Unseen Entity Handling in Complex Question Answering over Knowledge Base via Language Generation

Xin Huang and **Jung-jae Kim** and **Bowei Zou**
Institute for Infocomm Research, A*STAR, Singapore
huangx2,jjkim,zou_bowei@i2r.a-star.edu.sg

Abstract

Complex question answering over knowledge base remains as a challenging task because it involves reasoning over multiple pieces of information, including intermediate entities/relations and other constraints. Previous methods simplify the SPARQL query of a question into such forms as a list or a graph, missing such constraints as "*filter*" and "*order_by*", and present models specialized for generating those simplified forms from a given question. We instead introduce a novel approach that directly generates an executable SPARQL query without simplification, addressing the issue of generating unseen entities. We adapt large scale pre-trained encoder-decoder models and show that our method significantly outperforms the previous methods and also that our method has higher interpretability and computational efficiency than the previous methods.

1 Introduction

Answering user's questions via correct relation paths over a knowledge base may facilitate machine-human interaction to understand how the machine gets the answer. The *relation path* of a question is defined as the sequence of relations from the *topic entity* mentioned in a question to its answer entity in a knowledge base, which corresponds to the semantics of the question. While answering simple questions whose relation path has only one relation (or edge) without any other constraint has been largely resolved (Petrochuk and Zettlemoyer, 2018), answering complex questions over a knowledge base (called Complex KBQA) whose relation path contains more than one relation and/or other constraints remains as a difficult task (Zhou et al., 2018; Lan et al., 2019; Sun et al., 2019; Lan and Jiang, 2020).

Previous works on Complex KBQA cast it as a graph searching task. Yih et al. (2015), Xu et al. (2016), and Yu et al. (2017) identify the relation path of a question, by comparing the question with each candidate relation path. They should restrict the set of candidate relation paths (e.g. those with up to two relations), excluding any other constraints (e.g. *filter*, *order_by*), due to too big search space of all potential candidate relation paths. The methods thus show limited coverage for such datasets as ComplexWebQuestions, whose relation paths have up to three relations and other constraints. Sun et al. (2018, 2019) instead identify intermediate entities in the relation path iteratively until reaching the answer entity. However, the methods predict only one answer entity for a question and thus show low recall for questions with multiple answer entities. Chen et al. (2019), Lan et al. (2019), and Lan and Jiang (2020) extend the previous methods (Yih et al., 2015; Xu et al., 2016; Yu et al., 2017) by iteratively generating a query graph instead of ranking candidate relation paths. The methods predict one of the actions 'extend', 'connect' and 'aggregate' to grow a query graph by one more pair of edge and node, but yet do not cover such constraints as "*filter*" and "*order_by*". Please refer to Appendix A for detailed discussion of the previous works.

Inspired by the recent progress of adapting natural language generation (NLG) for various natural language processing (NLP) applications (Raffel et al., 2020; Brown et al., 2020), we approach Complex KBQA as a language generation task, fine-tuning large-scale pre-trained encoder-decoder models to generate executable SPARQL query from question. An issue of this approach is to generate *unseen* entities for questions of test dataset. The SPARQL queries in the KBQA datasets represent entities with their IDs (e.g. "ns:m.08x9_6"), but it is impractical to learn to generate *unseen* entity IDs. To address the issue, we leverage language generation models to learn the correlation between entity text labels (e.g. "1980 NBA Finals") and questions during the training process so as to generate *unseen* entities' text labels in the inference process. Specifically, our method learns to generate entity

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 547–557
November 7–11, 2021.

text labels instead of entity IDs, by replacing each entity ID in a SPARQL query with a placeholder (e.g. 'c1') and adding a string matching filter at the end of the SPARQL query (e.g. 'filter(str(?c1) = "1980 NBA Finals")').

The proposed approach has the following advantages over the previous works: 1) The proposed approach can optimize a model for the whole query sequence generation, while the iterative graph generation models are optimized for predicting one edge (or action) of query graph at a time; 2) the interpretability of sequence generation models is higher than that of iterative graph generation models (see Section 3.4 for details); 3) our method can utilize a large-scale pre-trained language model for learning SPARQL query generation, while the previous works can utilize such a model only for representing texts (e.q. question, entity and relation text labels); and 4) our method can learn to generate any constraints, while the previous works should define a new action type to deal with another unaddressed constraint type.

