorithms. This approach widens the receptive field of a graph encoder by exposing it to all possible graph paths. We explore how this path diversity affects performance across levels of AMR connectivity, demonstrating gains on AMRs of higher reentrancy counts and diameters. Analysis of generated sentences also supports high semantic coherence of our models for reentrant AMRs. Our best model achieves a 1.4 BLEU and 1.8 CHRF++ margin over a baseline that encodes only pairwise-unique shortest paths.

1 Introduction

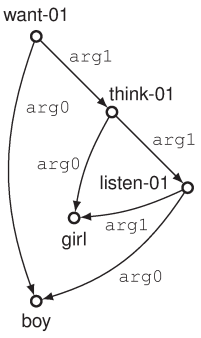
Text generation from sequences manifests in tasks such as story generation, summarization, and question-answering. Generation from semantic structures including Abstract Meaning Representation (Banarescu et al., 2013) instead involves graph inputs. An AMR is a rooted, tree-like structure that captures propositional meaning through labeled vertices and edges called concepts and relations. In this setting a model must recover the original sentence annotated by the AMR, which lacks syntactic specification (e.g., tense and plurality). The model must grapple with the denser connectivity of semantic graphs in addition to ambiguity in predicting the surface string.

The arbitrary structure of edges in a semantic graph allows it to express meaning compactly, yet also presents a challenge in learning global representations. Given sequential inputs, a neural encoder can repeatedly apply recurrent or convolutional operations to encode full strings. This local function sharing does not transfer easily to the varied neighborhoods of graphs, however.

Previous neural AMR-to-text models (Beck et al., 2018; Song et al., 2018) have favored graph neural networks (GNNs) that iteratively update each vertex's state as a function of those in its first-order neighborhood. To capture paths within the AMR, these models include gated recurrent units within the GNN vertex update function. Since paths may contain word order information, encoding them faithfully could help a model's decoder flatten the input graph into a target sequence. Though a GNN applies to arbitrary graphs, its number of stacked layers controls the length of encoded paths. Tuning this parameter is non-trivial since it depends on the size distribution of training graphs.



(a) The man is pleased with himself.



(b) The boy wants the girl to think he is listening to her.

Figure 1: Sample AMRs.

Rather than preserving state between distant vertices using RNNs, another approach is to bring vertex pairs closer together. This idea has been popularized by Transformer (Vaswani et al., 2017) self-attention networks (SANs) on sequences. Relaxing the linear nature of RNNs, SANs support parallelized attention on any subset of tokens per encoding step. The SAN's blindness to sequential positions seems ideal for unordered graph vertices. However, this lack of order also precludes the use of relative position embeddings (Shaw et al., 2018) to distinguish graph vertices within self-attention. Thus, the clique formed by vertex-level attention coefficients may obscure true graph structure.

In spite of the obstacles, recent AMR-to-text models have surpassed GNN-based encoders by using Transformer-based ones. Wang et al. (2020) and Yao et al. (2020) apply a form of graph attention networks (Veličković et al., 2017) that mask out non-adjacent vertices in self-attention layers. Similar to GNNs, these encoders only propagate vertex state through local neighborhoods. Zhu et al. (2019) remove the constraint by exploring relation encoders (e.g., SAN- or CNN-based) between all vertex pairs. Cai and Lam (2020) similarly learn relation embeddings from pairwise shortest paths fed to the model. They maintain that richer modeling of paths injects global graph structure into self-attention between vertices.

In this work, we propose to include a mixture of paths in the relation encoding of each vertex pair. This can be done by applying generalized shortest-paths algorithms—Floyd-Warshall and adjacency matrix multiplication. We draw inspiration from a generalized shortest-paths framework (Mohri, 2002) to learn relation embeddings. The benefits of this approach are threefold: (i) shortest paths need not be precomputed, (ii) the relation encoder can consider more than one path per vertex pair, and (iii) graph paths can be chosen based on properties besides distance. The latter is useful as shortest paths may oversimplify input structure of more densely connected AMRs.

A graph with reentrancies, or instances where a vertex has multiple incoming edges, can contain competing shortest paths between a vertex pair. For example, in Figure 1a the concept 'man' has two incoming edges from 'please-01'. Dropping either path between the two concepts would alter the AMR's meaning. Shortest paths may also inadequately express the relation between concept pairs. Figure 1b shows two possible paths between 'want-01' and 'boy'. The longer path in fact describes the boy's role in the subgraph rooted at 'think-01'—without which the model may fail to produce the pronoun *he*. Our relation encoder avoids both illustrated issues since it considers all graph paths.

