Query Graph Generation for Answering Multi-hop Complex Questions from Knowledge Bases

Yunshi Lan
Singapore Management University
yslan.2015@phdcs.smu.edu.sg

Jing Jiang
Singapore Management University
jingjiang@smu.edu.sg

Abstract

Previous work on answering complex questions from knowledge bases usually separately addresses two types of complexity: questions with constraints and questions with multiple hops of relations. In this paper, we handle both types of complexity at the same time. Motivated by the observation that early incorporation of constraints into query graphs can more effectively prune the search space, we propose a modified staged query graph generation method with more flexible ways to generate query graphs. Our experiments clearly show that our method achieves the state of the art on three benchmark KBQA datasets.

1 Introduction

Knowledge base question answering (KBQA) aims at answering factoid questions from a knowledge base (KB). It has attracted much attention in recent years (Bordes et al., 2014; Xu et al., 2016; Yu et al., 2017; Liang et al., 2017; Hu et al., 2018; Petrochuk and Zettlemoyer, 2018). Early work on KBQA focused on simple questions containing a single relation (Yih et al., 2014; Bordes et al., 2015; Dong et al., 2015; Hao et al., 2017). However, real questions are often more complex and recently some studies looked into complex KBQA.

Two different types of complexity have been studied: (1) Single-relation questions with *constraints*. For example, in the question "Who was the first president of the U.S.?" there is a single relation "president_of" between the answer entity and the entity "U.S.," but we also have the constraint "first" that needs to be satisfied. For this type of complex questions, a staged query graph generation method has been proposed, which first identifies a single-hop relation path and then adds constraints to it to form a *query graph* (Yih et al., 2015; Bao et al., 2016; Luo et al., 2018). (2) Questions with *multiple hops* of relations. For example, for the question "Who is the wife of the founder of Facebook?" the answer is related to "Facebook" through two hops of relations, namely, "wife_of" and "founder_of." To answer this type of multi-hop questions, we need to consider longer relation paths in order to reach the correct answers. The main challenge here is how to restrict the search space, i.e., to reduce the number of multi-hop relation paths to be considered, because the search space grows exponentially with the length of relation paths. One idea is to use beam search. For example, Chen et al. (2019) and Lan et al. (2019b) proposed to consider only the best matching relation instead of all relations when extending a relation path. However, little work has been done to deal with both types of complexity together.

In this paper, we handle both constraints and multi-hop relations together for complex KBQA. We propose to modify the staged query graph generation method by allowing longer relation paths. However, instead of adding constraints only *after* relation paths have been constructed, we propose to incorporate constraints and extend relation paths *at the same time*. This allows us to more effectively reduce the search space. On the ComplexWebQuestions dataset, which has a high percentage of complex questions with both types of complexity, our method substantially outperforms existing methods with an improvement of 3.3 percentage points in Prec@1 and 3.9 percentage points in F1. On two other benchmark KBQA datasets, our method also achieves the state of the art[1].

2 Method

2.1 Preliminaries

A KB can be represented as a set of triplets $\mathcal{K} = \{(h, r, t)\}$ where h and t are entities from $\mathcal{E}$ (the

[1]Our code is available at https://github.com/lanyunshi/Multi-hopComplexKBQA.

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 969–974
July 5 - 10, 2020.

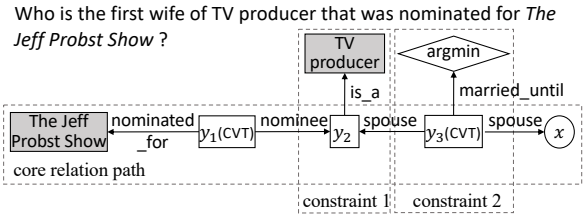

Figure 1: An example query graph for the question shown above. Assuming we start from the topic entity *The Jeff Probst Show*, the core relation path is the path linking *The Jeff Probst Show* to the lambda variable x. There are two constraints in the query graph. Note that y_1 and y_3 are CVT nodes used for n-ary relations.

entity set) and r is a relation from $\mathcal{R}$ (the relation set). Given a question Q, KBQA tries to find an entity $a \in \mathcal{E}$ that answers the question.

