A State-transition Framework to Answer Complex Questions over Knowledge Base

Sen Hu[1], **Lei Zou**[1,2], **Xinbo Zhang**[1]
[1]Peking University, China; [2]Beijing Institute of Big Data Research, China;
{husen,zoulei,zhangxinbo}@pku.edu.cn

Abstract

Although natural language question answering over knowledge graphs have been studied in the literature, existing methods have some limitations in answering complex questions. To address that, in this paper, we propose a *State Transition*-based approach to translate a complex natural language question N to a *semantic query graph* (SQG) Q^S, which is used to match the underlying knowledge graph to find the answers to question N. In order to generate Q^S, we propose four primitive operations (expand, fold, connect and merge) and a learning-based state transition approach. Extensive experiments on several benchmarks (such as QALD, WebQuestions and ComplexQuestions) with two knowledge bases (DBpedia and Freebase) confirm the superiority of our approach compared with state-of-the-arts.

1 Introduction

Complex Question Answering, in which the question corresponds to multiple triples in knowledge base, has attracted researchers' attentions recently (Bao et al., 2014; Xu et al., 2016; Berant and Liang, 2014; Bast and Haussmann, 2015; Yih et al., 2015). However, most of existing solutions employ the predefined patterns or templates in understanding complex questions. For example, (Bao et al., 2014; Xu et al., 2016) use dependency tree patterns to decompose a complex question to several simple questions. (Berant and Liang, 2014; Bast and Haussmann, 2015) rely on templates to translate natural language sentences to predefined logical forms. Obviously, it is impossible to cover all real-world cases using those handcrafted templates. The recent work STAGG (Yih et al., 2015) proposes a state transition strategy to tackle complex questions. However, the generated query graph's structure in STAGG is limited, as shown in Figure 1. Generally, the structure is formed by linking the topic entity e and the answer node t through a core relation. It only allows entity constraints (such as c) and one variable (the answer node t), the expressivity is quite limited. It cannot cover all types of complex questions such as the question in Figure 2, whose query graph contains variable constraint (v_4) and multi-edges (between v_1 and v_4). Therefore, in this paper, we propose a more flexible strategy to answer complex questions without handcrafted templates or restricting the query graph's structure.

1.1 Properties and Challenges of Complex Questions

We first analyze the inherent challenges for complex question answering, which has not been well studied in existing literatures.

Multi or Implicit relations. A complex question has multiple relations and some relations may be multi-hop or implicit. Consider the question N_1 = "*Which Russian astronauts died in the same place they were born in?*". Obviously there are two explicit relations represented by the two phrases "died in" and "born in", but many traditional methods only extract a single relation between topic entity and the answer node. On the other hand, there is an implicit relation between "Russian" and "astronauts" (i.e., ⟨Nationality⟩), which lacks a corresponding relation phrase.

Multi or No entities. Different from simple (one-hop) questions that only contain a single *topic entity* in the question sentence N, complex questions may have multiple entities. For example, "Which Russian astronauts were died in Moscow and born in Soviet Union". There are three entities "Russian", "Moscow" and "Soviet Union". In some cases, the question sentence does not include any entity, such as "Who died in the same place they were born in". Therefore, existing methods that

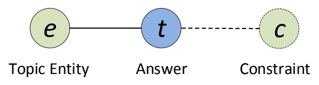

Figure 1: The query graph structure of STAGG. Node *e* is the "topic entity", path *e*-*t* is the "core inferential chain" and *c* represents "augmenting constraints". Notice we omit the CVT node as it is a virtual node only existed in Freebase.

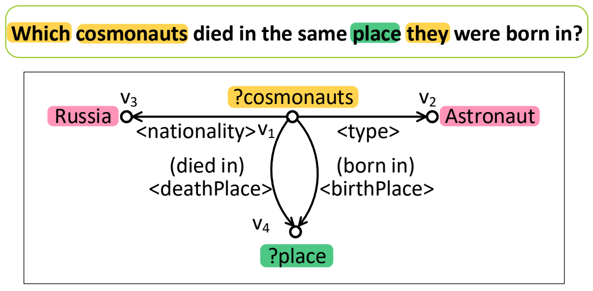

Figure 2: The example of complex questions with semantic query graph

only care about a single topic entity cannot work for such types of complex questions.

Variables and Co-reference. Existing solutions assume that there is a single variable in the question N, i.e., the wh-word, which is focus of the question (representing the answer node). However, complex questions may have other variables. Consider the question in Figure 2, there are four variables "Which", "cosmonauts", "place" and "they" recognized. While the three in yellow grid ("Which", "cosmonaut", "they") refer to the same thing, which is the co-reference phenomenon.

Composition. In the simple questions, the query graph's structure is determinate (one-hop edge). For complex questions, even when all entities, variables and relations are recognized, how to assemble them to logical forms or executable queries (query graphs) is still a challenge. As mentioned earlier, existing solutions that depend on predefined templates (such as (Xu et al., 2016; Bast and Haussmann, 2015)) or query graph structure patterns (such as (Yih et al., 2015)) may not be suitable due to the diversity of complex questions.

