Natural Answer Generation via Graph Transformer

Xiangyu Li, Sen Hu, and Lei Zou✉

Peking University, Beijing, China
{xiangyu_li,husen,zoulei}@pku.edu.cn

Abstract. Natural Answer Generation (NAG), which generates natural answer sentences for the given question, has received much attention in recent years. Compared with traditional QA systems, NAG could offer specific entities fluently and naturally, which is more user-friendly in the real world. However, existing NAG systems usually utilize simple retrieval and embedding mechanism, which is hard to tackle complex questions. They suffer issues containing knowledge insufficiency, entity ambiguity, and especially poor expressiveness during generation. To address these challenges, we propose an improved knowledge extractor to retrieve supporting graphs from the knowledge base, and an extending graph transformer to encode the supporting graph, which considers global and variable information as well as the communication path between entities. In this paper, we propose a framework called G-NAG, including a knowledge extractor, an incorporating encoder, and an LSTM generator. Experimental results on two complex QA datasets demonstrate the efficiency of G-NAG compared with state-of-the-art NAG systems and transformer baselines.

Keywords: question answering · natural answer generation · graph transformer.

1 Introduction

Natural Answer Generation (NAG), which devotes to providing fluent answers in the form of natural language sentences, has received much attention in recent years. Compared with traditional question answering (QA) systems that merely offer accurate Answer Semantic Units (*ASU*) [10], NAG could satisfy users in real-world scenarios where fluency is of strong demand.

Generally, the popular NAG framework consists of three modules, as shown in Figure 1-a. Knowledge extractor recognizes the topic entity and retrieves its related triples from the underlying Knowledge Base (KB). After Knowledge encoder representing these candidate triples and the question as two sequences, Generator could generate the natural answer with an attention mechanism. Existing NAG systems have achieved some success focused on simple problems (one topic entity), such as [27, 10, 6].

However, there are still many non-trivial issues due to linguistic complexity that the above systems do not perform well. (1) In Knowledge extractor. On the one hand, existing NAG systems recognize one topic entity and retrieve its one-hop neighbors related to the question. When a question contains more entities and multi-hop relations, they may leave out some critical entities. Take Q in Figure 1 as an example, the *ASU* `Jason Statham` should be retrieved

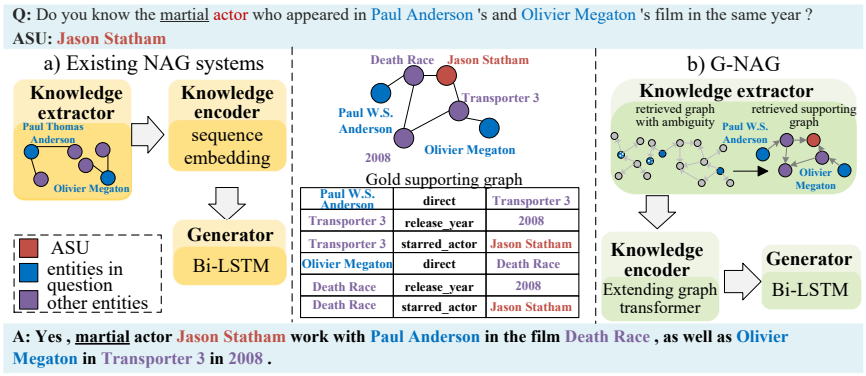

Fig. 1. Natural Answer Generation process of an example question.

through 2-hops from the mentioned entities in question so that it may be left by previous knowledge extractor with one-hop retrieval mechanism. On the other hand, without considering the global structure in KB, the above systems do disambiguation before retrieving triples. Thus, they may choose irrelevant entities far apart from others, such as `Paul Thomas Anderson` which may be confused with the correct entity `Paul W.S. Anderson` but unrelated to the question in Figure 1-a. (2) In Knowledge encoder. Previous NAG systems encode triples as a sequence, such as a list by LSTM [27, 10] or key-value structure by Memory Network [6], which is too simple to express complicated semantic information. For the same example, triple-list or key-value could not represent the topological structure of the supporting graph clearly, which is the key to generate answers logically.

We focus on these challenges above and propose some novel solutions. (1) In Knowledge extractor, we consider to retrieve multi-hop triples around mentioned entities, such as 2-hops, as some may not appear in questions but useful for answer generating. Since multi-hop retrieval may return a supporting graph with much redundancy, we propose a simplifying method based on semantic similarity, as shown in Figure 1-b. Meanwhile, we solve entity ambiguity after retrieving triples based on the global structure in KB to choose correct entities. (2) In Knowledge encoder. Since graph transformer [13] is proposed to generate summarization and achieve excellent performance, we employ an extending graph transformer as encoder, which has more capacity to encode complicated pair-wise relationships than the sequence structure. To fit the NAG problem, we introduce the communication path and two extra vertices to capture global or variable information, respectively (to be discussed in Section 2.3).

In this paper, we propose a framework called G-NAG (Graph-based NAG) to implement the generation process, which also consists of three modules, as shown in Figure 1-b. Compared with previous work, we enlarge the retrieval range before disambiguation and then propose a simplifying strategy based on semantic similarity in Knowledge extractor. Moreover, we replace the sequence encoder with a graph transformer considering communication path as well as global and variate vertices. Based on `Wikimovie` dataset [16] and `DBpedia`,

we reconstruct a QA dataset in movie domain aimed at multi-hop question answering. Experimental results on the original `Wikimovie` and new dataset demonstrate the efficiency of our model compared with state-of-the-art NAG systems and transformer baselines.