The language generation part of the proposed approach is in fact semantic parsing, which converts a question into a logical representation or an executable query (e.g. SQL) (Krishnamurthy et al., 2017; Dong and Lapata, 2018; Yin et al., 2020; Zeng et al., 2020). The key difference between Complex KBQA and semantic parsing is that Complex KBQA assumes a large knowledge base (e.g. Freebase) for the whole dataset, while semantic parsing aims at learning dynamic correlation between a question and any given table or relational database. Recent methods of semantic parsing (Yin et al., 2020; Zeng et al., 2020) learn the dynamic correlation by encoding the whole table together with the question. However, such knowledge base as Freebase is too large to be represented by a single encoder (see Table 5 for details). Instead, our method for Complex KBQA has two steps of topic entity location and executable query generation, jumping to a candidate topic entity and generating a SPARQL query starting from the entity.

We conduct experiments on three benchmark datasets: MetaQA (Zhang et al., 2018), ComplexWebQuestions (Talmor and Berant, 2018), and WebQuestionsSP (Yih et al., 2015). Evaluation results show that the proposed method significantly outperforms the state-of-the-art methods over all metrics on all three datasets. Besides, our method also outperforms the previous methods in terms of interpretability and computational efficiency.

We summarize the contributions that will be shown in this paper as follows:

- We adapt pre-trained language generation models for generating executable SPARQL queries for Complex KBQA questions, including all constraints (e.g. "filter", "order_by") without additional model architecture.
- We show that the issue of unseen entities causes simple adaptation of language generation for KBQA to have low performance and address the issue by learning to generate entity text labels instead of entity IDs.
- We show that the proposed method outperforms the previous methods in terms of interpretability and computational efficiency.

2 Methodology

Our method first recognises topic entities in a given question (Section 2.2), and then generates a list of SPARQL queries given the question and the category (or type) of each topic entity by training an encoder-decoder model (Section 2.3), and finally identifies the best valid SPARQL query that locates at least one answer entity in a given knowledge base at a post-processing step (Section 2.4). A question may mention multiple entities. Our method considers them all as *candidate* topic entities of the question and generates SPARQL queries with each of the candidate topic entities. If a SPARQL query has multiple entities, the entity whose ID is the first element of a triple (e.g. <entity ID, predicate, ?variable>) can be a topic entity. We select one topic entity at a time, while the other entities are considered as constraint entities. Our method is schematically described in Appendix B.1, and Figure 1 depicts how the method analyzes a question to generate an executable SPARQL query.

2.1 Data pre-processing

As mentioned in Introduction, our method generates entity text labels, specifically the text labels of constraint entities, and detects the position of topic entity in SPARQL query, while the SPARQL queries of the Complex KBQA datasets contain entity IDs. We thus modify the entity IDs in SPARQL queries as follows:

- Topic entity ID: Replaced with a special token ([*ENT*]). The query generation module

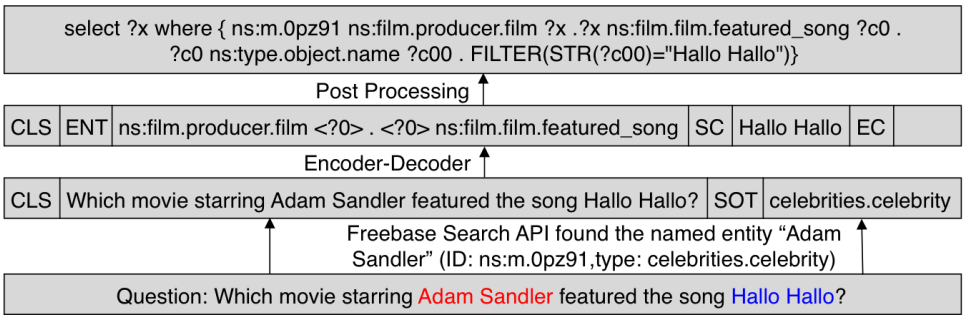

Figure 1: An example procedure of converting a question to an executable SPARQL query.

(Section 2.3) only identifies the position of topic entity ID in the SPARQL query, and the post-processing module (Section 2.4) replaces the special token with the ID of the topic entity identified by the topic entity identification module (Section 2.2).

- Constraint entity ID: Replaced with its text label surrounded by special tokens [*SC*] and [*EC*], which represent the start and end of the constraint entity's text label, respectively. The query generation module generates the text label and the post-processing module converts the generated text labels to identify their IDs.

Another issue is that different SPARQL queries may have different names of the variable for answer entity. We thus further modify the variables of SPARQL queries as follows:

- Answer entity variable: Replaced with '?0'
- Intermediate entity variable: Replaced with '?*n*' ($n > 0$), where *n* indicates that it is *n*-th hop away from the topic entity

Furthermore, we remove uninformative prefixes of SPARQL queries. Note that we do not change the other parts of SPARQL queries in the data pre-processing step, including operations like *filter* and *order_by*. For instance, Appendix B.2 shows the original SPARQL query of the question "Who were the 1980 NBA Finals champions that Lamar Odom is now playing for?" and its modified version by the data pre-processing module.