2 Related Work

Initial work in AMR-to-text generation relied on grammar-based approaches. Flanigan et al. (2016) used tree transducers to map between spanning tree representations of AMRs and strings. These transducers generalize finite-state transducers (FSTs) to recursively rewrite trees as strings. Song et al. (2017) bypassed tree extraction and learned a synchronous node replacement grammar (SNRG) to align graph and string fragments directly. Though such approaches were well-founded, the grammars suffered from data sparsity due to the structural complexity of graphs relative to corpus size.

Amid the dominance of attentive recurrent models, Konstas et al. (2017) linearized AMRs and trained sequence transduction models in both the parsing and generation directions. They applied a paired training procedure such that the generator could be trained on synthetic data produced by the parser (and vice versa). Gated GNNs (Beck et al., 2018; Song et al., 2018) soon replaced the sequential encoders, reducing the need to compensate for noisy linearizations with synthetic data. Damonte and Cohen (2019) evaluated the use of GNNs over simpler sequential and tree encoders, showing that encoding reentrancies boosts generation performance. They further found that stacking a recurrent layer above a GNN is superior to previous gated GNNs, verifying the importance of modeling AMR sequential dependencies.

Recently, Transformer self-attention has been adapted to encode AMRs for generation tasks. As previously noted, the worst-case communication cost between vertices is constant in SANs as opposed to linear in GNNs. Yet treating the input as a clique has the side-effect of weakening a model's grasp of AMR structure and long-range paths. We build on the relation encoder developed by Cai and Lam (2020) to bias self-attention on all paths in an AMR. Our methods mediate between the efficiency gains of SAN-based graph encoders and the need to capture fine-grained AMR topology. Progress in past models attests to the utility of high resolution path encoding for the language modeling objective.

New lines of work have modified aspects of AMR-to-text generation besides the graph encoder. Song et al. (2020) explore autoencoder-style loss terms that encourage decoder states to reconstruct the graph. Ribeiro et al. (2020) avoid graph encoding entirely, instead applying pretrained language models on linearized AMRs. These additions of input reconstruction and pretrained embedding are orthogonal to the graph encoder changes we introduce.

3 Graph Transformer

Here we establish a base adaptation of the Transformer to a graph-to-sequence model. Due to discrepancies between sequences and graphs, we highlight efforts to make the encoder SAN structure-aware.

3.1 Encoder

For sequence transduction let $\mathcal{X} = \{\mathbf{x}_1 \dots \mathbf{x}_n\}$ denote source token embeddings and $\mathcal{Y} = \{\mathbf{y}_1 \dots \mathbf{y}_m\}$ the targets. For source token $\mathbf{x}_j$ and context $\mathbf{x}_i$, the encoder SAN by Vaswani et al. (2017) applied dot-product attention between query vector $\mathbf{q}_j = W_q\mathbf{x}_j$ and key vector $\mathbf{k}_i = W_k\mathbf{x}_i$ as

$$\alpha_{i,j} = \text{softmax}_i \left(\mathbf{q}_j^\top \mathbf{k}_i / \sqrt{d} \right), \tag{1}$$

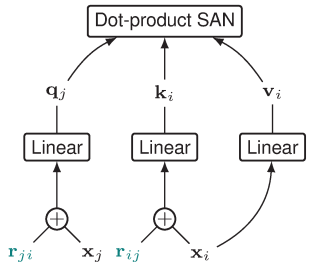


Figure 2: Encoder SAN with relation embeddings.

where W_q, W_k are learned projection matrices and d is the embedding dimension. The attention coefficients $\alpha_{\cdot,j}$ are used to retrieve a weighted average of value vectors $\mathbf{v}_i = W_v\mathbf{x}_i$ as $\mathbf{z}_j = \sum_{i=1}^{n} \alpha_{i,j}\mathbf{v}_i$ using parameters W_v. The token embeddings $\mathcal{X}$ are biased with encodings that indicate relative position to make the SAN sequence-aware. Multi-head attention allows for multiple sets of key, query, and value projection matrices to be learned and aggregated in parallel. After applying attention, each SAN layer also employs several linear projections, a residual connection, and layer normalization to update the embeddings $\mathcal{X}$ fed into the next layer.