We summarize our main contributions of this paper as follow:

- We design a generation framework G-NAG, which generates natural and logical answer based on KB. To our knowledge, it is the first framework that aims at addressing complex NAG problem.
- We present a novel knowledge extractor which enlarges retrieval range before disambiguation and simplifies triples based on semantic similarity to gain the supporting graph.
- We propose an extending graph transformer to represent the supporting graph, which considers the communication path and captures global or variable information by extra vertices.
- We implement experiments on two datasets in the movie domain. The results demonstrate that G-NAG performs better compared with existing NAG approaches and transformer baselines, especially in complex natural answer generation problems.

2 Methodology

In this section, we introduce the notations employed in this paper.

Basic definition: We denote a given question as Q, and its accurate Answer Semantic Units as ASU. There is only one type of ASU for each question, i.e., the ASU maybe two actors but not an actor and a writer. The generated natural answer is denoted as A, and knowledge triples in KB are in the form $\langle s, p, o \rangle$, where s,o are entity vertices (ent) and p is relation edge (rel).

Graph definition: We define the initial graph by multi-hop retrieval as a inter-connected graph set $\mathbb{G} = [G_i = (V_q, V_o, E_i)]$, where vertex $v \in V_q$ is mentioned in question, $v \in V_o$ denotes other retrieved vertex, and E_i is a set of relation edges that link vertices. After disambiguation and graph simplifying, the final supporting graph is denoted as G. In encoding section, we convert G to an unlabeled graph $G' = (V', P')$, where V' is a set of all vertices and P' is a matrix describing communication path among vertices.

2.1 Framework overview

Our G-NAG framework consists of three modules: Knowledge Extractor, Incorporating Encoder, and Generator. We depict an overview with a concrete example in Figure 2.

In Knowledge Extractor: Given the question Q, G-NAG maps each entity phrase to its candidate linking vertices $v \in V_q$ in underlying KB. Allowing for the ambiguity of phrase linking, k-hop neighbors of these $v \in V_q$ are retrieved from KB. These triples construct a large graph, which could be divided into an inter-connected graph set $\mathbb{G}$, as illustrated in Figure 2 a-I. Then for disambiguation, we employ a cost-distance strategy as a-II. Further, G-NAG removes redundant vertices and edges by semantic similarity to acquire a simplified supporting graph G as shown in Figure 2 a-III.

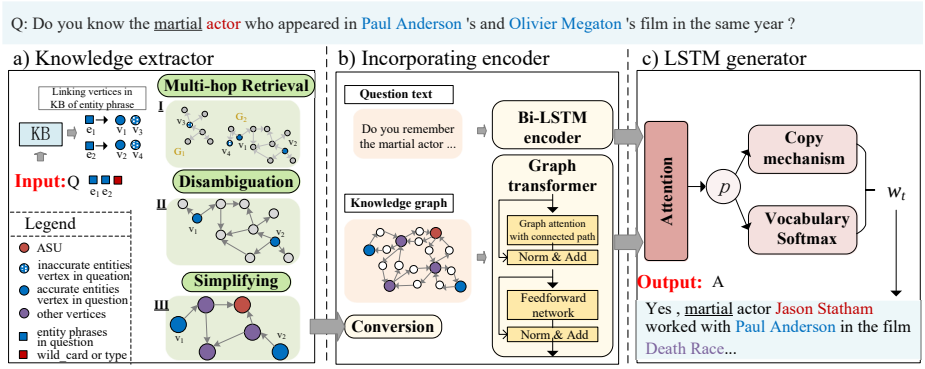

Fig. 2. Natural Answer Generation process of an example question.

In Incorporating Encoder: G-NAG obtains the embedding of supporting graph and question through concatenating a novel graph transformer (to be discussed in Section 2.3) and bi-LSTM, as shown in Figure 2-b. Specially, we consider the communication path between vertices-pair in graph attention calculation. Before encoding, we convert the supporting graph G to an unlabeled bipartite graph G', which contains global and variate vertices (to be discussed in Section 2.2).

In Generator: G-NAG predicts output word w_t at each time step t by generating from vocabulary or copying from supporting graph G and question Q via a soft switch p. As illustrated in Figure 2-c, token with underline is copied from question text, and the colored token is from the graph, while other ordinary words are generated from the vocabulary.

2.2 knowledge extractor

We propose an improved knowledge extractor in this section to provide a more accurate supporting graph. Specifically, we enlarge the extraction range by multi-hop retrieval before entity disambiguation, then solve the ambiguity based on the global graph structure, and simplify the graph by semantic similarity eventually.

In the offline phase, following a similar procedure to [25], G-NAG encodes the underlying KB into a common low-dimensional vector space. We employ TransE, which refers to that p of each triple is represented as translation operation from head vertex s to tail vertex o. Take triple $\langle s, p, o\rangle$ as an example, o should be the closest vertex of $s + p$, while semantically similar vertices should be closed as well.