1.2 Our Solution

Considering the above challenges, we propose a *state transition* based framework to translate users' question N into a semantic query graph (SQG) Q^S, further the answers of N can be found by matching Q^S over the knowledge graph G. Specifically, we first recognize nodes (such as entities and variables) from N as the initial state (Section 2.1). Then, to enable and speed up the state transition process, we propose four primitive operations with corresponding conditions (Section 2.2). The entity/relation candidates are extracted using existing entity linking algorithms and our proposed MCCNN model (Section 2.3). To guide the state transition, we learn a reward function using SVM ranking algorithm (Section 3) to greedily select a subsequent state.

Compared with existing solutions, our framework has stronger expressivity to solve more types of complex questions (considering the challenges introduced in Section 1.1) and does not rely on handcrafted templates.

2 Semantic Query Graph Generation

This section outlines the whole framework of our system. Given a natural language question N, we aim to translate N into a *semantic query graph* (SQG) (see Definition 1).

Definition 1 ***(Semantic Query Graph).*** *A semantic query graph (denoted as Q^S) is a connected graph, in which each vertex v_i corresponds a phrase in the question sentence N and associated with a list of entity/type/literal candidates; and each edge $\overline{v_i v_j}$ is associated with a list of relation candidates, $1 \leq i, j \leq |V(Q^S)|$.*

Given a question sentence N_1 = "Which cosmonauts died in the same place they were born in?", the corresponding semantic query graph Q^S_1 is given in Figure 2. In Q^S_1, the subgraph $\{v_1$, v_2 and $v_3\}$ corresponds "cosmonauts", v_4 corresponds "place". The relation phrases "born in" and "died in" denote two relations between v_1 and v_4. Each edge in Q^S has a list of relation(predicate) candidates with confidence probabilities. In this paper, we only draw one relation candidate of each edge for simplicity. Notice that the edge $\overline{v_1 v_3}$ and $\overline{v_1 v_2}$ have no relation phrases, however, they still have corresponding relation candidates $\langle$nationality$\rangle$ and $\langle$type$\rangle$.

After obtaining semantic query graph Q^S, we find matches of Q^S over the underlying knowledge graph G to find answers to the original question N. This stage is identical to gAnswer (Zou et al., 2014) and NFF (Hu et al., 2018). Thus, in this paper, we concentrate ourselves on discussing how to generate semantic query graph Q^S, which is different from NFF that relies on the paraphrasing dictionaries and heuristic rules to build Q^S.

2.1 Node Recognition

The first step is to detect the "nodes" (for building SQG Q^S) in users' question sentences. According to Definition 1, each node in the semantic query graph is a phrase corresponding to a subject or object in the SPARQL query. Generally, constant nodes (e.g., *Titanic*) can be divided into three categories according to their roles (entity, type, literal) in knowledge graph. Besides, a variable node (e.g., *?place*) has the potential to map any vertices in knowledge graph.

Generally, the node recognition task is regarded as a sequence labeling problem and we adopt a BLSTM-CRF model (Huang et al., 2015) to realize it. However, this model doesn't work very well on recognizing entity/type nodes as the entity phrases may be too long or too complex. Therefore, we first detect entity and type nodes by utilize existing entity linking algorithms of specific knowledge bases (see section 2.3), and then detect variable/literal nodes by the BLSTM-CRF model. A phrase w_{ij}, which contains the words from i to j in the question, is recognized as entity/type node if we can find entity/type mappings of w_{ij} through the entity linking algorithm. We check all possible phrases and select the longer one if two candidate nodes share any words. The detected entity/type nodes will be replaced by special tokens $\langle E \rangle$ or $\langle T \rangle$ before calling the BLSTM-CRF model.

In practice, we find that some nodes have hidden information, which can be expanded to a subgraph (i.e, one or multiple triples). For example, the hidden information of "?cosmonauts" in question N_1 in Figure 2 can extends two triples: ⟨?cosmonauts rdf:type dbo:Astronaut⟩ and ⟨?cosmonauts dbo:nationality dbr:Russia⟩. Such information can help us to reduce the matching space and make the answers more accurate.

We mine those nodes with hidden information from several QA benchmarks[1] and build a translation dictionary D^T in offline. Given a question N with the gold query graph Q^G, we try to match Q^G to the SQG Q^S generated by our approach. Each unmatched connected subgraph in Q^G is regarded as hidden information of the nearest matched node v. For the benchmarks only have gold question-answers pairs, we utilize the method in (Zhang and Zou, 2017) to mine the gold query graph first.

[1]QALD, WebQuestions and ComplexQuestions, only the training sets are used to build the translation dictionary D^T.