2.2 Topic Entity Identification

We retrieve candidate topic entities from a given question by using the FreeBase search API[1], and then select top-N candidate topic entities $e_i^{(0)}$, $i \in \{1, \ldots, N\}$ ranked by their scores. For each of the N candidate topic entities, we look up Freebase to find its category and use the category together with the given question as input to our generation model. If a topic entity is associated with multiple categories, we use the concatenation of all the categories as input.

2.3 SPARQL Query Generation

Given a question q and the type of a candidate topic entity $e_i^{(0)}$, we generate a list of SPARQL queries by using an encoder-decoder model with beam search. Specifically, we first concatenate q and $e_i^{(0)}$ and encode it to obtain a hidden representation denoted as $\mathbf{h}_{q'_i}$. Then, a decoder generates a list of SPARQL queries $\{o_{ij} | j \in [1, M]\}$ by $\mathbf{h}_{q'_i}$.

A decoder then generates a list of M SPARQL queries o_{ij}, $j \in \{1, \ldots, M\}$ given the hidden representations of the input string $\mathbf{h}_{q'}$.

We explore the following encoder-decoder models for the proposed method: GRU, Bert2Bert, GPT2GPT2 (Rothe et al., 2020) and BART (Lewis et al., 2020). The details and the fine-tuning process of the pre-trained models are described in Appendix B.3 and B.4, respectively.

2.4 Post-Processing

To convert the generated SPARQL query into a valid and executable form, we perform the following actions:

- Topic entity: Replace the special token ([*ENT*]) with the ID of the input topic entity
- Constraint entities: Assume a model generates *C* number of constraint entities, where the text label of each constraint entity is surrounded by the special tokens [*SC*] and [*EC*]. Replace them with variables ('?c*1*' $\cdots$ '?c*C*') and, for

[1] https://developers.google.com/freebase/v1/search-overview

Method	Beam size	MetaQA (3-hop)		WebQSP		CWQ (test)	
		hit@1	F1	hit@1	F1	hit@1	F1
Sun et al. (2018)	N/A	-	-	66.4	51.9	-	-
Sun et al. (2019)†	N/A	91.4	-	68.1	-	45.9	49.3
Yang et al. (2019)	N/A	83.4	-	-	-	-	-
Lan et al. (2019)	N/A	-	-	68.2	67.9	39.3	36.5
Lan and Jiang (2020)	N/A	-	-	73.3	74.0	44.1	40.4
GRU	100	**99.9**	**99.9**	64.4	64.6	33.6	34.5
	10	**99.9**	**99.9**	63.8	63.9	32.4	33.1
	1	**99.9**	**99.9**	60.2	60.2	25.2	25.8
BERT2BERT	100	98.2	98.2	**74.3**	74.4	59.9	61.8
	10	98.2	98.2	73.0	73.1	56.9	57.8
	1	98.2	98.2	70.3	70.3	50.7	51.3
GPT2GPT2	100	**99.9**	**99.9**	72.5	73.6	61.1	62.8
	10	**99.9**	**99.9**	71.1	71.2	54.7	55.7
	1	**99.9**	**99.9**	68.2	67.9	48.8	49.6
BART-large	100	**99.9**	**99.9**	74.1	**74.6**	**66.4**	**68.2**
	10	**99.9**	**99.9**	73.1	73.6	60.0	60.9
	1	**99.9**	**99.9**	67.4	67.5	54.9	55.5

Table 1: Performance comparison with the previous answer prediction methods. † denotes the model using the manually annotated topic entities.

each of them, add a relation of the Freebase type "ns:type.object.name" and a 'FILTER' statement, as exemplified in Figure 1. The filter will identify the constraint entities by exact string match to the generated text labels.

We finally add the common prefix to the SPARQL query. The final SPARQL query of the proposed method is shown in Appendix B.2.

3 Experiments

We conducted experiments on the three datasets of MetaQA, WebQuestionsSP (WebQSP) and ComplexWebQuestions (CWQ) (See Appendix C.1 for detailed descriptions and statistics of the datasets and their knowledge bases).

3.1 Evaluation Results

Table 1 summarizes the evaluation results of the proposed method and the existing methods against the datasets, when comparing their resultant answer entities against the ground truth. The results show that our method outperforms the previous methods on all datasets (e.g. as for Hit@1, MetaQA: 8.5%, WebQSP: 0.8%, CWQ: 20.5% improvements). We also evaluated our method with different beam sizes (1, 10, 100), and the results show that the larger beam size leads to the higher performance of the models, though slowing down model inference speed. In addition, the GRU model uses the vocabulary from the questions and SPARQL queries on the training set, so the performance is much lower compared to Transformer models on CWQ (test) because of many unknown words on the test set.