Our method is largely inspired by an existing staged query graph generation method (Yih et al., 2015; Bao et al., 2016; Luo et al., 2018), which we briefly introduce here first. A **query graph** has four types of nodes: A **grounded entity** (shaded rectangle) is an existing entity in the KB. An **existential variable** (unshaded rectangle) is an ungrounded entity. A lambda variable (circle) is also an ungrounded entity but it represents the answer. Finally, an **aggregation function** (diamond) is a function such as `argmin` and `count` that operates on a set of entities. The edges of a query graph are relations from $\mathcal{R}$. A query graph should have exactly one lambda variable to denote the answer, at least one grounded entity, and zero or more existential variables and aggregation functions. Figure 1 shows an example query graph for the question "Who is the first wife of TV producer that was nomiated for *The Jeff Probst Show*?"

We summarize the staged query graph generation method as follows. More details can be found in (Yih et al., 2015; Bao et al., 2016).

1) Starting from a grounded entity found in the question (referred to as a *topic entity*), identify a core relation path[2] linking the topic entity to a lambda variable. Existing work considers core relation paths containing a single relation (Yih et al., 2015; Bao et al., 2016; Luo et al., 2018).[3]

2) From a core relation path identified in Step 1, attach one or more constraints found in the question. A constraint consists of either a grounded entity or an aggregation function together with a relation. See examples in Figure 1.

3) With all the candidate query graphs generated from Step 1 and Step 2[4], rank them by measuring their similarities with the question. This is typically done through a neural network model such as a CNN (Yih et al., 2015; Bao et al., 2016).

4) Execute the top-ranked query graph against the KB to obtain the answer entities.

2.2 Motivation

The major challenge we face when directly applying the existing method outlined above to constrained multi-hop KBQA is that questions containing multiple hops of relations (such as the example in Figure 1) cannot be handled, because existing work considers only core relation paths with a single hop (or two hops with a CVT node). If we make a naive modification by allowing core relation paths to be longer, the search space suddenly becomes much larger. For example, on the ComplexWebQuestions dataset, if we allow core relation paths up to 3 hops, on average we will have around $10,000$ core relation paths per question, which is computationally very expensive.

Recent work on multi-hop KBQA tackles this problem by beam search, i.e., keeping only the top-K t-hop relation paths before generating the $(t+1)$-hop relation paths (Chen et al., 2019; Lan et al., 2019b). However, this approach ignores constraints when generating the relation paths. We observe that constraints found in a question can often help reduce the search space and guide the generation of the core relation paths towards the right direction.

Take the question in Figure 1 for example. Given a partial core relation path (*The Jeff Probst Show*, nominated_for, y_1, nominee, y_2), if we were to extend this path at y_2 with one more relation, we would need to consider all relations in the KB linked to bindings of y_2, which include *all entities* nominated for *The Jeff Probst Show*. But if we attach the constraint (is_a, TV producer) to y_2 first, then we would need to consider only those relations linked to *TV producers* nominated for *The Jeff Probst Show*.

We therefore propose a modified staged query graph generation method, which does not wait for each core relation path to be generated completely

[2]This path is called the *core inferential chain* by Yih et al. (2015) and *basic query graph* by Bao et al. (2016).

[3]They also consider paths with two relations connected by a so-called CVT node, which is a special dummy entity used in Freebase for n-ary relations. For simplicity, we treat these also as single-relation paths.

[4]In (Yih et al., 2015), a priority queue is used to keep only the top-ranked query graphs.

