IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 24, 2019, accepted April 15, 2019, date of publication May 23, 2019, date of current version June 21, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2918675

ComQA: Question Answering Over Knowledge Base via Semantic Matching

HAI JIN, (Fellow, IEEE), YI LUO, CHENJING GAO, XUNZHU TANG, AND PINGPENG YUAN, (Member, IEEE)

National Engineering Research Center for Big Data Technology and System/Services Computing Technology and System Lab/Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

Corresponding author: Pingpeng Yuan (ppyuan@hust.edu.cn)

This research was supported by The National Key Research & Development Program of China (No. 2018YFB1004002), NSFC (No. 61672255), Science and Technology Planning Project of Guangdong Province, China(No. 2016B030306003 and 2016B030305002), and the Fundamental Research Funds for the Central Universities, HUST.

ABSTRACT Question answering over knowledge base (KBQA) is a powerful tool to extract answers from graph-like knowledge bases. Here, we present ComQA—a three-phase KBQA framework by which end-users can ask complex questions and get answers in a natural way. In ComQA, a complex question is decomposed into several triple patterns. Then, ComQA retrieves candidate subgraphs matching the triple patterns from the knowledge base and evaluates the semantic similarity between the subgraphs and the triple patterns to find the answer. It is a long-standing problem to evaluate the semantic similarity between the question and the heterogeneous subgraph containing the answer. To handle this problem, first, a semantic-based extension method is proposed to identify entities and relations in the question while considering the underlying knowledge base. The precision of identifying entities and relations determines the correctness of successive steps. Second, by exploiting the syntactic pattern in the question, ComQA constructs the query graphs for natural language questions so that it can filter out topology-mismatch subgraphs and narrow down the search space in knowledge bases. Finally, by incorporating the information from the underlying knowledge base, we fine-tune general word vectors, making them more specific to ranking possible answers in KBQA task. Extensive experiments over a series of QALD challenges confirm that the performance of ComQA is comparable to those state-of-the-art approaches with respect to precision, recall, and F1-score.

INDEX TERMS Question answering, knowledge graph, semantic matching.

I. INTRODUCTION

With the advent of open data (open government data initiatives, open access to scientific data, Linked Open Data etc.), more and more knowledge graphs, including DBpedia [1] and Freebase [2] can be accessed publicly. The ever increasing public knowledge graphs on the Web are becoming a information backbone of many systems, such as Question answering over knowledge base (KBQA), a quite useful tool which helps people naturally retrieve answers from large-scale semi-structured knowledge bases.

Although knowledge graphs contain detailed information about entities and the connections between them—which are useful to figure out the true intention of the questions by offering a richer context—it is still difficult to build the map between the given question and the target answer in the underlying knowledge base. For example, if we pose a simple question "Who is the CEO of Apple?" to a KBQA framework, the first issue is "Which apple". It is nontrivial for KBQA framework to know the correct meaning of a word—⟨apple company⟩ instead of ⟨apple fruit⟩ in the question. When performing KBQA task in question answering systems, there are three challenges to be tackled and the previous example exemplifies the **first challenge**. That is how to locate the key entities and extract relations from input question while considering the underlying knowledge base. Since the entity or relation in the question may have a different mention in the underlying knowledge base, this mismatched mention problem can lead to returning wrong answer. Fig. 1(a) illustrates the mismatched mention problem when giving the question Q_{e2} to ComQA (Q_{e2} is introduced in next section). To address this challenge, first, the "seed

The associate editor coordinating the review of this manuscript and approving it for publication was Chang Choi.

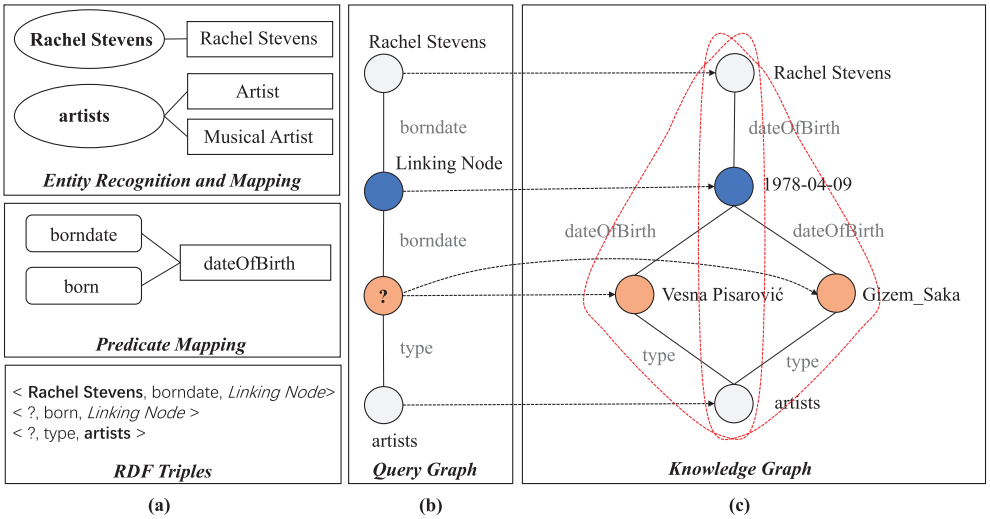

FIGURE 1. Overview of ComQA.

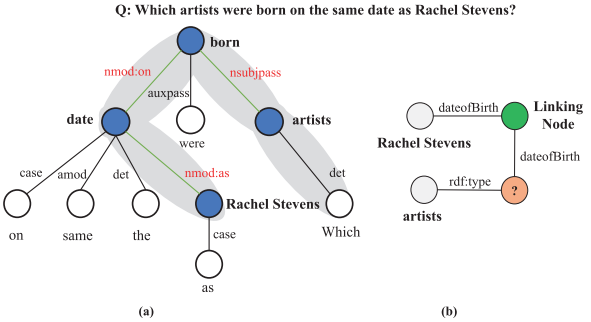

FIGURE 2. An example for dependency-tree and query graph. (a) Dependency Tree. (b) Gold Standard Query Graph.

entity'' is recognized by identifying the named entities [3] in the question while considering the underlying knowledge base. Further, a dependency-tree [4] is created by parsing the input question. Fig. 2(a) illustrates the dependency-tree of question Q_{e2}. With seed entity and dependency-tree, ComQA extracts ''seed relation'' from the question. By extending seed entity and seed relation semantically, ComQA generates *extended sets* (Section III, Definition 4) which can mitigate mismatched mention problem effectively.

Once the KBQA frameworks figure out the correct meaning of ⟨apple⟩, the following issue is How to locate the corresponding subgraphs which may contain target answer(s) in the knowledge base. Considering the magnitude of the knowledge base and the rich connections between any pair of entities, simply linking ⟨apple⟩ to the knowledge base will bring about many unrelated subgraphs. Let us consider the running example Q_{e2}, where the entity ⟨Rachel_Stevens⟩ has several related mentions in underlying DBpedia such as ⟨Rachel_Stevens⟩ and ⟨Rachel_Stevens_Song⟩. Subgraphs containing ⟨Rachel_Stevens_Song⟩ are obviously misleading candidates and should be discarded. This example exemplifies the **second challenge** in KBQA tasks—that is how to narrow down the search space of the final answer after linking the key entities to the underlying knowledge base. To address this challenge, we exploit the syntactic constraints containing in the question to filter out misleading subgraphs. By recognizing the *critical path* and the *linking node* in the dependency-tree, ComQA captures the syntactic pattern in the question and finally constructs the query graph which implies the syntactic constraints from the question. Fig. 1(b) illustrates the query graph of question Q_{e2}.