Note that the relative position of any token pair is equal to the length of the path connecting them, which lets the positional bias specify a token sequence. In the case of unordered graph vertices $\mathcal{X}$, defining the structure between any pair of vertex embeddings requires more care. The existence of numerous paths between vertex pairs nullifies the notion of relative position. Whereas an RNN limits the context of each token $\mathbf{x}_i$ to $\{\mathbf{x}_1 \dots \mathbf{x}_{i-1}\}$ and a GNN allows each vertex to receive messages from only its neighbors, an SAN lacks such spatio-temporal control.

Relation encoder To make the SAN structure-aware, several models (Zhu et al., 2019; Cai and Lam, 2020) have learned a unique relation embedding $\mathbf{r}_{ij}$ for each vertex pair. This embedding participates in computing the attention coefficients $\alpha_{i,j}$, typically via addition with $\mathbf{x}_j$ and/or $\mathbf{x}_i$. As the baseline we adopt the formulation of Cai and Lam (2020):

$$\alpha_{i,j} = \text{softmax}_i \left(\left(W_q \left(\mathbf{x}_j + \mathbf{r}_{ji} \right) \right)^\top W_k \left(\mathbf{x}_i + \mathbf{r}_{ij} \right) / \sqrt{d} \right).$$

They encode $\mathbf{r}_{ij}$ using bidirectional GRUs along the path of relation labels between vertices i and j. The AMR is augmented with backward relations (labeled accordingly), self loops, and a global vertex with vector $\mathbf{x}_0$ that connects to all existing vertices. In contrast to the original directed acyclic graph (DAG) structure, the authors claim that this unrestricted information flow aids both vertex- and graph-level representation learning.

3.2 Decoder

After encoding AMR vertices $\mathcal{X}$ using an SAN, the sequence decoder is largely unchanged from that of the original Transformer. At each output position $j \in [1, m]$, the decoder not only applies masked self-attention within target tokens $\{\mathbf{y}_1 \dots \mathbf{y}_j\}$ but also cross-attention on all n encoded vectors $\mathcal{X}$. Cross-attention is implemented using eq. 1, except that query vector $\mathbf{q}$ is a function of decoder outputs $\mathcal{Y}$ while key $\mathbf{k}$ and value $\mathbf{v}$ arise from $\mathcal{X}$. This way the decoder can efficiently attend to valid gold target tokens in tandem with all source tokens.

As in the model by Cai and Lam (2020), we initialize the decoder state at each step j with global vertex embedding $\mathbf{x}_0$ described in §3.1. This injects global content learned from the AMR along the length of the predicted token sequence. In a previous gated RNN model (Song et al., 2017), the first decoder state similarly consisted of an average over encoder vertex states. Following the decoder initialization, stacked layers of multi-head attention proceed as before.

During token prediction, the AMR-to-text decoder also includes a copy mechanism (Gu et al., 2016) that is common among neural text generation models. The mechanism addresses data sparsity of rare tokens (e.g., numbers, dates, named entities) by allowing the model to directly copy from AMR concept labels. The probability of drawing from the copy distribution as opposed to generating from the target vocabulary is found via a single feedforward layer: $p_{\text{copy}} = \text{softmax}(W_c \mathbf{h}_j)$ for state $\mathbf{h}_j \in \mathbb{R}^{d_t}$ of token j and learned weights $W_c \in \mathbb{R}^{2 \times d_t}$. Given target vocabulary distribution $p'(y \mid \mathbf{h}_j)$ stemming from decoder attention, the final token distribution is computed as

$$p(y \mid \mathbf{h}_j) = (1 - p_{\text{copy}}) p'(y \mid \mathbf{h}_j) + p_{\text{copy}} \sum_{i \in \mathcal{C}(y)} \alpha_{i,j}, \tag{2}$$

where $\mathcal{C}(y)$ contains the set of concepts whose labels correspond to y and attention distribution $\alpha_{\cdot,j}$ is over labeled source concepts.

4 Generalized Shortest-Paths Encoders

We now discuss how the relation encoder incorporates all possible AMR paths using generalized shortest-paths algorithms. The goal is to learn comprehensive pairwise path embeddings that inform the graph encoder SAN. This approach relegates the choice of 'best' paths between vertices to the model instead of merely including them as model input. $G = (V, E)$ denotes a directed AMR graph with vertex set $V = \{v_1 \dots v_n\}$ and edge set $E = \{(u, v) \mid u, v \in V\}$.