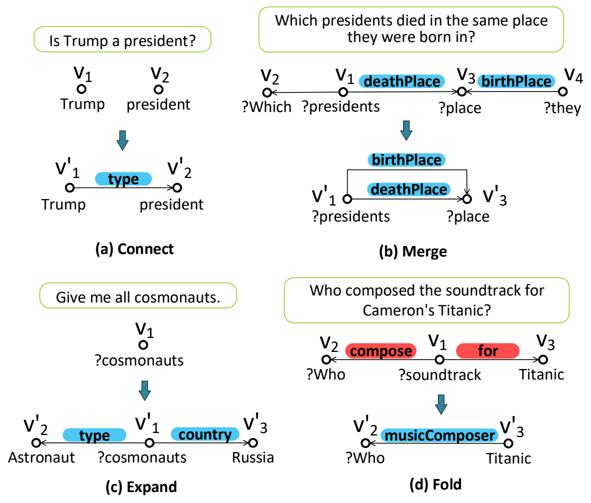

Figure 3: Samples of Operations

2.2 SQG's Structure Construction

As mentioned earlier, our SQG's structure is generated in a state-transition paradigm. Initially, the state only contains several *isolated* nodes that are recognized in the first step. To enable the state transition, we propose four primitive operations as follows.

1. **Connect**. Given two operate nodes u_1 and u_2, we introduce an edge $\overline{u_1 u_2}$ between them by the connect operation. Note that the relation mention and candidate relations (predicates) of the edge $\overline{u_1 u_2}$ can be found through relation extraction model (see section 2.3).

2. **Merge**. Given a SQG $Q^S = \{V, E\}$ and two operate nodes u,v, this operation is to merge the node u into v. The new SQG $Q'^S = \{V \setminus \{u\}, (E \setminus E^d) \cup E^a\}$, $E^d = \{\overline{uw} \in E\}$ and $E^a = \{\overline{vw} | \overline{uw} \in E \wedge w \neq v\}$. Notice that v also inherits the properties of u. The merge operation is designed to support co-reference resolution. The Figure 3(b) gives an example. For the question "Which presidents died in the same place they were born in?", the node v_2("which") is redundant and can be merged into v_1("presidents"). The node v_4("they") and v_1("presidents") are co-reference, we can merge v_4 to v_1. The edge $\overline{v_3 v_4}$(birthPlace) is replaced by edge $\overline{v'_3 v'_1}$(birthPlace).

3. **Expand**. Given a SQG $Q^S = \{V, E\}$ and the operate node $u \in V$, this operation is to expand u to a subgraph $Q^S_u = \{V_u, E_u\}$. The new SQG $Q'^S = \{V \cup V_u, E \cup E_u\}$. The expand operation is designed to those

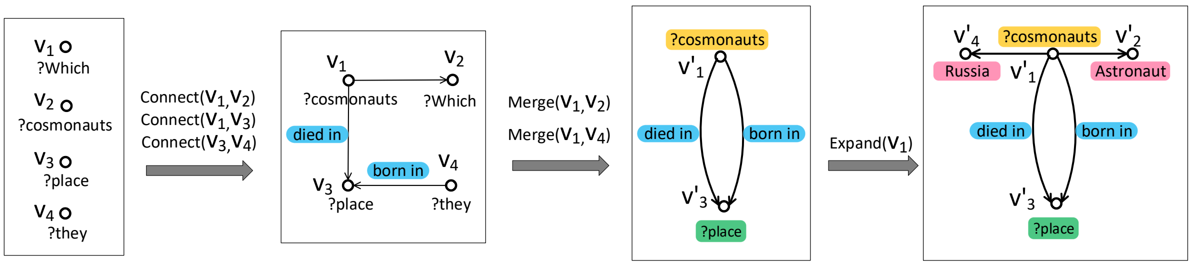

Figure 4: Semantic Query Graph Generation

nodes that have hidden information. Let us see Figure 3(c). The word "cosmonauts"(v_1) means "Russia astronauts", but it cannot be mapped to a certain entity or type in the knowledge base directly. To match "cosmonauts", we can expand it to an equivalent subgraph $\{v_1^{'},v_2^{'},v_3^{'}\}$ (according to the transition dictionary D^T) that can match the underlying knowledge base.

4. **Fold**. Given a SQG $Q^S = \{V, E\}$ and an operate node $u \in V$, this operation is to delete u and merge the associated relations. Formally, the new SQG $Q'^S = \{V \setminus \{u\}, (E \setminus E^d) \cup E^a\}$, in which $E^d = \{\overline{uv_i} \in E\}$ and $E^a = \{\overline{v_i v_j} | \overline{uv_i} \in E \wedge \overline{uv_j} \in E \wedge i \neq j\}$. The fold operation is designed to eliminate those nodes which are useless or mis-recognized. Figure 3(d) gives an example. For the question "Who composed the soundtrack for Cameron's Titanic?", SQG $\{v_1, v_2, v_3\}$ cannot find matches in knowledge base because there are no correct relations of $\overline{v_1 v_2}$ and $\overline{v_1 v_3}$. By applying a fold operation, these two edges with the redundant node v_1 are removed, while the new edge $\overline{v'_1 v'_2}$ can be matched to the correct relation ⟨musicComposer⟩ (according to the relation extraction model).