If the KBQA frameworks retrieve candidate subgraph(s) from the knowledge graph, the last issue is ''how to extract the answer by mapping the relation ⟨CEO of apple⟩ onto candidate subgraph(s) to find answer''. Inappropriate handling this issues can lead to a missing or wrong answer and it exemplifies the **third challenge** in KBQA—that is how to rank the retrieved subgraphs after mapping query graphs onto the knowledge base. To handle this challenge, we define a measurement of semantic similarity. Rather than simply using general word vectors to model the semantics, we fine-tune them by incorporating the information from the underlying knowledge base, making these vectors more specific to the ranking task at hand. By comparing the semantic similarity between the query graphs and retrieved subgraphs, ComQA can rank these candidate subgraphs and extracts the answer from the most promising one(s).

All the issues stem from how to evaluate the semantic similarity between the natural language question and the retrieved candidate subgraphs. To handle the problem, we develop ComQA, a three-phase KBQA framework focusing on semantic similarity. In ComQA, the process of question answering includes three phases—question parsing, query graph construction, and candidate subgraph evaluation. After parsing the question and construct the query graph, ComQA retrieves candidate subgraphs from the knowledge base and then excludes the subgraphs proved to be wrong by evaluating semantics similarity between subgraphs and questions. Then ComQA ranks final subgraphs to return the answer(s).

In this paper, our contributions are as follows:

1) We present ComQA, a three-phase KBQA framework. By offering end-users a natural language interface, they can ask complex questions and retrieve answers in a natural way, without any prior knowledge about the underlying knowledge base.
2) We devise a semantic-based extending method for entities and relations in the original question and incorporate it into the process of query graph construction, which is effective in capturing the compositional mention of entity (or relation) from the knowledge base.
3) By exploiting the syntactic pattern in the question, We propose an algorithm to construct the corresponding query graph, with which we can filter out topology-mismatch subgraphs to narrow down the search space of the answer in knowledge base.
4) We propose a semantic similarity measurement. Instead of using general word vectors to model the semantics, we fine-tune these vectors by incorporating the information from the underlying knowledge base.
5) We conduct extensive experiments against a series of QALD challenges. The results demonstrate that our ComQA is comparable with state-of-the-art approaches in terms of recall, precision and F1-score.

II. CONCEPTS AND FRAMEWORK OVERVIEW

The main function of ComQA is retrieving answers from the underlying knowledge base when given a natural language question Q. We will use the following questions Q_{e1} and Q_{e2} as our two running examples for demonstration. Fig. 1 illustrates the workflow of answering Q_{e2} with ComQA.

$$Q_{e1}: \textit{Who is the foreign minister of Ethiopia}?$$
$$Q_{e2}: \textit{Which artists were born on the same date as Rachel Stevens}?$$

To make our following description more precise, we first clarify the concept of *mention*.

Mention. In ComQA, we use mention to refer to an entity or a relation. An individual entity or a relation can have several mentions. For example, an entity mentioned as m_1 in the knowledge base may have another mention m_2 in the input question. Both m_1 and m_2 refer to the same entity.

Definition 1 (Relation): If given a RDF triple in the form of $\langle entity_1, rel, entity_2\rangle$ where $entity_1$ and $entity_2$ are two entities, *rel* is the relation which indicates how $entity_1$ is related to $entity_2$.

Generally, the workflow of our ComQA consists of three phases. Each phrase focuses on one of the three challenges. Phase I focuses on the extraction of key entities and relations. Phase II focuses on the construction of query graph. Phase III focuses on ranking the retrieved subgraphs. We will give a detailed explanation of all three phases in the following three sections.

In its first phase, ComQA takes as input a natural language question and performs the question parsing. The goal of question parsing is to locate the key entity (*ke*) in the question and extract the relation (*rel*) between *ke* and the target answer. For instance, in RDF triple ⟨Apple, hasCEO, Cook⟩ corresponding to "Who is the CEO of Apple?", ⟨Apple⟩ is the *ke* and ⟨hasCEO⟩ is the *rel*. In ComQA, *ke* links the question to potential subgraphs in the knowledge base, specifying the search space of the answer in knowledge base. While *rel* dominates ranking retrieved subgraphs. Simply using *ke* and *rel* extracted from the input question will encounter following problems:

1) The first one is what we called polysemy problem where a word or phrase may have several meanings. An example is ⟨apple⟩ in "Who is the CEO of Apple?"
2) The second one is what we called ambiguity problem. It means that *ke* can be linked to several related entities in the underlying knowledge base. Given the question Q_{e2}, the entity ⟨Rachel_Stevens⟩ can be linked to ⟨Rachel_Stevens⟩ or ⟨Rachel_Stevens_Song⟩ in DBpedia.
3) The last and worst is *composition problem* where there is improper entity or relation the underlying knowledge base being linked to *ke* or *rel*. Given the question "Who is the foreign minister of Ethiopia?", it is the *foreign_minister* that should be treated as the *rel*. Directly regrading *minister* as the *rel* will lead to returning a wrong answer.

Comparing with polysemy and ambiguity, composition problem is more critical. Although the first two problems introduce noise, it can be filtered out by a certain mechanism. While the composition problem will cause the loss of key information thus can make the following processing meaningless. Considering the fact that components of a composition (like *foreign* and *minister*) usually co-occurs in a corpora, we adopt GloVe [5], a global word vector representation algorithm which pays special attention to the co-occurrence of words. Based on GloVe, we present an approach to find entities most similar to *ke* and pack them with *ke* to create an extended set. In addition, the same process also applies to the extracted relation. With extended set, ComQA can mitigate the composition problem effectively.

After locating the key entities and extracting the possible relations, workflow of ComQA comes to its second phase. The goal of second phase is to construct the query graph revealing the syntactic pattern in the question, which also implies what kind of answer the end-user actually wants. The core issue in second phase is how to construct structured query graph from input question which is a sequence of words. Given a question, ComQA creates its dependency tree [6] to perform question parsing in phase I. By recognizing syntactic pattern and revealing hidden structure from dependency tree, ComQA can extract a series of RDF triples. After joining these RDF triples, the original query graph can be finally constructed. In addition, Considering the composition problem mentioned in the first phase, a set of extended query graphs will be constructed by replacing the key entities and

their corresponding relations in the original query graph with their corresponding extended sets.

With the set of query graphs constructed in the second phase, the goal of the last phase is mapping query graphs onto the knowledge base and evaluating matched subgraphs to figure out the answer. The first issue in this phase is how to avoid retrieving improper subgraphs from the knowledge base. For the sake of preventing the key information loss, we adopt the strategy—*setting a lower threshold for information to get in*. Anything relating to the target answer will be considered during the first two phases, which inevitably brings about some noise. To handle this noise-filtering issue, ComQA prunes those obviously improper subgraphs based on the topology information of query graphs. The second issue in this phase is how to rank filtered subgraphs to choose the most promising candidate(s). The ranking process in ComQA is hold by evaluating the semantic similarity between the query graphs and candidate subgraphs. For measuring the semantic similarity, we model the semantics of the vertices and edges in the knowledge base as word vectors. Further, considering that the general word vectors like GloVe, which is learned from a text-based corpora rather than a graph-like knowledge base, we adopt TransE [7] to fine-tune these pre-training word vectors, making them more specific to the underlying knowledge base. By calculating the cosine value of the angle between find-tuned vectors, ComQA can evaluate the their semantic similarity. Once the evaluation is finished, each candidate subgraph will receive a score with which ComQA determines the best candidate subgraph and returns the final answer based on this selected subgraph.

III. ENTITY AND RELATION EXTRACTION

In phase I, the core task is how to parse the question to locate key entities and extract relations from the question. These entities and relations are so important that with them we can link the input question to the underlying knowledge base. Specially, extracted entities from the question specify the search space of the answer in the knowledge base while relations dominate ranking retrieved subgraphs in phase III.