4.1 Floyd-Warshall

The Floyd-Warshall algorithm solves the all-pairs shortest-paths problem in time $\mathcal{O}(n^3)$. It iteratively relaxes the shortest path between each vertex pair i and j by introducing a new potential intermediate vertex $k \in [1, n]$. The shortest pairwise distance using only vertex set $\{1 \dots k\}$ is written as $d_{ij}^{(k)}$. At each step k the following update is made:

$$d_{ij}^{(k)} = \begin{cases} a_{ij} & \text{if } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{otherwise,} \end{cases} \tag{3}$$

where $A = (a_{ij}) \in \mathbb{R}^{\{0,1\}}$ is the adjacency matrix.

Though this procedure may not be the most efficient, it encounters all paths in G through its recursive bottom-up routine. As such, it provides a scaffold upon which a neural function can assign weights to paths for vertex self-attention. We replace the $(\min, +)$ semiring of eq. 3 on integer distances with $(+, f)$ on path embeddings, where f is a neural function that merges sub-path vectors. Parameterized by W_f and $\mathbf{b}_f$, the definition of f at step $k \in [1, n]$ is

$$f(\mathbf{r}_{ik}^{(k-1)}, \mathbf{r}_{kj}^{(k-1)}) = \text{ReLU}(W_f [\mathbf{r}_{ik}^{(k-1)}; \mathbf{r}_{kj}^{(k-1)}] + \mathbf{b}_f). \tag{4}$$

After linearly projecting the concatenated sub-path vectors, the result is passed through a ReLU nonlinearity. Path embeddings between vertices i and j are summed together over indices k via

$$\mathbf{r}_{ij}^{(k)} = \mathbf{r}_{ij}^{(k-1)} + \mathbb{1}\{k > 0\} \cdot f(\mathbf{r}_{ik}^{(k-1)}, \mathbf{r}_{kj}^{(k-1)}), \tag{5}$$

where $\mathbf{R}^{(0)} = (\mathbf{r}_{ij}^{(0)})$ contains relation label embeddings based on adjacency matrix A. This formulation allows the relation encoder to sum over all path representations, each computed using the neural function f. The final relation embedding combines the first and last k-indexed path embeddings through concatenation and linear projection, parameterized by W_o:

$$\mathbf{r}_{ij} = W_o [\mathbf{r}_{ij}^{(0)}; \mathbf{r}_{ij}^{(n)}] + \mathbf{b}_o. \tag{6}$$

4.2 Adjacency Matrix Multiplication

An alternative to Floyd-Warshall is repeated adjacency matrix multiplication. In this case, the algorithm extends the current shortest paths by one edge and compares them to those of the previous iteration. Indicating the path length by m, the assignment made for the shortest distance between vertex i and j is

$$d_{ij}^{(m)} = \begin{cases} a_{ij} & \text{if } m = 1 \\ \min(d_{ij}^{(m-1)}, \min_k\{d_{ik}^{(m-1)} + a_{kj}\}) & \text{otherwise.} \end{cases} \tag{7}$$

We try to shorten $d_{ij}^{(m-1)}$, the best path of length at most $m-1$, by extending prefix $d_{ik}^{(m-1)}$ by edge a_{kj}. This can be generalized as matrix multiplication if we replace $(\min, +)$ with $(+, f)$ as in §4.1:

$$d_{ij}^{(m)} = \sum_{k=1}^{n} d_{ik}^{(m-1)} a_{kj},$$

which is simply $D^{(m)} = D^{(m-1)}A$.

The operations between all rows of D and columns of A can be extended to compute relation embeddings $\mathbf{r}_{ij}$. For initial relation label embeddings $\mathbf{R}^{(0)}$, the below recurrence finds $\mathbf{R}^{(m)}$ for $m \in [1, n]$:

$$\mathbf{r}_{ij}^{(m)} = \sum_{k=1}^{n} f(\mathbf{r}_{ik}^{(m-1)}, \mathbf{r}_{kj}^{(0)}), \tag{8}$$

where f is the sub-path composition function in eq. 4. Similarly to eq. 6 an output projection layer is applied to the final $\mathbf{r}_{ij}^{(n)}$.