We focus on generating the correct SQG with the above operations. Obviously, it is impossible to enumerate all possible transitions in each step due to exponential search space. Therefore, we propose a greedy search algorithm[2] inspired by (Yih et al., 2015). The intention is that an better intermediate state should have more opportunities to be visited and produce subsequent states. As each state can be seen as a partial SQG, we propose a reward function using a log-linear model (see Section 3) to estimate the likelihood of each state. A priority queue is utilized to maintain the order of unvisited states. Each turn we choose the state with maximum correctness likelihood and try all possible transitions by each operation. Specifically, the connect and merge operations try every pair of nodes in current state as the two operate nodes, while the expand and fold operations try each node as the operate node. The newly generated states are estimated by the learning model and pushed into the priority queue. Note that reduplicated states will not be added. The algorithm ends when a complete SQG Q^S is found and Q^S can not generates any better subsequent states.

To reduce the search space of the state-transition process, we also propose several constraints for each operation. Only when the conditions are satisfied, the corresponding operation can be executed. Experiments show that those conditions not only speeds up the QA process but also improves the precision (see Section 4.4). The conditions are listed as follows.

Condition 1 (**Condition for Connect**) *Two nodes v_1 and v_2 can be connected if and only if there exists* no *other node v^* that occurs in the shortest path between v_1 and v_2 of the dependency parse tree*[3] *of question sentence N.*

Condition 2 (**Condition for Merge**) *For the merge operation, there should be at least one variable v in the two operate nodes. And v should be a wh-word or pronoun, which may co-reference*[3] *with an other node.*

Condition 3 (**Condition for Expand**) *For the expand operation, the operate node u should be a variable, and we need to be able to find u and its hidden information in the transition dictionary D^T.*

Condition 4 (**Condition for Fold**) *For the fold operation, we require at least one of the connected edges with the operate node have no relation candidates or the confidence probabilities of the corresponding relations are less than a threshold τ.*

[2]The formal algorithm can be found in Appendix A.

[3]We utilize the Stanford CoreNLP dependency parser and coreference annotator (Manning et al., 2014)

The following example illustrates the process of SQG generation. Note that it can have other transition sequences to get the final state.

Example 1 *Consider Figure 4. Given a question N_1 = "Which cosmonauts died in the same place they were born in?", we first extract four nodes v_1, v_2, v_3, v_4, which are all variable nodes. According the condition 1, we connect (v_1,v_2), (v_1,v_3), and (v_3,v_4). The simple path between v_1 and v_3 in the dependency parse tree can be regarded as the relation phrase of $\overline{v_1 v_3}$, which is "died in". Similarly, the relation phrase of $\overline{v_3 v_4}$ is "born in". As v_1 and v_2 are neighbors in the dependency parse tree, the corresponding relation phrase is empty. Then we merge (v_1,v_2) as they are recognized co-reference. Similarly v_1 and v_4 are merged. Notice that the associated edge $\overline{v_3 v_4}$ have been reserved, then there are two edges between v_1' and v_3' in the new SQG. In the end, we expand v_1' to the subgraph $\{v_1', v_4', v_2'\}$ and get the final SQG.*

2.3 Finding entity/relation candidates and matches of SQG

During SQG construction, there are two subtasks: entity extraction and relation extraction. We briefly discuss the two steps as follows.

Given a node phrase in Q^S, we find the candidate entities/types with confidence probabilities by using existing entity linking approaches. Specifically, we use S-MART (Yang and Chang, 2015) for Freebase dataset and DBpeida Lookup[4] for DBpedia dataset.

Given two connected nodes v_i and v_j in Q^S, we need to find the relation candidates between them. Inspired by the recent success of neural relation extraction models in KBQA (Yih et al., 2015; Dong et al., 2015; Xu et al., 2016), we propose a Multi-Channel Convolutional Neural Network (MCCNN) to extract all relations in the question N exploiting both syntactic and sentential information. We use the simple path between v_i and v_j in the dependency tree $Y(N)$ as input to the first channel. The input of second channel is the whole question sentence excluded all nodes, which represents the context of the relation we want to predict. If v_i and v_j are neighbors in $Y(N)$, we need to extract an implicit relation. To tackle this task, we use v_i, v_j and their types as input to the third channel. For v_i = "Chinese" and v_j = "actor", the input is "Chinese-Country-actor-Actor".

[4]http://wiki.dbpedia.org/projects/dbpedia-lookup

Different from (Xu et al., 2016) which can only extract one explicit relation, our model can not only extract multiple relations from N by taking different node pairs, but also extract the implicit relation considering the embedded information in the two nodes v_i and v_j.

After generating the SQG Q^S, we employ the subgraph matching algorithm (Hu et al., 2018) to find matches of Q^S over the underlying RDF knowledge graph G. Once the matches are found, it is straightforward to derive the answers.