Multi-hop Retrieval

Given Q, we map each entity phrase $enti$ to its candidate linking vertices $v \in V_q$ in KB, while a entity phrase $ent1$ may be matched more than one vertex as a set, such as $\mathbb{v}_{ent1} = [v_i^1] \subseteq V_q$. Allowing for the ambiguity of phrase linking temporarily, we retrieve k-hop neighbors of each linking vertex to construct a large graph. As some linking vertices are far apart in KB, the large graph could be divided into a graph set $\mathbb{G} = [G_i = (V_q, V_o, E_i)]$ where G_i are unconnected to each other, V_q denotes linking vertices for entity phrase in question and V_o denotes the other vertices retrieved. Factual questions usually have only a group

of core multi-hop relationships, that is, the distances between exact entities are all within a fixed number of hops, so target entities are rarely distributed on two completely different graphs G_i.

Entity Disambiguation
In this stage, G-NAG deals with the ambiguous vertices in V_q. To ensure the integrity of the supporting graph, we remain at most M graphs in $\mathbb{G}$ with more linking entity phrases (Assume n graphs cover all entity phrases and denote m as a parameter). Then considering one of the remaining graphs G_i, we compute its cost motivated by [25] formulated as follow:

$$M = max(m, n), \quad Cost_{G_i} = \sum_{(s,p,o) \in G_i} \|s + p - o\|_2^2 \tag{1}$$

Because of the cumulative effect of error, the candidate G with the minimum cost could be selected with the strongest internal relevance.

Moreover, we propose a minimum-distance method to delete redundant linking vertices in G for each entity phrase. Take $\mathbb{v}_{ent1} = [v_1, v_2]$ in Figure 3-a as an example, we define the shortest path (number of edges) between v_1 and each vertex of $\mathbb{v}_{ent2}$ as the minimum-distance between v_1 and $\mathbb{v}_{ent2}$. Then we rank vertices v in the same $\mathbb{v}_{entj}$ according to the minimum-distance sum of v and other $\mathbb{v}_{entj}$. Further, we only keep the vertex with the minimum sum in each $\mathbb{v}$.

Graph Simplifying
In this stage, G-NAG deletes redundant vertices in V_o. For each vertex $v \in V_o$ in graph, we keep it if there exists a communication path between two linking vertices v_i, $v_j \in V_q$ containing it. In other words, we remove $v \in V_o$ only related to one entity phrase, which means a weak correlation with Q. Here, we regard two vertices as isomorphic if they share the same neighborhood and connect every common neighbor vertex with edges of the same relation. Then we merge isomorphic vertices and concatenate their text attributions as one vertex.

G-NAG further deletes redundant vertices in V_o using aggregated semantic similarities based on word embedding [9]. For this step, we only consider the alternative vertex $v \in V_o$ that could be removed without affecting the connectivity of the graph. Specifically, for each vertex, we concatenate triples containing it as a word sequence T, then use Word2Vec [15] to compute string similarities between T and question Q following [20]. where, w represents a word of the string, and the average is used as the aggregation functions. Finally, we keep the top-k alternative vertices in V_o with a higher score.

$$Similarity(Q, T) = Agg\ cos(w_Q, w_T) \tag{2}$$

Different from existing NAG systems, which match triples with the question directly, G-NAG performs multi-hop retrieval in entity-level without considering relation phrases, then simplifies the graph based on semantic similarity. This strategy allows G-NAG to handle implicit relations, where predicates are missed in question, more effectively.

In addition, we identify the wild-card, i.e., *who*, *when* or main type phrase, i.e., *actress* in Figure 1, in question text, which will be the text attribution of variate vertex described in next section.

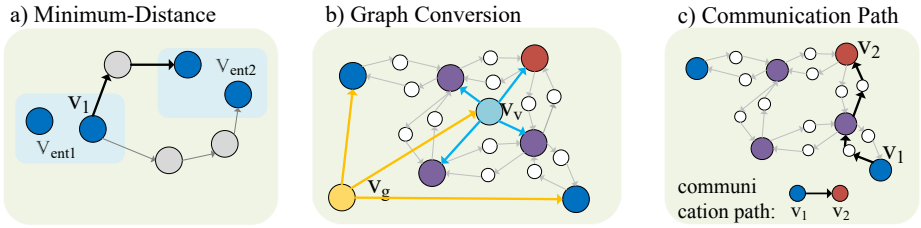

Fig. 3. Example of minimum distance, graph conversion and communication path

2.3 Incorporating encoder

The encoder receives discrete inputs (question text and supporting graph defined before) and encodes them into numerical representations jointly [3], to accomplish neural network feeding.

Graph Conversion
Inspired by [2, 13], we convert the extracted graph to an unlabeled bipartite graph. Specifically, we replace each relation edge with two vertices, where one represents the forward direction of the relation and the other represents the reverse. The key difference with the above work is G-NAG introduces two extra vertices v_v, v_g to capture effective information.

Specifically, global vertex connects to vertices $v \in V_q$ mentioned in the question to update the global state following the reasoning perspective as humans. Besides, variate vertex connects to other retrieved vertices $v \in V_o$ and global vertex, where its text attribution is a wild-card, i.e., who, or *ASU* type recognized by knowledge extractor, i.e., actor. Therefore, the variate vertex v_v concerns the other retrieved vertices $v \in V_o$ except for these mentioned in question, while global vertex v_g mainly grasps the whole graph via mentioned vertices $v \in V_q$ and variate vertex v_v.