Relation self-attention Since f in eq. 8 is an expensive linear projection over two embeddings, we propose to push the sum over k into f. This means that f is computed once per path, and we sum over all possible subpath pairs $\langle i \dots k \rangle$, $\langle k \dots j \rangle$ using attention over k. Specifically, at step m, the attention mechanism uses step $(m-1)$'s path embedding for $\langle i \dots j \rangle$ to softly select path prefixes $\langle i \dots k \rangle$, $\forall k \in [1, n]$. The attention coefficient for a particular path $\langle i \dots k \dots j \rangle$ is found as

$$\alpha'_{i,k,j} = \text{softmax}_k \left(\left(W'_q \mathbf{r}_{ij}^{(m-1)} \right)^\top W'_k \mathbf{r}_{ik}^{(m-1)} \right).$$

Now eq. 8 can be written in terms of path prefix $\mathbf{p}_{ij}$ and suffix $\mathbf{s}_{ij}$, which are each a function of $\alpha'_{i,k,j}$:

$$\mathbf{r}_{ij}^{(m)} = \text{ReLU}\left(W_p \left[\mathbf{p}_{ij}; \mathbf{s}_{ij} \right] + \mathbf{b}_p \right),$$

$$\mathbf{p}_{ij} = \sum_{k=1}^{n} \alpha'_{i,k,j} \mathbf{r}_{ik}^{(m-1)} \qquad \mathbf{s}_{ij} = \sum_{k=1}^{n} \alpha'_{i,k,j} \mathbf{r}_{kj}^{(0)},$$

where $W_p \in \mathbb{R}^{d \times 2d}$ is a linear projection matrix. We also apply a residual connections and layer normalization across successive steps of m. As $\langle i \dots k \rangle$ and $\langle i \dots j \rangle$ belong to the same row we can apply row-wise self-attention over $\mathbf{R}^{(m-1)}$ to efficently compute $\alpha'_{i,\cdot,j}$. Thus, our model requires $\mathcal{O}(n^2 d^2 + n^3)$ operations for eq. 8 instead of $\mathcal{O}(n^3 d^2)$. See Figure 3 for an example application of the relation SAN.

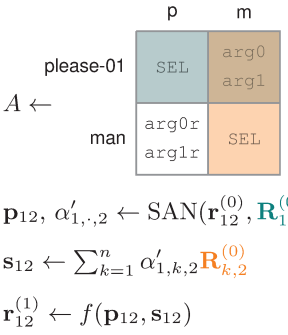
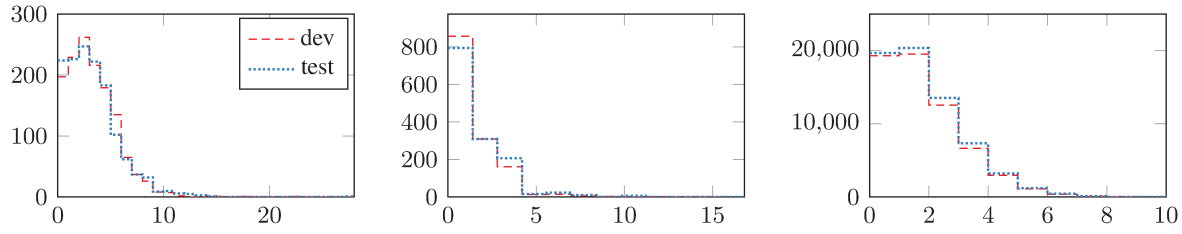


Figure 3: Relation SAN.

5 Experiments

We evaluate two generalized shortest-paths encoders, using the model by Cai and Lam (2020) as a baseline. All components besides the relation encoder are held constant to test whether feeding the graph encoder a wider scope of path embeddings improves downstream language modeling. Experiments are run on the LDC2017T10 dataset that consists of 36,521 training, 1,368 development, and 1,371 test AMR-sentence pairs. Results are reported in terms of BLEU (Papineni et al., 2002) and CHRF++ (Popović, 2017) for direct comparison to the baseline.



(a) Reentrancy count per AMR.

(b) Competing shortest path count (maximum) per AMR.

(c) Shortest path length per vertex pair.

Figure 4: Histograms for graph connectivity metrics.

5.1 Preprocessing

We follow the same vertex- and token-level preprocessing steps as the baseline model—entity simplification and anonymization—that were introduced by Konstas et al. (2017). The former includes removing sense tags and 'instance-of' relations, while the latter replaces subgraphs rooted at entity types with indexed entity labels (e.g., 'country_0') and standardizes 'date' entities. Postprocessing reverses these effects after model predictions are made.

