Knowledge-Based Systems 253 (2022) 109576

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Staged query graph generation based on answer type for question answering over knowledge base

Haoyuan Chen, Fei Ye *, Yuankai Fan, Zhenying He *, Yinan Jing, Kai Zhang, X. Sean Wang

Fudan University, Songhu Road 2005, Shanghai, 200438, China

ARTICLE INFO

Article history:
Received 9 January 2022
Received in revised form 23 July 2022
Accepted 27 July 2022
Available online 1 August 2022

Keywords:
Knowledge base
Question answering
Semantic parsing
SPARQL
RDF

ABSTRACT

Question answering over knowledge base (KBQA) enables users to query over the knowledge base without the need to know the details. A range of existing KBQA approaches treats the entities mentioned in the given question as the starting point to find the answers. While helpful in achieving improvements on the existing benchmarks, they have some limitations on the strategy of query graph generation, which creates too many candidate queries and makes it hard to select the best-matching one to get the answer. In this paper, we propose a staged query graph generation approach based on the answer type, which exploits the correlation between questions and answer types to reduce the size of the candidate set and further improve the performance. Besides, we construct a question/answer-type (QAT) dataset aiming to predict the answer type of a given question. Extensive experiments demonstrate our method outperforms existing methods on both simple questions and complex questions.

1. Introduction

Facts are descriptive knowledge that can be stored in knowledge bases (KB) for use by many applications [1,2]. The knowledge graph (KG) is a knowledge base that can be used for knowledge representation and management. Recently, there are many knowledge graphs constructed and released, such as Freebase [3], YAGO [4] and DBpedia [5], which contain billions of facts stored as RDF [6–8] triples (i.e., subject, predicate, object) that can be queried by SPARQL [9,10]. The advent of knowledge graph creates many downstream tasks, one of which named question answering over knowledge base (KBQA) [11], has received widespread attention. KBQA aims to answer the input questions expressed in natural language (NL) based on the given knowledge base. Most of the current approaches treat KBQA as a semantic parsing task [12–19]. They first transform the NL question into a logical form. Then they convert the logical form to an executable query statement (i.e., SPARQL). Finally, they get the answer by executing the query statement over the given knowledge base.

Most previous semantic parsing-based methods [16–18] propose to use query graph (Definition 2) to represent the logical form of the given NL question and make a pipeline to generate the candidate query graphs: **S1: Entity linking.** These methods first get all entities ("*Family Guy*" and "*Meg*") from the NL question and then select one entity ("*Family Guy*") as the topic entity, as shown in Fig. 1(*S*1). **S2: Core inferential chain** [16]. These methods assume that the distance between the topic entity and the answer is no more than two hops. Therefore, they start from the topic entity and generate the core inferential chain by exploring all one-hop or two-hop paths, as shown in Fig. 1(*S*2). **S3: Candidate query graph generation.** They add some self-defined constraints (Definition 1) to the core inferential chain to get all the candidate query graphs. For example, there is a candidate query graph in Fig. 1(S3). **S4: Ranking candidate query graphs.** They compute the similarity between the candidate graph with the NL question and hence get the top-1 candidate query graph as the result.

Definition 1 (*Constraint*)**.** A constraint is defined as a triple<*e*, *p*, *c*>, where *e* is a vertex in the core inferential chain, *c* is a constraint vertex and *p* is the path between *e* and *c*. The constraints can be divided into four types: entity constraint, type constraint, time constraint and aggregation constraint.

Definition 2 (*Query Graph*)**.** A query graph is defined as a set of triples containing the core inferential chain and constraints extracted from the given NL question, and it can be directly converted into an executable query.

While the accuracy of answering the questions has improved through these approaches, one problem remains: they generate too many noisy query graphs. Thus it is hard to get the most

* Corresponding authors.
E-mail addresses: haoyuanchen19@fudan.edu.cn (H. Chen), fye21@m.fudan.edu.cn (F. Ye), ykfan19@fudan.edu.cn (Y. Fan), zhenying@fudan.edu.cn (Z. He), jingyn@fudan.edu.cn (Y. Jing), zhangk@fudan.edu.cn (K. Zhang), xywangcs@fudan.edu.cn (X.S. Wang).

https://doi.org/10.1016/j.knosys.2022.109576
0950-7051/

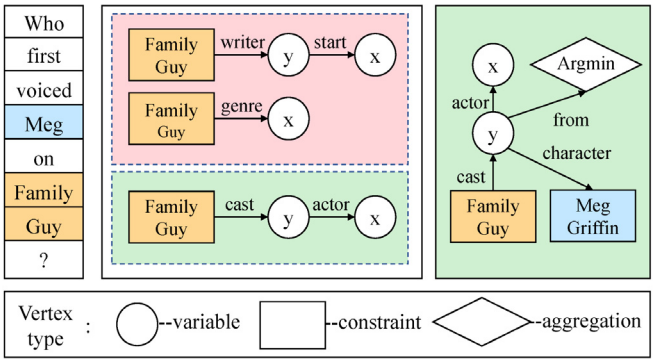


Fig. 1. An example query to explain the existing approach [16] and its problem.

similar query graph to the question. For example, as shown in *S*2 in Fig. 1, there are three core inferential chains, but two are wrong (red boxes), resulting in many noisy query graphs.

To solve the above problem, we propose a staged query graph generation method by introducing the answer types of the questions. More specifically, our approach consists of three stages. In the first stage, we train an answer type classification model. In the second stage, we propose a staged query graph generation based on answer type, which starts from the answer and can filter many "noisy" query graphs. In the final stage, we rank the candidate query graphs, select the top-1 as the most similar query graph to the NL question, and execute the corresponding SPARQL query to get the answer.

However, there is a challenge to train a model to predict the answer type of the given question which lies in the dataset. To the best of our knowledge, there is no question/answer-type (QAT) dataset for Freebase. To address this problem, we construct a dataset called CWQAT from ComplexWebQuestions (CWQ) [20], which contains 31158 QAT pairs divided into 280 classes.

In summary, the contributions of this paper are as follows:

- We propose an answer type-based method to generate candidate query graphs, which can filter many noisy candidate query graphs and make our approach more accurate.
- We construct a new QAT dataset to explore the correlation between the questions and answer types. It has been released and can be downloaded freely.[1]
- We conduct comprehensive experiments on multiple QA datasets, and our approach outperforms previous works on both simple questions and complex questions.

The rest of this paper is organized as follows: Section 2 discusses the related work of the KBQA system. Section 3 introduces our approach. Section 4 presents the details of the experiments and shows the experimental results. And Section 5 concludes this paper.

2. Related work

KBQA has been studied for several decades, and most approaches can be divided into two classes: information retrieval-based methods and semantic parsing-based methods.