As shown in Figure 3-b, global vertex v_g, which concentrates information via two mentioned vertices and the variate vertex, could reach all vertices in G'. Therefore, we initialize the decoder state using global vertex representation after self-attention following [26, 13]. Moreover, since the variate vertex has specific text attribution, it focuses more on other involved vertices $v \in V_o$ especially ASU, which is of vital importance for the generation. The conversion result is $G' = (V', P')$, where V' is a vertex set and P' is a matrix storing communication path between every vertices-pair. Take vertices pair v_1, v_2 as an example in Figure 3-c, a sequence of vertex text attribution along the path from v_1 to v_2 expresses the communication path. Note we choose the shortest path between vertices-pair (numbers of edges) and adopt embedding average when two or more equal length paths exist.

Graph Transformer
In this section, the text attribution of vertices is embedded as $V = [v_i], v_i \in R^d$ in a dense continuous space using bi-LSTM described in the Question encoder section, which is the input of graph transformer. Same as typical transformer, each vertex has 3 vector representations $\mathbf{q}(query)$, $\mathbf{k}(key)$, $\mathbf{v}(value)$.

Our graph transformer maintains a similar architecture as that in [13], which is proposed to generate summarization in the scientific domain. Compared with summarization generation, there are two differences in our task that is also challenges for encoding.

- Entities in supporting graph could be divided into mentioned vertices $v \in V_q$ and other retrieved vertices $v \in V_o$, while the former linked by entity phrases are related to answers but the latter may be *ASU* or supplementary information that could be omitted.
- There are closer relationships between entities than that of a summarization since the supporting graph is for a specific question in NAG, not for a scientific topic.

Therefore, we improve the graph transformer as following.

For each original vertex, G-NAG employs self-attention over the whole converted graph while [13] only calculates on local neighborhoods. This design allows vertex to capture more information except for neighborhoods since the supporting graph in the NAG task is smaller and more logically connected than that in the long-text summarization generation task.

Besides, we extend the conventional self-attention architecture by explicitly encoding the communication path between vertices-pair v_i, v_j in the alignment model motivated by [22]. Specifically, we encode the communication path described following into d-size dimension space and add it to vertex v_j's $\mathbf{k}(key)$ vector for calculation. Thus, we represent v_i as the weighted sum of all vertices' $\mathbf{v}(value)$ vectors with the consideration of communication path, formulated as follow:

$$\hat{v}_i = \Big\Vert_{n=1}^{N} \sum_{j \in V} \alpha_{ij}^n W_V^n v_j, \ where\ a_{ij}^n = \frac{exp((\mathbf{W_k k_j} + \mathbf{W_R r_{ij}})^\top W_Q q_i)}{\sum_{z \in V} exp((\mathbf{W_k k_z} + \mathbf{W_R r_{iz}})^\top W_Q q_i)} \quad (3)$$

where $\|$ represents concatenation, α_{ij}^n is normalized attention coefficient computed by self-attention mechanism per head, and W_V^n is transformation's weight matrix of $\mathbf{v}(value)$. For each attention function α, W_K, W_Q are transformation's weight matrix of $\mathbf{k}$ (key) and $\mathbf{q}$ $(query)$, where r_{ij} denotes the embedded communication path between v_i, v_j and $W_R \in R^{d*d}$ is a parameter matrix of $\mathbf{r}$.

For global and variate vertex, we compute their representation over neighbor vertices without path encoding respectively. As discussed before, we capture retrieved information by variate vertex and obtain global state by global vertex, which allows graph transformer to better articulate global patterns and *ASU* location. Since the edges around each extra vertex do not represent real relation in KB, we only contextualize global and variate vertices' representation by attending over their neighborhoods. As a result, these two vertices' representations are calculated attending over their neighborhoods in G' formulated as follows. Here, $\mathcal{N}_g$ denotes the neighborhoods of v_g and the representation calculation of v_v is the same as v_g.

$$\hat{v_g} = \Big\Vert_{n=1}^{N} \sum_{j \in \mathcal{N}_g} \alpha_j^n W_V^n v_j, \quad where\ a_j^n = \frac{exp((\mathbf{W_k k_g})^\top W_Q q_i)}{\sum_{z \in \mathcal{N}_g} exp((\mathbf{W_k k_z})^\top W_Q q_g)} \quad (4)$$

Finally, we adopt the conventional transform architecture, which is composed of a stack of D = 6 identical layers. As illustrated in Figure 2 b, each layer consists of a self-attention mechanism and feed-forward network, both around by a residual connection. The final representation of vertices is denoted as $V^D = [v^D]$.

In the following, we describe the representation of the communication path between vertices-pair. Given a vertex sequence along the communication path between two vertices, we concatenate the text attribution p_i of each vertex as sequence $p = [p_i]$. Then, we acquire d-sized corresponding embedding sequence $s = [s_i]$ inspired by the label sequence embedding procedure in [28]. Considering continuous or discrete representations separately, we employ the average method and self-attention method to calculate representation vector r_{ij}.

Average method: calculate the averaged embedding as the representation vector of the communication path.