3 Learning the reward function

Given an intermediate state s during SQG generation process, we may transit to multiple subsequent states s' by applying different operations. Thus, we greedily select a subsequent state s' with the maximum $\gamma(s')$, where $\gamma()$ is a reward function taking the features and outputting the reward of corresponding state. In this work, the reward function $\gamma()$ is a linear model trained with the SVM ranking loss. Below we describe the features, learning process and how to generate the training data.

3.1 Features

Note that for any intermediate state s, we also regard it as a SQG $Q^S(s)$ even though it is a disconnected graph. For a given SQG Q^S, we extract the following features which are passed as an input vector to the SVM ranker.

Nodes. For each entity/type node v, we utilize entity linking systems to find the candidate entities or types and corresponding confidence probabilities. We use the largest confidence probability as the score of v, and use the average score of all entity nodes as the feature. In addition, the number of constant nodes and variable nodes in current Q^S are considered as features.

Edges. For each edge $\overline{v_i v_j}$, the relation extraction model returns a list of candidate predicates with the corresponding confidence probabilities. We use the largest confidence probability as the score of $\overline{v_i v_j}$, and use the average score of all edges as the feature. We also consider the total number of edges in Q^S, the number of edges which have relation phrase and which have no relation phrase.

Matches. We also consider the matching between Q^S and the knowledge graph G. A new state s would get a low score if its corresponding

SQG Q^S could not find any matches in G. We do not discard s directly from the search queue as we allow the fold operation of it. To speed up the matching process, we build the index of all entities and relations in offline. For a given SQG Q^S, we use the offline index to check whether it is valid.

Given a valid state s, if all of its possible subsequent states s' have smaller reward function value than that of s, we will terminate the state transition process and return s as the final state. The corresponding SQG $Q^S(s)$ is used to match the knowledge base to retrieve answers.

3.2 Learning

The task of reward function can be viewed as a ranking problem, in which the appropriate SQGs should be ranked in the front. In this work, we rank them using a SVM rank classifier (Joachims, 2006).

Given a SQG $Q^S = \{V, E\}$, the ranking score that we use to train our reward function $\gamma()$ is calculated by the following function.

$$R(Q^S) = \frac{|P(V)|}{|V'|} + \frac{|P(E)|}{|E'|} + \max(F(A_i, A')) \tag{1}$$

V' and E' are the gold node set and relation set extracted from the gold SPARQL query. $P(V)$ contains the nodes existing in both V and V'. $P(E)$ contains the relations existing in both E and E'. $F(A_i, A')$ is the F-1 measure where A' is the gold answer set and A_i is one of the subgraph matches between Q^S and knowledge graph G obtained by the executing method in (Hu et al., 2018). Notice that for the benchmark without gold SPARQL queries, $R(Q^S) = max(F(A_i, A'))$.

3.3 Generating Training Data

To generate the training data of the reward function, given a question $N = \{w_1, w_2, ..., w_n\}$, we first check all possible phrases w_{ij} by existing entity linking system and get the candidate entity list $C_{v_i} = \{e_1, e_2, ..., e_m\}$ for each entity node v_i. If two entity node v_i and v_j have conflict, i.e, v_i and v_j share one or more words, we reserve the longer one. We replace each entity node v_i with a special token to denote the type of entity e_j, where $e_j \in C_{v_i}$ and has the largest confidence possibility. Then the whole node set V can be predicted by the well-trained node recognition model. $Q^S = \{V, \phi\}$ is the initial state. Further we do state transition by applying the predefined operations using a BFS algorithm. Meanwhile, only the states satisfied with corresponding conditions are considered if enabling the condition mechanism. Once we get a new Q^S we call the well-trained relation extraction model to get the relation candidate list of each new edge $\overline{v_i v_j} \in E$, while the features are collected according Section 3.1 and the ranking score is calculated using Equation 1.

4 Evaluation

We evaluate our system on several benchmarks with two knowledge base DBpedia and Freebase. For DBpedia, we use the QALD-6 benchmark. We compare our method STF (State Transition Framework) with all systems in QALD-6 competition as well as Aqqu (Bast and Haussmann, 2015), gAnswer (Zou et al., 2014) and NFF (Hu et al., 2018). For Freebase, we use WebQuestions (Berant et al., 2013) and ComplexQuestions (Abujabal et al., 2017) as benchmarks and compare our method with corresponding state-of-art systems.

4.1 Setup

4.1.1 Knowledge Bases

DBpedia RDF repository is a knowledge base derived from Wikipedia (Bizer et al., 2009). We use the version of DBpedia 2014 and the statistics are given in Table 1.

Freebase is a collaboratively edited knowledge base. We use the version of Freebase 2013, which is same with (Berant and Liang, 2014). The statistics are given in Table 1.

4.1.2 Benchmarks

QALD is a series of open-domain question answering campaigns, which have several different tasks. We only consider the task of English question answering over DBpedia in QALD-6 (Unger et al., 2016), which has 350 training questions and 100 test questions, with gold SPARQL queries and answer sets.

WebQuestions (Berant et al., 2013) consists of 3778 training questions and 2032 test questions, which are collected using the Google Suggest API and crowdsourcing. It only provides the pairs of question and answer set.