2.1. Information retrieval-based methods

The information retrieval-based methods try to get candidate answers to the questions and rank them. GCN and network embedding [21–24] use models to embed a graph into a vector space. Knowledge Tracing [25–27] is a popular knowledge embedding method, which estimates how well students have mastered a concept based on their multi-modal historical records to embed a graph. These methods allow the information retrieval-based methods to embed knowledge graph information into a low-dimensional space and find answers to the questions. The information retrieval-based methods always differ in both the way to get candidate answers and the method to rank them. Yao and Durme [28] adopt some rules to extract hand-crafted features from the dependency parse results of questions and then feed them into a classification model. Bordes et al. [29] first propose an embedding model which can learn the low-dimensional vector representations of the question words and knowledge base to predict the confidence of the candidate answers. Later Bordes et al. [30] propose a subgraph embedding-based method to learn the low-dimensional features, which can get more information about the candidate answers. Dong et al. [31] utilize a CNN-based model for encoding the questions, and Hao et al. [32] apply an LSTM-based model [33] to encode them. Xu et al. [34] leverage Wikipedia free text to improve the performance of the system. Saxena et al. [35] embed the knowledge graph into the model to compare the similarity between given question and candidate answers. Li et al. [36] propose a graph summarization technique using Recurrent Convolutional Neural Network (RCNN) and GCN to improve KBQA. To find the answer to the question as much as possible, recent researches retrieve answers over knowledge graph with multiple hops of reasoning. Chen et al. [37] use external knowledge to enrich the embeddings of current knowledge graph information for multi-hop KBQA. Bi et al. [38] use question encoding and relation encoding to expand the search space of answers.

The methods mentioned above often focus on extracting relations and topic entities. Then they adopt an encoding model to embed the questions, knowledge base, and answers into a low-dimensional space. Finally, they compute each score of candidate answers to the question, such as Saxena et al. [35]. However, these methods obtain target answers directly from the question and knowledge base without leveraging the query structure to the question.

2.2. Semantic parsing-based methods

To make the approaches of KBQA more interpretable, the semantic parsing-based methods translate the given NL question into a logical form, which has much meaning and can be directly translated into an executable query statement by rules. These methods differ in several ways, such as the logical form and the way to compute the similarity between the logical form and NL question. Liang et al. [12] define Lambda Dependency-Based Compositional Semantics (λ-DCS) as the logical form, which can be generated by using all logical operations. Zou et al. [13] and Hu et al. [14] propose to use Semantic Query Graph (SQG). They extract entities and relations from the question firstly and combine them to construct SQG. Abujabal et al. [15] generate templates that can map dependency parse tree to a query graph template. Yih et al. [16] define the query graph as the formal meaning representation. They first extract entities from the question and then build the query graph guided by the knowledge graph. Bao et al. [17] define some constraints to construct the query graph based on [16]. Luo et al. [18] come up with a new method to compare the similarity between the question and query graphs. Chen et al. [19] utilize abstract query graph (AQG) to guide query graph generation. Qin et al. [39] propose a new way to generate query graphs. They start with the entire knowledge base and gradually shrink it to the desired query graph. Bakhshi et al. [40]

[1] https://https://github.com/chy000000/CWQAT

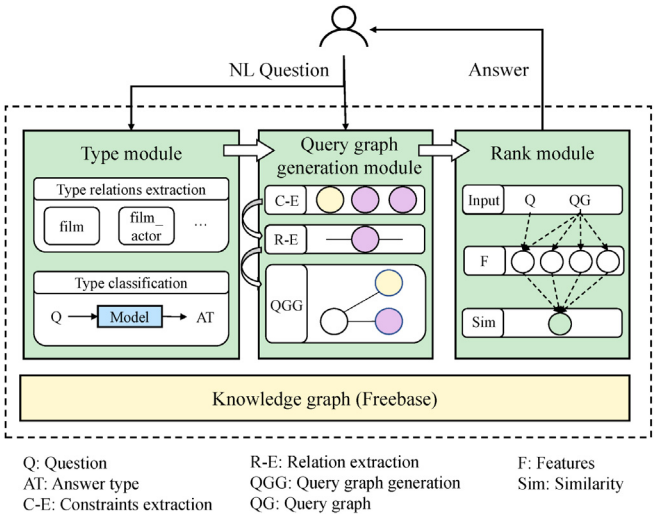


Fig. 2. Overview of the proposed approach.

propose to reorder the words of the question and then locally parses the new sequence of words backward. Finally, they refine the uncertain question graph by eliminating the useless nodes and triples. To explore multi-hop query graph generation, Lan et al. [41] allow a longer relation path in the query graph generation. They use an iterative search method to generate the query graph, which inserts new relations in each iteration to expand the scope of the query graph. Xiong et al. [42] propose DPG to efficiently compute the path similarity, which can dynamically expand the candidate path set with increasing hop number. Some researchers use relation prediction to improve KBQA, such as Li et al. [43] and Zhao et al. [44].

The methods mentioned above use a pipeline to answer the given NL question. The key to these methods is how to generate the defined candidates of logical form from the NL question and how to rank them, such as Yih et al. [16] and Luo et al. [18]. The main difference between our method and the previous methods is that we use the specific answer type to reduce the size of the candidate set, and thus it is easier to rank them.

3. Approach

In this section, we introduce our approach. We first show the overview of our approach in Section 3.1. Then we talk about the details of the type module in Section 3.2. Next, we discuss the process of our candidate query graph generation in Section 3.3. Finally, we present the strategy of ranking in Section 3.4.

3.1. Overview

Our approach follows existing approaches [16–18] to extract all kinds of constraints from the NL questions and generate the candidate query graphs. Unlike the existing semantic parsing-based methods, our approach predicts the answer type of a given NL question and then leverages the answer type to generate all candidate query graphs.

The overview of our approach is shown in Fig. 2. Without loss of generality, in this paper, we use Freebase as the knowledge base, which has more than 46 million topics and 2.6 billion facts. We divide our approach into three modules:

(1) **Type module**. In this module, we first extract the entity type relations (Definition 3) of each entity type in Freebase. Then we predict the answer type of the questions and hence get the corresponding type relations.

(2) **Query graph generation module**. In this module, different from the existing works, which generate one-hop or two paths as the core inferential chains starting from a topic entity, our approach starts from the answer node. Then it adds constraints to the graph by the same relation between the constraints and the answer node. We aim to use this strategy to reduce the noise of the candidate query graphs, and the output of this module will be input to the last module.

(3) **Rank module**. We apply an existing LSTM-based model to compute the similarity between the candidate query graphs and the given questions. Then we train a ranking model by a set of meaningful features to get the best query graph.

Definition 3 (*Type Relations*)**.** Type relations are a set of relations that every entity has with the same entity type. For example, the entity type "actor" has many entities in Freebase, like "Natalie Portman", "John Noble", etc., and we will extract all of one or two-hop relations of the entities as this type relations.

3.2. Type module

We divide the type module into two parts, type relations extraction and type classification. We first extract all entity types' relations from Freebase. Then, for a given question, we use a neural network model to predict the answer type and hence get the corresponding type relations. We introduce how to generate the type relations and three features of type relations in Section 3.2.1, the way to collect the dataset in Section 3.2.2, and the detail of type classification in Section 3.2.3.

3.2.1. Type relations extraction

We extract relations of every entity type and three features of each relation from Freebase. Specifically, for a given entity type, we look up all entities of the entity type in Freebase. Then we look up all one-hop or two-hop relations for each entity and count the frequency of each relation. Inspired by TF-IDF, we propose three features to estimate the importance of each relation to a entity type, namely Relation Proportion (RP, Definition 4), Type Relevance (TR, Definition 5) and Relation Rate (RR, Definition 6). Fig. 3 shows two entity types and part of their relations with the three features.

Definition 4 (*Relation Proportion*)**. RP** is the proportion of a relation to a specific entity type, and it can describe the degree of relation associated with an entity type. We define RP as follows:

$$RP_{ij} = \frac{Count(R_i, T_j)}{Count(T_j)} \tag{1}$$

where RP_{ij} is the relation proportion of ith relation to jth entity type, $Count(R_i, T_i)$ is the number of $Relation_i$ occurring in all entities of $Type_j$, and $Count(T_i)$ is the number of entities of $Type_j$ in Freebase.