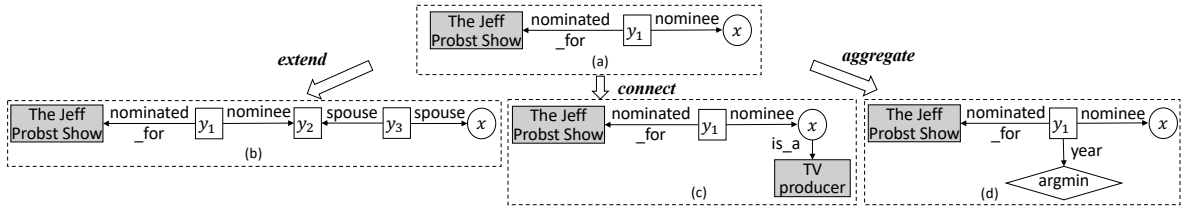

Figure 2: Examples of the *extend*, *connect* and *aggregate* actions. Note that query graph (d) corresponds to the question "Who is the first person that was nominated for *The Jeff Probst Show*?"

before attaching a constraint to it. This more flexible way of generating query graphs, coupled with a beam search mechanism and a semantic matching model to guide pruning, explores a much smaller search space while still maintaining a high chance of finding the correct query graph.

2.3 Query Graph Generation

Formally, our method uses beam search to generate candidate query graphs iteratively. We assume that the t-th iteration produces a set of K query graphs, denoted as $\mathcal{G}_t$. At the $(t+1)$-th iteration, for each $g \in \mathcal{G}_t$, we apply one of the {*extend*, *connect*, *aggregate*} actions (explained below) to grow g by one more edge and one more node. We do this for all $g \in \mathcal{G}_t$ and all actions that are applicable to each g. Let $\mathcal{G}'_{t+1}$ denote the set of all resulting query graphs. We then use a scoring function (explained in Section 2.4) to rank all the query graphs in $\mathcal{G}'_{t+1}$ and place the top-K of them in $\mathcal{G}_{t+1}$. We continue the iterations until there is no $g \in \mathcal{G}_{t+1}$ that is scored higher than any $g \in \mathcal{G}_t$.

We allow the following actions to grow a query graph. Figure 2 shows examples of these actions.
(1) An ***extend*** action extends the core relation path by one more relation in $\mathcal{R}$. If the current query graph contains only a topic entity e, an *extend* action finds a relation r linked to e in the KB and grows the path by r[5]. It also makes the other end of r the lambda variable x. If the current query graph has a lambda variable x, an *extend* action changes x into an existential variable y, finds all bindings of y in the KB by executing the current query graph against the KB, finds a relation r linked to one of these entities, and finally attaches r to y. The other end of r becomes the new lambda variable x.
(2) Besides the topic entity at the start of the current core relation path, there are oftentimes other grounded entities found in the question. A ***connect*** action links such a grounded entity e to either the lambda variable x or an existential variable connected to x that is a CVT node.[6] To decide which relation r to use to link e and x, again we can find all bindings of x by executing the current query graph and then find a relation that exists between one of these entities and e.
(3) Following Luo et al. (2018), we can detect an aggregation function from the question using a set of predefined keywords. An ***aggregate*** action attaches the detected aggregation function as a new node to either the lambda variable x or an existential variable connected to x that is a CVT node.

The novelty of our method is that the *extend* action can be applied after the *connect* and *aggregate* actions, which previous methods do not allow.

2.4 Query Graph Ranking

At the end of the t-th iteration, we rank the candidate query graphs in $\mathcal{G}'_t$ by deriving a 7-dimensional feature vector $\mathbf{v}_g$ for each graph $g \in \mathcal{G}'_t$ and feeding these vectors into a fully-connected layer followed by softmax to derive $p(g|Q)$.

The first dimension of $\mathbf{v}_g$ comes from a BERT-based semantic matching model. Specifically, we convert g into a sequence of tokens by following the sequence of actions that has been taken to construct g and adding the textual descriptions of the entities and relations involved at each step sequentially to the sequence. Existential variables and lambda variables are ignored. For example, the query graph shown in Figure 2(a) of the paper is converted to the following sequence: (the, jeff, probst, show, nominated, for, nominee).[7]

[5] We also allow r to be two relations connected through a CVT node.