Our method performs especially well on CWQ. To understand it well, we divide the questions according to the following perspectives: 1) Questions with 1-hop or 2-hops of relation path; 2) Questions with or without constraints; and 3) Question with the two most complex constraint types, *filter* and *order_by*. Table 2 shows the results of our method and the state-of-the-art method (Lan and Jiang, 2020) on those question subsets.[2] We find the followings: 1) If a question has a relation path with more hops, it is more difficult to get its correct answer, which is intuitive; 2) our method shows consistent performance for questions with or without constraints; and 3) our method shows approximately 25% higher performance over the state-of-the-art method for the questions with the two constraint types.

3.2 Ablation Study

To prove that our method is effective in handling the issue of *unseen* entities, we evaluated the method without the data pre-processing module, which learn to generate the original SPARQL query with entity IDs. Table 3 summarizes our models' performance on CWQ (test) and WebQSP in terms of Hit@1 with different model settings. 1) The system performance drops significantly (16% for CWQ, 7%∼8% for WebQSP) without the data pre-

[2]Note that the results in Table 2 are based on the beam size of 10 due to the training efficiency.

Method	1-hop (53.8%)	2-hop (42.8%)	non-CONS (17.3%)	CONS (82.7%)	CONS: *filter* (11.8%)	CONS: *order_by* (7.7%)
Lan and Jiang (2020)	41.6	30.6	25.8	38.7	23.3	22.8
BART-large	**62.4**	**58.9**	**60.6**	**59.8**	52.2	58.5
GPT2GPT2	57.3	52.7	57.5	54.1	45.4	58.5
BERT2BERT	58.3	55.8	52.9	57.4	**52.4**	**60.0**

Table 2: Performances for various categories of questions on CWQ (Hit@1). The proportion in parentheses indicates the ratio of the corresponding category of questions to the total number of questions. The work of Lan and Jiang (2020) is the state-of-the-art method on CWQ. CONS stands for constraints.

Setting	CWQ (test)		WebQSP	
	BERT	Bart	BERT	Bart
Proposed settings	**56.9**	**60.0**	**74.3**	**74.1**
w/ orig. SPARQL query	41.3	44.0	67.2	66.3
w/o TE type as input	56.1	58.5	73.5	72.4
w/ TE label (not type)	56.3	58.8	73.3	73.2
w/ TE type+label	56.2	59.8	72.6	73.4

Table 3: Performances based on different model settings. 'TE' stands for topic entity, and 'orig.' stands for 'original'. 'BERT' indicates BERT2BERT model, and 'BART' indicates BART-large model.

Error type	Proportion (%)
Incorrect Topic Entities	39.0
Incorrect Main Relations	22.3
Incorrect Constraint Relations	24.3
Incorrect Constraint Values	14.4

Table 4: Percentage of errors from BART-large model for CWQ dataset.

processing module, which learns to generate the original SPARQL query. These results show that our proposal of generating entity's text labels and retrieving entities by the labels is much better than directly generating entity IDs, effectively addressing the issue of *unseen* entities. 2) We tested variants of topic entity input to the query generation model, including no input of topic entity information, using the text label of topic entity instead of its type, and using both the type and the text label of topic entity. Using the type of topic entity shows the best performance.

3.3 Error Analysis

Table 4 shows the proportion of error types on the CWQ questions for our best performing BART-large model. The results show that about half of the errors are due to the incorrect relation path prediction, while majority of the rest of errors are due to the external tool of entity linking (FreeBase search API). We thus plan to work on, for instance, joint learning of SPARQL query generation and entity linking to address the latter error type.

3.4 Interpretability and Training Efficiency

Even if a model predicts an answer entity correctly, it may reach the answer entity accidentally via incorrect path in a knowledge base. We measure how well a model identifies relation path from topic entity and constraint entities to answer entity. Appendix C.2 shows that our models outperform the state-of-the-art method (Lan and Jiang, 2020) on the two datasets of CWQ and WebQSP. In particular, the Bart-large model shows 9% improvement over (Lan and Jiang, 2020) in terms of relation path prediction, compared to 0.8% improvement in terms of Hit@1. This result may indicate that (Lan and Jiang, 2020) optimizes for answer prediction, while our method optimizes for relation path prediction (in fact, for SPARQL query generation).