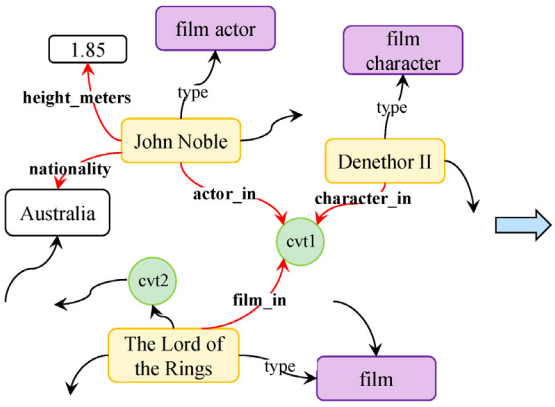
Definition 5 (*Type Relevance*)**. TR** describes the ability to distinguish different entity types of relations. If most entity types have the same relation, it will have a low type relevance. We define TR as follows:

$$TR_i = log\frac{Sum(T)}{1 + Count(TRC_i)} \tag{2}$$

where TR_i is the type relevance of ith relation, $Sum(T)$ is the number of all entity types in Freebase, and $Count(TRC_i)$ is the number of entity types that contain the $Relation_i$.

Definition 6 (*Relation Rate*)**. RR** combines RP and TR into a single metric. It can reflect how important a relation is to an entity type. We can compute it as follows:

$$RR_{ij} = RP_{ij} * TR_i \tag{3}$$



(a) A subgraph in Freebase

type	relation	RP	TR	RR
film actor	height_meters	0.013	0.223	0.002
	nationality	0.274	0.133	0.036
	actor_in-film_in	0.281	0.618	0.173
	actor_in-character_in	0.292	0.784	0.228
	.....			
film	film_in-actor_in	0.273	0.556	0.151
	film_in-character_in	0.286	0.784	0.224
	.....			
.....				

(b) Type relations and their three features

Fig. 3. Example of type relations extraction. **(b)** shows two entity types and their part relations can be extracted from **(a)**.

Algorithm 1 Question Clustering

Input: NL question set $Q = \{q_1, q_2, ..., q_n\}$
the corresponding sets of answer type $S = \{s_1, s_2, ..., s_n\}$
the similarity threshold ε
Output: The clusters $C = \{c_1, c_2, ..., c_k\}$

$C = \emptyset$
$cur = 1$
while *hasUnlabeledQuestion*(Q) **do**
  $target = chooseUnlabeledQuestion(Q)$
  $c_{cur} = \{q_{target}\}$
  $label(q_{target})$
  foreach $q_i \in Q$ **do**
    if not isLabeled(q_i) **then**
      $similarity = Jaccard(s_{target}, s_i)$
      if $similarity \geq \varepsilon$ **then**
        $c_{cur}.add(q_i)$
        $label(q_i)$
      end if
    end if
  end for
  $cur+ = 1$
end while
return C

3.2.2. CWQAT dataset collection

Recently, the Semantic Answer type prediction (SMART) task [45] was organized as a challenge at the 2020 International Semantic Web Conference (ISWC'20), which aims at predicting the answer type of a given NL question. However, it only contains the DBpedia dataset and the Wikidata dataset. There are no QAT pairs for Freebase. Therefore, we aim to build a QAT dataset whose answer types belong to Freebase. In addition to questions and answers, CWQ also provides the corresponding entities in Freebase for the answers. Therefore, we can get the types of each answer in Freebase. However, each answer may correspond to more than one type, and thus we have to choose one or more relevant types to the question.

Towards tackling the problem, we first cluster the questions by answer types. More specifically, we can get a set of answer types corresponding to each question through the answer type. Then, we think the questions belong to the same category, whose answer types are similar. Thus, we use the similarity between the set of answer types to represent the distance between questions. We use Jaccard to calculate the similarity between two sets of answer types:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{4}$$

Then, we can get the top-10 types with the highest frequency of each cluster. Finally, for each question, we label it with the answer types, which are in both its corresponding answer type set and the top-10 types of its cluster.

Algorithm 1 shows the process of question clustering. Given an NL question set Q, the corresponding sets of answer type S, and the similarity threshold ε, we first choose an unlabeled question as the target of a new cluster and mark it. Then, for each $q \in Q$, we calculate the Jaccard score between it and the target. If the score is greater than ε, we add it to the same cluster and mark it. Finally, if there are unlabeled questions, we go through the above process; otherwise, we return the clustering results.

The details of CWQAT can be found in Section 4.1 and Appendix.

3.2.3. Type classification

The answer type of a given NL question is essential for the precision of the final prediction [46]. For example, in the question "What movies does Michael Jeffrey Jordan play in?", if we do not know the answer type of this question, we may generate a candidate query "*select ?x where { Michael Jeffrey Jordan play_in ?x}*", which is the most similar to the original question. But this query statement will generate not only the answers of the movie but also some wrong answers of other types, such as the Chicago Bull, an American professional basketball team. Therefore, this strategy to generate candidate query graphs will hurt the precision without answer type. Walker et al. [46] have shown that identifying the answer type of a given NL question can boost the performance of their system.

Previous works use some manual rules [17] or some static methods [18] to obtain the answer type, which is limited and hard to expand. Instead of using these static methods, we propose a model which uses both information of NL question and knowledge base. Fig. 4 shows an example of our model predicting the answer type of the given NL question. First, we use a pre-trained model to get the sentence information of the NL question. Then we add the knowledge information according to the entities extracted from the NL question. Finally, we use a linear classifier to predict the answer type of the given NL question. The detail of our model is followed.

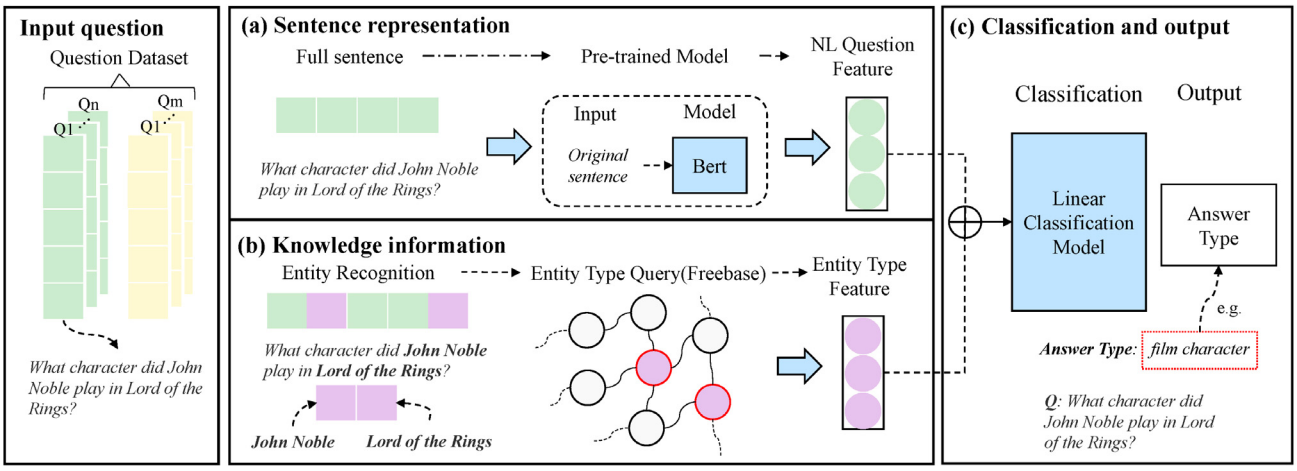


Fig. 4. A running example of our model.