Data attributes Since our approach targets densely connected AMR, we examine a few properties indicative of such in the dataset. Figure 4a shows the distribution of AMR reentrancy counts for development and test splits after preprocessing. 14.4% of the former and 16.3% of the latter has no reentrancies. Among reentrant AMRs, the average number of reentrancies is 3.33 and 3.43 for the two splits. Figure 4b reveals how the number of competing shortest paths is distributed over AMRs; 37.4% of development and 42.0% of test split AMRs have at least one set of tied paths. Over all AMRs the mean (maximum) number of ties is 1.63 and 1.81. In Figure 4c is the distribution of shortest path lengths. Per split, 30.8% and 29.8% of the paths are zero-length (i.e., self-loops)—without which the mean lengths are 1.97 and 2.00, respectively. In summary, most AMRs are reentrant, ∼40% of AMR have tied shortest paths, and pairwise unique vertices are on average two hops away. We explore performance relative to these attributes in §5.4.

5.2 Setup

The Floyd-Warshall (FLOWAR) and adjacency matrix multiplication (ADJMATMUL) model variants inherit the same base settings. Encoder vertex embeddings and decoder token embeddings of size 300 each are randomly initialized. Both embeddings types also include output from a character-level CNN. Model hyperparameters excluding those of the relation encoder are replicated from Cai and Lam (2020). Notably, the encoder and decoder SANs share a set of settings: hidden states of size 512, feed-forward projections of size 1024, and eight attention heads. Parameters of the model are optimized using *Adam* (Kingma and Ba, 2015) with default settings of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

FLOWAR includes relation embeddings of size 200 and a final relation projection of size 512. We found that larger relation embeddings degrade performance—perhaps contributing to overfitting. As Floyd-Warshall combines subproblems in a tree-like fashion, the model may struggle to generalize with complex embeddings across large values of k. We tried adding gated recurrent operations to eq. 5 to address this suspicion; however, it was not helpful.

ADJMATMUL has an added maximum m-value hyperparameter that limits the length of candidate shortest paths. Empirically a maximum m of 6 performed the best with performance dropping precipitously at higher values. The problem may stem from a mismatch with the corpus graph distribution; the average diameter (i.e., maximum pairwise vertex distance) per AMR is 3.43 for both evaluation splits. The best model also featured relation embeddings of size 216 and a final relation projection of size 512. The inter-relation SAN employs hidden states of size 216 and eight attention heads.

5.3 Results

Performance on AMR-to-text generation from the LDC2017T10 corpus is reported relative to the baseline model results. We outperform all previous systems in §2 that primarily target the graph encoder except those of Zhu et al. (2019) and Yao et al. (2020). The former uses a different preprocessing method that includes BPE (Sennrich et al., 2016) applied to a shared concept-token vocabulary. Their ablation studies show that this method raises their baseline's scores by 6.1 BLEU and 8.4 CHRF++ on LDC2015E86. Although these vocabulary changes may complement our proposed models, we do not include them since the authors do not release their preprocessing code. Yao et al. (2020) also applies BPE to AMR concept labels, without which their model suffers a 2.0 drop in BLEU on LDC2015E86. In addition, they apply Levi graph transformation and encode several views of the AMR graphs (e.g., reverse edges only, fully connected). Since their work is concurrent to ours, we were unable to explore the effects of these changes.

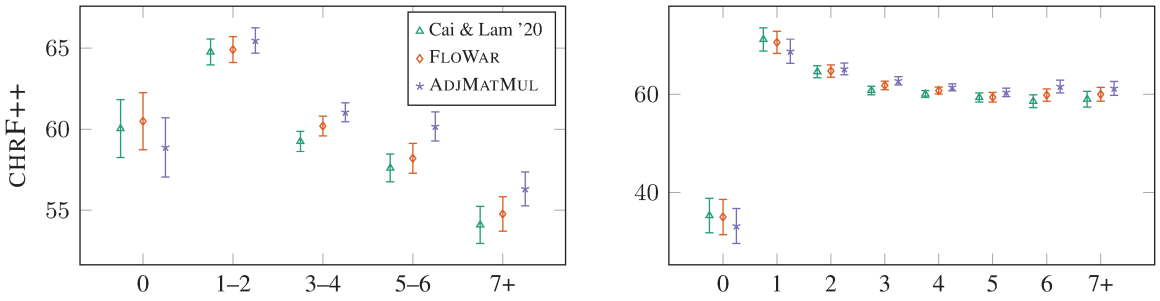
	BLEU	CHRF++
Cai and Lam (2020)	29.8	59.4
FLOWAR	30.0	60.1
ADJMATMUL	**31.2**	**61.2**

Table 1: LDC2017T10 test split scores.