Our method also shows better training efficiency than the existing methods because it does not need to retrieve subgraphs like Sun et al. (2018, 2019). Please refer to Appendix C.3 for details of the training efficiency comparison.

4 Conclusion

We propose to improve complex KBQA by utilizing pre-trained encoder-decoder models to generate a normalized SPARQL query from questions. The proposed method outperforms previous models on all of three complex KBQA benchmarks and addresses unseen entities by translating entity IDs to SPARQL queries. In the future, we will explore combining relation classification with the constraint generation to reduce the space of beam search.

Acknowledgements

This research is supported by the Agency for Science, Technology and Research (A*STAR) under its AME Programmatic Funding Scheme (Project #A18A2b0046).

References

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *NeurIPS*.

Yu Chen, Lingfei Wu, and Mohammed J. Zaki. 2019. Bidirectional Attentive Memory Networks for Question Answering over Knowledge Bases. In *NAACL-HLT*, pages 2913–2923.

Rajarshi Das, Manzil Zaheer, Siva Reddy, Andrew Mccallum, and Computer Sciences. 2017. Question Answering on Knowledge Bases and Text using Universal Schema and Memory Networks. In *ACL*, pages 358–365.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *ACL*, pages 731–742.

A. Kalyanpur, S. Patwardhan, B. K. Boguraev, A. Lally, and J. Chu-Carroll. 2012. Fact-based question decomposition in DeepQA. *IBM Journal of Research and Development*, 56(3-4):1–11.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural Semantic Parsing with Type Constraints for Semi-Structured Tables. In *EMNLP*, pages 1516–1526.

Yunshi Lan and Jing Jiang. 2020. Query Graph Generation for Answering Multi-hop Complex Questions from Knowledge Bases. In *ACL*, pages 969–974.

Yunshi Lan, Shuohang Wang, and Jing Jiang. 2019. Knowledge base question answering with topic units. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5046–5052. International Joint Conferences on Artificial Intelligence Organization.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*, pages 7871–7880.

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 *ACL*, pages 23–33.

Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *EMNLP*, pages 1400–1409.

Michael Petrochuk and Luke Zettlemoyer. 2018. SimpleQuestions Nearly Solved: A New Upperbound and Baseline Approach. In *EMNLP*, pages 554–558.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*.

Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. 2020. Leveraging Pre-trained Checkpoints for Sequence Generation Tasks. *Transactions of the Association for Computational Linguistics*, 8:264–280.

Haitian Sun, Tania Bedrax-Weiss, and William W. Cohen. 2019. PullNet: Open Domain Question Answering with Iterative Retrieval on Knowledge Bases and Text. In *ACL*, pages 2380–2390.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. Open Domain Question Answering Using Early Fusion of Knowledge Bases and Text. In *EMNLP*, pages 4231–4242.

Alon Talmor and Jonathan Berant. 2018. The Web as a Knowledge-Base for Answering Complex Questions. In *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 *ACL*, pages 2326–2336.

Haihong Yang, Han Wang, Shuang Guo, Wei Zhang, and Huajun Chen. 2019. Learning to decompose compound questions with reinforcement learning.

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 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China. Association for Computational Linguistics.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL*, pages 8413–8426.

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 *ACL*, pages 571–581.

Jichuan Zeng, Xi Victoria, Lin Caiming, Xiong Richard, Michael R Lyu, Irwin King, and Steven C H Hoi. 2020. PHOTON : A Robust Cross-Domain Text-to-SQL System. In *ACL*, pages 204–214.

Haoyu Zhang, Jingjing Cai, Jianjun Xu, and Ji Wang. 2019. Complex Question Decomposition for Semantic Parsing. In *ACL*, pages 4477–4486.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *AAAI*, pages 6069–6076.

Mantong Zhou, Minlie Huang, and Xiaoyan Zhu. 2018. An interpretable reasoning network for multi-relation question answering. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2010–2022, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

A Related Work

Some works for multi-hop question answering over knowledge base (KBQA) adapt semantic parsing to convert question into knowledge base (KB) query for answer retrieval. Liang et al. (2017) presented a weakly supervised semantic parsing framework based on reinforcement learning algorithm, supporting language compositionality by augmenting seq2seq model with key-variable memory and executing the program of semantic parse in a high-level programming language for answer generation. Zhang et al. (2019) adapted the idea of question decomposition, as a part of end-to-end learning of semantic parsing. However, their resultant logical representation of question needs further steps before being converted into executable SPARQL queries.