To begin question parsing, we employ DBpedia Spotlight to recognize the ''seed entity'' from input question. Since the input question can be in any reasonable form, extracting relation is not a simple task. Considering the fact that a relation is often denoted by a verb or a verb complemented by other words, syntax parsing tools are quite helpful in extracting relations since these tools can reveal the syntactic structure of the input sentence. In ComQA, we adopt Stanford Parser to create the dependency-tree for the input question. In this dependency-tree, we denote the seed entity recognized by the DBpedia Spotlight as a starting point and the predicate verb as an end point. Vertices in the path connecting the starting point and the end point are extracted as the complement of the predicate verb. With the predicate verb and its complements, ComQA can extracted the ''seed relations''. For example, considering the dependency-tree of Q_{e2} in Fig. 2(a), the predicate verb is ''born'' while ''date'' complements ''born'', ComQA will regard ⟨born, borndate⟩ as the seed relations.

Although we can extract the seed entities and seed relations from the question, knowing in advance their corresponding mentions in the knowledge base is nearly impossible. Thus it is hard to make sure that a mention in the question is exactly the same as its counterpart in the knowledge base, which makes the mismatched mention problem of entity (relation) be the most challenging part in phase I.

When linking key entities to the underlying knowledge base, the main problem lies in the diverse mentions that an individual entity can have—an entity may have a mention which is different from the corresponding mention in the question. If these two mentions do mismatch, tools like DBpedia Spotlight will fail to recognize this entity from the question. For example, if the end-user asks Q_{e2} with the nickname of ⟨Rachel_Stevens⟩, it is difficult to link this nickname to ⟨Rachel_Stevens⟩, even though the entity is lying in the knowledge base.

Except for the cases of mismatched entity mentions, a relation may also have different mentions between the input question and the underlying knowledge base. A typical case of mismatched mentions is the composition problem. For example, the relation ⟨borndate⟩ in Q_{e2} has a mention ⟨dateofBirth⟩ in DBpedia. If we simply search ⟨born⟩ against the RDF graph, no answer will be found. The underlying knowledge base, however, actually contains the target relation denoted by ⟨dateofBirth⟩.

Clearly, we need a semantic-based extending strategy. We denote the mention from the knowledge base as m^G and the mention from the question as m^Q. There are two useful observations:

1) *While m^Q may vary from person to person who use the KBQA framework, the corresponding m^G remains unchanged*. Thus m^G could be treated as the target. So long as we can turn the m^Q into a collection of mentions which are highly related to m^Q, it is promising that the target m^G will be matched with one of the mentions in this collection. Just like playing darts game, throwing several darts once a time is more likely to hit the target than merely throwing one dart.
2) *The underlying knowledge base is built from the textual data*. For example, DBpedia is built from the Wikipedia corpus. This textual corpus are also represented in the form of natural language like English, which implies that m^G is highly similar to m^Q in semantics. If we can build a collection of mentions which are highly close to m^Q both in semantics and lexical forms, it is likely that m^G will be in this collection.

Based on these two observations, we adopt GloVe, a global word vector representation algorithm. By representing m^Q in its GloVe word vector, we can find the top-k mentions which are most similar to m^Q in semantics (Definition 2) and take these mentions as the *Candidate Set* of m^Q. Considering the tradeoff between performance and overhead, k is 10. Since GloVe is learned from the real textual corpus and focuses on

the co-occurrence of words, it is safe to say that mentions in the candidate set of m^G co-occur with m^G with highest frequency thus candidate set can provide m^G with a richer context and complement the information of m^G.

Definition 2 (Similarity of Two Mentions): Give two mentions m_1 and m_2. Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be the corresponding word vector for m_1 and m_2, the similarity of m_1 and m_2 is defined as follows:

$$similarity(m_1, m_2) = \frac{\mathbf{c}_1 \cdot \mathbf{c}_2}{||\mathbf{c}_1|| \, ||\mathbf{c}_2||} \tag{1}$$

Definition 3 (Candidate Set): A candidate set is a collection of top-k mentions which are most similar to m^Q in semantics, where m^Q is the mention from the input question Q. If m^Q is an entity, its candidate set is denoted as CS_e. If m^Q is a relation, its candidate set is denoted as CS_r.

Algorithm 1 shows how to build the corresponding candidate set for a certain mention m^Q.

Algorithm 1 Building the Candidate Set

Input: A mention m^Q, The dictionary of GloVe *GW*, The GloVe vectors set *GV*
Output: The candidate set for m^Q
define a variable *CS* for the candidate set;
for each element *e* in *CS* **do**
  e ← *(NULL: 0)*;
define a variable *n_vec*;
n_vec ← *GV*[m^Q];
for each word vector *w* in *GW* **do**
  w_vec ← *GV[w]*;
  sim ← calculating the similarity between *w* and m^Q;
  for each element *e* in *CS* **do**
    min_sim ← find the minimum similarity in *CS*;
    min_w ← find the word indexing *min_sim*;
    if *sim* > *min_sim* **then**
      delete *min_w* from *CS*;
      CS[w] ← *sim*;
return *CS*;

Although the candidate set consists of mentions which are similar to m^Q, it can not be directly used in matching the m^G. One reason is that our choice, GloVe, focuses on the co-occurrence of words. mentions in candidate set are words which frequently co-occur with m^Q thus some of them may not be lexically similar to m^G. For example, in Q_{e2}, the CE_r of relation ⟨born⟩ contains mentions like ⟨footballer⟩ and ⟨ΦΔΣ-822-8448⟩ which clearly have nothing in common with the final answer to Q_{e2}. The other reason is that the actual entity or relation could be a phrase rather than a single word. When creating the dependency-tree of the input question, the question is finally broken into words. It is nontrivial to rebuild the correct phrase merely with syntax information. As shown in Q_{e1}, the actual relation is ⟨foreign_minister⟩. It is difficult to evaluate how important the adjective "foreign" is in forming the actual relation. But the CS_e of ⟨minister⟩ contains a mention ⟨foreign⟩, which means that ⟨foreign⟩ actually contains useful semantic information and should be reserved to form the actual relation—⟨foreign_minister⟩.

Take these two reasons into consideration, in order to filter noises out and handle the phrase issue, further processing for the candidate set of m^Q is necessary. In this case, We introduce the *Extended Set* for m^Q. The detail of creating the extended set is explained in Algorithm 2.

Definition 4 (Extended Set): An extended set for m^Q is a collection of mentions. Each mention is either a single word or a phrase which is selected according to the Algorithm 2. If m^Q is an entity, its extended set is denoted as ES_e. If m^Q is a relation, its extended set is denoted as ES_r.

Algorithm 2 Creating the Extended Set

Input: Dependency tree *T*, A mention *l*, and its extended set *CS*
Output: The extended set for m^Q
join(a, b) means join two words *a*, *b* to form a new word;
define a variable *ES* for extended set;
for each element *e* in *CS* **do**
  if *e* is *noun* **then**
    ES ← *ES* + *join(e, l)*;
    if *l* is *noun* **then**
      ES ← *ES* + *e*;
  if *e* is *verb* and *l* is *verb* **then**
    ES ← *ES* + *e*;
  if *e* is *adjective* and *l* is *noun* **then**
    ES ← *ES* + *join(e_l)*;
  if *(e* is *adverb* or *preposition)* and *l* is *verb* **then**
    ES ← *ES* + *join(e_l)*;
if *w* is *noun* and *w* is on the same layer with *l* in *T* **then**
  ES ← *ES* + *w*, *ES* ← *ES* + *(w_l)*;
return *ES*;

In a word, by creating the extended sets, ComQA can mitigate the mismatched mention problem when extracting entities and relations from the input question.

