Question Answering Over Knowledge Graphs: Question Understanding Via Template Decomposition

Weiguo Zheng[1], Jeffrey Xu Yu[1], Lei Zou[2], Hong Cheng[1]
[1] *The Chinese University of Hong Kong, China;*
[2] *Peking University, China.*
{wgzheng,yu,hcheng}@se.cuhk.edu.hk, zoulei@pku.edu.cn

ABSTRACT

The gap between unstructured natural language and structured data makes it challenging to build a system that supports using natural language to query large knowledge graphs. Many existing methods construct a structured query for the input question based on a syntactic parser. Once the input question is parsed incorrectly, a false structured query will be generated, which may result in false or incomplete answers. The problem gets worse especially for complex questions. In this paper, we propose a novel systematic method to understand natural language questions by using a large number of binary templates rather than semantic parsers. As sufficient templates are critical in the procedure, we present a low-cost approach that can build a huge number of templates automatically. To reduce the search space, we carefully devise an index to facilitate the online template decomposition. Moreover, we design effective strategies to perform the two-level disambiguations (i.e., entity-level ambiguity and structure-level ambiguity) by considering the query semantics. Extensive experiments over several benchmarks demonstrate that our proposed approach is effective as it significantly outperforms state-of-the-art methods in terms of both precision and recall.

PVLDB Reference Format:
Weiguo Zheng, Jeffrey Xu Yu, Lei Zou, and Hong Cheng. Question Answering Over Knowledge Graphs: Question Understanding Via Template Decomposition. *PVLDB*, 11 (11): 1373-1386, 2018.
DOI: https://doi.org/10.14778/3236187.3236192.

1. INTRODUCTION

Using natural language to query knowledge bases has become a research hotspot of database (DB) and natural language processing (NLP) communities [25, 54, 53, 44], since it provides an intuitive and expressive way to explore knowledge bases. As there is a gap between the structured knowledge base and the plain natural language question, most of the existing methods follow the framework that generating a structured query for the input question and then performing the subgraph match over the knowledge graph. If the generated structured query is incorrect, the methods are very likely to find false or incomplete answers. Hence, generating a structured query for the input question is a crucial step.

To generate a structured query, a widely used approach is parsing the input question into the syntactic dependency representation by employing some NLP tools, e.g., Stanford Parser [15]. Based on the parsing result, a query skeleton is constructed [54, 28, 53]. Once the question is parsed incorrectly, it will be difficult to produce the correct structured query. Instead of adopting the existing parsers, the literatures in NLP community mainly focus on learning a grammar that can parse natural language into a sophisticated meaning representation language [5, 44]. However, they suffer from mismatches between grammar of predicted structures and the structure of knowledge bases [27, 6]. Although these methods can deal with most simple questions (the question that only contains one relation) [13], the effect over complex questions (the question that contains multiple relations) is not satisfactory [44]. The main reason is that relations and the corresponding arguments may not be identified correctly.

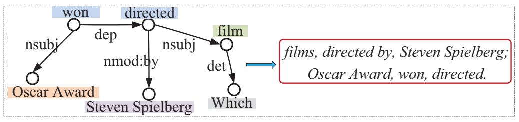


Figure 1: The dependency tree and triples.

1.1 Motivating Example

Given the question "*Which films directed by Steven Spielberg won the Oscar Award*", we obtain the syntactic dependency representation as shown in Figure 1 by using the state-of-the-art Stanford Parser [9]. Obviously, the parsing results are not correct as "won" is assigned two incorrect arguments "Oscar Award" (nsubj) and "directed" (dep), which will lead to incorrect triples as presented in the right part of Figure 1. Hence, the input question cannot be answered by employing the existing Stanford Parser.

Recently, two template-based approaches [1, 12] have been proposed to deal with complex questions. The underlying intuition is that *a complex question consists of multiple simple subquestions*, where each simple subquestion involves only one relation that is mapped to a predicate in the knowledge graph. Abujabal et al. obtain the dependency representation by using the existing syntactic parser first and then perform simple automated rewriting to get two separate interrogative propositions following some predefined rules [1].

Proceedings of the VLDB Endowment, Vol. 11, No. 11
Copyright 2018 VLDB Endowment 2150-8097/18/07.
DOI: https://doi.org/10.14778/3236187.3236192.

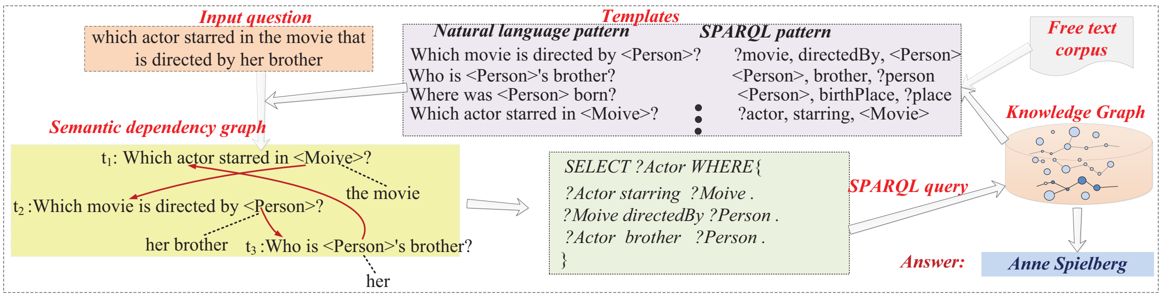

Input question
which actor starred in the movie that is directed by her brother
Templates
Natural language pattern
SPARQL pattern
Which movie is directed by <Person>?
?movie, directedBy, <Person>
Who is <Person>'s brother?
<Person>, brother, ?person
Where was <Person> born?
<Person>, birthPlace, ?place
Which actor starred in <Moive>?
?actor, starring, <Movie>
Free text corpus
Knowledge Graph
Semantic dependency graph
t1: Which actor starred in <Moive>?
the movie
t2 :Which movie is directed by <Person>?
her brother
t3 :Who is <Person>'s brother?
her
SELECT ?Actor WHERE{
?Actor starring ?Moive .
?Moive directedBy ?Person .
?Actor brother ?Person .
}
SPARQL query
Answer:
Anne Spielberg

Figure 2: A running example.

Finally, each subquestion is answered independently and the non-empty intersection of the answer sets is returned as final answers. Clearly, the method has a limited ability to handle complex questions since it highly depends on the syntactic parser and manually defined rewriting rules. Instead of leveraging the dependency representation resulting from syntactic parsers, KBQA decomposes the question q into a sequence of binary factoid question (BFQ) $\{\breve{q}_1, \dots, \breve{q}_k\}$ and then answers each BFQ sequentially [12]. It demands that the answer to $\breve{q}_{i-1}$ is the value of the variable in $\breve{q}_i$ and $\breve{q}_0$ contains an entity that is equal to the entity of q. The definition is too rigorous to capture more general cases. For instance, the question "who acted in Mission Impossible and Vanilla Sky?" can be decomposed into two subquestions: "who acted in Mission Impossible" and "who acted in Vanilla Sky", neither of which does not contain variables. Beyond that, the question "which actor starred in the movie that was directed by her brother?" has no specific entity. Furthermore, these subquestions form a *dependency graph* (the bottom left in Figure 2) ***rather than just a sequence***. Moreover, KBQA learns question answering through a QA corpus that consists of question-answer pairs. However, it is difficult to acquire sufficient high-quality QA pairs.

Motivated by the shortcomings above, we propose a novel approach to answering complex questions, that is, using templates to understand the input questions. Specifically, *we decompose the input question into a set of subquestions via templates, where these subquestions form a dependency graph.* The template used in this paper consists of two parts: the question pattern and the SPARQL query pattern (as shown in the upper middle of Figure 2). We model the task of answering complex questions as a template selection and assembly problem: Given an input question and a set T of simple templates, we choose several templates from T to cover the input question and assemble them into a semantic dependency graph (formally defined in Definition 4).

1.2 Challenges and Contributions

Challenge 1: *Generating sufficient simple templates is the premise for the task of understanding question through templates.* Templates have been confirmed to be effective in answering natural language questions [1, 53] since they bridge the gap between questions and knowledge graphs. However, obtaining sufficient templates, especially complex templates (the templates contain multiple relations) is non-trivial. In general, none of the template-based methods can answer the question if the required templates are missing. In this paper, we adopt simple templates (also called binary templates) and propose a novel approach to build templates by exploiting the knowledge graph and a large-scale text corpus. Since it does not require laborious labelling, the proposed approach is low-cost and feasible compared with the existing question-answer pair based methods, e.g., [12].

Challenge 2: *How to select target templates from millions of templates and reduce the time cost.* Since we use templates to understand the input question q, a critical step is to select a set of templates from the pre-generated templates to match q. Note that these selected templates should form a semantic structure which can describe the question. By integrating the two procedures, we define a notion *semantic dependency graph* that is an intermediate representation between the input question and structured query, and design an effective strategy to decompose q in an iterative manner. Since the number of templates may be too large, to enhance the online user experience, we carefully devise an index, based on which the search space can be reduced significantly.

Challenge 3: *How to deal with the ambiguities during the decomposition of the input question.* To understand an input question, a huge challenge is to deal with the ambiguities generated during the question decomposition and the semantic dependency graph construction. Besides the widely known entity-level ambiguity (that is a mention in the question may be linked to multiple entities in the knowledge graph), we identify the structure-level ambiguity that has received relatively little attention. We devise a systematic method that screens the decompositions by considering query semantics.