Another approach is to learn matching between question vector and answer vector, by incrementally updating the question vector using key-value memory network. Miller et al. (2016) and Das et al. (2017) adapted key-value memory network for multi-hop QA, where the memory network stores knowledge base triples in form of (subject, relation, object) such that a subject and a relation are a key, and their object is the value of the key, and stores a window centered at an entity mention such that the entity is a key and the window is a value. The memory network incrementally updates question embedding with a weighted sum of value embeddings whose keys are relevant to the current question embedding, and the final updated question embedding is matched to the best answer entity embedding. Chen et al. (2019) presented a bidirectional attentive memory network (BAMnet) for KBQA, which extends the key-value memory network to learn two-way interactions between question and a KB. The approach offers better interpretability and performance with the attention mechanism of BAMnet than the key-value memory network (Miller et al., 2016; Das et al., 2017). However, those methods do not utilize existing semantic representations of known questions, which may explicitly guide them in the path from topic entity to answer entity.

The next approach is to identify the sequence of entities in the path from the topic entity of question to its answer entity in the KB. Sun et al. (2018) extracted a subgraph for given question out of the graph of a KB and documents, in which KB entities are linked to documents that contain the entity mentions, by using personalized PageRank and then learnt graph node representation conditioned on the question to classify if each node is an answer or not and to search the subgraph for the answer of the question. Sun et al. (2019) upgraded his previous method (Sun et al., 2018) for multi-hop KBQA by jointly learning iterative subgraph expansion and graph node classification, dynamically selecting the candidate nodes of the next hop. They also compared the interpretability of the two models by checking if the identified relation paths from topic entities to the predicted answers are correct or not when the predicted answers are correct.

The fourth approach is to explicitly predict relation types of question (Xu et al., 2016; Yu et al., 2017). Yu et al. (2017) presented a method that uses hierarchical bi-LSTM to get representations of question and each candidate relation path, estimates the similarity of the two representations by using cosine similarity and selects the relation path with the highest score.

The last approach to introduce is to decompose a complex question into a sequence of simple questions and to identify the answer of the complex question by merging the answers of the simple questions. Kalyanpur et al. (2012) presented a question decomposition framework that utilizes handwritten rules, which improves IBM Watson's QA system for Jeopardy. Talmor and Berant (2018) presented a seq2seq model, which learns to split a complex question into a sequence of sub-questions by using ComplexWebQuestions, and a search engine combined with a reading comprehension model for answering the sub-questions, and computed the final answer by applying symbolic operations such as union and interaction to the answers of the sub-questions. However, those methods cannot be straightforwardly adapted for KBQA as they find the answers of the decomposed simple questions from relevant documents retrieved by a search engine, but not from a knowledge base.

B Details of Methodology

B.1 Algorithm

The proposed method is schematically described in Algorithm 1.

B.2 Example SPARQL queries

1. The original SPARQL query of the question "Who were the 1980 NBA Finals champions that Lamar Odom is now playing for?" is as follows:

Algorithm 1 Relation Path Prediction by Text Generation Method

Given a question q, recognize all topic entities $\{e_i\}$ for $i \in \{1, \cdots, N\}$
$S = []$ // Initialize the set of candidate SPARQL queries
for $i = 1, \cdots, N$ **do** // for each topic entity
  $t_{e_i} = retrieve\_type\,(e_i)$ // Retrieve topic entity type
  $q' = [q, t_{e_i}]$ // Concatenate the question and the topic entity type into a single string
  $\{o_{ij}\}_{j=1}^{M} = generate(q')$ // Generate M candidate SPARQL queries given the concatenated string
  for $j = 1, \cdots, M$ **do** // for each candidate SPARQL query
    $s_{ij} = post\_process(o_{ij}, e_i)$ // Replace a placeholder with the ID of the topic entity
    $S = S + [s_{ij}]$ // Append the SPARQL query to the candidate set
  end for
end for
for $S_i \in \{S\}$ **do** // for each candidate SPARQL query
  $a_i = retrieve\_answers(S_i)$ // Retrieve the answers with given SPARQL query in the knowledge base
  if $count(a_i) > 0$ **then** // Terminate when the answer set is not empty
    Select a_i as final answers for question q
    break
  end if
end for

```
SELECT DISTINCT ?x WHERE {
        ns:m.02_nkp ns:sports.pro_athlete.
             teams ?y .
        ?y ns:sports.sports_team_roster.team ?
             x .
        ?x ns:sports.sports_team.championships
              ns:m.08x9_6 .
}
```

The "ns:m.02_nkp" is the topic entity named "Lamar Odom", the $?y$ denotes the intermediate entities at the first hop, and the $?x$ denotes the target answer entities. The answer entities are also constrained by the entity "m.08x9_6" named "1980 NBA Finales".