IV. QUERY GRAPH CONSTRUCTION

In phase I of ComQA, we create the extended set ES_e for each extracted entity and ES_r for each extracted relation. Although ES_e specifies the search space of the answer in the knowledge base, simply linking ES_e to the knowledge base is not a reasonable choice. As we mentioned in Section I, there are three problems including polysemy, ambiguity, and composition when associating the question with the knowledge base. ES_e can mitigate the composition problem but the polysemy and ambiguity can introduce noise which makes the search space too big. To narrow down the search space, we exploit the syntactic pattern hidden in the question. We introduce graph query to represent the syntactic pattern. So the core task of phase II is constructing the query graph from the question. In addition, considering the mismatched mention problem, it is necessary to generate more query graphs with extended sets by regarding the original query graph as a template.

In order to construct the query graph form the question which is a sequence of words, we exploit the topology information drawn from the dependency-tree T. Based on the T, we manage to construct the query graph to represent the intention of the input question. If a plausible subgraph is not isomorphic to the query graph, this subgraph will be discarded. A basic idea behind this criterion is that the dependency-tree T reveals the syntactic structure of the input question and T actually imposes syntactic constraints on the entity e^A which could represent the answer. Thus we can extract the "syntactic pattern" of e^A from T and encode it into the query graph. If a subgraph and the query graph are not isomorphic, this subgraph is high likely just the noise and should be filtered out.

As we mentioned above, the dependency-tree imposes the syntactic constraints on the entities which could form the answer but the dependency-tree still contains some useless noise. We should extract the syntactic pattern from the dependency-tree and construct the more compact query graph. There is an observation that the wh-words such as *what, how, and which* highly contribute to the intention of question while the query graph must include all the possible entities in the question, no matter whether the entity is already determined. Based on this observation, we presents the concept of *critical path*. Critical path captures the syntactic pattern which the entities in the question should comply with. We also presents a heuristic algorithm (Algorithm 3) to extract the critical path from the dependency-tree.

Definition 5 (Critical Path in Dependency-tree): Given a Dependency-tree T of the input question Q, Let e be a leaf entity at the bottom of T and let w be the wh-word of Q. The critical path is the shortest path which starts from e and ends at w which only consists of entities and the root verb.

Algorithm 3 Extracting Critical Path

Input: All the Extended sets *ES*, Dependency tree T
Output: The critical path p
$p \leftarrow \varnothing$;
define a variable N and $N \leftarrow$ all of leaf nodes in T;
define a variable *loc* and *loc* $\leftarrow$ NULL;
while *loc* $\neq$ *wh-word* **do**
  for each entity e in *ES* **do**
    if e is in N **then**
      $p \leftarrow p +$ <e>;
      loc $\leftarrow e$;
      delete e from *ES*;
    if *loc* $\neq$ *root node* **then**
      $N \leftarrow$ parents of *loc*;
    else
      $N \leftarrow$ (*children of loc - ES*);
    break;
define a variable *term* and *term* $\leftarrow$ last node in p;
if *term* == *root node* **then**
  $p \leftarrow p +$ <*wh-word*>;
return p;

Generally speaking, once the critical path is extracted from the dependency-tree, we can construct the query graph. When it coming to the complex question, however, there is another problem to be addressed. Here the complex question refers to the question whose query graph is consists of at least *two* RDF triples. In order to answer complex question like, we need to join several RDF triples to specify the pattern of the query graph, which will bring about the issue of *linking node*. The linking node plays a key role in constructing query graph of complex question—it joins the triple containing the target answer and the triple containing the already-known facts. In addition, a linking node can impose topology constraints on the query graph to further limit the scope of candidate subgraphs.

Definition 6 (Linking Node): Let Q be a input question. Let p be the critical path of Q. Let e be the entity in the graph query of Q and let w be the corresponding node of e in p. Node e is a linking node if and only if its corresponding e can not be directly located in p while e is linking two triples in the query graph.

A good example of the linking node can be found in Q_{e2} *"Which artists were born on the same date as Rachel Stevens?"* If we manually write the query of Q_{e2} in SPARQL, one possible way is as follows:

```
1 SELECT DISTINCT ?uri
2 WHERE {
3         ?uri rdf:type Artist .
4         ?uri dateOfBirth ?date .
5         Rachel_Stevens dateOfBirth ?date .
6 }
```

It is clear that the triple in line 4 and the triple in line 5 share a same entity denoted by *?date*. To make it more concrete, the corresponding query graph of Q_{e2} is shown in Fig. 3 (b). The green node in Fig. 3 (b) joins two triples. Although there is node mentioned by *date* in the critical path in Fig. 3 (a), it should be noted that this *date* node is just a kind of syntactic structure complementing the relation between the node *born* and the node *Rachel Stevens* thus it corresponds no entity. According to Definition 6, the green node in Fig. 3 (b) is the linking node of this query graph.

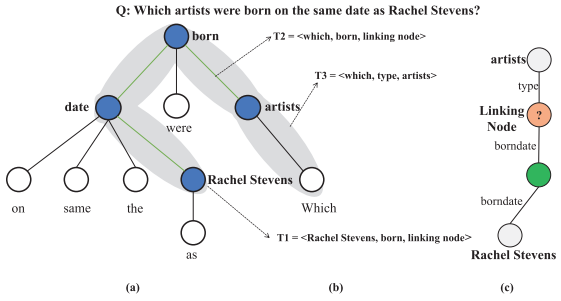

FIGURE 3. The process of constructing graph query. (a) Dependency Tree. (b) Triples. (c) Constructed Query Graph.

With the concept of linking node, we can finally construct the query graph of (complex) question from its critical path. Once we get the critical path of a question by applying

Algorithm 3, Algorithm 4 can be used to construct the query graph. Moreover, in Algorithm 4, we assume that the critical path remains the tree-like structure when it is extracted from the dependency-tree.

Algorithm 4 Constructing the Query Graph

Input: Critical path p, Dependency-tree T, and Extended set ES of the root node in T
Output: Query graph q, ES
$root \leftarrow$ the root node in T;
$id \leftarrow$ the bottom leaf node of p;
$end \leftarrow$ the wh-word in p;
$triple\_1 \leftarrow (s, NULL, NULL)$;
$s \leftarrow (s \rightarrow next)$;
while $s \neq root$ **do**
  $tmp \leftarrow s$, $s \leftarrow (s \rightarrow next)$;
  $tmp \leftarrow tmp + s$;
$ES \leftarrow ES + tmp$;
$triple\_1 \leftarrow$ merge $triple\_1$ and $(NULL, tmp, link)$;
$triple\_2 \leftarrow (link, tmp, ?)$;
$s \leftarrow (s \rightarrow next)$, $tmp \leftarrow NULL$;
while $s \neq end$ **do**
  $tmp \leftarrow tmp + s$, $s \leftarrow (s \rightarrow next)$;
if $type \neq NULL$ **then**
  $triple\_3 \leftarrow (?, type, tmp)$;
else
  $triple\_3 \leftarrow (?, type, end)$;
$q \leftarrow$ join $triple\_1$, $triple\_2$, and $triple\_3$;
return q, ES;

We will give an example to illustrate how Algorithms 4 works. This example is based on Q_{e2} and is shown in Fig. 2. The subtree in Fig. 2 (a) colored by gray corresponds to the critical path of Q_{e2}. According to Algorithm 4, the whole process begins with the bottom leaf node *Rachel Stevens*. Moving upward, the next node is *date*. This node is not the root node, we simply continue to moves upward. The next node is the root node—*born*. We concatenate the mention *born* with the *date* and put this new mention into the extended set of *born*. At the same time, we can build the fact triple *(Rachel Stevens, born, linking node)* and the linking node triple *(?, born, linking node)*. Then we move downward from root node. The next node is artists, then we move on. The next node is the terminal node—which. The traversal of the critical path is ended. With the information obtained from DBpedia Ontology [8], the node artist can indicates a "is-a" relation in the input question. Thus we can build the last triple *(which, rdf:type, artists)*. It is obvious that *which* and *?* correspond to a same entity. By joining these three triples together, we can construct the query graph of Q_{e2} as shown in Fig. 2 (c).