ComplexQuestions (Abujabal et al., 2017) is composed of 150 test questions that exhibit compositionality through multiple clauses, which is constructed using the crawl of WikiAnswers (http://wiki.answers.com) and human annotator. It has no training questions and SPARQL queries.

Table 1: Statistics of Knowledge Bases

	DBpedia	Freebase
Number of Entities	5.4 million	41 million
Number of Triples	110 million	596 million
Number of Predicates	9708	19456
Size of KBs (in GB)	8.7	56.9

Table 2: The average F1 score of WebQuestions and ComplexQuestions benchmark

	WQ	CQ
STF (Our approach)	**53.6%**	**54.3%**
STAGG (Yih et al., 2015)	52.5%	-
QUINT (Abujabal et al., 2017)	51.0%	49.2%
NFF (Hu et al., 2018)	49.6%	-
Aqqu (Bast and Haussmann, 2015)	49.4%	27.8%
Aqqu++ (Bast and Haussmann, 2015)	49.4%	46.7%

4.2 Results on WebQuestions and ComplexQuestions

Table 2 shows the results on the test set of WebQuestions(WQ) and ComplexQuestions(CQ), which are based on the Freebase. We compare our system with STAGG (Yih et al., 2015), QUINT (Abujabal et al., 2017), NFF (Hu et al., 2018) and Aqqu(Bast and Haussmann, 2015). QUINT is the model used in the original ComplexQuestions paper and Aqqu is the best publicly available QA system on WebQuestions. The average F1 of our system (53.6% for WQ and 54.3% for CQ) are better than the other systems.

Aqqu defines three query templates for WebQuestions and try to match test questions to predefined templates. It has a poor performance (27.8%) on ComplexQuestions when answering the test questions directly. Aqqu++ shows the result (46.7%) by taking manually decomposed subquestions as input and getting the intersection of subquestions' answer sets. QUINT is a system that automatically learns utterance-query templates from the pairs of question and answer set. NFF builds a relation paraphrase dictionary and leverages it to extract relations from the questions. STAGG proposes a state-transition based query graph generation method. However, its query graph structure is limited as in the Figure 1, which is very similar with the templates of Aqqu.

To generate the training data of relation extraction model on the WebQuestions, for each two nodes u and v in the training question, we explore all simple relations between u and v in Freebase. Note that if u or v is the answer node, we will anchor the answer entity first and select the relation which can handle most of answer entities as the gold relation. Each entity will be replaced in text by a special token representing its type before training. As there are no training data in ComplexQuestions, we use the models trained on WebQuestions directly.

Generally, WebQuestions benchmark has low diversity and contains some incorrect or incomplete labeled answers. Most of questions in WebQuestions are simple questions, which can be translated to a "one-triple" query. The results on WebQuestions benchmark prove our approach is able to obtain good performance on simple questions. ComplexQuestions benchmark is more complex than WebQuestions. The experiments results show that the two template based methods QUINT and Aqqu can solve a part of the complex questions. However, it still lack of the capacity to tackle all questions due to the error of dependency parse or the mismatch of the templates. On the other hand, our system outperforms them because we do not rely on templates and have better generalization ability to solve complex questions.

Table 3: Evaluating QALD-6 Testing Questions

	Processed	Right	Recall	Precision	F-1
Our approach	100	70	0.72	0.89	0.80
CANaLI	100	83	0.89	0.89	**0.89**
UTQA	100	63	0.69	0.82	0.75
KWGAnswer	100	52	0.59	0.85	0.70
SemGraphQA	100	20	0.25	0.70	0.37
UIQA1	44	21	0.63	0.54	0.25
UIQA2	36	14	0.53	0.43	0.17
NFF	100	68	0.70	0.89	0.78
gAnswer	100	40	0.43	0.77	0.55
Aqqu	100	36	0.37	0.39	0.38

4.3 Results on QALD

Table 3 shows the evaluation results on the test set of QALD-6. We compare our system with all participants of QALD-6 as well as gAnswer (Zou et al., 2014), NFF (Hu et al., 2018) and Aqqu (Bast and Haussmann, 2015). To enable the comparison, we show the experiment results in the QALD competition report format. "Processed" denotes the number of questions can be processed and "Right" refers to the number of questions that were answered correctly. Our approach can answer 70 questions correctly, and get the 0.80 F-1 score.