(a) Sentence representation

We leverage BERT to get the features of the NL questions, which is an NLP model developed by Google for pre-trained language representations and is trained with a vast amount of text data publicly on the web in an unsupervised manner. Specifically, we transform the NL question into a formal representation and feed it into the model. Finally, we can obtain a vector as the features of the NL question. The process of this part can be formalized as follows:

$$Q = \{w_1^s, w_2^s, \ldots, w_n^s\} \tag{5}$$

$$W_i^s = embed(w_i^s), i \in [1, n] \tag{6}$$

$$f_q = BERT(W_i^s), i \in [1, n] \tag{7}$$

where Q is an NL question, and w_i^s is each word in Q, which is embedded into a vector $W_i^s \in R^{d_v}$ and fed into the BERT to get the sentence features.

(b) Knowledge information

To build a bridge between NL question and the knowledge graph, we add the knowledge graph information related to the question through the entities extracted from the given NL question. For an NL question q, we generate a Freebase entity type vector $T = \{t_1^q, t_2^q, \ldots, t_k^q\}$ as the information of the knowledge graph, and the length k of this vector equals the number of entity types in Freebase. This vector represents what types of entities are included in the question, and we use the one-hot encoding method to encode it. For example, for the question "What character did John Noble play in Lord of the Rings?", we can get the entities "John Noble, The Lord of the Rings" and their corresponding entity type "actor, film". Then we set the positions of these entity types in the entity type vector to 1, and the other entity type positions are set to 0. We use entity type vector because we believe that the types of entities in question greatly influence the answer type of the question. As in the example above, the entity types "actor" and "film" are relevant to the answer type "character". The improvement of the experimental results also proves the effectiveness of the method. Then we use a linear layer to get this part of the features. The feature extraction in this part is as follows:

$$T = \{t_1, t_2, \ldots, t_k\}, k \in [1, K] \tag{8}$$

$$f_k = ReLU(W * T + b) \tag{9}$$

where K is the number of entity types and f_k is the output feature.

(c) Classification and output

To predict the answer type of the given NL question, we concatenate the NL question features and the knowledge features together, and feed them into a fully connected layer. Finally, we utilize softmax as the activate function, and we use multiple classification cross-entropy loss as our objective function. The final process is as follows:

$$f = [f_q; f_k] \tag{10}$$

$$p = softmax(W_p * f + b_p) \tag{11}$$

$$L = -\sum_{i=1}^{K} y_i \log(p_i) \tag{12}$$

where y is the label of data and p is the predicted probability.

Algorithm 2 Candidate Query Graphs Generation

Input: NL question Q, the knowledge graph G, the type relations set T

Output: Candidate Query Graphs CQG

```
CQG(Q) = ∅
t = TypePrediction(Q)
τ = TypeRelationMapping(t, T)
C = ConstraintDetection(Q)
foreach c ∈ C do
    tmp = ∅
    R = ConstraintRelationGeneration(c, G)
    foreach r ∈ R do
        if r ∈ τ then
            foreach g ∈ CQG(Q) do
                g_c = ContraintBinding(g, c)
                tmp.append(g_c)
            end for
        end if
    end for
    CQG(Q).append(tmp)
end for
return CQG(Q)
```

3.3. Query graph generation

This section introduces the details of our staged candidate query graphs generation method. We do not follow the general approach proposed by [16] and used in [17,18] to generate the candidate query graphs. We propose a new candidate query graphs generation strategy which starts from the answer node

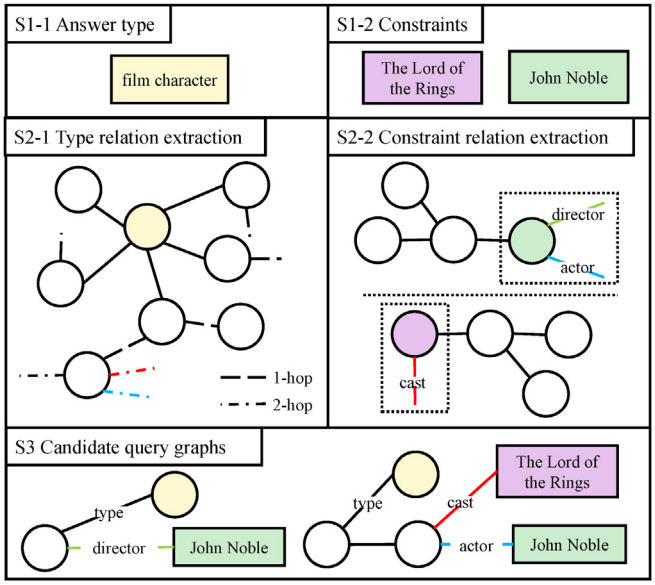


Fig. 5. An example of candidate query graph generation for the question "What character did John Noble play in Lord of the Rings?".

by the answer type. Fig. 5 shows an example of generating a candidate query graph by our method.

Step 1: Constraints extraction. We can extract all possible constraints from the given NL question, refer to S1-1 and S1-2 in Fig. 5. For entity constraints extraction, we extract all the entities mentioned in the NL question by using the state-of-the-art entity tool S-MART [47]. For type constraints extraction, we leverage an NLP model proposed in Section 3.2.3 to predict the answer type of the natural question. For time linking, referring to [18], we extract time mentions by matching year regex and month regex.

Step 2: Relation extraction. We get the type relations of the answer type and extract one-hop relations of each entity constraint. For example, in Step 2 in Fig. 5, we can get the type relations of "film character", which can be one or two hops, and regard them as our core inference chains. Then we can get the one-hop relations of "John Noble" and "The Lord of the Rings".

Step 3: Candidate query graph generation. Inspired by Mintz [48] that if two entities have a relation in a knowledge base, the sentence which includes these two entities will express the relation in some way, we assume that if both the answer and constraint have the same relation, the question will express the relation between the answer and constraint in some way. Therefore we can add these constraints to the core inference chains by the same relations. Step 3 in Fig. 5 shows one of the candidate query graphs that might be generated. We can add the entity constraints "John Noble" to the basic graph by the relation "actor", and we can add the other entity constraints "The Lord of the Rings" to this graph by the relation "cast".

Algorithm 2 describes how to generate all candidate query graphs $CQG(Q)$. Firstly, given the NL question Q, the knowledge graph G and the set of all type relations, we set $CQG(Q)$ empty, get the answer type t of Q and the corresponding type relations τ, and we extract all constraints C from Q. Secondly, for each constraint $c \in C$, we first set a temp set tmp empty to store the query graphs after adding c to previous candidate query graphs. Then we extract all relations R of c from G. Next, for each $r \in R$, if $r \in \tau$, which means both the answer and c have r, we can add c to previous candidate graphs g to generate new candidate query graphs. Finally, we add all graphs of tmp to $CQG(Q)$. By doing this, we can get all candidate query graphs $CQG(Q)$ of Q.

Table 1
Features and description.