2. The original SPARQL query is modified by the data pre-processing module as follows:

```
SELECT DISTINCT ?0 WHERE {
        [ENT] ns:sports.pro_athlete.teams ?1 .
        ?1 ns:sports.sports_team_roster.team
             ?0 .
        ?0 ns:sports.sports_team.championships
              [SC] "1980_NBA_Finals" [EC]
}
```

3. The SPARQL query generated by the query generation modules is modified by the post-processing module as follows:

```
SELECT DISTINCT ?0 WHERE {
        ns:m.02_nkp ns:sports.pro_athlete.
             teams ?1 .
        ?1 ns:sports.sports_team_roster.team
             ?0 .
        ?0 ns:sports.sports_team.championships
              ?c0 .
        ?c0 ns:type.object.name ?c00
        FILTER(STR(?c00)="1980_NBA_Finals")
}
```

The variables ?0 and $?c0$ in the generated SPARQL query denote the answer entities and the constraint entities respectively. The query includes a "filter" statement to filter correct constraint entities using exact string matching.

B.3 Pre-trained encoder-decoder models

- **Bert2Bert** (Rothe et al., 2020): The encoder is a pre-trained BERT-base model (Devlin et al., 2019). The decoder is another pre-trained BERT-base model and connected to the encoder via cross-attention, though fine-tuned independently from the encoder.
- **GPT2GPT2** (Rothe et al., 2020): Both the encoder and the decoder are pre-trained GPT2 models and connected via cross-attention, though fine-tuned separately.
- **BART** (Lewis et al., 2020): This model is a pre-trained BART-base encoder-decoder model (Lewis et al., 2020).

B.4 Training details

The learning rate we chose for training the Transformer models is $\alpha = 5 \times 10^{-5}$, and the learning rate for GRU model is $\alpha = 1 \times 10^{-4}$. We used the number of epochs $E = 45$ for Transformer models and $E = 100$ for GRU models, and the Adam optimizer with $\epsilon = 10^{-8}$ and $\beta_1 = 0.9, \beta_2 = 0.999$. For the BART model, we set the dropout rate to 0 for both activation layers and attention layers. For the GRU model, we set the dropout rate to 0.4 for both dense layers and GRU layers. We select the model with the highest accuracy of SPARQL query generation on the dev set.

C Details of Experiments

C.1 Datasets

MetaQA: Zhang et al. (2018) constructed more than 400k single and multi-hop (up to 3-hop) questions, as an extension of single-hop questions of

Dataset	Number of questions/answers			Knowledge base statistics			
	Train	Dev	Test	KB	Entity	Relation	Triple
MetaQA 3-hop	114,196	14,274	14,274	WikiMovies	43K	9	135K
ComplexWebQuestions	27,623	3,518	3,531	Freebase	7.62M	8,664	33.8M
WebQuestionSP	2848	250	1639	Freebase	10.3M	646	17.45M

Table 5: Dataset and knowledge base statistics.

Coverage	Pattern
29.71%	$(?y \xleftarrow{r_1^{(1)}} B) \xrightarrow{r_0^{(1)}} ?x$
23.29%	$A \xrightarrow{r_0^{(1)}} ?y \xrightarrow{r_0^{(2)}} ?x \xleftarrow{r_1^{(1)}} B$
22.60%	$A \xrightarrow{r_0^{(1)}} ?x \xleftarrow{r_1^{(1)}} B$
6.75%	$((?y \xleftarrow{r_1^{(1)}} B_1) \xrightarrow{r_0^{(1)}} ?z \xleftarrow{r_2^{(1)}} B_2) \xrightarrow{r_0^{(2)}} ?x$
5.64%	$(?y \xleftarrow{r_1^{(1)}} B) \xrightarrow{r_0^{(1)}} ?z \xrightarrow{r_0^{(2)}} ?x$

Coverage	Pattern
54.53%	$A \xrightarrow{r_0^{(1)}} ?x$
16.85%	$A \xrightarrow{r_0^{(1)}} ?y \xrightarrow{r_0^{(2)}} ?x$
16.12%	$(A \xrightarrow{r_0^{(1)}} ?y \xleftarrow{r_1^{(1)}} B) \xrightarrow{r_0^{(2)}} ?x$
6.21%	$A \xrightarrow{r_0^{(1)}} ?x \xleftarrow{r_1^{(1)}} B$
5.02%	$A \xrightarrow{r_0^{(1)}} ?y \xrightarrow{r_0^{(2)}} ?x \xleftarrow{r_1^{(1)}} B$

Table 6: Top-5 patterns of relation paths and their coverage in CWQ (left) and WebQSP (right).

the WikiMovies dataset Miller et al. (2016). The MetaQA dataset does not provide SPARQL queries, but question types from which we can identify relation paths and write corresponding SPARQL queries. Note that the SPARQL queries of the MetaQA dataset do not contain any constraint. We report evaluation results against the whole dataset with up to 3-hop questions.