Self-attention method: use attention function as presented in Eq. 4 to acquire the representation of s as $h^s = [h_i^s]$, then define a weight γ to calculate weighted sum of h^s as r:

$$\gamma_i = \frac{exp(e_i)}{\sum_{k=0}^{L} exp(e_k)}, e_i = v^\top tanh(W_{h^s} h_i^s + b) \tag{5}$$

where L denotes the length of communication path.

Question encoder

The question encoder transforms the question text into a vector representation by Recurrent Neural Network (RNN). The tokens of the question q_i are fed into a single-layer bidirectional LSTM [11] one by one, producing a sequence of concatenated encoder hidden states hq_i. While hq_i is expressed by $[\overrightarrow{hq_i}, \overleftarrow{hq_{L-i+1}}]$, which are encoded by a forward and a backward RNN independently. We use encoder state $hq_L = [\overrightarrow{hq_L}, \overleftarrow{hq_1}]$ to represent the whole question, while encoder output list hq_i is stored for attention distribution calculation.

2.4 Generator

To predict answer words y_t in each time step, we use LSTM decoder. During training, decoder accepts the embedding of previous output words $y_{<t} = y_1, y_2, ..., y_{t-1}$, a context vector c_t with attention on inputs, and decoder hidden state of previous step s_{t-1} to update hidden state: $s_t = f(y_{t-1}, s_{t-1}, c_t)$. Inspired by Copynet [7], we apply the copy mechanism to deal with the unknown or special words expected to appear in the answer sentence. In the following, we describe the generation process in decoder at each time step.

Firstly, we initialize the decoder state using global vertex representation as s_0. Then we compute the graph context vector c_g using N-headed attention as follows, which is a weighted sum of vertex representations.

$$c_g = s_t + \Big\|_{n=1}^{N} \sum_{i \in V} \alpha_i^n W_G^n v_i^D, \qquad where \alpha_i = \frac{exp((\mathbf{W_k k_i})^\top W_Q s_t)}{\sum_{z \in V} exp((\mathbf{W_k k_z})^\top W_Q s_t)} \tag{6}$$

Similarly, the question context vector c_q is computed attending over the question text as in [1]. Then we concatenate c_g and c_q as final context vector c_t.

Below, parameters W_h, W_s, b^* are learned during training, and L indicates the length of question sequence.

$$c_q = \sum_{j=1}^{L} \beta_j^n h_j,\ where \beta_j = \frac{exp(e_j)}{\sum_{k=0}^{L} exp(e_k)}, e_j = v^\top tanh(W_h h_j + W_s s_t + b^*) \quad (7)$$

G-NAG model generates answer words both from vocabulary based on attention and copying words via pointing. Therefore, we define a soft switch g within 0 to 1, which chooses between predicting a vocabulary word by distribution P_v or copying a word via attention distribution $[\alpha_i, \beta_j]$. Eventually, we acquire a final probability distribution over the extend vocabulary as follows.

$$P(w) = gP_{copy}(w) + (1-g)P_v(w),\ where\ g = sigmoid(W_h^\top h_t + W_s^\top s_t + b_g) \quad (8)$$

$$P_{copy} = \sum\nolimits_{j:w_j=w} (\alpha_j + \beta_j) \qquad P_v = softmax(W_{v1}(W_{v2}[s_t, c_t] + b_1) + b_2) \quad (9)$$

Besides, we minimize negative log-likelihood of the target word w_t^* for each time step, and the overall loss is defined as their sum.

$$\mathcal{L} = \frac{1}{T} \sum\nolimits_{t=0}^{T} (-logP(w_t^*)) \quad (10)$$

3 Experiment

3.1 Datasets

Our model attempts to generate natural answers, especially for complex questions that contain logical relations between entities. To our knowledge, there is not an existing dataset naturally fitted to this problem. Thus, we tailor the `Wikimovie`[1] dataset [16] according to our requirements as `wikimovie`*. Moreover, we reconstruct a multi-hop dataset `wikimovie-multihop` from the `Wikimovie` and `DBpedia` by manual annotation. The original `Wikimovie` dataset consists of simple question-ASU pairs, external KB and natural sentences from Wikipedia about the movie, which covers 10 topics. To expand knowledge, we search cast members' related triples in DBpedia by DBpedia Lookup Service . Statistics of the two datasets are available in Table 1.

Table 1. Data statistics of dataset

Dataset	Total movie num	QA-pairs	Avg length of question	Avg length of answer	Avg triples per QA-pair
wikimovie*	6429	12037	17	14	4.7
multihop	13066	34472	15	15	5.5

`wikimovie`*: Take each natural sentence in `Wikimovie` as an ideal answer, we search the related triples in underlying KB and choose one `o` (*object*) among the triples as *ASU*. Then let annotators generate the corresponding question, which contains the triple information mentioned in the answer without variate and movie name as Example 1. We remove the QA-pair if its *ASU* is not unique. Since each natural sentence in `Wikimovie` is around one movie, the related graph is star-like and within 2-hops.

[1] http://fb.ai/babi

Example 1. Given the natural answer "*Resident Evil is a 2013* ***English*** *movie directed by Paul Anderson and starring Li_Bingbing*". One possible question is "*What is the language of the 2013 film by Paul Anderson and Li_Bingbing*".