Contributions. We make the following contributions.

- We present a systematic framework to understand natural language questions by using templates.
- We propose a novel approach to automatically generate a larger number of templates rather than relying on enormous high-quality labeled data (e.g., pairs of questions and answers) that is expensive to obtain.
- We define an intermediate representation for the natural language question and structured query, and design an efficient method to decompose input questions based on a carefully devised index.
- We propose effective strategies to perform both the entity-level and structure-level disambiguations.
- We evaluate the proposed methods by conducting extensively experimental evaluations on real datasets.

2. PRELIMINARY AND OVERVIEW

In this section, we first formalize the problems studied in this paper. Then we present the overview of our approach.

2.1 Problem Definition

Definition 1. (Knowledge Graph). A knowledge graph is a directed graph $G = \{V, E, L\}$ consisting of a set of vertices V (including entities/types/literals) and a set of edges E that are functionally labeled (predicates/properties) by L.

Each edge in E is called a *triple* $\langle v_1, r, v_2 \rangle$, where $v_1, v_2 \in V$, and r is a predicate/property. If the type of an entity is unknown, we can employ the existing type detection techniques, e.g., [32], to discover it. If an entity has multiple types, we use the one with the largest depth (i.e., the shortest path distance from the root type) in the type ontology graph, where the ontology graph describes the subTypeOf relation among types. The questions are divided into two categories, i.e., "*simple*" and "*complex*" as follows.

Definition 2. (Simple and Complex Questions). A simple question is a question that just contains one fact. A complex question contains more than one fact, where the *fact* is a triple that is not equipped with the predicate "type".

For instance, the question "who is the alumni of Princeton University and Harvard University?" is a complex question since it contains two facts, i.e., ⟨*?p, alumni, Princeton University*⟩, and ⟨*?p, alumni, Harvard University*⟩.

Definition 3. (Binary template). A template t contains two parts of patterns: the natural language question pattern $t.n$ and the SPARQL query pattern $t.s$, where the corresponding entities in $t.n$ and $t.s$ are replaced by their types (denoted by slots), and there are correspondences between the slots in $t.n$ and $t.s$. A binary template refers to the template that contains just one fact.

Some examples of templates are presented in Figure 2, where only the basic graph patterns are retained for ease of presentation. If the slots are replaced with specific entities, we can obtain the *instantiated template* (i.e., a subquestion). We say a system "understands" the question q in the view of G, where "understand" means generating a SPARQL query for q following the terms (including entities, predicates, and literals) used in G. To that end, we propose to construct a semantic dependency graph for q.

Definition 4. (Semantic Dependency Graph). Given an input question q, its semantic dependency graph is a directed graph, denoted as $SDG(q) = \{V_q, E_q\}$, where each vertex $v \in V_q$ represents an instantiated template, and there is an edge from v_1 to v_2 if the answer to v_2 is v_1's value (for filling the slot in v_1) or v_1 and v_2 share the common answer.

To build a system that understands the natural language question, we need to solve two tasks.

Problem Statement 1. *Given a knowledge graph G and a text corpus D, we need to generate binary templates for G.*

The text corpus D is a set of documents that relate to the knowledge graph. Generally, D is easy to obtain. For example, Wikipediais the resource for DBpedia [29].

Problem Statement 2. *Given a knowledge graph G, a set of templates T, and an input question q, the goal is to construct a semantic dependency graph for q.*

After building $SDG(q)$, it is straightforward to generate the SPARQL query for q since each question pattern corresponds to a SPARQL pattern. At last, conducting the SPARQL query over G will return answers to q.

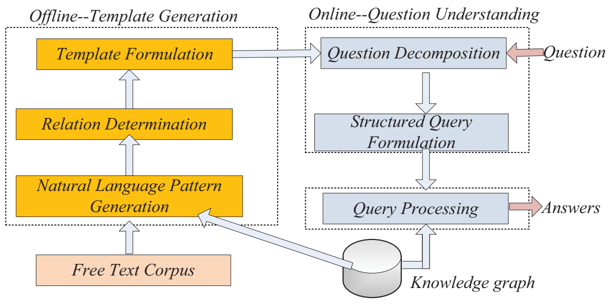


Figure 3: Block diagram of the proposed system.

2.2 Overview of Our Approach

Figure 3 provides an overview of the proposed approach.

2.2.1 *Template Generation*

Due to the flexibility of natural language, a predicate in the knowledge graph can be described using different expressions, which highly affects the question understanding ability. To tackle the problem, we propose a low-cost approach to build templates, which just exploits the knowledge graph G and a large-scale text corpus D. The main idea is to find natural language patterns from D for each triple $e \in G$. There are three steps to generate templates.

Natural language pattern generation. First, we need to find the mentions for each triple $e = \langle v_1, r, v_2 \rangle$ from the text corpus D. If a sentence $s \in D$ contains both v_1 and v_2, s is a candidate mention for e. Then we replace v_1 and v_2 in s with their types to obtain a natural language pattern.

EXAMPLE 1. *For instance, given the triple ⟨Mac OS X, PrecededBy, Mac OS⟩, we can find two sentences containing the two entities: "The original Mac OS was replaced with a completely new Mac OS X." and "Mac OS X succeeded classic Mac OS.". After the entity substitution, we acquire two natural language patterns as follows: The original $\langle OS_1 \rangle$ was replaced with a completely new $\langle OS_2 \rangle$ and $\langle OS_2 \rangle$ succeeded classic $\langle OS_1 \rangle$. Note that OS_1 and OS_2 are both the type OS, where the subscripts are used to distinguish the two entities.*

We also replace entities in the triple e with their types. For example, the triple pattern for the triple in Example 1 is ⟨OS_1, *PrecededBy*, OS_2⟩.

Relation determination. The found natural language pattern p may not correspond to the predicate in triple e. Moreover, the pattern p may be identified through different triple patterns. Therefore, we should map the natural language patterns to the triple patterns. In general, a triple pattern can correspond to multiple natural language patterns, but a natural language pattern only corresponds to a single triple pattern. In order to compute the mapping, we construct a tripartite graph consisting of the supporting instances I, natural language patterns S, and predicates R. Then we can compute the mapping between S and R based on the tripartite graph.

Template formulation. The natural language question patterns generated from the free text corpus may be declarative statements. Thus we need to transform the declarative natural language patterns into that of question form. Then we parameterize the question patterns and triple patterns to get templates of interest.

2.2.2 *Question Decomposition*

Since the templates T bridge the gap between q and G, we propose to construct a semantic dependency graph for q by using these templates. To that end, we employ the templates to decompose q. A straightforward method is to exhaust each subsentence q' of q and compute the similarity between q' and the question pattern of each template. However, it is very time-consuming since the number of templates can be very large. Hence, we propose type-based and order-based techniques that can greatly reduce the search space.

During the process, another challenge is to deal with ambiguities. The ambiguities lie in two aspects, i.e., the entity-level ambiguity and the structure-level ambiguity. Similar to the existing discussion [54, 53], the entity-level ambiguity refers to the case that a phrase may have multiple candidate mappings in the underlying knowledge graph. Using templates can solve the problem easily because slots in templates are constrained with specific types. The structure-level ambiguity refers to that a sentence may be interpreted into more than one structural expression, which is less studied previously. To perform the structure-level disambiguation, we design effective techniques that detect false decompositions by considering query semantics.

2.2.3 *Structured Query Formulation*

After obtaining the set S of decomposing templates, we need to organize them into a semantic dependency graph, which can be easily converted to the final structured query. To build the semantic dependency graph for q, the key task is to link the templates in S to each other if required. A template t_1 can be linked to another template t_2 if the answer to t_1 is the instance of one slot in t_2 and vice versa. For example, for the question "which actor starred in the movie that is directed by her brother", we get three templates as shown in Figure 2. There is an edge between t_1 and t_3 since the answer to t_1 is the value of the slot in t_3. To find such connections, the coreference resolution may be invoked.

3. AUTOMATIC TEMPLATE GENERATION

We propose a novel approach to build templates automatically that directly uses the knowledge graph and text corpus. It comprises three steps to complete template generation.

3.1 Natural Language Pattern Generation

Given the knowledge graph G and a text corpus D, the goal is to generate templates that can capture the mapping relations between the structured G and unstructured natural language questions. Thus the basic idea of our method is to find the mentions for each triple e=$\langle v_1, r, v_2\rangle$, based on which templates are generated.

The text corpus D is better to come from the same source as G. Otherwise, the generated templates may be bad since the entities in G may not be found in D, which will lead to a very limited number of templates. Thus, it may degrade the ability of answering questions. Generally, we can find such a corpus for G since many knowledge graphs are constructed from unstructured data or semi-structured data. Handling the case that there is no such an appropriate corpus for G is beyond the scope of the paper.