Category	Feature	Description
Entity	S_{ent}	The average score of all entities linking;
	N_{ent}	Number of entities;
Graph	N_{con}	Number of each kind of constraints in G;
	S_{RP}	The average relation proportion of relations in G;
	S_{TR}	The average type relevance of relations in G;
	S_{RR}	The average relation rate of relations in G;
	$S_{sim(q,G)}$	The similarity score between NL question and query graph;
	N_{ans}	Number of output answers.

3.4. Query graph ranking

To get the most similar query graph to the given question from the candidate query graphs, we define several features shown in Table 1 and use a one-layer neural network model to rank them. Specifically, we compute two types of features: entity features and graph features. Entity features measure the quality of entities extracted from NL questions. Graph features consider the association between relation and the question in the query graph, and the relevance between relation and the answer type, etc. For the similarity score $S_{sim(q,G)}$ between NL question and candidate query graph, we use LSTM to compute it. Instead of regarding it as a binary classification task, we view it as a ranking task. Therefore, during training, we adopt margin ranking loss to maximize the distance between positive query graphs G^+ and negative query graphs G^-:

$$L = max\{0, margin - S(q, G^+) + S(q, G^-)\} \tag{13}$$

The detail of training data generation can be found in Section 4.1.

4. Experiments

In this section, we introduce the settings of our experiments and the two data sets we use to evaluate our approach, followed by the main experimental results and detailed analysis.

4.1. Experimental setup

QA datasets: We evaluate our approach on ComplexQuestions (CompQ) [17] and WebQuestion (WebQ) [12]. CompQ is a QA dataset over Freebase, which contains 2100 QA pairs collected from a three-month search query log. The training set has 1300 QA pairs, and the testing set has 800 QA pairs.[2] WebQ is also a QA dataset built on Freebase, which contains 5810 QA pairs collected from Google Suggest API. The training set has 3778 QA pairs, and the testing set has 2032 QA pairs.[3]

CWQAT dataset: We collect 31158 QAT pairs from CWQ and split them into 27639 for training and 3518 for testing. We divide these data into 204 clusters by clustering, and finally, we get 104 different classes of them. Specifically, we add a "None" class to represent the answers that have no type in Freebase, and the number of different entity types is 616.

Candidate query graph ranking dataset: Because CompQ and WebQ do not contain the ground truth query graphs, we use our approach to generate all candidate query graphs for every question in CompQ and WebQ. Then we pick a candidate graph as positive data, if the F1 score of its answer is the highest of all the candidate query graphs and is larger than a threshold (set to

[2] https://github.com/JunweiBao/MulCQA/tree/ComplexQuestions
[3] https://github.com/brmson/dataset-factoid-webquestions

Table 2
The parameter settings of the baselines in end-to-end experiment.

Method	Parameter	Value
Liang et al., 2013	The k-best derivations for each span	k = 50
Yih et al., 2015	The number of nodes in the convolutional layer	1000 ± 200
	The number of nodes in the semantic layer	300 ± 100
	Learning rate	0.05∼0.005
	Training epoch	800
Jain et al., 2016	The dimension of word vectors	64
	Max-norm cutoff	4
Luo et al., 2018	The dimension of word vectors	300
	The size of Bi-GRU hidden layer	300
	Margin λ	{0.1,0.2,0.5}
	The ensemble threshold K	{1,2,3,5,10,+INF}
	The batch size B	{16,32,64}
Chen et al., 2020	The dimension of word vectors	300
	The size of the hidden states	256
	The layer number of Bi-LSTM	1
	The layer number of graph Transformer	3
	Learning rate	2×10^{-4}
	Training epoch	30

0.1 in our work), and view others with an F1 score less than a threshold (set to 0.1 in our work) as negative graphs. Finally, we get 813 questions with positive query graphs in CompQ and 5004 questions in WebQ for training.

Knowledge base: In our experiments, we follow the settings of [18] to use the full Freebase dump[4] as the knowledge base for ComQ and WebQ. We host the knowledge base with Virtuoso engine.[5]

Implementation detail: For question clustering in data collection, the similarity threshold is set to 0.3. For the answer type model, we fine-tune the pre-trained parameters of BERT, and the embedding dimension of knowledge type is set to 64. We use the Adam with a learning rate of 0.00001. For the graph similarity model, we use GloVe [49] word vectors with 300-d to initialize word embedding, and the layer number of LSTM is set to 1. During the training, we also use the Adam with a learning rate of 0.001, the margin is set to 0.1, and the number of the training epochs is set to 30.

4.2. End-to-end results

Because each question can have more than one answer, we use precision, recall and F1 score as our evaluation metric.[6]

We compare our approach with the following methods: Berant et al. [12] use λ-DCS as the logical form to solve KBQA. Bast and Haussmann [50] define three templates for WebQ and try to match test questions to them. Yih et al. [16], and Bao et al. [17] leverage query graph as the logical form and construct pipelines to generate the candidate query graphs. Jain et al. [51] use memory networks to achieve a state-of-the-art result on WebQ. Abujabal et al. [15] generate templates based on statistics to get the logical form for a given question. Hu et al. [14] use state-transition over dependency parsing. Luo et al. [18] propose a semantic matching model to calculate the similarity between the candidate query graphs and the NL question to improve the performance of their system. Chen et al. [19] achieve the state-of-the-art result on CompQ by AQG. We only statistic the parameter settings of baselines given by the authors in their works. For convenience, the parameter settings of the baselines in the end-to-end experiment are listed in Table 2.

Table 3
Average F1 scores on CompQ and WebQ.

Method	CompQ	WebQ
Liang et al., 2013	–	35.7
Yih et al., 2015	36.9	52.5
Bast and Haussmann, 2015	–	49.4
Bao et al., 2016	40.9	52.4
Jain et al., 2016	–	55.6
Abujabal et al., 2017	–	51.0
Luo et al., 2018	42.8	52.7
Hu et al., 2018	–	53.6
Chen et al., 2020	43.1	53.4
Our approach	**43.8**	**56.6**

The results of our experiments are reported in Table 3. Our approach achieves state-of-the-art results on both CompQ and WebQ. We note that the method Jain [51] uses to achieve the highest F1 score before on WebQ is not the semantic parsing-based method, and thus it has poor interpretability. Yih et al. [16] and Bao et al. [17] propose a pipeline to generate the candidate query graphs and make several features to rank them. Luo et al. [18] get the similarity between the query graphs and NL question by encoding the whole query graph to compare with the NL question, thus getting better results. However, traversing all the one-hop and two-hop core inference chains creates a lot of noisy query graphs. We propose to use specific answer types to reduce the noisy query graphs and achieve better results.

Hu et al. [14] extract relations and add them to the query graph to generate the candidate set. However, the representation of the relations is diverse in NL, and there are millions of relations in the knowledge base. Thus it is hard to improve the accuracy of the system by adding relation extracted from the NL to the query graph. Instead of relation extraction, we use the answer type to improve our system, the number of which in the knowledge base is much smaller, and thus it is more realizable to predict.

4.3. Detailed analysis

4.3.1. Impact of answer type

This section shows the performance of the answer type-based query graph generating method. We evaluate the performance of our method from two aspects: the ability to filter noisy query graphs and the quality of the candidate query graphs generated by different approaches.

(1) Since many works do not have the process of query graph generation, and the candidate query graphs are mostly obtained

[4] https://developers.google.com/freebase
[5] https://virtuoso.openlinksw.com/
[6] http://www-nlp.stanford.edu/software/sempre/

Table 4
The results of filter noisy query graphs.