`wikimovie-multihop`: We extract sub-graph randomly in underlying KB with limited size, while the sub-graph should contain more than 2-hop relations between entities, but no more than 4-hop for the longest path. Then for each sub-graph, we mask one vertex to be *ASU* that is not in the border. Based on the sub-graph, let annotators generate QA pairs in natural language sentences, while the question must be answerable and the answer should contain all information without missing. Note that entities not essential for reasoning *ASU* could be omitted or replaced, i.e., in 2008 replaced by in the same year in section 1. After annotators providing 460 QA pairs, we extend the dataset by replacing the sub-graph in underlying KB with the same graph structure.

3.2 Evaluation Metrics

Automatic Evaluation:Similar to existing NAG systems [27, 10], we compute ASU-acc to evaluate the correctness of *ASU*. Following [5], we adopt some word-overlap based metrics (WBMs)[2] including BLEU-4 [19], and METEOR [4] to measure the co-occurrences of references and generated answers.

Manual Evaluation: Further, it is hard to automatically evaluate the naturalness and correctness of generated answers. Following [17], we employ a manual evaluation to measure the Naturalness and Correctness respectively by a score among 0-5, where the higher the score, the better the evaluation. The Kappa coefficient for inter-annotator is 0.744, and the p-value for scores is less than 0.01.

3.3 Comparison Models

Throughout existing researches on the natural language answer generation problem, we compare our model (G-NAG) with state-of-the-art NAG models from different perspectives.

- GenQA [27], a standard seq2seq model with attention using encoder-decoder structure. It retrieves the best-matched triple by MLP and encodes it with the question encoded by LSTM to generate a natural answer.
- COREQA [10], a similar structure to GenQA. Moreover, it retrieves more one-hop triples and introduces the copy mechanism.
- HM-NAG [6], an improvement of COREQA. It encodes all related triples in key-value structure without matching with the question and selects proper triples completely by attention during generation.

Except for existing NAG systems, we compare several baselines containing graph attention or transformer. Since these models have no knowledge extractor module, we feed the same simplified graph after converting as input.

[2] WBMs are implemented in https://github.com/Maluuba/nlgeval.

– GraphWriter [13], a graph2seq model for summarization containing graph transformer without variate vertex and communication path during the self-attention calculation.
– Tranformer [23], a sequence transformer proposed originally without graph structure.

3.4 Implementation Details

In knowledge extractor, we recognize entity phrases by StanfordCoreNLP tools and use Word2Vec [15] with 300 dimension vectors trained on the EN-wiki dataset to compute string similarities. Besides, we keep the top-2 alternative vertices in V_o with a higher score, and set $k = 2$, $m = 3$ in extractor module.

In experiments, G-NAG and baseline models are trained for about 40 epochs with the learning rate as 0.03, where gradients are updated by Adam [12] learning rule. In both datasets, we add word occurring more than 5 times into vocabulary and the state size of word embedding and batch size are both set to 256. For the transformer, we set layer D as 6, attention heads as 4, following the setting in [13], and use a self-attention based method to encode the communication path described in section 2.3.

3.5 Result

Table 2. Performances on dataset *Wikimovie**

Model	GenQA	COREQA	HM-NAG	GraphWriter	Transformer	G-NAG
ASU-acc	0.6506	0.6680	0.6818	0.8171	0.7913	**0.8310**
BLEU-4	0.3421	0.3792	0.3879	0.4282	0.4014	**0.4419**
METEOR	0.3722	0.3990	0.4113	0.4527	0.4371	**0.4809**
Natural	2.5	2.7	2.7	**3.4**	3.1	**3.4**
Correctness	2.0	2.5	2.7	3.4	3.1	**3.6**

Table 2 shows the answer generation performance on the `wikimovie*` dataset. From the result, we can see that G-NAG performs better than NAG baselines[3] both in the automatic or manual evaluation due to the improved knowledge extractor. Meanwhile, G-NAG outperforms GraphWriter and Transformer in ASU-acc, BLEU-4, and METEOR with stronger information express-ability of graph embedding method.

Considering manual evaluation, both G-NAG and GraphWriter, employing graph transformer, could generate fluent natural answers with the same score in Naturalness. Moreover, our G-NAG obtains a higher score in Correctness as it introduces two extra vertices and communication path embedding into the self-attention calculation.

Next, we prove the effectiveness of our model in `wikimovie-multihop` dataset in Table 3. Compared with G-NAG, ASU-acc metrics of NAG baselines are unsatisfactory as they use one-hop triple retrieval, which solves complex relations hardly in a multi-hop situation. Meanwhile, we see that G-NAG achieves higher ASU-acc than GraphWriter and Transformer which are fed with the same supporting graph since G-NAG has more ability to capture the *ASU* by variate vertex representation.

[3] Since different tailoring for the dataset, the result of HM-NAG is not the same as it reported

Table 3. Performances on dataset *Wikimovie-multihop*

Model	GenQA	COREQA	HM-NAG	GraphWriter	Transformer	G-NAG
ASU-acc	0.3071	0.4129	0.4513	0.7544	0.7403	**0.7816**
BLEU-4	0.1608	0.2011	0.2106	0.3322	0.3078	**0.3471**
METEOR	0.2034	0.2351	0.2509	0.3777	0.3541	**0.3912**
Natural	2.1	2.3	2.4	3.1	3.0	**3.2**
Correctness	1.7	2.1	2.2	3.2	2.9	**3.3**

A comparison in manual evaluation between sequence-based knowledge representation, such as NAG baselines or Transformer, and graph transformer-based framework proves the express-ability of graph transformer. We analyze that sequence-based systems may miss information during retrieving or generating stage, therefore the generated answers get a low score. Further, as for graph transformer, G-NAG could generate more logical and perfect answers than GraphWriter, which is reflected in BLEU-4 and Correctness metrics.