For each triple e=$\langle v_1, r, v_2\rangle \in G$, we find all sentences from D that are related to e. A sentence s is said to be *related to* e if s contains the two incident entities v_1 and v_2. Note that the text corpus could be very large. Hence, it

Algorithm 1 *NLPattGeneration*(G, D)

```
Input:    The knowledge graph $G$ and the text corpus $D$;
Output:   Natural language patterns $NLPatt$.
1: for each sentence $s \in D$ do
2:     if $|s| \leq maxLen$ then
3:         $X \leftarrow$ entities in $s$ that are contained in $G$
4:         find the type $c$ from $G$ for each entity $v \in X$
5:         for each two entities $v_1$ and $v_2$ in $X$ do
6:             $s' \leftarrow$ replace $v_1$ in $s$ with $c_1$
7:             $s' \leftarrow$ replace $v_2$ in $s'$ with $c_2$
8:             add $s'$ together with $(v_1, v_2)$ into $NLPatt$
9: return $NLPatt$
```

is very costly to explore the whole corpus D exhaustedly if each time a triple is checked. Thus we propose an efficient way of constructing natural language patterns as depicted in Algorithm 1. For each sentence s in the corpus D, we first identify the set of entities, denoted by X, which can be completed via the existing tools, such as DBpedia Spotlight [14]. Then each two entities v_1 and v_2 are replaced with their types c_1 and c_2, respectively. Finally, we add the sentence pattern s' together with the pair of entities (v_1, v_2) into $NLPatt$. Materializing the entity pair (v_1, v_2) is used to generate candidate relations for the sentence pattern. The materialized entity pair (v_1, v_2) is called a *supporting instance* for the sentence pattern s', denoted by *supp*(s').

The generated natural language patterns above may not be simple (i.e., they may contain multiple relations), which indicates that they may not be employed to decompose the input question. Thus we need to discard some natural language patterns of length larger than a threshold $maxLen$. To determine $maxLen$, we can analyze the simple question patterns that are identified by the existing work, e.g., [1, 12]. We replace entities in each triple with their types to obtain the entity pattern. For instance, the entity pattern for (v_1, v_2) is (c_1, c_2). Then we group the sentence patterns together that share the common entity pattern.

3.2 Relation Determination

After obtaining the natural language patterns, we should map them to the specific predicates (i.e., relations). In general, two entities may have multiple relations in the underlying knowledge graph. For example, Charlie Chaplin is the director, writer, and starring of the movie City Lights. Thus we have the triples as follows: ⟨*City_Lights, director, Charlie_Chaplin*⟩, ⟨*City_Lights, writer, Charlie_Chaplin*⟩*, and* ⟨*City_Lights, starring, Charlie_Chaplin*⟩. Actually, a question pattern corresponds to only one relation at most. In contrast, one relation may have multiple mappings of natural language question patterns due to the variability of natural language. To determine the relation for a natural language pattern, we construct a pattern tripartite graph.

Definition 5. (Pattern Tripartite Graph). A pattern tripartite graph, denoted PTG, consists of three sets of vertices I, S and R, where I represents the set of supporting instances, S represents the set of natural language patterns, and R represents the set of predicates. There is an edge between (v_1, v_2) and $s \in S$ if (v_1, v_2) is a supporting instance for s, and there is an edge between $s \in S$ and $r \in R$ if r is a relation of two entities in a supporting instance *supp*(s) for the natural language pattern s.

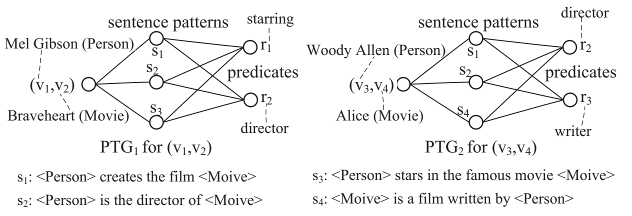


Figure 4: Pattern tripartite graphs.

We can build a PTG for each supporting instance as presented in Figure 4. It is straightforward to filter out some edges between the sentence patterns and predicates based on their semantic similarity, which is defined in Equation (1),

$$sim(s, r) = \max_{ph \in s} sim(ph, r) \tag{1}$$

where ph is a phrase in s and $sim(ph, r)$ is the similarity between a phrase ph and a predicate r. We adopt the widely used semantic similarity measure word2vec [30] in the paper.

Based on the similarity we can refine the tripartite graph. Specifically, if the similarity $sim(s, r)$ is larger than a threshold θ_u, it is convinced that s matches the predicate r. In contrast, if the similarity $sim(s, r)$ is smaller than a threshold θ_l, we can remove the candidate mapping between s and r. Then the remaining mappings with similarity score between θ_l and θ_u should be determined further.

We propose to consider the candidate mappings across different pattern tripartite graphs. Let us recall the pattern tripartite graphs in Figure 4. The two different supporting instances (v_1, v_2) and (v_3, v_4) share two sentence patterns s_1 and s_2. At the same time, they share the predicate r_2. Hence, we can conclude that it is very likely that the patterns s_1 and s_2 match the predicate r_2. To justify the conclusion, we compute the precision of cases linking the shared predicate to the shared sentence patterns for two different supporting instances. Note that we only consider the case that there is just one common predicate between two supporting instances. The precision is 0.87 in the experiment. Based on the basic idea, we can verify the candidate mappings. First, we maintain a counter for each natural language pattern s, denoted as $\mu(s, r)$, which is set to be 0 initially. Since templates contain natural language question patterns in which specific entities are replaced with types, we can group entity pairs according to their types. Let $EP(c_1, c_2)$ denote the set of entity pairs sharing the same types c_1 and c_2. For any two entity pairs in $EP(c_1, c_2)$, we compute the common sentence patterns $CP(c_1, c_2)$ and common relations $CR(c_1, c_2)$. Then we increase $\mu(s, r)$ by 1 if $s \in CP(c_1, c_2)$ and $r \in CR(c_1, c_2)$. Finally, the relation r_i with largest $\mu(s, r_i)$ is selected as the matching relation for the sentence pattern s, i.e., $f(s) = argmax\ \mu(s, r_i)$, where $f(s)$ represents the matching function for s.

3.3 Template Formulation

We need to transform the declarative statements into question form since the sentence patterns acquired from open-domain text corpus are likely to be declarative.

First, we determine the wh-word according to the entity type. If the target type is a descendent of "Person" in the ontology, "who" or "whose" is selected. If the target type is a descendent of "Location" in the ontology, "where" is chosen. If the target type is a time, "when" is used. In the remaining cases, "what" is used. In general, "which" can match any type. For the wh-word 'who", "whose", "where", or "what", we remove the corresponding type. For the wh-word "which", we move the type behind "which". Finally, the auxiliary verb "do", "does", or "can" is inserted into the sentence according to the subject of the sentence.

Algorithm 2 $AnswerSimpleQuestions(q, G, T)$

```
Input: Input question $q$, knowledge graph $G$, templates $T$;
Output: Answers to $q$.
1: identify entities in $q$
2: $q' \leftarrow$ replace entities in $q$ with their types
3: for each template $t \in T$ do
4:     if there is bijection between $q'.C$ and $t.n.C$ then
5:         compute the similarity between $q'$ and $t.n$
6: $t_i \leftarrow$ the template with largest similarity score
7: $\lambda(q) \leftarrow$ instantiate $t_i.s$
8: $\mathbb{A} \leftarrow$ execute the SPARQL query $\lambda(q)$
9: return $\mathbb{A}$
```

Example 2. *Let us consider the pattern "The $\$v_1\langle River\rangle$ starts from the $\$v_2\langle Country\rangle$". Assume that the target entity is "Country", where $\$v_1$ has two parameters to be instantiated based on the input question. We can generate two question patterns as follows: "which $\langle Country\rangle$ does the $\$v_1\langle River\rangle$ start from?" and "where does the $\$v_1\langle River\rangle$ start from?".*

It is straightforward to generate general questions as no wh-word is required to be introduced. Notice that the question templates are not necessary to follow the syntax strictly since we will consider the similarity between the question pattern of a template and the input question q. Moreover, we need to generate SPARQL patterns based on the corresponding triple patterns. Each triple pattern $\langle c_1, r, c_2\rangle$ actually corresponds to three triples, two of which are the type constraints, i.e., $\langle \$v_1, type, c_1\rangle$ and $\langle \$v_2, type, c_2\rangle$. In the last triple, we set a variable that corresponds to the target entity in the question pattern.

Example 3. *Following Example 2 above, since the target entity is v_2 the corresponding SPARQL query pattern is:*
SELECT $?v_2$ WHERE
$\{\$v_1, type, River.\ ?v_2, type, Country.\ \$v_1, sourceCountry, ?v_2.\}$

4. ONLINE QUESTION DECOMPOSITION

In the online phase, we need to build a structured query, i.e., SPARQL query, for the input question q. Moreover, the SPARQL query should comply with the schema and vocabulary that are used in G. In real scenario, q may be a simple question or a complex question, which will be described in Subsections 4.1 and 4.2, respectively.

4.1 Answering Simple Questions

If q is a simple question that only contains one relation, it is easy to deal with. We can select a template from T to match q. Algorithm 2 outlines the process.