The query graph Q^mconstructed from the critical path can be treated as the template to generate other query graph. Based on the extended sets of entities and relations, the method of generating new query graph is listed as follows:

1) For a certain entity in Q^m, replace this entity with all the elements in its extended set once a time.
2) Based on the new query graphs generated from the step above, choose another entity not selected in the previous step, replace this entity with all the elements in its extended set once a time.
3) Repeat step 2 until all the possible entities in the original query graph are extended.

By generating new but good quality query graphs from the original one, more candidate subgraphs can be retrieved when mapping these generated query graphs onto the underlying knowledge base. Thus there should be a much better chance of retrieving the subgraph containing the target answer. And this is the main target of phase II in ComQA.

V. CANDIDATE SUBGRAPHS EVALUATION

At the end of phase II, ComQA finally generates a set of query graphs. The core task of phase III in ComQA is mapping these query graphs on the underlying knowledge base and evaluating these retrieved candidate subgraphs to locate the answer. By filtering and ranking these candidate subgraphs, ComQA can return the target answer to the input question.

When associating the question to the knowledge base, ComQA will encounter the problems of polysemy and ambiguity. So query graphs are constructed to filter out the misleading subgraphs introduced by these two problems. In order to explain how query graphs work, we first introduce the concept of *neighbor*, which is the initial search space of the answer.

To make it succinct, we denote the entity in the question as e^Q, the corresponding entity in the knowledge base as e^G, and the entity representing the answer as e^A. There is a *critical assumption* that once e^Q is correctly linked to the e^G, e^A should be the neighbor of e^G. If not, the input question can not be answered over the knowledge base. With this assumption, the search space of the e^A is limited to the neighbor of e^G. Further, we set the range of neighbor within 3-hop.

Definition 7: (Neighbor in the Knowledge Base): In a knowledge base G, Let e be an entity in G and let N be a set of entities in G. N is the neighbor of e if and only if the following conditions hold:

1) $N = \varnothing$;
2) $\forall n \in N$, n is in the range of n-hop from e;

Holding this "neighbor containing answer" assumption, ComQA starts mapping the query graph onto the underlying knowledge base with the already known entity in the *fact triple*. For example, the query graph of Q_{e2} includes three triples—$\langle$*Rachel Stevens, born, linking node*$\rangle$, $\langle$*?, born, linking node*$\rangle$, and $\langle$*which, rdf:type, artists*$\rangle$. According to Definition 8, $\langle$*Rachel Stevens, born, linking node*$\rangle$ is the fact triple and $\langle$*Rachel Stevens*$\rangle$ is the recognized entity and clearly in its extended set. We refer to $\langle$*Rachel Stevens*$\rangle$ as e. To begin the query graph mapping, e can be selected as the starting point. For e and its extended set, ComQA will search their possible counterparts in the knowledge base, which is an entity linking task. Once ComQA finds all the

candidate entities in the knowledge base, A BFS algorithm will be applied on these entities and all the neighbors of these candidate entities can be retrieved from the knowledge base as candidate subgraphs. However, entity linking is never a simple task which will introduce ambiguity, lots of those candidate entities in the knowledge graph are unrelated to the intention of Q_{e2}. Further, instead of only mapping e, ComQA also maps the extended set of e onto the knowledge base, which can get a much better chance of retrieving the target answer but inevitably brings extra noise. Thus before ranking these candidate subgraphs, Filtering those obviously unrelated subgraphs out is of great necessity. ComQA performs filtering task both with query graph information and semantics.

Definition 8 (Fact Triple): Given extended set of entity ES_e. Let q be the original query graph of the question Q and let t be a triple in q. t is a fact triple of q if t is in the form of $\langle$*entity, predicate, linking node*$\rangle$ *where entity* $\in ES_e$.

1) PRUNING UNRELATED PATH VIA QUERY GRAPH

As we argued in the previous section, the query graph of a input question imposes syntactic constraints on the entities which can form the target answer. The main body of these syntactic constraints resides in the topology of the query graph. If a path consisting of several triples is not isomorphic to the query graph, this path is supposed to contain no answer and should be pruned from the retrieved subgraphs. In addition, if a path is isomorphic to the query graph and one of its endpoints indicates the relation of "rdf:type", we can double check this endpoint against the query graph to make the result more precise. The details of pruning unrelated paths can be seen in Algorithm 5.

An example over Q_{e2} can help to illustrate how to prune unrelated paths in the retrieved subgraph. As shown in Fig. 4, the staring point is *Rachel Stevens*, and the length of the query is three (hops). It is clear the two paths listed in the leftmost of Fig. 4 (a) are not isomorphic to the query graph. Both of these two paths are shorter than the query graph and should be pruned from the retrieved subgraph. Fig. 4 (b) gives the results of pruning the unrelated paths.

2) RANKING PATHS ACCORDING TO SEMANTICS

By pruning unrelated paths from the candidate subgraphs, ComQA actually transforms the original subgraph into a

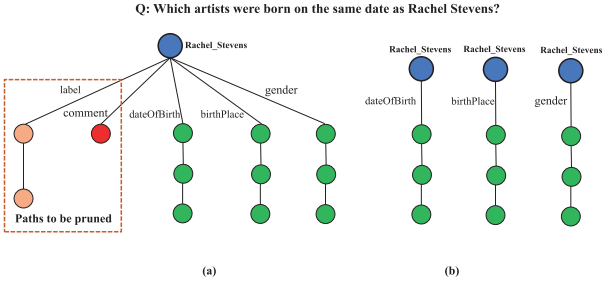

FIGURE 4. Pruning unrelated paths. (a) Retrieved Subgraph. (b) Candidate Paths.

Algorithm 5 Pruning Unrelated Paths via Query Graph

Input: Query graph q, retrieved subgraph G_s
Output: A set of path may contain the answer P
$s \leftarrow$ the *central node* of G_s;
$len \leftarrow$ *length of longest path* in q;
$N_a \leftarrow$ *all nodes* except s in G_s;
while $N_a \neq \varnothing$ **do**
  select an adjacent node i of s;
  $N_i \leftarrow$ *longest paths*starting from e while passing i;
  for each path p_i in N_i **do**
    $P_i \leftarrow \varnothing$;
    if *length* of $p_i \geq len$ **then**
      $P_i \leftarrow P_i + p_i$;
  $P \leftarrow P + P_i$;
  delete node i and all *its neighbors* except s;
return P;

set of discrete paths in the underlying knowledge base. We denote these paths as candidate paths. Considering the magnitude of candidate paths, we need to rank them to find the most likely path containing the answer. There is an observation that the edge joining two entities is actually a predicate and indicates a certain relation between this pair of entities. Given two RDF triples, the predicate can be used to measure the semantic similarity between the triples, which is helpful in ranking candidate paths. This task is known as predicate mapping.

When performing the predicate mapping, there exists two issues:

1) The first issue is that the predicate may have different mention in the query graph from its mention in the candidate path. For example, in Fig 5, the input question is "Which country did Bill Gates born in?". The predicate of the fact triple in query graph is "born in" while its counterpart in the answer path is "birthPlace". These different mentions of an identical predicate may leads to filtering out the answer path.
2) The second issue is the hidden linking node. As shown in Fig 5 (c), the answer path contains a linking node (colored by green) and consists of three triples. The query graph, however, fails to reveal the linking node thus consists of only two triples. In order to rank the candidate paths correctly, we need to expand the hidden link node in Fig 5 (a) to the triple denoted by the dashed rectangle in 5 (c).