Table 4. Performances on implicit relation dataset *Wikimovie**

Model	GenQA	COREQA	HM-NAG	GraphWriter	Transformer	G-NAG
ASU-acc	0.5217 (-0.129)	0.5513 (-0.117)	0.5904 (-0.091)	0.7744 (-0.0430)	0.7502 (-0.041)	**0.7909 (-0.040)**
BLEU-4	0.2904 (-0.052)	0.3122 (-0.067)	0.3212 (-0.067)	0.3884 (-0.040)	0.3571 (-0.044)	**0.4037 (-0.038)**
METEOR	0.3317	0.3520	0.3688	0.4243	0.3914	**0.4427**
Natural	2.3	2.6	2.6	**3.4**	3.0	**3.4**
Correctness	1.8	2.2	2.3	3.2	2.7	**3.5**

As mentioned in section 2.2, G-NAG can handle implicit relations in questions, which is a challenge to NAG but the common situation in daily life, i.e., Q in Figure 2. Thus, we select the QA pairs in `Wikimovie*` where the questions have no obvious attribute or relational predicates. As shown in Table 4, G-NAG performs better than NAG baselines as it extracts triples depending more on the entity, not the relation, which is reflected in the decline value of ASU-acc and BLEU-4 compared with Table 3. Moreover, G-NAG keeps retrieved vertices as well as relation edges with higher scores in graph simplifying so as to identify these implicit relations. Furthermore, we can see that although the automatic metrics have fallen, the Naturalness and Correctness of G-NAG stay essentially flat because of the ability of generator module. However, G-NAG may generate redundant information in this situation, which will be discussed in the case study.

3.6 Case Study

Table 5 gives some outputs from our model, GraphWriter, and HM-NAG which performs better than the other two NAG baselines. *ASU* and other entities in this table are marked as **bold** and *italics* separately, while copy words marked as underline and superscript denotes the dataset QA pairs from. Besides, we use (movie-1,movie-2) to denote mentioned movies in the order that they appear in gold answers.

In Case 1, though HM-NAG recognizes entities correctly, it fails to generate *ASU* and accurate movie names because of triple missing. Meanwhile, because

Table 5. Example outputs of various systems versus Gold.

Question*	Do you remember the *César-winner* actress who appeared in director Rupert Sanders' and Drew Goddard's film in the same year?
Knowledge	(movie-1, release_year, 2012), (movie-1, directed_by, Rupert Sanders), (movie-1, starred_actor, ASU),(movie-2, release_year, 2012), (movie-2, directed_by, Drew Goddard),(movie-2, starred_actor,ASU)
HM-NAG	*César-winner* Rupert Sanders worked with Rupert Sanders in the film Snow White and the Huntsman, and Drew Goddard in the film Bad Times at the El Royale in the same year .
GraphWriter	*César-winner* **Kristen Stewart** worked with Rupert Sanders in the film Snow White and the Huntsman in 2012, and Drew Goddard in the film The Cabin in the Woods .
G-NAG	*César-winner* **Kristen Stewart** worked with Rupert Sanders in the film Snow White and the Huntsman, and Drew Goddard in the film The Cabin in the Woods in 2012 .
Gold	Yes, *César-winner* **Kristen Stewart** worked with Rupert Sanders in the film Snow White and the Huntsman, as well as Drew Goddard in The Cabin in the Woods in 2012 .
Question^{multi}	What is the release date of the *animated* movie by Kurt Frey and Ben Stassen?
Knowledge	(movie-1,directed_by,Ben Stassen),(movie-1,written_by,Ben Stassen) , (movie-1, written_by, Kurt Frey),(movie-1, release_year, ASU)
HM-NAG	Haunted Castle is a *animated* film written by writer Kurt Frey and Ben Stassen .
GraphWriter	Haunted Castle is a **2001** *animated* horror film written by writer Kurt Frey and directed by Ben Stassen .
G-NAG	Haunted Castle is a **2001** *animated* horror film written by writer Kurt Frey and directed by co-writer Ben Stassen .
Gold	Written by Kurt Frey and directed by co-writer Ben Stassen, Haunted Castle is a **2001** *animated* film .

GraphWriter does not consider path information in the attention calculation, it has not a comprehensive grasp of graph structure to generate *year* in the right position. In Case 2, when the given question contains implicit relations, it is hard for HM-NAG to recognize all accurate relations and *ASU*. Moreover, even fed with a more accurate graph, GraphWriter misses the relation reflected in gold answer by *co-writer*. As implicit relation affects the simplifying stage, G-NAG obtains a supporting graph with more redundant entities while it generates extra information as *horror*.

4 Related Work

Our work belongs to the NAG task and draws inspiration from the research fields of graph-to-sequence, and copying mechanism.