	CompQ				WebQ			
	N	P	R	F1	N	P	R	F1
Luo et al., 2018	204.4	40.76	54.98	42.47	147.5	52.16	61.25	52.73
Chen et al., 2020	**88.3**	**42.11**	54.59	43.06	76.1	53.77	61.81	53.43
Our approach	102.9	40.28	**62.51**	**43.80**	**62.7**	**55.49**	**67.15**	**56.62**

Table 5
The results of candidate query graph generation.

	CompQ				WebQ			
	NQ	P'	R'	$F1'$	NQ	P'	R'	$F1'$
Luo et al., 2018	698	41.51	82.35	46.19	1806	61.43	84.08	64,67
Our approach	**705**	**49.72**	**83.15**	**53.34**	**1849**	**63.91**	**85.55**	**66.78**

Table 6
Accuracy of answer type prediction by different language models.

Model	CWQAT	CompQ & WebQ
BERT, 2018	92.83	87.11
RoBERTa, 2019	90.57	84.31
DistilBERT, 2020	93.86	87.69
Our approach	**94.20**	**88.11**

by brute force search in the past works, we only select the most recent algorithm proposed by Luo et al. [18] which generates the candidate query graphs by brute force search as the baseline. In addition, we find that only Chen et al. [19] have done research on filtering noise query graphs by AQG, so we also used this work as the baseline. We use the average number of candidate query graphs, precision, recall, and F1 score as the evaluation metrics. The average number of candidate query graphs (N) means the average size of the candidate query graph set generated by the candidate query graph generation method for each NL question. Thus, the lesser the value of N is, the smaller the average size of the candidate set is, and the fewer noise query graphs generated, which means the candidate query graphs generation method has better performance. Precision (P), recall (R) and F1 score are conventional metrics for evaluating KBQA algorithms. The higher these three metrics are, the better performance of the candidate query graph generation method has. Table 4 shows the size of the candidate set generated by our approach is much smaller than that generated by Luo et al. [18]. It is smaller than that generated by AQG used by Chen et al. [19] on WebQ, but larger than it on CompQ. Note that Chen et al. [19] use graph neural network to achieve AQG, which is harder to train than the model to predict the answer type. In addition, compared with these two works, our answer type-based method improves the performance of the whole system.

(2) Since Chen et al. [19] do not give the results of candidate query graph generation on CompQ and WebQ, we only select Luo et al. [18] as the baseline to compare the quality of the candidate query graphs with our approach. We use the number of questions with at least one candidate query graph that has an F1 score greater than 0 (NQ), the average precision (P'), the average recall (R') and the average F1 score ($F1'$) of the best candidate query graphs for each NL question as the evaluation metrics. The value of NQ is greater means the candidate query graph generated by the candidate query graph generation method can answer more questions, and it also shows that the candidate query graph generation method has better performance. P', R' and $F1'$ of the best candidate query graphs are the best performance of each KBQA algorithm, the higher they are, the better performance the candidate query graph generation method has. Table 5 shows that our approach can get more questions with as least one candidate query graph greater than 0, and the average precision and F1 score of the query graph with the highest F1 score for each question is higher than Luo et al. on both CompQ and WebQ.

4.3.2. Evaluation of answer type prediction

This section shows the performance of our type classification model and the results of answer type prediction. Specifically, we select three commonly used pre-trained models as the baselines, which can be applied to text classification tasks: BERT, RoBERTa and DistilBERT. BERT is proposed by Google and achieved a significant improvement in the effect of 11 tasks in NLP at that time. It adopts the pre-trained language model training task based on the Mask mechanism and the next sentence prediction task (NSP), so the model can capture semantic information of words and sentences. RoBERTa is jointly published by Facebook and the University of Washington, and achieves better results by improving the BERT training process, such as using dynamic Mask and not using NSP training methods. DistilBERT uses a lightweight pre-trained model for the limited GPU. The main method is to use the knowledge distillation method, which makes the model parameters 40% less than BERT, 60% faster than BERT and retains BERT 97% language comprehension. Specifically, we train different models by multi-label classification training on CWQAT. During the training, we train the baseline models with different parameters and select the best results to compare. Finally, we use Adam with a learning rate of 0.0005 to train the baseline models, and the number of the training epochs is set to 30. For prediction, we only take the label with the highest probability as the prediction, and we use accuracy to measure these models, which means if the prediction is in the corresponding answer type set, it is a true positive. Otherwise, it is a false positive. We evaluate these models on CWQAT and the data collected from CompQ and WebQ, as shown in Table 6. It shows that our model achieves the highest scores in both datasets.

4.3.3. Evaluation of CWQAT dataset

In this section, we explore the quality of our QAT dataset. Specifically, we select three methods as the baselines: (1) we randomly select an answer type as the label from the corresponding answer type set for each question, (2) we use the full corresponding answer type set as the label for each question, (3) we use the answer type with the highest frequency in each cluster to label each question in the cluster. For the "random" dataset and the "top-1" dataset, we train a single-label classification model. For the full label dataset and our dataset, we train a multi-label classification model. During the training, we use Adam with a learning rate of 0.001 to train the baseline models, and the number of the training epochs is set to 30 to get the best results. For prediction and evaluation, we adopt the strategy of the previous section. As shown in Table 7, the model trained by our dataset achieves the best performance in both evaluation datasets.

Table 7
Accuracy of the model trained by QAT dataset collected by different collection strategies.

Label choose method	CWQAT	CompQ & WebQ
Random	92.75	85.26
Full labels	93.15	86.00
Top 1	91.90	87.16
Our approach	**94.20**	**88.11**

Table 8
The results of ranking models trained on different features.

Feature	CompQ F1	WebQ F1
raw	40.2	52.7
raw+RP	41.6	53.5
raw+TR	41.3	53.8
raw+RR	42.9	55.4
raw+RP+TR+RR	**43.8**	**56.6**

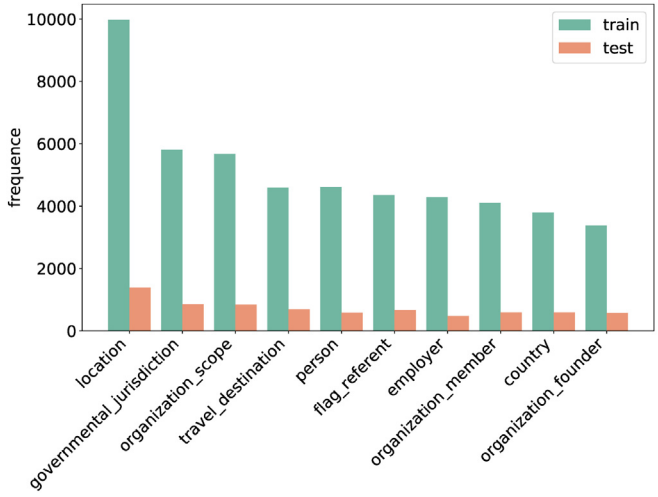


Fig. 6. The 10 most frequent entity types in CWQAT.

4.3.4. Impact of relation features

This section shows the impact of the three features extracted from type relations on our ranking performance. Specifically, we select four methods as the baseline for comparison with the three features: raw features, raw features with RP, raw features with TR and raw features with RR, which use the raw features, Relation Proportion, Type Relevance and Relation Rate to rank the candidate query graphs. During the training, we use Adam with a learning rate of 0.001 to train the baseline models, the margin is set to 0.1, and the number of the training epochs is set to 30 to get the best results. The results are shown in Table 8. Raw features with the three features achieve the best performance in both CompQ and WebQ. This demonstrates the effectiveness of the three features.