[6] Here we only consider the existential variable connected to the lambda variable as we should have already considered the other existential variables in past iterations.

[7] This example is for illustration purpose. In the actual data, the relation descriptions are different from what we show in Figure 1. Therefore the actual token sequence is different for this example. We also convert the question into a sequence of tokens. For example, the question "Who is the wife of the founder of Facebook?" becomes (who, is, the, wife, of, the, founder, of, facebook). We then concatenate the query graph sequence and the question sequence into a single sequence,

The other 6 dimensions of $\mathbf{v}_g$ are as follows: The first one is the accumulated entity linking scores of all grounded entities in the query graph. The second one is the number of grounded entities appearing in the query graph. The third to the fifth ones are the numbers of entity types, temporal expressions and superlatives in the query graph, respectively. The last feature is the number of answer entities of the query graph.

2.5 Learning

To train our model, we make use of paired questions and their correct answers without any ground truth query graphs. Following the framework of Das et al. (2018), we use REINFORCE algorithm to learn a policy function $p_\theta(g|Q)$ in an end-to-end manner, where θ is the set of parameters we want to learn, including the BERT parameters to be updated and the parameters of the fully-connected layer for the 7-dimensional vector $\mathbf{v}_g$. We use F1 score of the predicted answers with respect to the ground truth answers as reward.

3 Experiments

3.1 Implementation Details

Our method requires entities to be identified from the questions and linked to their corresponding entries in the KB. For named entity linking, we use an existing linking tool[8] for the ComplexWebQuestions dataset and the already extracted topic entities released together with the dataset for the other two datasets. For entity type linking, we make use of the training questions and their answers to learn a linking model. For temporal expressions and superlative linking, we simply use regular expressions and a superlative word list. The superlative words are manually mapped to two aggregation functions: `argmax` and `argmin`.

We initialize the BERT module in the ranker with the BERT base model[9]. Other parameters are initialized randomly. For the hyper-parameters in BERT model, we set the dropout ratio as 0.1, the hidden size as 768. The number of layers and the number of multi-attention heads are set as 6 and 12, respectively. we use the latest dump of Freebase[10] as our KB for all the datasets. For beam search, we set the beam size K to be 3.

3.2 Datasets

We use three datasets to evaluate our method: ComplexWebQuestons (CWQ) (Talmor and Berant, 2018), WebQuestionsSP (WQSP) (Yih et al., 2015) and ComplexQuestions (CQ) (Bao et al., 2016). We treat CWQ as the major evaluation dataset because CWQ has a significantly higher percentage of complex questions with multiple hops of relations and constraints, as shown in Table 1a.[11] For example, more than 30% of the questions in CWQ has 2-hop relations with constraints, compared with just 0.5% in WQSP. Note that we cannot collect similar statistics for the CQ dataset because it does not provide the ground truth query graphs, but we observe that major questions in CQ have 1-hop relations.

3.3 Methods for Comparison

We compare our method with the following existing work. First, we compare with existing staged query graph generation methods (Yih et al., 2015; Bao et al., 2016; Luo et al., 2018), which cannot handle multi-hop questions. Next, we compare with (Lan et al., 2019a), which handles constraints and considers multi-hop relation paths, but uses neither beam search nor constraints to reduce the search space. We also compare with (Chen et al., 2019), which uses beam search with a beam size of 1 to handle multi-hop questions but does not handle constraints. Finally, we compare with (Bhutani et al., 2019) and (Ansari et al., 2019). Bhutani et al. (2019) decomposed complex questions into simple questions and achieved the SOTA in terms of Prec@1 on CWQ[12]. Ansari et al. (2019) generated query programs from questions token by token and achieved the SOTA on WQSP.