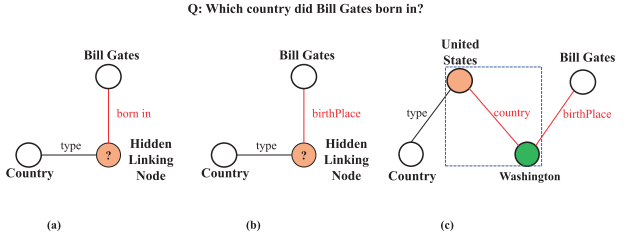

FIGURE 5. Locating the hidden linking node. (a) Original Query Graph. (b) Predicate Extended. (c) Path in Knowledge Base.

In order to handle these two issues above, we exploit word vector to represent the predicate in the query graph and in the candidate path. Thus we can measure the semantic similarity (Definition 2) between two predicates. Since the general word vector like GloVe is learning from the text-based data, it may not perform very well in representing the word from the knowledge base. We adopt TransE to fine-tune GloVe, making it more specific to the task related to the knowledge base. Based on word vector representation, ComQA can handle the mentioned two issues above.

For handling the first issue, the extended set of relation is critical. We can extend the predicate in the query graph to match the mention in the candidate path. Considering the fact that the most reliable triple in query graph is the fact triple and the candidate path sometimes is longer than the query graph (Fig. 5), the predicate extending process is only performed on the fact triple of query graph. We illustrate the whole process with an example. Fig. 5 (a) shows the query graph of question "Which country did Bill Gates born in?" and the fact triple is ⟨Bill Gates, born in, ?⟩. The predicate "born in" should be extended to match the predicate "birthPlace" in Fig. 5 (c). By applying the Algorithm 2, the word "birthin" is added to the extended set of "born in". By calculating the similarity between "birthin" and "birthPlace", ComQA will replace the predicate "born in" with "birthPlace". And the extended query graph is shown in Fig. 5 (b).

For handling the second issue, we exploit a particular property of TransE. For example, $\langle \mathbf{a}, \mathbf{h}, \mathbf{b} \rangle$ and $\langle \mathbf{c}, \mathbf{h}, \mathbf{d} \rangle$ are two RDF triples mapped into a vector space via TransE representation. Since these two triples share a same relation, vector $\langle \mathbf{b} - \mathbf{a} \rangle$ is very similar to vector $\langle \mathbf{d} - \mathbf{c} \rangle$. With this property, we can reveal the hidden linking node. As shown in Fig. 5 (b), we can get the TransE vector of ⟨country⟩ by using a formula $\overrightarrow{country} = \overrightarrow{Bill\_Gates} - \overrightarrow{Country}$. Then we calculate the TransE vector of ⟨ Country ⟩ in Fig. 5 (c) with a formula, $\overrightarrow{Country} = \overrightarrow{United\_States} + \overrightarrow{type}$. We denote the vector of the first ⟨country⟩ in Fig. 5 (b) as v_1 and denote the other as v_2. If $\langle v_1, v_2 \rangle$ is more similar than any other pairs of $\langle v_1, v_o \rangle$, ComQA will judge that the query graph in Fig. 5 (b) contains a hidden linking node and the candidate path in Fig. 5 (c) is the most likely mapping results of the query graph in Fig. 5 (a).

VI. EVALUATION

In this section, we conduct extensive experiments against a series of QALD challenges, which includes QALD-3, QALD-4 and QALD-5. We compare the performance of our ComQA with some representative KBQA systems. It should be noted that since squall2sparql puts constraints on the question and not directly uses the free natural language, we will not compare with it or give its results.

A. EXPERIMENTAL SETUP

The experiment setup includes dataset and software/hardware environment.

1) DATASET

All of the QALD challenges are running on the *DBpedia knowledge base* which is actually a large RDF knowledge graph built based on Wikipedia. Each QALD has its corresponding DBpedia version. To be specific, QALD-3 runs on DBpedia 3.8. QALD-4 runs on the DBpedia 3.9 and QALD-5 runs on DBpedia 2014-10.

2) ENVIRONMENT

All the experiments are conducted on a server computer, equipped with 2-way Intel Xeon CPU and 256 GB memory. The DBpedia knowledge base is treated as a whole RDF graph and managed by TripleBit [9] running on Centos 6.5.

B. PERFORMANCE OF QUESTION ANSWERING

In order to evaluate our ComQA extensively, we conduct experiments against a series of QALD challenges including QALD-3, QALD-4, and QALD-5.

After each QALD challenge, a official report of the challenge results will be released on its website.[1] The performance reports of other participants are excerpted from the corresponding official report. Since QALD challenge is held annually, the participants may vary from year to year.

The performance metrics in QALD are *precision*, *recall* and *F1-measure*. For each question q, its three performance metrics are as follows:

$$Precision(p_i) = \frac{number\ of\ correct\ answers}{number\ of\ all\ system\ answers}$$

$$Recall(p_i) = \frac{number\ of\ correct\ answers}{number\ of\ gold\ standard\ answers}$$

$$F1 - measure(p_i) = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

The results reported in this section are overall metrics, which are calculated by taking the mean of all individual precision, recall and F1-measure.

In addition, some questions in QALD challenge may contain server answering, if a certain system can not return all the correct answers, this question will be marked as "Partially".

Table 1 reports the results against QALD-3 challenge. From the table, we can see that ComQA can answer the most questions correctly (38 questions) and partially correctly (13 questions) while remain the highest *precision* (0.48). Our *recall* and *F1-measure* are still comparable with the leading systems. As a contrast, CASIA answers 29 questions correctly and 8 questions partially correctly.

Table 2 reports the results against QALD-4 challenge. According to this table, ComQA can handle the most questions (46 questions). It should be noted that Xser incorporate a KB-independent model which needed to be learned offline. Except for Xser, ComQA is the leading competitor among the remaining systems. We achieve a result of *0.34 recall*, *0.58 precision* and *0.36 F1-measure*. As a contrast, gAnswer achieves *0.37 recall*, *0.37 precision*, and *0.37 F1-measure*.

[1] https://qald.sebastianwalter.org

TABLE 1. QALD-3 on DBpedia 3.8.

	Proceed	Right	Partially	Recall	Precision	F1-measure
CASIA	52	29	8	**0.36**	0.35	**0.36**
Scalewelis	70	32	1	0.33	0.33	0.33
RTV	55	30	4	0.34	0.32	0.33
Intui2	**99**	28	4	0.32	0.32	0.32
SWIP	21	15	2	0.16	0.17	0.17
ComQA	87	**38**	**13**	0.31	**0.48**	0.31

TABLE 2. QALD-4 on DBpedia 3.9.

	Proceed	Right	Partially	Recall	Precision	F1-measure
Xser	40	**34**	6	**0.71**	**0.72**	**0.72**
gAnswer	25	16	4	0.37	0.37	0.37
CASIA	26	15	4	0.40	0.32	0.36
Intui3	33	10	4	0.25	0.23	0.24
ISOFT	28	10	3	0.26	0.21	0.23
RO_FII	50	6	0	0.12	0.12	0.12
ComQA	**46**	25	**7**	0.34	0.58	0.36

TABLE 3. QALD-5 on DBpedia 2014-10.