4.4. Error analysis

We randomly select 100 questions with an F1 score of 0 from CompQ and WebQ and analyzed the reason. The major causes of errors are shown as follows:

Entity linking error (23%): This type of error occurs when the NL question contains highly ambiguous mentions. For example, the question "who plays Edward on Deception?" contains two entities, "Edward" and "Deception". However, there are many entities with the same name as "Edward" and "Deception" in Freebase.

Answer type error (10%): This type of error occurs when our model fails to understand the semantics of the questions and thus gives the wrong answer types. For example, the answer type of the question "Who does Mike marry at the end of season 5 Desperate Housewives?" is "film character", but the answer type that our model outputs is "person".

Complex question (17%): This type of error occurs when the questions involve superlative qualifiers like "richest", "oldest", etc., or contain clauses, which are implicit constraints. For example, the question "Who was the president when the first man landed on the Moon?" contains a time clause, "when the first man landed on the Moon", which is an implicit time constraint, and thus it is hard to answer.

Incomplete answers (12%): This type of error occurs when the ground truth answers to the question are incomplete. For example, the right answer to the question 'Where did Tim Tebow grow up?" is "Manila", but also "Philippines". However, the ground truth of the question is only "Manila".

Semantic ambiguity (38%): This type of error occurs when the question is ambiguous. For example, our approach generates an answer as "profession" to the question "What to do in Paris?", while the ground truth is as "tourist attraction".

These errors lead to 705 questions in CompQ and 1849 questions in WebQ with at least one candidate query graph that has an F1 score greater than 0, and thus they limit the performance of our method to an upper bound.

5. Conclusion and future work

In this paper, we present an answer type-based method for KBQA and collect a QAT dataset for Freebase. Our approach can predict the answer type of the given NL question to filter "noisy" candidate query graphs. To the best of our knowledge, this is the first work to start from the answer to generate the candidate query graphs. Our approach achieves the state-of-the-art on both CompQ and WebQ.

For the future work, we set the following tasks: (1) find a more meaningful way to choose the most related answer type from the answer type set for each question, (2) find a method to generate multi-hop query graphs instead of one-hop or tow query graphs, (3) it can be seen from the error analysis that the error of complex question and semantic ambiguity account for the majority, and thus it is helpful to break the complex question into many simple questions and combine them to answer it.

Table 9
The comparison between CWQAT and existing QAT datasets, Number(AT) is the number of different answer types in the dataset, Average(AT) is the average number of answer types corresponding to each NL question.

Data set	Size	Target KG	Number(AT)	Average(AT)
SMART (DBpedia)[45]	21952	DBpedia	310	2.86
SMART (Wikidata)[45]	22822	Wikidata	3129	1.62
EAT [52]	72656	DBpedia	27	1
CWQAT (Ours)	31158	Freebase	616	2.07

CRediT authorship contribution statement

Haoyuan Chen: Data curation, Investigation, Writing – original draft, Conceptualization, Methodology, Formal analysis, Validation, Software. **Fei Ye:** Formal analysis, Conceptualization, Methodology, Writing – review & editing, Supervision. **Yuankai Fan:** Methodology, Writing – review & editing, Supervision. **Zhenying He:** Conceptualization, Methodology, Writing – review & editing, Supervision. **Yinan Jing:** Writing – review & editing. **Kai Zhang:** Writing – review & editing. **X. Sean Wang:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was mainly supported by National Natural Science Foundation of China under Grant Nos. 61732004, 62072113.

Appendix. Cwqat details

CWQAT is present in our repository. It is stored in JSON format, and each question corresponds to one or more answer types. Table 9 compares CWQAT with existing QAT datasets. It shows that there is only CWQAT built on Freebase. The number of different entity types is 616, and the average number of answer types corresponding to each NL question is 2.07, which shows that the dataset covers many entity types, and each question corresponds to the complete answer types as much as possible.

Fig. 6 shows the 10 most frequent entity types in CWQAT. It can be seen that the higher-level entity types are more frequent, such as "location" and "person". For the staged query graph generation based on answer type, the more specific answer types, the more noisy query graphs are filtered, but higher-level entity types can also filter out part of the noisy query graphs. And each question may correspond to multiple labels. For the questions with high-level labels, they will also contain some specific types, such as the answer type of the question "Who play Denethor II in the Lord of the Rings?" is "person", and the question may also correspond to the "film actor" answer type. So CWQAT will also allow the model to learn the relation between questions and the specific types.