3.4 Main Results

We show the overall comparison in Table 1b. We can see that on the CWQ dataset, our method clearly achieves the best performance in terms of

with the special token [CLS] separating them, as how BERT is used typically to handle two sequences. We then use the standard BERT model (Devlin et al., 2019) to process the entire sequence and derive a score at the top layer. Note that we fine-tune the pre-trained BERT parameters during learning.

[8]The tool can be found at https://developers.google.com/knowledge-graph.

[9]The pre-trained BERT base model could be found at https://github.com/huggingface/pytorch-transformers.

[10]The KB can be downloaded from https://developers.google.com/freebase/.

[11]Note that we treat 2-hop relation paths with CVT nodes as 1-hop paths.

[12]We note that on the leaderboard of CWQ the best Prec@1 was achieved by Sun et al. (2019). However, their method uses annotated topic entities and is thus not comparable here.

QType	CWQ	WQSP
1-hop w/o CONS	0.1%	71.3%
1-hop w/ CONS	35.9%	28.2%
2-hop w/o CONS	33.5%	0.0%
2-hop w/ CONS	30.5%	0.5%

(a)

Method	CWQ Prec@1/F1	WQSP F1	CQ F1
Yih et al. (2015)	−/−	69.0	–
Bao et al. (2016)	−/−	–	40.9
Luo et al. (2018)	−/−	–	42.8
Lan et al. (2019a)	39.3/36.5	67.9	–
Chen et al. (2019)†	30.5/29.8	68.5	35.3
Bhutani et al. (2019)	40.8/33.9	60.3	–
Ansari et al. (2019)	−/−	72.6	–
Our Method	**44.1/40.4**	**74.0**	**43.3**

(b)

Method	CWQ Prec@1/F1
SOTA	40.8/36.5
w/ BERT	44.1/40.4
w/ LSTM	42.1/38.7
w/o extend	25.2/22.8
w/o connect	33.2/31.3
w/o aggregate	42.4/39.6

(c)

Table 1: (a) Some statistics of CWQ and WQSP. CONS stands for constraints. (b) Comparison between our method and existing work. † denotes our re-implementation. (c) Ablation study on the CWQ dataset.

both Prec@1 and F1. The amount of improvement is also substantial, with 3.3 percentage points in Prec@1 and 3.9 percentage points in F1. This validates our hypothesis that our method works particularly well for complex questions with both constraints and multi-hop relations. For the other two datasets, WQSP and CQ, our method also achieves the SOTA, outperforming previous methods, demonstrating the robustness of our method.

3.5 Ablation Study

We also conduct an ablation study to better understand our model. To verify that the effectiveness of our method is not mainly due to the use of BERT, we replace BERT with LSTM. We can see in Table 1c that the LSTM-based version of our method can still outperform the previous state of the art. This shows that the effectiveness of our model is not simply because of the use of BERT. We also test three versions of our method, each with one action removed, in order to understand if all three actions are necessary. The results are also shown in Table 1c. We can see that the *aggregate* action is the least important action whereas the *extend* action is the most important one. However, we need to combine all three actions together to achieve the best performance.

3.6 Error Analysis

We randomly sampled 100 error cases for manual inspection. We summarize the errors into the following categories.

Ranking Error: There are 65% of errors coming from mis-prediction of query graphs. We look at these error cases closely. We find that some relations are hard to be detected even with human judgment. For example, our model mis-predicts the relation in the question "Who was VP for Nixon?" as "profession" while the correct relation is "vice president". To understand "VP" is the abbreviation of "vice president" needs external knowledge, if this mapping has not been observed in the training data. **Topic Linking Error**: We observe that there are 27% of errors occurring due to the entity or expression linking error. E.g., "What guitar does Corey Taylor play?" has the constraint type "guitar", but it is not detected in the linking procedure. **Generation Limitation**: The limitation of query graph generation strategies leads to 6% of errors. For the question "What jobs did John Adams have before he was president", we are unlikely to find a matched query graph with our strategies.