As shown in Table 1, ADJMATMUL improves on Cai and Lam (2020) by 1.4 BLEU and 1.8 CHRF++. FLOWAR achieves a 0.2 BLEU and 0.7 CHRF++ boost over the baseline. It is surprising that FLOWAR is inferior to ADJMATMUL; the former captures all possible paths in an AMR, while the latter halts upon reaching a maximum path length m. The sheer number of paths encountered in FLOWAR may in fact be harmful for downstream text generation. Since consecutive sentence-aligned AMR concepts are often close together in the graph, encoding long paths may not be critical. FLOWAR's higher CHRF++ compared to the baseline indicates that it still offers a modest advantage at the character level.

5.4 Analysis

To delve deeper into how our relation encoders improve upon the baseline, we analyze how their performance varies over measures of graph connectivity. We hypothesized earlier that our generalized shortest-paths encoders offer a more complete view of AMR structure to the graph encoder.



(a) AMR reentrancy count.

(b) AMR diameter.

Figure 5: Test split average sentence-level CHRF++ over binned graph connectivity metrics.

Reentrancies In §1 we asserted that introducing multiple paths between vertex pairs (e.g., Figure 1) can be problematic for models that encode only shortest paths. We found in §5.1 that a majority of the evaluation AMRs exhibit reentrancies (Figure 4a). To show the impact of encoding such structure, Figure 5a plots average sentence-level CHRF++ for AMRs over varying bins of reentrancy counts. FLOWAR maintains a lead over the baseline across all reentrancy counts, with noticeable gains for AMRs with above two reentrancies. We speculated in §5.3 that the high volume of paths FLOWAR encodes may

hurt its performance. However, it does not appear to suffer at higher reentrancy counts even though the number of paths increases exponentially in graph size (correlated with reentrancy). The gap of 1.97 CHRF++ between ADJMATMUL and the baseline is widest for AMRs containing five or six reentrancies. This margin is present across AMRs with between one and six reentrancies, though it vanishes for non-reentrant AMRs.

Diameter The maximum distance between vertex pairs, or diameter, of an AMR is another metric of interest. It captures the worst-case communication cost between two vertices in terms of path length. Figure 5b compares trends in CHRF++ over different AMR diameters. FLOWAR consistently outperforms the baseline on diameters above one, with performance waning at lower diameters. ADJMATMUL follows the same pattern relative to the baseline as FLOWAR. It is possible that the density of paths encoded by FLOWAR and ADJMATMUL dilutes graph structure of lower diameter AMRs, whose vertices may be highly interconnected. The fact that both generalized shortest-paths encoders perform well at higher graph diameters supports their efficacy on AMRs with distant vertices.

Manual inspection For a more holistic view of performance, we inspect a few model-generated sentences. The examples in Table 2 are chosen with special attention to graph complexity. The AMR of example (2) has three reentrancies and a diameter of four, while those of (1) and (3) have three reentrancies and a diameter of three. We observe in (1) that the baseline substitutes the phrase *as opposed to* with the word *but*, which lends the reverse meaning to the sentence. FLOWAR and ADJMATMUL choose the appropriate phrases *rather than* and *instead of*, respectively. In (2) the preposition *at* in the phrase *look at the script* is replaced with *like* by the baseline. It appears that the model misplaces *like* from *life is like a show*. Both FLOWAR and ADJMATMUL are able to recover the correct reference phrase. Finally, (3) consists of several interrelated clauses and may test models' ability to synthesize global AMR information. The baseline fails to mention *chavez* in the first clause, instead linking *castro* to *cuba*. In the second clause it then inserts the irrelevant word *india* in place of *castro*. Neither FLOWAR nor ADJMATMUL drop *chavez*, and they both curiously use a pronoun to refer to *castro* in the second clause—though FLOWAR applies the wrong gender.