The experiment results show that we can beat all systems in QALD-6 campaign in F-1 except for CANaLI (Mazzeo and Zaniolo). Note that CANaLI is not an automatic QA system, as it needs users to specify the precise entities and predicates in the question sentences. gAnswer (Zou et al., 2014) proposes a relation first framework to generate the semantic query graph while it can not detect implicit relations. NFF (Hu et al., 2018) proposes a node first framework which performs well on answering complex ques-

Table 4: The performance comparison of state transition with conditions or without conditions in QALD-6 test set

	Right	Recall	Precision	F-1
with condition	70	0.72	0.89	0.80
no condition	62	0.64	0.84	0.72

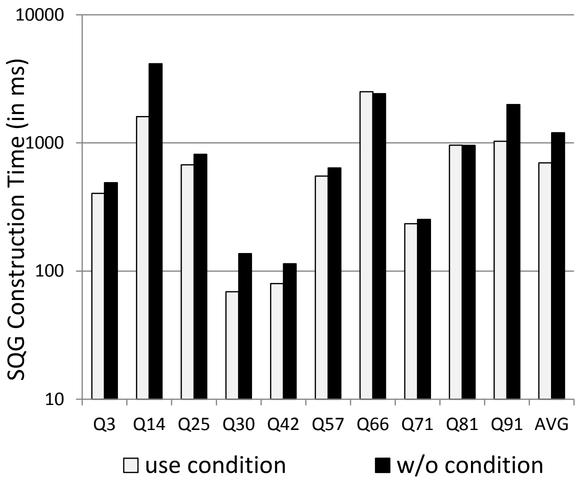

Figure 5: The elapsed time comparison of state transition with conditions or without conditions in QALD-6 test set

tions (0.78 F1 score on QALD-6). However it relies on the predefined paraphrasing dictionaries to extract relations and could not tackle with redundant nodes.

As QALD has the gold SPARQL queries, we can obtain the training data of node recognition and relation extraction model directly. Different from the WebQuestions and ComplexQuestions which have only special questions, the QALD benchmark also has general questions and imperative sentence. Besides, there are some questions that have no entities or multiple entities.

Generally, QALD benchmark has both high diversity and quality. The performance of Aqqu in QALD is worse than its performance in WebQuestions and ComplexQuestions as it has only designed the templates to WebQuestions. In contrast, our approach do not rely specific benchmark or knowledge base. The evaluation results show that our approach performs well on various types of complex questions.

4.4 Comparison experiments of the four conditions

We run experiments on QALD-6 test set to verify the effectiveness of the four conditions proposed in Section 2.2. We first compare the elapsed time of state transition with conditions or without conditions. Figure 5 shows the results of 10 questions selected randomly and AVG means the average time consumption among all 100 test questions. The average elapsed time with conditions is 700 ms while it rises to 1200 ms without the conditions. The results declare that using the conditions in the search process can avoid unnecessary searches and save SQG construction time. In some questions (such as Q66 and Q81), the gap of elapsed time are very little because the number of recognized nodes are less than three. In other words, those simple questions do not need the conditions to speed up the SQG construction process.

On the other hand, using conditions can improve the evaluation performance (see Table 4). That because the conditions help system to reduce search space and avoid local optimum, especially the condition of connect operation. When the question has multiple nodes, if we do not use the condition of connect operation, it tries to connect each two nodes and enrolls too many mistaken states.

Table 5: Failure Analysis on QALD-6

Reason	# (Ratio)	Sample question
Structure Failure	6 (20%)	Q55: In which countries do people speak Japanese?
Entity Mapping	5 (17%)	Q88: What color expresses loyalty?
Relation Mapping	10 (33%)	Q44: What is the full name of Prince Charles?
Complex Aggregation	9 (30%)	Q80: How short is the shortest active NBA player?

4.5 Error Analysis

We provide the error analysis of our approach on QALD-6 test set (100 questions), which contains gold SPARQL queries and more diversified questions. The ratio of each error type and the corresponding examples are given in Table 5.

There are mainly four reasons for the failure of some questions in our approach. The first reason is the structure failure, which means we generate a wrong SQG. That usually occurred when the correct SQG needs a fold operation. For example, the correct SQG of question "In which countries do people speak Japanese" has only two nodes "countries", "Japanese". However, we recognized the phrase "people" as a variable node by mistake and connected it to both "countries" and "Japanese" to generate a wrong SQG $Q^{S'}$. It should be eliminated by the fold operation to get the correct SQG Q^S. However, the score of $Q^{S'}$ that we get from the reward function is higher than the score of Q^S. The inherent reason is the lack of training data.

The second reason is the failure of entity linking, for example, we could not find the correct entity of the node "loyalty" in the question "What color expresses loyalty?". It has smallest proportion as the existing entity linking algorithms already have high accuracy on QA task.

The third reason is the failure of relation extraction. For example, the correct relation between "What" and "Prince Charles" in the question 'What is the full name of Prince Charles?" is $\langle$alias$\rangle$. However, we got a wrong relation $\langle$name$\rangle$. The relation extraction problem is very important to QA task, which can be improved by designing more elegant models and combining more useful information.

The last one is that our method cannot answer some complex aggregation questions, which need to detect the aggregation constraint first and then infer the related node and corresponding relation. Using predefined templates or utilizing textual evidence maybe works but not good enough.