WebQuestionsSP (WebQSP): It contains 4.7k questions which require up to 2-hops of reasoning in the KB and are answerable against Freebase (Yih et al., 2015).

ComplexWebQuestions (CWQ) (v1.1): It consists of multi-hop questions against Freebase, which are constructed by increasing the complexity of SPARQL queries from WebQuestionsSP and collecting the corresponding complex questions via crowdsourcing (Talmor and Berant, 2018). The questions require up to 3-hops of reasoning on the KB. We report evaluation results against the test subset of the dataset.

Table 5 shows basic statistics of those datasets and their knowledge bases.

C.2 Interpretability

Table 6 shows top-5 frequent patterns of relation paths in a linear form and their coverage in CWQ and WebQSP. An arrow denotes a relation, A is a topic entity, B or B_i is a constraint entity or constraint value, $?y$ and $?z$ are intermediate entities, and $?x$ indicates the answer entity. $r_0^{(j)}$ is the j-th relation in the relation path from the topic entity to the answer entity. $r_i^{(j)}$ is the j-th relation in the relation path from i-th constraint entity to one of intermediate entities on the path from the topic entity to the answer entity. As shown in Table 6, most of the questions in the complex KBQA datasets can be semantically represented as combinations of relation paths. We thus evaluate the interpretability of complex KBQA models in terms of how correctly the models identify all the relation paths given a question.

Table 7 summarizes the interpretability evaluation results of our models against the state-of-the-art method (Lan and Jiang, 2020) on WebQSP and CWQ, showing that our models outperform (Lan and Jiang, 2020) on both datasets. In particular, the Bart-large model shows 9% improvement over (Lan and Jiang, 2020) in terms of relation path prediction, compared to 0.8% improvement in terms of Hit@1. This result may indicate that our models show significantly higher interpretability than (Lan and Jiang, 2020).

In Table 7, *Acc. R* indicates the accuracy of identifying the main relation path from topic entity to answer entity, and *Acc. C* indicates the accuracy of identifying a relation path from a constraint entity to one of intermediate entities on the path from topic entity to answer entity. *Joint Acc* is the accuracy of identifying all the relation paths correctly. For the simplicity of comparison, we do not compare constraint entity or value but only compare relation types in the paths. For the CWQ dataset, Lan and Jiang (2020) use a query graph different from our query graph from the original SPARQL query. To ensure fair comparison, we consider the

Method	WebQSP			CWQ		
	Joint Acc	Acc. R	Acc. C	Joint Acc	Acc. R	Acc. C
Lan and Jiang (2020)	57.6	70.7	78.3	24.0	52.5	42.2
Bart-large	**66.9**	**72.2**	85.7	**67.6**	**77.9**	**78.6**
BERT2BERT	66.4	71.8	**86.1**	65.5	77.1	76.6
GPT2GPT2	65.7	71.1	85.3	66.4	77.5	77.5

Table 7: Interpretability evaluation results

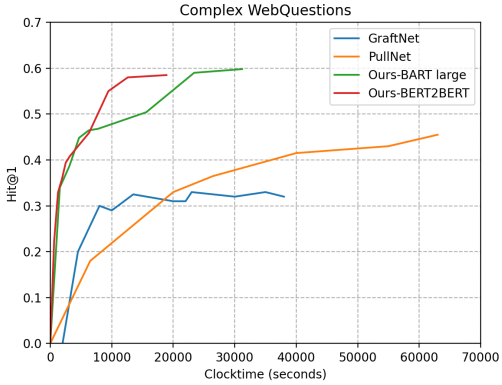

Figure 2: Training efficiency of our model compared to baselines under clock training time.

accuracy of relation paths either from the original SPARQL query as presented above or from the gold query graph of (Lan and Jiang, 2020).

C.3 Training Efficiency

Figure 2 depicts the training process speed of our models and two baselines (Sun et al., 2018, 2019) as the models grow to show higher performance in terms of Hit@1. The BART-large model performs the best but is slower than the BERT2BERT model. Our methods show better efficiency than other baselines because the methods don't need to retrieve subgraphs like Sun et al. (2018, 2019). Lan and Jiang (2020) needs to retrieve subgraphs of query graphs in every step of the training stage and takes much longer time to train than the other baselines. Therefore, we did not include it in the comparison for training efficiency.