	Proceed	Right	Partially	Recall	Precision	F1-measure
Xser	42	26	**7**	**0.72**	**0.74**	**0.73**
APEQ	26	8	5	0.48	0.40	0.44
QAnswer	37	9	4	0.35	0.46	0.40
SemGraphQA	31	7	3	0.32	0.31	0.31
YodaQA	33	8	2	0.25	0.28	0.26
ComQA	**46**	**30**	4	0.39	0.65	0.40

Table 3 reports the results against QALD-5 challenge. From this table, we can find that Xser still performs best on *Recall*, Precision, and *F1-measure* with the same reason mentioned above. ComQA, however, can process the the most questions (46 questions) and can answer the most questions correctly (30 questions). As a contrast, Xser can process 42 questions and answers 26 questions correctly. Except Xser, our ComQA achieves the highest *precision* (0.65) and is comparable with other systems when considering *recall* and *F1-measure*. In addition, we report the 30 questions correctly answered in Table 4.

In addition, for some questions in three QALD challenges, especially those in QALD-3 challenge, it is difficult to extract the correct mention of the relation which can match the corresponding mention in DBpedia, only by exploiting the syntactic and semantic information. Without commonsense knowledge, answering such questions is a nontrivial task. A good example is "What is the total amount of men and women serving in the FDNY?" from QALD-3. The corresponding mention of the relation in DBpedia is ⟨dbp:strength⟩. Lacking commonsense knowledge, it is hard to match the ⟨dbp:strength⟩ with ⟨total amount of men and women⟩. Since our ComQA only exploits the syntactic and semantic information, it can hardly answer such kind of question. How to use commonsense knowledge in KBQA will be explored in the future research.

VII. RELATED WORK

QA (question answering) has a long history dating back to the 1970s in IR. Representative tasks of QA are benchmarks

TABLE 4. The Questions in QALD-5 which can be completely answered by ComQA.

ID	Questions
Q1	Give me all ESA astronauts.
Q2	Give me all Swedish holidays.
Q3	Who is the youngest Pulitzer Prize winner?
Q4	Which animals are critically endangered?
Q6	Did Arnold Schwarzenegger attend a university?
Q8	Is Barack Obama a democrat?
Q9	How many children does Eddie Murphy have?
Q10	Who is the oldest child of Meryl Streep?
Q12	In which city is Air China headquartered?
Q14	Which artists were born on the same date as Rachel Stevens?
Q18	Who is the manager of Real Madrid?
Q19	Give me the currency of China.
Q22	How many companies were founded by the founder of Facebook?
Q25	How many airlines are members of the Star Alliance?
Q27	Which musician wrote the most books?
Q30	In which country is Mecca located?
Q31	What is the net income of Apple?
Q32	What does the abbreviation FIFA stand for?
Q33	When did the Ming dynasty dissolve?
Q34	Which museum in New York has the most visitors?
Q37	What is the highest mountain in Italy?
Q39	Which Greek parties are Pro-European?
Q41	Who is the mayor of Rotterdam?
Q46	List the seven kings of Rome.
Q47	Who were the parents of Queen Victoria?
Q48	Who is the heaviest player of the Chicago Bulls?
Q50	Who is the tallest basketball player?

provided by TREC and SemEval. These tasks fall in the classical IR domain which focus on retrieving answers from text-based data sets. Thus, the requirements of the answer are not complex, even for the recently released TREC CAR (complex answer retrieve) data set [10]. IBM Waston [11] begins to exploit the expressiveness of structured database, but textual data still mainly contributes to Waston's knowledge sources. In order to handle complex question answer over text-based data, [11]–[13] make efforts in question decomposition. Although these technologies of reasoning and re-composition work well in conventional text-based QA tasks [12], the infrastructure of KBQA borrows heavily from the Semantic Web, which makes it hard to simply apply these technologies to the KBQA task. Another effort to answer complex question comes from [14] which pays attention to compositional semantics and can deal with several special types of complex questions.

KBQA receives lots of attention recently, especially when knowledge graph becomes more and more popular. [15]–[19] could be treated as the starting point of KBQA. These early work focus on answering simple questions. [15] also presents a widely-used benchmark—WebQuestions. Superior systems against WebQuestions benchmark typically adopt templates and grammars [20]–[23], incorporate supplementary information [24], [25], or directly train an end-to-end model [25]–[29]. Although all these systems can handle the binary factoid questions, they do not perform well in answering the complex questions presented in another famous benchmark QALD. Leading systems against QALD benchmark often regard the underlying knowledge base as a huge RDF graph. These systems usually parse the question

into entities and predicates, then manage to find a way to map these entities and predicates into the knowledge base correctly. In order to reduce the ambiguity from the natural language, [30] presents an approach to extract key phrases using knowledge graphs. Systems like [31], [32] restrict the vocabulary and grammar in the process of parsing the input question. These restrictions help to improve the answering quality, but also limit the expressiveness of the system. Other systems like [33]–[35] directly parse the input question without any restrictions. Xser [33] employs a method of ad-hoc lexicon to perform the mapping between the input question and the knowledge base. CASIA [34] adopts Markov Logic Network to perform the Disambiguation. gAnswer [35] builds a dictionary to perform the mapping between the predicates. All these three pay little attention to the context of the entities and predicates.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we present ComQA, a KBQA framework which can answer complex questions. Different from current work, we devise a semantic-based extending method which is effective in capturing the compositional entities or relations from the knowledge base. We also exploit syntactic pattern in the question to facilitate semantic matching. In addition, we fine-tune the pre-training general word vector by incorporating the information from the underlying knowledge base, making the vector more specific to the ranking task in KBQA. Extensive experiments over QALD confirm that our ComQA framework is comparable with state-of-the-art approaches in terms of recall, precision and F1-score.

In the future, we will explore how to represent the commonsense knowledge of the world in KBQA tasks. Then we will aim to incorporate the appropriately represented commonsense knowledge into the process of KBQA, which helps a lot in evaluating the semantic similarity of different entities (or relations) at a deeper level than merely considering the semantic similarity at the word level in our current work.

REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Lves, "Dbpedia: A nucleus for a web of open data," in *Proc. ISWC*, Busan, South Korea, 2007, pp. 722–735.

[2] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proc. SIGMOD*, Vancouver, BC, Canada, Jun. 2008, pp. 1247–1250.

[3] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes, "Improving efficiency and accuracy in multilingual entity extraction," in *Proc. ISEM*, Graz, Austria, Sep. 2013, pp. 121–124.

[4] S. Schuster and C. D. Manning, "Enhanced english universal dependencies: An improved representation for natural language understanding task," in *Proc. LREC*, Portorož, Slovenia, May 2016, pp. 23–28.

[5] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. EMNLP*, Doha, Qatar, 2014, pp. 1532–1543.

[6] M. Marneffe and C. D. Manning. (Sep. 2016). *Stanford Typed Dependencies Manual, Version 3.7.0*. [Online]. Available: https://nlp.stanford.edu/software/dependencies_manual.pdf

[7] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Proc. AAAI*, Austin, TX, USA, Feb. 2015, pp. 2181–2187.

[8] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Kleef, S. Auer, and C. Bizer, "DBpedia—A large-scale, multilingual knowledge base extracted from Wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015.

[9] P. Yuan, P. Liu, B. Wu, H. Jin, W. Zhang, and L. Liu, "TripleBit: A fast and compact system for large scale RDF data," *Proc. VLDB Endowment*, vol. 6, no. 7, pp. 517–528, May 2013.

[10] L. Dietz, B. Gamari, and J. Dalton. (2018). *Trec Car: A Data Set for Complex Answer Retrieval Version 2.1*. [Online]. Available: http://trec-car.cs.unh.edu

[11] D. A. Ferrucci, "Introduction to 'this is watson'," *IBM J. Res. Develop.*, vol. 56, nos. 3–4, pp. 1:1–1:15, May/Jun. 2012.

[12] E. Saquete, J. L. Vicedo, and P. Martínez-Bar, R. Muñoz, and H. Llorens, "Enhancing QA systems with complex temporal question processing capabilities," *JAIR*, vol. 35, no. 1, pp. 775–811, May 2009.