We first identify the entities in q and replace the entities with their types to obtain q' (lines 1-2 in Algorithm 2). A template $t \in T$ is a candidate for q only if there is a bijection between $q'.C$ and $t.n.C$, where $q'.C$ is the set of entity types in q' and $t.n.C$ is the set of entity types in the natural language pattern $t.n$. If there is no such a bijection, the SPARQL pattern $t.s$ cannot be instantiated properly. For

Algorithm 3 *Base-QD*(q, T, τ)

```
Input: The input question $q = \{w_1, \cdots, w_{|q|}\}$, templates $T$, and the similarity threshold $\tau$;
Output: The decomposed subquestions $D(q)$ and the set of templates $T(q)$ matching $q$.
1: $q_0 \leftarrow$ replace entities in $q$ with the corresponding types
2: for $i$ from 1 to $|q|$ do
3:     for $j$ from $i$ to $|q|$ do
4:         $q' \leftarrow$ the substring $\{w_i, \cdots, w_j\}$ of $q_0$
5:         for each template $t \in T$ do
6:             if a bijection between $q'.C$ and $t.n.C$ exists then
7:                 $\rho(q', t.n) \leftarrow$ the similarity between $q'$ and $t.n$
8:         $t' \leftarrow$ the template with the largest $\rho(q', t.n)$
9:         if $\rho(q', t'.n) \geq \tau$ then
10:            $q'_0 \leftarrow$ replace $q'$ in $q_0$ with the answer type of $t'$
11:            if $|q'_0| = 1$ or Base-QD$(q'_0, T, \tau) \neq NULL$ then
12:                $D(q) \leftarrow D(q) \cup q'$, $T(q) \leftarrow T(q) \cup t'$
13:                return $D(q)$ and $T(q)$
14: return $NULL$
```

each candidate template t we compute the similarity between q' and $t.n$. In this paper, we shall exploit the widely used Jaccard similarity coefficient since it is highly related to several other similarity measures, e.g., string edit distance [50], Dice similarity [18], and Overlap similarity [41], as discussed in the work [41].

The template with the largest similarity score is selected, denoted by t_i (lines 3-6). Then we instantiate the SPARQL query pattern $t.s$ to obtain the SPARQL query $\lambda(q)$ by using the entities found through q. Finally, executing $\lambda(q)$ over G can lead to the answers to q (lines 7-9).

4.2 Answering Complex Questions

As shown in Definition 2, a complex question contains more than one fact, which can be classified into two categories: plain complex questions and "And-questions". The "And-question" is a complex question that contains a conjunction, such as "and", "or", and "but". The rest complex questions are called plain complex questions. For example, the question "Who is married to an actor and starred in Philadelphia ?" is an "And-question".

Unlike answering simple questions, it is unfeasible to select just one single template to match the input question q since there are multiple relations in q. The underlying principle is that *a complex question can be decomposed into multiple simple questions*. Hence, we need to retrieve a set S of templates from T to cover the semantics of q such that the templates in S form a semantic dependency graph. We first discuss how to deal with plain complex questions in this section, and then handle the And-questions in Section 5.2.

4.2.1 Question Decomposition

Our task is to break the complex question q into constituent parts based on the pre-generated templates T. Algorithm 3 gives a straightforward solution. The basic idea is to try each subsentence of q and compute its similarity between the question pattern of each template.

We replace the entity in q with the corresponding type if any in line 1. If q does not contain any entity, we just identify the type involved in q. Similar to that in Section 4.1, a template t can match q' only if there is a bijection between

Algorithm 4 *TypeQD-SubquestionGen*(i, j, q, G)

```
Input: The starting position $i$, ending position $j$, input question $q = \{w_1, \cdots, w_{|q|}\}$ and knowledge graph $G$;
Output: The candidate subquestion $q_0$.
1: $q_0 \leftarrow w_i, \cdots, w_j$
2: if $q_0$ contains pronouns then
3:     $q_0 \leftarrow$ perform coreference resolution
4: if $i \neq 1$ then
5:     $q_0 \leftarrow$ introduce wh-words
6: return $q_0$
```

$q'.C$ and $t.n.C$ (lines 6-7). Then the template t' with the largest similarity score to q' is selected (line 8). If it holds that (1) $\rho(q', t'.n) \geq \tau$, and (2) q'_0 is decomposed correctly or q'_0 contains just an answer type, i.e., $|q'_0| = 1$, the decomposed subquestions and templates can be returned (lines 9-13).

Example 4. *Let us consider the question "where was the wife of the US president born?". Although the substring "where was the wife" may match a template, there will be no templates that can match the remaining substring. When the substring "the US president" is taken as q', it matches the template "Who is the $\$\langle$Country$\rangle$ president?". Then q'_0 is "where was the wife of $\langle$Person$\rangle$ born". Recursively, we can obtain the decomposed subquestions as shown in Figure 6.*

Time complexity. Because Algorithm 3 iteratively exhausts all possible subsentences of q and considers the whole set of templates sequentially, the time complexity is $O(|q|^2 \cdot 2^{|q|} \cdot |T| \cdot \gamma)$ in the worst case, where γ is the time cost of computing the similarity between q'_0 and $t.n$. Since $|q|$ is very small and $|T|$ may be very large, $|T|$ dominates the time complexity and Algorithm 3 is not efficient. To improve the online performance, we propose an efficient method next.

4.2.2 Type-based Question Decomposition

The algorithm above considers all subsequences and templates, where the search space can be reduced greatly by employing the entity types.

Subquestion generation. Since there must be a bijection between the types of a subquestion and a template, we can identify the entities in q first and find their corresponding types, which can help determine the subquestions and compute the candidate templates. Specifically, each subquestion should contain one type at least. Instead of sequentially enumerating all possible subsequences, we can obtain a more promising subquestion $q_0 = w_i, \cdots, c_j$ immediately by splitting the input question q at the type c_j (c_j is the nearest type to w_i and the position of c_j is larger than that of w_i) as shown in Algorithm 4 and lines 1-4 of Algorithm 5, where $pos(x, w)$ represents the position of word w in the record x. It is common that a complex question may contain pronouns, such as "it", "they", "he", "she", "that", "who", and "which". In this case, we should determine which entity the pronoun refers to, which is a well-known problem coreference resolution in NLP community. There have been lots of efforts devoted to the problem [34, 22, 10].

Candidate template computation. As the types in a template should match those in the subquestion, we build an inverted index $\mathbb{I}(c_i)$ for these templates. $\mathbb{I}(c_i)$ is a data structure that maps a type c_i to a list of templates such that the corresponding question pattern contains c_i. Accordingly, $\mathbb{I}(c_i)[j]$ indicates the j-th entry in the inverted index

Algorithm 5 *TypeQD*(q, G, τ)

```
Input: The input question $q = \{w_1, \cdots, w_{|q|}\}$, templates $T$, knowledge graph $G$, and the similarity threshold $\tau$;
Output: The decomposed subquestions $D(q)$ and the set of templates $T(q)$ matching $q$.
1: $\delta(q) \leftarrow$ entity types in $q$
2: for $i$ from 1 to $pos(q, c_{|\delta(q)|})$ do
3:     $c_j \leftarrow$ the first type following $w_i$
4:     $q_0 \leftarrow$ TypeQD-SubquestionGen$(i, pos(q, c_j), q, G)$
5:     for $k$ from $pos(q, c_j)$ to $|q|$ do
6:         if $k \neq pos(q, c_j)$ then
7:             $q_0 \leftarrow q_0 + w_k$
8:         $cand(q_0) \leftarrow \bigcap_{c_i \in q_0} \mathbb{I}(c_i)$
9:         $MaxJ \leftarrow \tau$, $t_0 \leftarrow null$
10:        for each $t \in cand(q_0)$ do
11:            $J(q_0, t.n) \leftarrow$ the similarity between $q_0$ and $t.n$
12:            if $J(q_0, t.n) \geq MaxJ$ then
13:                $MaxJ \leftarrow J(q_0, t.n)$, $t_0 \leftarrow t$
14:        if $t_0 \neq null$ then
15:            $q' \leftarrow$ replace $q_0$ in $q$ with the answer type of $t_0$
16:            if $|q'| = 1$ or Type-QD$(q', G, \tau) \neq NULL$ then
17:                $D(q) \leftarrow D(q) \cup q_0$, $T(q) \leftarrow T(q) \cup t_0$
18:                return $D(q)$ and $T(q)$
19: return $NULL$
```

of type t_i. Given the subquestion q_0 that has been conceptualized (i.e., entities are replaced with types), we can retrieve the candidates by the types in q_0. Specifically, the intersection of $\mathbb{I}(c_i)$ for all types in q_0 form the candidate templates $cand(q_0)$ of q_0 as presented in line 8 of Algorithm 5.

Template refinement. Each candidate template t in $cand(q_0)$ should be verified by computing the similarity between q_0 and $t.n$. Then the template t_0 with largest similarity is selected. If the corresponding similarity score is larger than the threshold and the newly generated subquestion q' is valid, t_0 is the returned as the matching template for q_0 (lines 9-18). Otherwise, we need to extend q_0 by including more words as depicted in line 7 of Algorithm 5, where "+" means extending q_0 by concatenating the word w_k.

Time complexity. As shown in Algorithm 5, it may exhaust the input question and compute the similarity for each candidate templates, the time complexity is $O(|q|^2 \cdot 2^{|q|} \cdot |\mathbb{I}(c)_{max}| \cdot \gamma)$, where $|\mathbb{I}(c)_{max}|$ denotes the largest index size for each type c. Note that $|\mathbb{I}(c)_{max}|$ is smaller than $|T|$ greatly.

4.2.3 Order-based Optimization

Clearly, the subquestion generation and candidate template computation are highly inter-related to each other, and both of them can be improved by considering the other procedure. As the Jaccard similarity is not sensitive to the order of words in each record. We can reorder the words in each natural language question pattern $t.n$ according to a global ordering $\mathcal{O}$. Let $w_1[i]$ and $w_2[i]$ denote the first pair of different characters of two words w_1 and w_2. If $w_1[i]$ is smaller than $w_2[i]$ ($i \leq min\{|w_1|, |w_2|\}$) in the alphabetical order or w_1 is a substring of w_2 such that $w_1[j] = w_2[j]$ for all j ($j \leq min\{|w_1|, |w_2|\}$) , w_1 is smaller than w_2. Let $\mathcal{O}(w)$ denote the global order of word w.