References

[1] Peng Lin, Qi Song, Yinghui Wu, Fact checking in knowledge graphs with ontological subgraph patterns, Data Sci. Eng. 3 (4) (2018) 341–358.
[2] Yuzhuo Wang, Hongzhi Wang, Junwei He, Wenbo Lu, Shuolin Gao, TAGAT: Type-aware graph attention networks for reasoning over knowledge graphs, Knowl.-Based Syst. 233 (2021) 107500.
[3] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, Jamie Taylor, Freebase: a collaboratively created graph database for structuring human knowledge, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, 2008, pp. 1247–1250.
[4] Fabian M. Suchanek, Gjergji Kasneci, Gerhard Weikum, Yago: a core of semantic knowledge, in: Proceedings of the 16th International Conference on World Wide Web, 2007, pp. 697–706.
[5] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, Zachary Ives, Dbpedia: A nucleus for a web of open data, in: The Semantic Web, Springer, 2007, pp. 722–735.
[6] Leslie F. Sikos, Dean Philp, Provenance-aware knowledge representation: A survey of data models and contextualized knowledge graphs, Data Sci. Eng. 5 (3) (2020) 293–316.
[7] Baozhu Liu, Xin Wang, Pengkai Liu, Sizhuo Li, Qiang Fu, Yunpeng Chai, UniKG: A unified interoperable knowledge graph database system, in: 2021 IEEE 37th International Conference on Data Engineering, ICDE, IEEE, 2021, pp. 2681–2684.
[8] Zhixin Qi, Hongzhi Wang, Ziming Shen, Donghua Yang, PreKar: A learned performance predictor for knowledge graph stores, World Wide Web (2022) 1–21.
[9] Eric Prud'hommeaux, SPARQL query language for RDF, W3C recommendation, 2008, http://www.w3.org/tr/rdf-sparql-query/.
[10] Pengkai Liu, Xin Wang, Qiang Fu, Yajun Yang, Yuan-Fang Li, Qingpeng Zhang, KGVQL: A knowledge graph visual query language with bidirectional transformations, Knowl.-Based Syst. (2022) 108870.
[11] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yangqiu Song, Seung-won Hwang, Wei Wang, KBQA: learning question answering over QA corpora and knowledge bases, 2019, arXiv preprint arXiv:1903.02419.
[12] Jonathan Berant, Andrew Chou, Roy Frostig, Percy Liang, Semantic parsing on freebase from question-answer pairs, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 1533–1544.
[13] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, Dongyan Zhao, Natural language question answering over RDF: a graph data driven approach, in: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, 2014, pp. 313–324.
[14] Sen Hu, Lei Zou, Xinbo Zhang, A state-transition framework to answer complex questions over knowledge base, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 2098–2108.
[15] Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, Gerhard Weikum, Automated template generation for question answering over knowledge graphs, in: Proceedings of the 26th International Conference on World Wide Web, 2017, pp. 1191–1200.
[16] Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, Jianfeng Gao, Semantic parsing via staged query graph generation: Question answering with knowledge base, 2015.
[17] Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, Tiejun Zhao, Constraint-based question answering with knowledge graph, in: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, 2016, pp. 2503–2514.
[18] Kangqi Luo, Fengli Lin, Xusheng Luo, Kenny Zhu, Knowledge base question answering via encoding of complex query graphs, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 2185–2194.
[19] Yongrui Chen, Huiying Li, Yuncheng Hua, Guilin Qi, Formal query building with query structure prediction for complex question answering over knowledge base, 2021, arXiv preprint arXiv:2109.03614.
[20] A. Talmor, J. Berant, The web as a knowledge-base for answering complex questions, in: North American Association for Computational Linguistics, NAACL, 2018.
[21] Chaokun Wang, Binbin Wang, Bingyang Huang, Shaoxu Song, Zai Li, FastSGG: efficient social graph generation using a degree distribution generation model, in: 2021 IEEE 37th International Conference on Data Engineering, ICDE, IEEE, 2021, pp. 564–575.
[22] Zhao Li, Xin Liu, Xin Wang, Pengkai Liu, Yuxin Shen, TransO: a knowledge-driven representation learning method with ontology information constraints, World Wide Web (2022) 1–23.
[23] Gaoyang Guo, Chaokun Wang, Bencheng Yan, Yunkai Lou, Hao Feng, Junchao Zhu, Jun Chen, Fei He, Philip Yu, Learning adaptive node embeddings across graphs, IEEE Trans. Knowl. Data Eng. (2022).
[24] Yaming Yang, Ziyu Guan, Jianxin Li, Wei Zhao, Jiangtao Cui, Quan Wang, Interpretable and efficient heterogeneous graph convolutional network, IEEE Trans. Knowl. Data Eng. (2021).
[25] Guotong Xue, Ming Zhong, Jianxin Li, Jia Chen, Chengshuai Zhai, Ruochen Kong, Dynamic network embedding survey, Neurocomputing 472 (2022) 212–223.
[26] Xiangyu Song, Jianxin Li, Yifu Tang, Taige Zhao, Yunliang Chen, Ziyu Guan, Jkt: A joint graph convolutional network based deep knowledge tracing, Inform. Sci. 580 (2021) 510–523.
[27] Xiangyu Song, Jianxin Li, Qi Lei, Wei Zhao, Yunliang Chen, Ajmal Mian, Bi-CLKT: Bi-graph contrastive learning based knowledge tracing, 2022, arXiv preprint arXiv:2201.09020.
[28] Xuchen Yao, Benjamin Van Durme, Information extraction over structured data: Question answering with freebase, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2014, pp. 956–966.
[29] Antoine Bordes, Sumit Chopra, Jason Weston, Question answering with subgraph embeddings, 2014, arXiv preprint arXiv:1406.3676.
[30] Antoine Bordes, Jason Weston, Nicolas Usunier, Open question answering with weakly supervised embedding models, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2014, pp. 165–180.
[31] Li Dong, Furu Wei, Ming Zhou, Ke Xu, Question answering over freebase with multi-column convolutional neural networks, 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, 2015, pp. 260–269.

[32] Yanchao Hao, Yuanzhe Zhang, Kang Liu, Shizhu He, Zhanyi Liu, Hua Wu, Jun Zhao, An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, Volume 1: Long Papers, 2017, pp. 221–231.
[33] Sepp Hochreiter, Jürgen Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[34] Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, Dongyan Zhao, Question answering on freebase via relation extraction and textual evidence, 2016, arXiv preprint arXiv:1603.00957.
[35] Apoorv Saxena, Aditay Tripathi, Partha Talukdar, Improving multi-hop question answering over knowledge graphs using knowledge base embeddings, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 4498–4507.
[36] Sirui Li, Kok Wai Wong, Chun Che Fung, Dengya Zhu, Improving question answering over knowledge graphs using graph summarization, in: International Conference on Neural Information Processing, Springer, 2021, pp. 489–500.
[37] Qianglong Chen, Feng Ji, Haiqing Chen, Yin Zhang, Improving commonsense question answering by graph-based iterative retrieval over multiple knowledge sources, 2020, arXiv preprint arXiv:2011.02705.
[38] Xin Bi, Haojie Nie, Xiyu Zhang, Xiangguo Zhao, Ye Yuan, Guoren Wang, Unrestricted multi-hop reasoning network for interpretable question answering over knowledge graph, Knowl.-Based Syst. 243 (2022) 108515.
[39] Kechen Qin, Cheng Li, Virgil Pavlu, Javed Aslam, Improving query graph generation for complex question answering over knowledge base, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 2021, pp. 4201–4207.
[40] Mahdi Bakhshi, Mohammadali Nematbakhsh, Mehran Mohsenzadeh, Amir Masoud Rahmani, SParseQA: Sequential word reordering and parsing for answering complex natural language questions over knowledge graphs, Knowl.-Based Syst. 235 (2022) 107626.
[41] Yunshi Lan, Jing Jiang, Query Graph Generation for Answering Multi-Hop Complex Questions from Knowledge Bases, Association for Computational Linguistics, 2020.
[42] Haobo Xiong, Shuting Wang, Mingrong Tang, Liping Wang, Xuemin Lin, Knowledge graph question answering with semantic oriented fusion model, Knowl.-Based Syst. 221 (2021) 106954.
[43] Jianbin Li, Ketong Qu, Jingchen Yan, Liting Zhou, Long Cheng, TEBC-net: An effective relation extraction approach for simple question answering over knowledge graphs, in: International Conference on Knowledge Science, Engineering and Management, Springer, 2021, pp. 154–165.
[44] Fen Zhao, Yinguo Li, Jie Hou, Ling Bai, Improving question answering over incomplete knowledge graphs with relation prediction, Neural Comput. Appl. (2022) 1–18.
[45] Nandana Mihindukulasooriya, Mohnish Dubey, Alfio Gliozzo, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Ricardo Usbeck, SeMantic answer type prediction task (SMART) at ISWC 2020 semantic web challenge, 2020, CoRR/arXiv, arXiv:2012.00555.
[46] Andrew D Walker, Panos Alexopoulos, Andrew Starkey, Jeff Z Pan, José Manuel Gómez-Pérez, Advaith Siddharthan, Answer type identification for question answering, in: Joint International Semantic Technology Conference, Springer, 2015, pp. 235–251.
[47] Yi Yang, Ming-Wei Chang, S-mart: Novel tree-based structured learning algorithms applied to tweet entity linking, 2016, arXiv preprint arXiv:1609.08075.
[48] Mike Mintz, Steven Bills, Rion Snow, Dan Jurafsky, Distant supervision for relation extraction without labeled data, in: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2009, pp. 1003–1011.
[49] Jeffrey Pennington, Richard Socher, Christopher D. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, 2014, pp. 1532–1543.
[50] Hannah Bast, Elmar Haussmann, More accurate question answering on freebase, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, 2015, pp. 1431–1440.
[51] Sarthak Jain, Question answering over knowledge base using factual memory networks, in: Proceedings of the NAACL Student Research Workshop, 2016, pp. 109–115.
[52] Aleksandr Perevalov, Andreas Both, Improving answer type classification quality through combined question answering datasets, in: International Conference on Knowledge Science, Engineering and Management, Springer, 2021, pp. 191–204.