5 Related Work

Generally, the solutions of KBQA can be divided into *information retrieval-based* and *semantic parsing-based* methods. The general process of *information retrieval-based* methods is to select candidate answers first and then rank them by various methods(Veyseh, 2016; Dong et al., 2015; Xu et al., 2016; Bordes et al., 2014; Yao and Durme, 2014). The main difference among *information retrieval-based* methods is how to rank the candidate answers. (Bordes et al., 2014) utilizes subgraph embedding to predict the confidence of candidate answers. (Dong et al., 2015) maximize the similarity between the distributed representation of a question and its answer candidates using Multi-Column Convolutional Neural Networks (MCCNN), while (Xu et al., 2016) aims to predicate the correct relations between topic entity and answer candidates with text evidence.

The *semantic parsing-based* methods try to translate the natural language question into semantic equivalent logical forms, such as simple λ-DCS (Berant et al., 2013; Berant and Liang, 2014), query graph (Yih et al., 2015; Zou et al., 2014), or executable queries such as SPARQL (Bast and Haussmann, 2015; Unger et al., 2012; Yahya et al., 2012). Then the logical forms are executed by corresponding technique and get the answers from knowledge base. For answering complex questions, the key is how to construct the logical form. Different from existing systems relying on the predefined templates, we build the query graph by state transition, which has stronger representation power and more robustness.

(Dong and Lapata, 2016) trains a sequence to tree model to translate natural language to logical forms directly, which needs a lot of training data. (Yu et al., 2017) uses deep bidirectional LSTMs to learn both relation-level and word-level question/relation representations and match the question to candidate relations. The pipeline of (Yu et al., 2017) is similar with (Yih et al., 2015), which detect topic entity and relation first and then recognize constraints by enumerating all connected entities of answer node or CVT node in knowledge base. The issue is they assume the question N only have one relation, which is from the topic entity to the answer. Different from (Yu et al., 2017), we consider the question can have many nodes (including entities and variables) and recognize them by a BLSTM model. We also allow multi relations between these nodes and extract them by the MCCNN model.

6 Conclusions

In this paper, we propose a state transition framework to utilize neural networks to answer complex questions, which generates semantic query graph based on four primitive operations. We train a BLSTM-CRF model to recognize nodes including entities and variables from the question sentence. To extract the relations between those nodes, we propose a MCCNN model which can tackle both explicit and implicit relations. Comparing with existing solutions, our framework do not rely on handcrafted templates and has the potential to generate all kinds of SQGs given enough training data. Extensive experiments on multi benchmark datasets confirm that our approach have the state-of-art performance.

Acknowledgments

This work was supported by The National Key Research and Development Program of China under grant 2016YFB1000603 and NSFC under grant 61622201 and 61532010. Lei Zou is the corresponding author of this work. This work is also supported by Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology).

References

Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. 2017. Automated template generation for question answering over knowledge graphs. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 1191–1200.

Jun-Wei Bao, Nan Duan, Ming Zhou, and Tiejun Zhao. 2014. Knowledge-based question answering as machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 967–976.

Hannah Bast and Elmar Haussmann. 2015. More accurate question answering on freebase. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, pages 1431–1440.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 1415–1425.

Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 2009. Dbpedia - a crystallization point for the web of data. *J. Web Sem.*, 7(3):154–165.

Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 615–620.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. 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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 260–269.

Sen Hu, Lei Zou, Jeffrey Xu Yu, Haixun Wang, and Dongyan Zhao. 2018. Answering natural language questions by subgraph matching over knowledge graphs. *IEEE Trans. Knowl. Data Eng.*, 30(5):824–837.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

Thorsten Joachims. 2006. Training linear svms in linear time. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 217–226.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*.

Giuseppe M. Mazzeo and Carlo Zaniolo. Answering controlled natural language questions on RDF knowledge bases. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 608–611.

Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. 2012. Template-based question answering over rdf data. In *WWW*, pages 639–648.

Christina Unger, Axel-Cyrille Ngonga Ngomo, and Elena Cabrio. 2016. 6th open challenge on question answering over linked data (QALD-6). In *Semantic Web Challenges - Third SemWebEval Challenge at ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers*, pages 171–177.

Amir Pouran Ben Veyseh. 2016. Cross-lingual question answering using common semantic space. In *Proceedings of TextGraphs@NAACL-HLT 2016: the 10th Workshop on Graph-based Methods for Natural Language Processing, June 17, 2016, San Diego, California, USA*, pages 15–19.

Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question answering on freebase via relation extraction and textual evidence. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. 2012. Natural language questions for the web of data. In *EMNLP-CoNLL*, pages 379–390.

Yi Yang and Ming-Wei Chang. 2015. S-MART: novel tree-based structured learning algorithms applied to tweet entity linking. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 504–513.

Xuchen Yao and Benjamin Van Durme. 2014. Information extraction over structured data: Question answering with freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 956–966.

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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1321–1331.

Mo Yu, Wenpeng Yin, Kazi Saidul Hasan, Cícero Nogueira dos Santos, Bing Xiang, and Bowen Zhou. 2017. Improved neural relation detection for knowledge base question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 571–581.

Xinbo Zhang and Lei Zou. 2017. Improving the precision of RDF question/answering systems: A why not approach. In *Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia, April 3-7, 2017*, pages 877–878.

Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural language question answering over RDF: a graph data driven approach. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 313–324.