Thus we design another index that considers the word order, denoted by $\mathbb{IO}$, which consists of two layers. The first layer $\mathbb{IO}(c, *)$ maps a type c to a set of candidate templates. The other layer $\mathbb{IO}(c, w)$ maps a word w to a group of lists $\{\mathbb{L}_{k_1}, \mathbb{L}_{k_2}, \cdots, \mathbb{L}_{k_m}\}$ such that 1) The natural question pattern of each template in $\mathbb{L}_{k_i}$ contains the word w; 2) The natural question patterns in $\mathbb{L}_{k_i}$ share the same number of words, i.e., k_i; 3) The templates in $\mathbb{L}_{k_i}$ are sorted by the increasing order of the position of w in $t.n$.

We also reorder the words in subquestion q_0. Figure 5(b) presents the second layer of the index structure $\mathbb{IO}$ for the input records in Figure 5(a), where the first number in each cell is k_i. By applying the index, we can reduce the search space. Let us consider the following example first.

EXAMPLE 5. *Let us consider the query $q_0 = [F, H, J, M]$, and the similarity threshold $\tau = 0.85$. By using the first word F, we can locate the group of lists $\mathbb{IO}(F)$. Assume that we consider the list of candidates of size 5, i.e., $\{t_4, t_5, t_2, t_1\}$. Currently, we can only see the first word F in $q_0 = [F, ?, ?, ?]$. In the template $t_4 = [?, F, ?, ?, ?]$, there is no common words before F, which indicates that the maximum number common words between q_0 and t_4 is 4. Then the maximum possible similarity of the pair $\langle q_0, t_4 \rangle$ is at most $\dfrac{4}{4+5-4} = 0.8$. Therefore, this pair cannot meet the similarity threshold. More importantly, we do not need to consider all the remaining candidates in the list since their common words between q_0 will not be more than 4.*

The example above indicates that we may not need to exhaust the candidates sequentially. We formally state the pruning principle in the following theorem.

LEMMA 1. *Given a query q_0 and the list $\mathbb{L}_{k_i}$ of candidate templates of size k_i located by the word w in q_0. If the upper bound $J_u(q_0, t_0)$ (as computed in Equation 2) of $J(q_0, t_0)$ is less than the threshold τ, where $t_0 \in \mathbb{L}_{k_i}$, the remaining candidates after t_0 in the list $\mathbb{L}_{k_i}$ can be filtered out safely.*

$$J_u(q_0, t_0) = \frac{\sigma_0 + 1}{|q_0| + |t_0| - \sigma_0 - 1} \tag{2}$$

where $\sigma_0 = \min\{|q_0| - pos(q_0, w), |t_0| - pos(t_0, w)\}$ and $pos(x, w)$ represents the position of word w in the record x.

PROOF. Let t_1 be a candidate that is after t_0 in the list $\mathbb{L}_{k_i}$. 1) Assume that $|q_0| - pos(q_0, w) \geq |t_0| - pos(t_0, w)$. Thus $\sigma_0 = |t_0| - pos(t_0, w)$. Since $pos(t_0, w) \leq pos(t_1, w)$ and $|t_0| = |t_1|$, $\sigma_1 = |t_1| - pos(t_1, w) \leq \sigma_0$. Therefore, we have $J_u(q_0, t_1) = \dfrac{\sigma_1 + 1}{|q_0| + |t_1| - \sigma_1 - 1} \leq J_u(q_0, t_0)$. 2) Assume that $|q_0| - pos(q_0, w) \leq |t_1| - pos(t_1, w)$. Then $\sigma_0 = \sigma_1$. So it holds that $J_u(q_0, t_0) = J_u(q_0, t_1)$. 3) Provided that $|t_1| - pos(t_1, w) < |q_0| - pos(q_0, w) < |t_0| - pos(t_0, w)$.

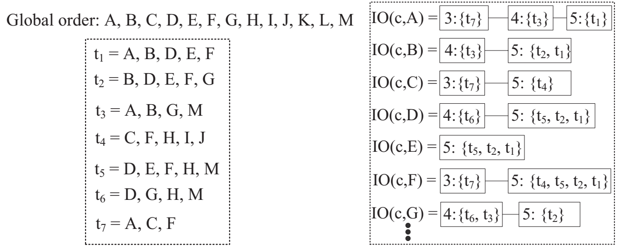

Global order: A, B, C, D, E, F, G, H, I, J, K, L, M
t1 = A, B, D, E, F
t2 = B, D, E, F, G
t3 = A, B, G, M
t4 = C, F, H, I, J
t5 = D, E, F, H, M
t6 = D, G, H, M
t7 = A, C, F
IO(c,A) = 3:{t7} 4:{t3} 5:{t1}
IO(c,B) = 4:{t3} 5: {t2, t1}
IO(c,C) = 3:{t7} 5: {t4}
IO(c,D) = 4:{t6} 5: {t5, t2, t1}
IO(c,E) = 5: {t5, t2, t1}
IO(c,F) = 3:{t7} 5: {t4, t5, t2, t1}
IO(c,G) = 4:{t6, t3} 5: {t2}
(a) Templates containing type c (b) $\mathbb{IO}(c, *)$

Figure 5: Example of index structure

Then we have $\sigma_0 > \sigma_1$. $J_u(q_0, t_0) = \frac{\sigma_0 + 1}{|q_0| + |t_0| - \sigma_0 - 1}$. $J_u(q_0, t_1) = \frac{\sigma_1 + 1}{|q_0| + |t_1| - \sigma_1 - 1}$. $J_u(q_0, t_0) - J_u(q_0, t_1) = \frac{(|q_0| + |t_0|) \cdot (\sigma_0 - \sigma_1)}{(|q_0| + |t_0| - \sigma_0 - 1) \cdot (|q_0| + |t_1| - \sigma_1 - 1)} > 0$. In summary, the lemma holds. □

Note that the threshold τ is updated when there is a new candidate template, whose similarity is larger than τ, is found. The candidate template computation can facilitate the subquestion generation. Note that we need to extend the subquestion q_0 if no templates can match it as shown in Algorithm 5. Based on the upper bound above, we can determine whether the newly extended subquestion is valid instead of performing the candidate template computation and template refinement.

Let $J_u(q_0, t_*)$ denote the maximum upper bound derived by each word in q_0. If $J_u(q_0, t_*) < \tau$, we have the following lemma to reduce the search space.

LEMMA 2. *If $J_u(q_0, t_*) < \tau$ and $\mathcal{O}(w') > \mathcal{O}(w_{|q_0|})$, the newly generated subquestion $q'_0 = q_0 + w'$ will not be valid, where w' is the next word to add into q_0 and $w_{|q_0|}$ is the last word in q_0.*

PROOF. Since$\mathcal{O}(w') > \mathcal{O}(w_{|q_0|})$, we have $\sigma_* \geq \sigma' = \min\{|q_0| + 1 - pos(q_0, w'), |t_*| - pos(t_*, w')\}$. $J_u(q_0, t_*) - J_u(q', t_*)$
$= \frac{\sigma_* + 1}{|q_0| + |t_*| - \sigma_* - 1} - \frac{\sigma' + 1}{|q_0| + 1 + |t_*| - \sigma' - 1}$
$= \frac{(|q_0| + t_*) \cdot (\sigma_* - \sigma') + \sigma_* + 1}{(|q_0| + 1 + |t_*| - \sigma' - 1) \cdot (|q_0| + |t_*| - \sigma')} > 0$. Hence, the newly generated subquestion q' is not valid. □

By integrating the two pruning techniques into Algorithm 5, the search space will be reduced greatly. Specifically, (1) instead of exploring each candidate template as shown in lines 10-11, we locate the candidate templates by each word in q_0, and compute $J_u(q_0, t_0)$ based on Equation 2. If $J_u(q_0, t_0) < \tau$, the remaining templates in $\mathbb{L}_{k_i}$ will be pruned. (2) Before extending the subquestion q_0 as shown in line 7, $J_u(q_0, t_*)$ is computed. If $J_u(q_0, t_*) < \tau$ and $\mathcal{O}(w_k) > \mathcal{O}(w_{|q_0|})$, we can skip the subquestion $q'_0 = q_0 + w_k$.

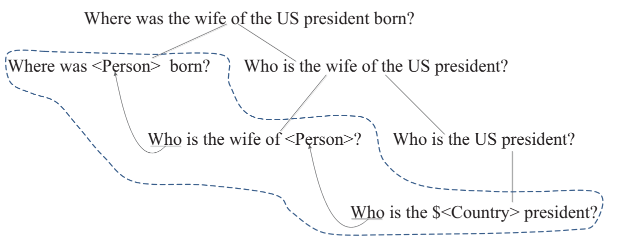
5. STRUCTURED QUERY FORMULATION

To answer the questions, we need to build a semantic dependency graph $SDG(q)$ for the input question q (Section 5.1), based on which a structured query is easy to be generated. Then we deal with the ambiguities involved in the construction of $SDG(q)$ in Section 5.2.

5.1 Semantic Dependency Graph Construction

After acquiring decomposed subquestions and the corresponding templates, we can build the $SDG(q)$ for q, which contains two steps, making connections (i.e., adding edges) between two templates and filling the slots in templates.