[13] P. Yin, N. Duan, B. Kao, J. Bao, and M. Zhou, "Answering questions with complex semantic constraints on open knowledge bases," in *Proc. CIKM*, Melbourne, VIC, Australia, Oct. 2015, pp. 1301–1310.

[14] P. Liang, M. I. Jordan, and D. Klein, "Learning dependency-based compositional semantics," *Comput. Linguistics*, vol. 39, no. 2, pp. 389–446, Jun. 2013.

[15] J. Berant, A. Chou, R. Frostig, and P. Liang, "Semantic parsing on freebase from question-answer pairs," in *Proc. EMNLP*, Washington, DC, USA, 2013, pp. 1533–1544.

[16] Q. Cai and A. Yates, "Large-scale semantic parsing via schema matching and lexicon extension," in *Proc. ACL*, Sofia, Bulgaria, 2013, pp. 423–433.

[17] A. Fader, L. Zettlemoyer, and O. Etzioni, "Open question answering over curated and extracted knowledge bases," in *Proc. KDD*, New York, NY, USA, Aug. 2014, pp. 1156–1165.

[18] C. Unger and L. Bühmann, J. Lehmann, A.-C. N. Ngomo, D. Gerber, and P. Cimiano, "Template-based question answering over RDF data," in *Pro. WWW*, Lyon, France, Apr. 2012, pp. 639–648.

[19] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, and G. Weikum, "Natural language questions for the web of data," in *Proc. EMNLP-CoNLL*, Jeju Island, South Korea, Jul. 212, pp. 379–390.

[20] S. Reddy, M. Lapata, and M. Steedman, "Large-scale semantic parsing without question-answer pairs," *Trans. Assoc. Comput. Linguistics*, vol. 2, pp. 377–392, Dec. 2014.

[21] H. Bast and E. Haussmann, "More accurate question answering on freebase," in *Proc. CIKM*, Melbourne, VIC, Australia, Oct. 2015, pp. 1431–1440.

[22] A. Abujabal, M. Yahya, M. Riedewald, and G. Weikum, "Automated template generation for question answering over knowledge graphs," in *Proc. WWW*, Perth, WA, Australia, Apr. 2017, pp. 1191–1200.

[23] A. Abujabal, R. S. Roy, M. Yahya, and G. Weikum, "Never-ending learning for open-domain question answering over knowledge bases," in *Proc. WWW*, Lyon, France, Apr. 2018, pp. 1053–1062.

[24] D. Savenkov and E. Agichtein, "When a knowledge base is not enough: Question answering over knowledge bases with external text data," in *Proc. SIGIR*, Pisa, Italy, Jul. 2016, pp. 235–244.

[25] K. Xu, S. Reddy, Y. Feng, S. Huang, and D. Zhao, "Question answering on freebase via relation extraction and textual evidence," 2016, *arXiv:1603.00957*. [Online]. Available:https://arxiv.org/abs/1603.00957

[26] H. Li, C. Xiong, and J. Callan, "Natural language supported relation matching for question answering with knowledge graphs," in *Proc. KG4IR*, Tokyo, Japan, 2017, pp. 43–48.

[27] W. Yih, M. Chang, X. He, and J. Gao, "Semantic parsing via staged query graph generation: Question answering with knowledge base," in *Proc. ACL-IJCNLP*, Beijing, China, 2015, pp. 1321–1331.

[28] Z. Xie, Z. Zeng, G. Zhou, and W. Wang, "Topic enhanced deep structured semantic models for knowledge base question answering," *Sci. China Inf. Sci.*, vol. 60, no. 11, Nov. 2017, Art. no. 110103.

[29] T. Shao, Y. Guo, H. Chen, and Z. Hao, "Transformer-based neural network for answer selection in question answering," *IEEE Access*, vol. 7, pp. 26146–26156, 2019.

[30] W. Shi, W. Zheng, J. Yu, H. Cheng, and L. Zou, "Keyphrase extraction using knowledge graphs," *Data Sci. Eng.*, vol. 2, no. 4, pp. 275–288, Dec. 2017.

[31] G. M. Mazzeo and C. Zaniolo, "Answering Controlled Natural Language Questions on RDF Knowledge Bases," in *Proc. EDBT*, Bordeaux, France, 2016, pp. 608–611.

[32] S. Ferré, "squall2sparql: A translator from controlled english to full SPARQL 1.1," *Workshop of Multilingual Question Answering Over Linked Data (QALD-3)*, Valencia, Spain, Sep. 2013. [Online]. Available: https://hal.inria.fr/hal-00943522
[33] K. Xu, Y. Feng, S. Huang, and D. Zhao, "Question answering via phrasal semantic parsing," in *Proc. CLEF*, Toulouse, France, 2015, pp. 414–426.
[34] S. He, Y. Zhang, K. Liu, and J. Zhao, "CASIA V2: A MLN-based question answering system over linked data," in *Proc. CLEF*, Sheffield, U.K., Sep. 2014, pp. 1249–1259.
[35] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao, "Natural language question answering over RDF: A graph data driven approach," in *Proc. SIGMOD*, Snowbird, Utah, USA, Jun. 2014. pp. 313–324.

HAI JIN received the Ph.D. degree in computer engineering from Huazhong University of Science and Technology, in 1994. He is a Cheung Kung Scholars Chair Professor of Computer Science and Engineering with Huazhong University of Science and Technology. In 1996, he was awarded a German Academic Exchange Service Fellowship to visit the Technical University of Chemnitz in Germany. He worked at The University of Hong Kong from 1998 to 2000, and as a Visiting Scholar at the University of Southern California from 1999 to 2000. He received the Excellent Youth Award from the National Science Foundation of China, in 2001. He is the Chief Scientist of ChinaGrid, the largest grid computing project in China, and also the Chief Scientists of National 973 Basic Research Program Project of Virtualization Technology of Computing System and Cloud Security.

Dr. Jin is a CCF Fellow, and a member of the ACM. He has coauthored 22 books and published over 700 research papers. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

YI LUO received the B.S. degree in material forming and control engineering from Huazhong University of Science and Technology, Wuhan, China, in 2011. He is currently pursuing the Ph.D. degree in computer science and technology with Huazhong University of Science and Technology, Wuhan, China.

Since 2013, he has been a Ph.D. student with Services Computing Technology and System Lab, Wuhan, China. His research interests include question answering, knowledge graph representation, and graph data mining.

CHENJING GAO received the B.S. degree in computer science and technology from Wuhan University of Science and Technology, Wuhan, China, in 2018. She is currently pursuing the M.S. degree in computer technology with Huazhong University of Science and Technology, Wuhan, China.

Since 2018, she has been a M.S. student with Services Computing Technology and System Lab, Wuhan, China. Her research focuses on question answering and knowledge graph representation.

XUNZHU TANG received the B.S. degree in information management and information system from Guangxi University, Nanning, China, in 2016. He is currently pursuing the Ph.D. degree in computer software and theory with Huazhong University of Science and Technology, Wuhan, China.

Since 2016, he has been a Ph.D. student with Services Computing Technology and System Lab, Wuhan, China. His research interest includes question answering and natural language processing.

PINGPENG YUAN received the Ph.D. degree in computer science from Zhejiang University. Currently, he is a Professor with the School of Computer Science and Technology, Huazhong University of Science and Technology. His research interests include databases, knowledge representation, and reasoning and information retrieval, with a focus on high-performance computing. During exploring his research, he implements systems and innovative applications in addition to investigating theoretical solutions and algorithmic design. Thus, he is the Principle Developer in multiple system prototypes, including TripleBit, PathGraph, and SemreX.