4 Conclusion

In this paper we proposed a modified staged query graph generation method to deal with complex questions with both multi-hop relations and constraints. By incorporating constraints into query graphs early, coupled with the help of beam search, we are able to restrict the search space. Experiments showed our method substantially outperformed existing methods on the ComplexWebQuestions dataset and also outperformed the previous state of the art on two other KBQA datasets.

Acknowledgment

This research is supported by the National Research Foundation, Singapore under its International Research Centres in Singapore Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

Ghulam Ahmed Ansari, Amrita Saha, Vishwajeet Kumar, Mohan Bhambhani, Karthik Sankaranarayanan,

and Soumen Chakrabarti. 2019. Neural program induction for kbqa without gold programs or query annotations. In *Proceedings of the IJCAI*, pages 4890–4896.

Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-based question answering with knowledge graph. In *Proceedings of the COLING*, pages 2503–2514.

Nikita Bhutani, Xinyi Zheng, and H V Jagadish. 2019. Learning to answer complex questions over knowledge bases with query composition. In *Proceedings of the CIKM*, pages 739–748.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075.

Antonie Bordes, Sumit Chopra, and Jason Weston. 2014. Question answering with subgraph embeddings. In *Proceedings of the EMNLP*, pages 615–620.

Zi-Yuan Chen, Chih-Hung Chang, Yi-Pei Chen, Jijnasa Nayak, and Lun-Wei Ku. 2019. UHop: An unrestricted-hop relation extraction framework for knowledge-based question answering. In *Proceedings of the NAACL-HLT*, pages 345–356.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2018. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the NAACL*, pages 4171–4186.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over freebase with multi-column convolutional neural networks. In *Proceedings of the ACL-IJCNLP*, pages 260–269.

Yanchao Hao, Yuanzhe Zhang, Kang Liu, Shizhu He, Zhanyi Liu, Hua Wu, and Jun Zhao. 2017. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In *Proceedings of the ACL*, pages 221–231.

Sen Hu, Lei Zou, and Xinbo Zhang. 2018. A state-transition framework to answer complex questions over knowledge base. In *Proceedings of the EMNLP*, pages 2098–2108.

Yunshi Lan, Shuohang Wang, and Jing Jiang. 2019a. Knowledge base question answering with topic units. In *Proceedings of the IJCAI*, pages 5046–5052.

Yunshi Lan, Shuohang Wang, and Jing Jiang. 2019b. Multi-hop knowledge base question answering with an iterative sequence matching model. In *Proceedings of the ICDM*, pages 359–368.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: learning semantic parsers on freebase with weak supervision. In *Proceedings of the ACL*, pages 23–33.

Kangqi Luo, Fengli Lin, Xusheng Luo, and Kenny Zhu. 2018. Knowledge base question answering via encoding of complex query graphs. In *Proceedings of the EMNLP*, pages 2185–2194.

Michael Petrochuk and Luke Zettlemoyer. 2018. Simple questions nearly solved: a new upperbound and baseline approach. In *Proceedings of the EMNLP*, pages 554–558.

Haitian Sun, Tania Bedrax-Weiss, and William W. Cohen. 2019. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of the EMNLP*, pages 2380–2390.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the NAACL-HLT*, pages 641–651.

Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question answering on Freebase via relation extraction and textual evidence. In *Proceedings of the ACL*, pages 2326–2336.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the ACL-IJCNLP*, pages 1321–1331.

Wen-tau Yih, Xiaodong He, and Christopher Meek. 2014. Semantic parsing for single-relation question answering. In *Proceedings of the ACL*, pages 643–648.

Mo Yu, Wenpeng Yin, Kazi Saidul Hasan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. 2017. Improved neural relation detection for knowledge base question answering. In *Proceedings of the ACL*, pages 571–581.