Based on Definition 4, there is a type constraint between two joint nodes. Specifically, the answer type of v_2 should be identical to the type of the slot in v_1, or v_1 share the answer as v_2. Note that this constraint has been guaranteed by the decomposition algorithm as it adds the answer type into the similarity computation when a template is identified. That is, we can obtain the templates as well as the dependency relations among the templates. Figure 6 gives the decomposed results for the input question "where was the wife of the US president born".



Where was the wife of the US president born?
Where was <Person> born?
Who is the wife of the US president?
Who is the wife of <Person>?
Who is the US president?
Who is the $<Country> president?

Figure 6: Decomposed subquestions and templates.

Then we need to instantiate the templates with entities in the corresponding decomposed subquestions. Notice that the input question is not necessary to contain an entity. Then a semantic dependency graph is constructed. For example, the structure enclosed by the dashed line in Figure 6 forms a semantic dependency graph.

SPARQL query construction. With $SDG(q)$, it is easy to create the structured query. Specifically, we can generate a triple for each instantiated template based on the correspondences between each natural language question pattern and its corresponding SPARQL query pattern. The dependency relation in a semantic dependency graph means that the two incident templates share the same variable or entity. As the subquestions and templates are computed in the bottom up manner, the variable of the last template is taken as the variable of the whole SPARQL query.

EXAMPLE 6. *The generated SPARQL query for the semantic dependency graph in Figure 6 is as follows.*
SELECT ?place WHERE
$\{?person_1, birthPlace, ?place.$
$?person_1, spouse, ?person_2.$
$United\_States, president, ?person_2.\}$

Finally, conducting the generated SPARQL query over the knolwedge graph can lead to answers to the input question.

5.2 Two-level Ambiguities

During the construction of semantic dependency graphs, a big challenge is to solve the ambiguities in the question decomposition. Generally, there are two kinds of ambiguities, i.e., entity-level ambiguity and structure-level ambiguity.

Rule-based Disambiguation. Entity-level ambiguity is a well-known problem, that is, a mention in the question may correspond to multiple distinct entities in the underlying knowledge graph. For example, in the question "who wrote the House of Flying Daggers" the entity "House of Flying Daggers" may map to a movie or music. Beyond that, the ambiguity may arise in the process of coreference resolution. There have been lots of efforts devoted to solve the entity-level ambiguity these years [24, 11, 45, 33].

Besides entity-level ambiguities, we identify structure-level ambiguities, which refers to the case that a question may be decomposed into different sets of subquestions. Let us consider the example next.

EXAMPLE 7. *For the input question "which actor starred in the movie that wins Oscar Award?", we can obtain two sets of decomposed subquestions: D_1 ={which actor starred in ⟨Movie⟩?, which ⟨Movie⟩ wins ⟨Award⟩?}. D_2 ={which actor starred in ⟨Movie⟩? which actor wins ⟨Award⟩? }.*

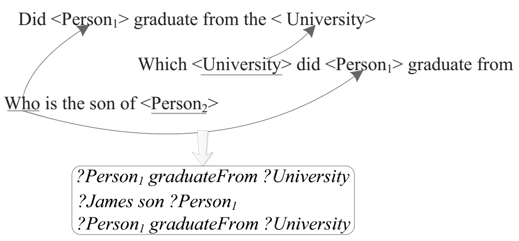

Did <Person1> graduate from the < University>
Which <University> did <Person1> graduate from
Who is the son of <Person2>
?Person1 graduateFrom ?University
?James son ?Person1
?Person1 graduateFrom ?University

Figure 7: Illustration of logic-based disambiguation.

Both the two sets of decompositions in Example 7 satisfy the type constraint, which indicates that there are two possible semantic dependency graphs. In the first decomposition D_1, the word "that" refers to "movie". Meanwhile, in the second decomposition D_2, the word "that" refers to the whole subquestion "which actor starred in $\langle Movie \rangle$?". Since the question is a "that clause", "that" should refers to an entity or a type but not a subquestion. The similar rule is also applicable to the "who clause" and "which clause".

Logic-based Disambiguation. In some cases, the pronouns (e.g., "that, which, who", and "he") may correspond to multiple entities or types. Thus the rule above does not work. So we propose the logic-based disambiguation. The main principle is to find some logical problems in $SDG(q)$.

Example 8. *Let us consider the question "Did the son of James go to the university that he graduated from?", where "he" may refer to "son of James" or "James". The set of decomposed templates is $\{$Did $\langle Person \rangle$ graduate from the $\langle University \rangle$? Who is the son of $\langle Person \rangle$?, Which $\langle University \rangle$ did $\langle Person \rangle$ graduate from?$\}$. Figure 7 presents the semantic dependency graph for case that "he" refers to "son of James", based on which the generated SPARQL triples are listed in the bottom of the figure.*

We identify two logic problems, i.e., the *duplicate statement* and *weak semantic relevance*, as defined next.

Definition 6. (Duplicate Statement). If two triples are identical to each other in the generated SPARQL query, they are called duplicate statements.

Definition 7. (Weak Semantic Relevance). Weak semantic relevance means that the triples are not correlated, i.e., the generated SPARQL query is not a connected graph.

The two problems above betray logic and common sense. If one of them occurs in a SPARQL query, the corresponding set of decomposed templates will be incorrect. As shown in Example 8, the first triple is identical to the third one, i.e., the duplicate statements. Hence, "he" refers to "James".

Answering And-Questions. To answer And-questions, a key task is to perform completion, that is, to add the omitted components explicitly. Generally, the conjunction (e.g., "and" and "or") connects two or more components with similar semantic roles. Based on the nature of conjunction, there are two cases of completions, adding entities (i.e., subjects or objects) and adding relations (i.e., predicates).

Example 9. *Given the question "who is married to an actor and starred in Philadelphia", we find that the subquestion "starred in Philadelphia" lacks a subject. However, there are two possible subjects, "who" and "actor", in the question. By considering the conjunction nature, "starred in" has the similar semantic role to "is married to". Thus the omitted subject should be "who".*

As shown in the example above, to distinguish the ambiguous completions for a target component s_1, we first discover the component s_2 that shares the same semantic role to target component. Then we add the entities or relations (that are omitted in s_1) in s_2 to s_1. Finally, a SPARQL query can be constructed.

Note that in some cases, the input question is ambiguous inherently such that persons may have different understandings. For instance, in the question "*who is married to an actor that starred in $Movie_1$ and directed $Movie_2$*", it is hard to determine the subject for the subquestion "directed $Movie_2$", i.e., the subject may be "who" or "actor". If the ambiguous decompositions pass all the disambiguation techniques above, the final barrier is to perform the data-driven disambiguation. Specifically, we conduct each SPARQL query over the knowledge graph, if some answers are returned, the corresponding SPARQL query is correct. That is, we distinguish the queries according to the underlying knowledge graph, which is similar to the technique used in the work [54].

Remark. Several NLP tasks, such as entity linking, phrase similarity, and coreference resolution, are invoked in the process of template generation and question answering. Each of the tasks may affect the ability of answering questions. It is better to use the solutions that convey semantics and can be incorporated into techniques proposed in this paper. For instance, as a template contains type constraints on entities, if an entity linking solution can be integrated into the template decomposition so that they can reinforce each other, the overall performance may be improved.

6. EXPERIMENTAL STUDY

We evaluate the proposed method via empirical study in this section. Section 6.1 describes the experimental setup, followed by evaluation of template construction in Section 6.2. Section 6.3 and Section 6.4 report the effectiveness and efficiency results, respectively.

6.1 Experimental Setup

6.1.1 Datasets

Knowledge Graphs. We use two well-known public knowledge graphs DBpedia [29] and Freebase [7] in the experiments. DBpedia is an open-domain knowledge base, which contains about 21 million vertices (including 6.6 million entities, 0.38 million concepts, and 14.2 million literals) and 155 million edges (i.e., triples). Freebase contains 116 million entities and 2.9 billion triples.

Free text corpus. We use the free online encyclopedia Wikipedia as the text corpus, based on which we can build templates automatically.

QA datasets. We use the datasets QALD-5 [39], WebQuestions [5], and ComplexQuestions [1] in the experiments.

- **QALD**. QALD is a benchmark delivered for the evaluation campaign answering over knowledge bases. It provides a set of a natural language questions, the corresponding SPARQL queries and answers. QALD-5 contains 468 questions including 418 training questions and 50 test questions.

Table 1: Effect of similarity threshold θ_u

	DBpedia			Freebase		
θ_u	FR	precision	$\|T\|$ (M)	FR	precision	$\|T\|$ (M)
0.6	0.53	53.23%	4.9	0.42	37.28%	7.6
0.7	0.47	64.37%	4.5	0.37	53.54%	7.3
0.8	0.35	79.53%	4.1	0.29	72.36%	6.8
0.9	0.21	93.78%	3.8	0.17	91.65%	6.5
1.0	0.08	91.36%	3.8	0.06	88.81%	6.3

Table 2: Effect of similarity threshold θ_l

	DBpedia			Freebase		
θ_l	FR	precision	$\|T\|$ (M)	FR	precision	$\|T\|$ (M)
0.1	0.19	78.92%	4.3	0.21	75.29%	7.8
0.2	0.23	81.67%	4.1	0.24	79.56%	7.4
0.3	0.29	84.67%	3.9	0.31	82.19%	7.1
0.4	0.35	93.78%	3.8	0.37	91.65%	6.5
0.5	0.41	94.24%	2.9	0.48	92.33%	5.4

- **WebQuestions**. It is an open dataset created by Berant et al. There are 5,810 natural language questions.
- **ComplexQuestions**. It is constructed by Abujabal et al. to evaluate the answering ability on complex questions [1]. It consists 150 compositional questions, e.g., "who was married to billy thornton and jonny miller?".