(1)	REF	to me that just sounds like you being overly anxious , **as opposed to having** really bad ocd .
	Cai & Lam '20	that just sounds over anxious to me , **but you have** a really bad ocd .
	FLOWAR	that just sounds over anxious to me **rather than having** really bad ocd .
	ADJMATMUL	that just sounds like you are overly anxious to me **instead of having** really bad ocd .
(2)	REF	no one is easy . life is like a show , you must have a **look at the script** to know how to perform
	Cai & Lam '20	no one is easy as a show (you have to **look like a script** to know how the performance would be performed .)
	FLOWAR	life is - like a show , but no one is easy , you have to **look at a script** to know how the performance is .
	ADJMATMUL	life like a show , you have to **look at a script** to know how performed . no one 's easy .
(3)	REF	cuban leader fidel castro is chavez 's his major ally and chavez is growing **closer to castro** .
	Cai & Lam '20	cuba 's leader fidel castro is a major ally of cuba and has grown **closer to india** .
	FLOWAR	cuba 's leader fidel castro is a major ally and chavez is growing **closer to her** .
	ADJMATMUL	cuba leader fidel castro is a major ally with chavez and has grown **closer to him** .

Table 2: Sample sentences.

In the selected sentences, FLOWAR and ADJMATMUL exhibit more semantic accuracy and cohesion than the baseline does. This difference is especially apparent in (3), which contains several relations between the entities *fidel castro* and *chavez*. Perhaps the ability of FLOWAR and ADJMATMUL to model multiple pairwise paths allows them to preserve meaning in such scenarios. Moreover, the generalized shortest-paths encoders are robust to the preposition-related errors shown in (1) and (2).

6 Conclusion

We presented two relation encoders based on generalized shortest-paths algorithms for graph encoders. Due to their expansive graph path encoding, we proposed that the encoders' more global context would

especially benefit complexly structured AMRs for text generation. Experiments revealed that their performance gains are most consistent on higher-diameter AMRs, verifying their efficacy on encoding long-distance vertex relations. Our adjacency matrix multiplication relation encoder scores above the baseline shortest-paths model in terms of both BLEU and CHRF++. Furthermore, sentence-level inspection showed that both our model variants produce more semantically sound output. Encouraging the model to learn encoder-specific parameters (e.g., maximum path length) may improve its adaptivity to variations in AMR structure and remedy its sensitivity to low AMR reentrancy. We leave this experimentation for future work.

Acknowledgments This work was funded in part by NSF award 1813823. We are also grateful for comments from Parker Riley.

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 *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL-18)*, pages 273–283.

Deng Cai and Wai Lam. 2020. Graph Transformer for graph-to-sequence learning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-20)*, pages 7464–7471.

Marco Damonte and Shay B Cohen. 2019. Structural neural encoders for AMR-to-text generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-19)*, pages 3649–3658.

Jeffrey Flanigan, Chris Dyer, Noah A Smith, and Jaime Carbonell. 2016. Generation from Abstract Meaning Representation using tree transducers. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-16)*, pages 731–739.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL-16)*, pages 1631–1640.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR-15)*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL-17)*, pages 146–157.

Mehryar Mohri. 2002. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-02)*, pages 311–318.

Maja Popović. 2017. chrF++: words helping character n-grams. In *Proceedings of the 2nd Conference on Machine Translation*, pages 612–618.

Leonardo F. R. Ribeiro, Martin Schmitt, Hinrich Schütze, and Iryna Gurevych. 2020. Investigating pretrained language models for graph-to-text generation. *arXiv preprint arXiv:2007.08426*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL-16)*, pages 1715–1725.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-18)*, pages 464–468.

Linfeng Song, Xiaochang Peng, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2017. AMR-to-text generation with synchronous node replacement grammar. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL-17)*, pages 7–13.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for AMR-to-text generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL-18)*, pages 1616–1626.

Linfeng Song, Ante Wang, Jinsong Su, Yue Zhang, Kun Xu, Yubin Ge, and Dong Yu. 2020. Structural information preserving for graph-to-text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL-20)*, pages 7987–7998.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS-17)*, pages 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. In *Proceedings of 5th International Conference on Learning Representations (ICLR-17)*.

Tianming Wang, Xiaojun Wan, and Hanqi Jin. 2020. AMR-to-text generation with Graph Transformer. *Transactions of the Association for Computational Linguistics*, 8:19–33.

Shaowei Yao, Tianming Wang, and Xiaojun Wan. 2020. Heterogeneous graph transformer for graph-to-sequence learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL-20)*, pages 7145–7154.

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 *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP-19)*, pages 5462–5471.