NAG: [27, 10] propose an end-to-end model to encode question and related knowledge as a sequence. Further, [6] put these triples into Key-Value memory proved effective by [16]. The above work provides a feasible framework consists of retrieving and generating that is followed by G-NAG. However, limited by the simple retrieval and sequence representation structure, these systems do not perform well in complex questions, which stimulates us to make improvements.

Graph-to-sequence: To find alternative representation structure for NAG, we notice that converting graph to sequence is wildly studied from different aspects. The above work proves that the graph is an effective structure to encode complex information [14], which fits our requirements. As for graph representation, the key idea is to learn a mapping to embed nodes as points in a low-dimensional vector space. Motivated by [24, 13, 28], we employ graph transformer considering communication path to encode the supporting graph.

Attention and Copy Mechanism: Since unknown or special words in source text may impede predicting, Copying based on Attention has been proven extremely useful for a broad range of text generation tasks. To judge where to copy from, Copynet [7] utilizes the soft attention distribution to produce an output sequence containing elements from the input. This solution is applied to dialogue system [7], NMT [8], summarization [18, 21], QA system [10], etc.

5 Conclusion and Future Work

In this paper, we propose a novel generating framework based on graph transformer to address the natural answer generation problem (NAG). The model we put forward, named G-NAG, improves knowledge extraction by multi-hop retrieval before disambiguation and simplifying strategy. Besides, it mainly increases express-ability by an extending graph transformer to encode the supporting graph for generating. Experimental results on two closed-domain datasets demonstrate that our model significantly outperforms existing NAG models, and prove the effectiveness of graph attention and transformer meanwhile. In the future, we expect G-NAG to find the balance between enlarging retrieval range and controlling graph size. Moreover, we try to solve the repetition problems by coverage model or other approaches.

Acknowledgement

This work was supported by NSFC under grant 61932001 and 61961130390.

References

1. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)
2. Beck, D., Haffari, G., Cohn, T.: Graph-to-sequence learning using gated graph neural networks. pp. 273–283. Association for Computational Linguistics, Melbourne, Australia, https://www.aclweb.org/anthology/P18-1026
3. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
4. Denkowski, M., Lavie, A.: Meteor universal: Language specific translation evaluation for any target language. pp. 376–380. ACL
5. Elsahar, H., Gravier, C., Laforest, F.: Zero-shot question generation from knowledge graphs for unseen predicates and entity types. ACL (2018)
6. Fu, Y., Feng, Y.: Natural answer generation with heterogeneous memory. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (2018)
7. Gu, J., Lu, Z., Li, H., Li, V.O.: Incorporating copying mechanism in sequence-to-sequence learning. arXiv preprint arXiv:1603.06393 (2016)

8. Gulcehre, C., Ahn, S., Nallapati, R., Zhou, B., Bengio, Y.: Pointing the unknown words. arXiv preprint arXiv:1603.08148 (2016)
9. Hasibi, F., Balog, K., Bratsberg, S.E.: Dynamic factual summaries for entity cards. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 773–782. ACM (2017)
10. He, S., Liu, C., Liu, K., Zhao, J.: Generating natural answers by incorporating copying and retrieving mechanisms in sequence-to-sequence learning. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
13. Koncel-Kedziorski, R., Bekal, D., Luan, Y., Lapata, M., Hajishirzi, H.: Text generation from knowledge graphs with graph transformers. arXiv preprint arXiv:1904.02342 (2019)
14. Lin, P., Song, Q., Wu, Y.: Fact checking in knowledge graphs with ontological subgraph patterns. Data Science and Engineering **3**(4), 341–358 (2018)
15. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
16. Miller, A., Fisch, A., Dodge, J., Karimi, A.H., Bordes, A., Weston, J.: Key-value memory networks for directly reading documents. arXiv preprint arXiv:1606.03126
17. Mohammed, S., Shi, P., Lin, J.: Strong baselines for simple question answering over knowledge graphs with and without neural networks. pp. 291–296. ACL
18. Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al.: Abstractive text summarization using sequence-to-sequence rnns and beyond. arXiv preprint arXiv:1602.06023
19. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: a method for automatic evaluation of machine translation. Association for Computational Linguistics
20. Reinanda, R., Meij, E., de Rijke, M.: Mining, ranking and recommending entity aspects. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 263–272. ACM (2015)
21. See, A., Liu, P.J., Manning, C.D.: Get to the point: Summarization with pointer-generator networks. arXiv preprint arXiv:1704.04368 (2017)
22. Shaw, P., Uszkoreit, J., Vaswani, A.: Self-attention with relative position representations. arXiv preprint arXiv:1803.02155 (2018)
23. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
24. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
25. Wang, R., Wang, M., Liu, J., Chen, W., Cochez, M., Decker, S.: Leveraging knowledge graph embeddings for natural language question answering. In: International Conference on Database Systems for Advanced Applications
26. Xu, K., Wu, L., Wang, Z., Feng, Y., Witbrock, M., Sheinin, V.: Graph2seq: Graph to sequence learning with attention-based neural networks. arXiv preprint arXiv:1804.00823 (2018)
27. Yin, J., Jiang, X., Lu, Z., Shang, L., Li, H., Li, X.: Neural generative question answering. arXiv preprint arXiv:1512.01337 (2015)
28. Zhu, J., Li, J., Zhu, M., Qian, L., Zhang, M., Zhou, G.: Modeling graph structure in transformer for better amr-to-text generation (2019)