6.1.2 Metrics and Competitors

Let $\#QC$ denote the number of questions that are correctly answered. Let $\#QA$ denote the total of number input questions. Let $\#QF$ denote the number of questions that are fed to a system. Actually, a system may only return a partially correct answers to a given question. In this case, we say the question is partially correctly answered, the number of which is denoted by $\#QP$. Then we define the *precision* P, *partial precision* P^*, *recall* R, *partial recall* R^*, and F1 measure as follows.

$$P = \frac{\#QC}{\#QF}; \quad P^* = \frac{\#QC + \#QP}{\#QF}; \quad R = \frac{\#QC}{\#QA};$$
$$R^* = \frac{\#QC + \#QP}{\#QA}; \quad F1 = \frac{2}{1/P + 1/R}.$$

Beyond that, since QALD-5 provides SPARQL queries for the questions, we are interested in another metric, the ratio of correct SPARQL query patterns (i.e., ignoring the specific entities) that are generated by the method, denoted by $P^\sharp$.

To confirm the effectiveness, we compare our method, denoted by *TemplateQA*, with the existing methods Xser [43], KBQA [12], QUINT [1], UncertainTQA [53], QAnswer [36], DEANNA [45], APEQ, SemGraphQA [4], and YodaQA.

The experiments are conducted on an Intel(R) Xeon(R) CPU E5504 @ 2.00GHz and 64G RAM, on Windows Server 2008. The programs for building templates were implemented in Python. Other programs were implemented in C++.

6.2 Template Evaluation

By using the Wikipedia documents and two knowledge graphs, we construct 3.8 million and 6.5 million templates under the default settings for DBpedia and Freebase, respectively. Figure 8 presents a small partial set of the templates that are generated for DBpedia.

To evaluate the quality of these templates quantitatively, we recruit five students to rate the templates. If the natural language question pattern describes the corresponding relation (i.e., predicate) in the SPARQL query pattern, the template is regarded as "true". Otherwise, the template is "false". Each time we randomly sample 200 templates for each student. Tables 1 and 2 give the results by varying the similarity thresholds θ_u and θ_l, respectively. The default settings for θ_u and θ_l are 0.9 and 0.4. FR is ratio of the pairs of question patterns and relations that can be determined as "match" (for θ_u) or "unmatched" (for θ_l).

First, we fix θ_l and vary θ_u from 0.6 to 1.0. As shown in Table 1, the ratio FR decreases as we increase θ_u, which indicates that only a small fraction of the candidates have high similarity scores. Interestingly, we find that it achieves the best performance when θ_u is around 0.9. If θ_u is too large, it will impose heavy burdens on the module that deals with middle-grade candidates (i.e., the candidates whose similarity score lies between θ_l and θ_u). At the same time, the number of returned templates $|T|$ decreases accordingly.

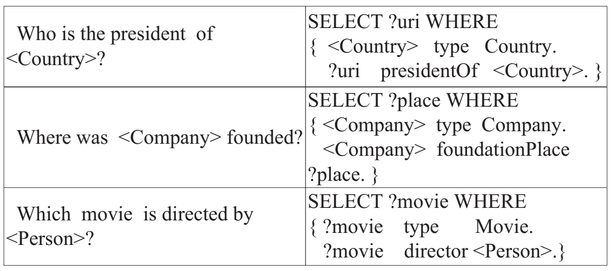
Then we fix θ_u and vary θ_l from 0.1 to 0.5. The precision grows with the increasing of θ_l. That is because more low-quality candidates are screened out by a larger θ_l. Meanwhile, some promising candidates are discarded as well, which leads to the decreasing of the returned templates.

Actually, in order to evaluate the templates, another straightforward way is to answer natural language questions by using these templates. The details are reported in Section 6.3.

6.3 Effectiveness Evaluation

Results on QALD-5. To evaluate the effectiveness, we compare TempalteQA with the existing methods. Table 3 gives the results on QALD-5. For the competitors, we directly report their results in their papers. The recently proposed method can only answer a few answers as it focuses on the binary factoid questions (i.e., the simple questions). However, most questions in reality are complex questions. It is clear our method TempalteQA beats all the competitors in terms of both precision and recall, which indicates that our method is effective and the generated templates have high quality. Note that $\#QF$ equals $\#QA$, i.e., $\#QF = \#QA = 50$, in the experiments. Hence, precision is equivalent to recall. Moreover, we find the precision of generated SPARQL query patterns $P^\sharp$ is pretty high, which shows that TempalteQA is effective and promising.

Results on WebQuestions. We use the evaluation metric that is adopted in the official evaluation. Different from the metric used in QALD-5, the precision, recall, and F1 are the average precision, recall, and F1 across all test questions. Besides KBQA and UncertainTQA, we also compare



Who is the president of <Country>?	SELECT ?uri WHERE { <Country> type Country. ?uri presidentOf <Country>. }
Where was <Company> founded?	SELECT ?place WHERE { <Company> type Company. <Company> foundationPlace ?place. }
Which movie is directed by <Person>?	SELECT ?movie WHERE { ?movie type Movie. ?movie director <Person>.}

Figure 8: Case study over DBpedia.

Table 3: Results on QALD-5

Methods	QC	QP	P	P^*	R	R^*	$P^\sharp$
Xser	26	7	0.52	0.66	0.52	0.66	–
APEQ	8	5	0.16	0.26	0.16	0.26	–
QAnswer	9	4	0.18	0.26	0.18	0.26	–
SemGraphQA	7	3	0.14	0.20	0.14	0.20	–
YodaQA	8	2	0.16	0.20	0.16	0.20	–
DEANNA	9	3	0.18	0.24	0.18	0.24	–
UncertainTQA	24	6	0.48	0.60	0.48	0.60	–
KBQA	8	0	0.16	0.16	0.16	0.16	–
TempalteQA	34	4	0.68	0.76	0.68	0.76	0.86

Table 4: Results on WebQuestions

Methods	F1	Methods	F1
Bast and Haussmann [3] (2015)	0.49	QUINT	0.51
Yih et al. [48] (2015)	0.53	UncertainTQA	0.41
Reddy et al. [35] (2016)	0.50	KBQA	0.34
PARA4QA [20] (2017)	0.51	TempalteQA	0.62

TempalteQA with QUINT [1], Yao [46] (2015), Bast and Haussmann [3] (2015), Yih et al. [48] (2015), and Reddy et al. [35] (2016). Table 4 presents the results on WebQuestions. QUINT highly depends on the dependency trees and POS tag annotations. Once the output is incorrect, false answers will be generated. Different from that, our proposed TempalteQA understands input questions by template decomposition. It is clear that TempalteQA outperforms the other state-of-the-art algorithms significantly.

Results on ComplexQuestions. More interestingly, we test our method over the dataset ComplexQuestions and compare it with QUINT [1] and Bast and Haussmann++ [3] (2015), where the question is manually decomposed into subquestions and each subquestion is answered by Bast and Haussmann [3] (2015). Answering complex questions can reveal the ability of question understanding for a system. As reported in Table 5, TempalteQA achieves the best performance (with F1 of 0.71) among these methods, which confirms the potential and effectiveness of TempalteQA.

To evaluate the effectiveness of our proposed disambiguation techniques, we turn off the techniques one by one. Let R, L, and A denote the rule-based, logic-based, and and-question techniques, respectively. For example, TempalteQA/RLA represents turning off all the three techniques. As shown in Table 5, the performance degrades significantly if none of the proposed disambiguation techniques is used (i.e., the F1 of TemplateQA/RLA is 0.52). Moreover, the rule-based technique has the most effect on the quality of answers (i.e., the F1 of TemplateQA/R is 0.62).

Effect of τ. We also study the effect of the threshold τ used in the decomposition procedure. As depicted in Figure 9(a), the F1 score increases first and then decreases slightly with the growth of τ. That is because when τ is small the recall may increase but the precision decrease correspondingly. But if τ gets too larger, some subquestions may not be matched by any templates. Hence, we set τ to be 0.8 in the experiments.

Effect of θ_l and θ_u. Table 6 reports the effect of thresholds θ_l and θ_u on precision over QALD-5 (note that precision equals recall). With non-default threshold values (e.g., $\theta_l = 0.5$ and $\theta_u = 0.9$), the number of generated templates may decrease greatly, which will degrade the precision. Although the templates increase with some thresholds (e.g., $\theta_l = 0.1$ and $\theta_u = 0.9$), the quality of templates may degrade

Table 5: Results on ComplexQuestions

Methods	F1	Methods	F1
Bast and Haussmann++ [3]	0.47	TemplateQA/LA	0.61
QUINT	0.49	TemplateQA/R	0.62
TemplateQA/RLA	0.52	TemplateQA/L	0.67
TemplateQA/RL	0.59	TemplateQA/A	0.64
TemplateQA/RA	0.55	TemplateQA	0.71

Table 6: Effect of θ_l and θ_u

$\theta_u = 0.9$				$\theta_l = 0.4$			
θ_l	P	IT (s)	OT (s)	θ_u	P	IT (s)	OT (s)
0.1	0.62	504.81	0.049	0.6	0.60	582.52	0.082
0.2	0.66	472.56	0.037	0.7	0.64	518.34	0.063
0.3	0.66	441.33	0.024	0.8	0.68	474.15	0.038
0.4	0.68	428.72	0.016	0.9	0.68	428.72	0.016
0.5	0.58	336.18	0.011	1.0	0.66	426.81	0.014

significantly. It achieves the best performance when $\theta_l = 0.4$ and $\theta_u = 0.9$ by taking the indexing time (shorted as IT) and online running time (shorted as OT) into consideration.

Failure analysis. The failures can be classified into three main categories, i.e., entity linking, similarity compuation, and lack of templates. The average ratios over the datasets are listed in Table 7. It is obvious that the errors resulted from entity liking dominate all the failure cases. For instance, in the question "Give me all cosmonauts" (from QALD-5), we do not identify the nationality constraint "Russia" or "Sovie_Union". The second reason is the similarity computation problem. It fails to find the question pattern "what is the date of $\langle Holiday \rangle$" for "Halloween" in the question "Show me everyone who was born on Halloween." since there is only a word. The third reason is that some templates may not be contained in the current version. For example, the question "What is the height difference between Mount Everest and K2?" demands the subtraction operation of two elevations, which is not included in the templates. Note that only a partial set of answers for some questions can be found since some patterns are missing in the construction of $SDG(q)$. For instance, the triple pattern ⟨*res:Perl dbo:influenced ?uri*⟩ is missing for the question "Which programming languages were influenced by Perl?".

6.4 Efficiency Evaluation

Offline Performance. For ease of presentation, let TemplateQA-basic, TemplateQA-type, TemplateQA-order denote the basic decomposition method, the type-based decomposition, and order-based optimization, respectively. Table 8 reports the time cost on building index (denoted by IT) and the memory consumption (denoted by IS). Note that TemplateQA-basic does not construct index. TemplateQA-type only needs to index types, but TemplateQA-order constructs a two-layer index involving all words in the natu-

Table 7: Failure analysis

Reason	Ratio	Example
entity linking	41%	Give me all cosmonauts. Which animals are critically endangered?
similarity	33%	Show me everyone who was born on Halloween. Is Lake Baikal bigger than the Great Bear Lake?
lack of templates	18%	What is the height difference between Mount Everest and K2?

Table 8: Efficiency performance over DBpedia

Methods	IT (s)	IS (MB)	OT-D (ms)	OT-S (ms)
TemplateQA-basic	–	–	3356	5
TemplateQA-type	14.85	73.6	213	6
TemplateQA-order	428.72	506	14	5

ral language question patterns. Hence, the index size of TemplateQA-type is much smaller than that of TemplateQA-order. TemplateQA-type takes less time to build the index. The results on Freebase are similar to that on DBpedia.

Online Performance. The average online running time is presented in Table 8, where OT-D denotes the time on question decomposition, and OT-S denotes the time on SDG and SPARQL query construction. It is clear that OT-D dominates OT-S, which indicates that improving the time cost on the question is very critical. TemplateQA-basic is the most inefficient as it does not employ any index to facilitate the question decomposition. In contrast, TemplateQA-order substantially outperforms the other two methods, which confirms the effectiveness of the pruning techniques in accelerating the decomposition processing.

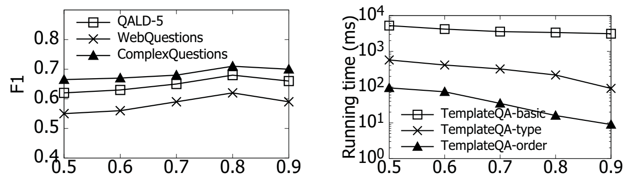
Effect of τ. We also study the effect of the similarity threshold τ (in question decomposition) on the running time. As depicted in Figure 9(b), it has slight effect on TemplateQA-basic. That is because it exhausts all possible candidates and then selects the candidate with the largest similarity score to perform threshold check. In contrast, smaller τ will lead to larger search space for the other two methods.

7. RELATED WORK

Natural language question answering. There has been a stream of research on answering natural language questions. They can be divided into the three categories.

<u>Retrieval based methods</u>. The methods integrate the techniques used in information extraction to answer natural language questions as it can provide the syntactic framework for pattern identification [31]. The answer extraction module identifies all the matches for this pattern, based on which the final answers are extracted. Delmonte proposes a hybrid system, a combination of statistical and symbolic processing with reference to a specific problem [16]. Yao and Durme identify the subgraph, called topic graph, that consists of the entities contained in a question[47]. The answer extraction process is maximally automated by combining discriminative features for both the question and the topic graph. Xu et al. present a neural network based relation extractor to retrieve the candidate answers [44].

<u>Semantic parsing and paraphrase based methods.</u> They often adopt a domain independent meaning representation derived from the combinatory categorial grammar (CCG) parses [5, 23, 48]. Berant and Liang generate canonical text descriptions for candidate logical forms and then rank the logical forms by computing paraphrase scores [6]. Xu et al. propose an efficient pipeline framework to model a user's query intention as a phrase level dependency DAG which is then instantiated regarding a specific KB to construct the final structured query [42]. Borders et al. and Dong et al. use question paraphrases in a multi-task learning framework to train the neural networks to output similar vector representations for the paraphrases [8, 21]. Yin et al. propose to use tree-structured neural networks constructed based on the constituency tree to model natural language queries [49].



(a) Effect on $F1$ (b)Running time (QALD-5)

Figure 9: Effect of the threshold τ

Recently, a general framework for learning paraphrases for question answering, called PARA4QA, is proposed, where paraphrase scoring and QA models are trained end-to-end on question-answer pairs [20]. Usually, these methods demand question-answer pairs as the training dataset, which is critical and expensive in reality.

<u>Template based methods.</u> They transform the input question into a structured query by employing templates. Conducting the structured query leads to final answers. Unger et al. propose to rely on a parse of the question to produce a SPARQL template that directly mirrors the internal structure of the question [38]. UncertainTQA [53] builds the templates automatically through the workloads of natural language questions and SPARQL queries. Both KBQA [12] and QUINT [1] take the pairs of questions and answers as the input, based on which the templates are generated. To handle the complex questions QUINT exploits the manually defined rules to rewrite the dependency parsing results, and then performs subquestion answering and answer stitching.

String similarity computation. String similarity search has been well studied there years [2, 40, 52, 37]. Xiao et al. study the top-k set similarity join, that is, it returns the top-k pairs of records ranked by their similarities [41]. Kim and Lee propose a partitioning technique that considers multiple token orderings based on token co-occurrence statistics [26]. Deng et al. also partitions a string into a set of segments and creates inverted indices for the segments [17]. However, these methods cannot be used directly in our problem as we need to guarantee the bijection between types.

For more discussions on question answering and string similarity search, please refer to two surveys [19] and [51].

8. CONCLUSIONS

Using natural language questions to query knowledge graphs provides an easy and natural way for common users to acquire useful knowledge. However, it is challenge to understand the input question especially complex questions in the view of the specified knowledge graph. In this paper, we propose a novel approach that understands natural language questions via binary templates. In order to generate adequate templates automatically, we present a low-cost method that does not demand expensive question-answer pairs as training data. We also design a systematic technique to accelerate the question decomposition and handle the two-level ambiguities effectively. Extensive empirical evaluations over several benchmarks demonstrate that our proposed method is very effective and promising.

Acknowledgements. This work was supported by grants from the Research Grant Council of the Hong Kong Special Administrative Region, China [Project No.: CUHK 14205617] and [Project No.: CUHK 14221716]. Lei Zou was supported by NSFC under grant 61622201 and 61532010.

9. REFERENCES

[1] A. Abujabal, M. Yahya, M. Riedewald, and G. Weikum. Automated template generation for question answering over knowledge graphs. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1191–1200, 2017.

[2] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 918–929, 2006.

[3] H. Bast and E. Haussmann. More accurate question answering on freebase. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*, pages 1431–1440, 2015.

[4] R. Beaumont, B. Grau, and A. Ligozat. Semgraphqa@qald5: LIMSI participation at qald5@clef. In *Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum*, pages 1–10, 2015.

[5] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, 2013.

[6] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 1415–1425, 2014.

[7] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM International Conference on Management of Data*, pages 1247–1250, 2008.

[8] A. Bordes, S. Chopra, and J. Weston. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 615–620, 2014.

[9] D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 740–750, 2014.

[10] K. Clark and C. D. Manning. Improving coreference resolution by learning entity-level distributed representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 643–653, 2016.

[11] S. Cucerzan. Large-scale named entity disambiguation based on wikipedia data. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 708–716, 2007.

[12] W. Cui, Y. Xiao, H. Wang, Y. Song, S. Hwang, and W. Wang. KBQA: learning question answering over QA corpora and knowledge bases. *International Conference on Very Large Data Bases*, 10(5):565–576, 2017.

[13] W. Cui, Y. Xiao, and W. Wang. KBQA: an online template based question answering system over freebase. In *IJCAI*, pages 4240–4241, 2016.

[14] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems (I-Semantics)*, 2013.

[15] M.-C. de Marneffe, B. MacCartney, and C. D. Manning. Generating typed dependency parses from phrase structure parses. In *LREC*, pages 449–454, 2006.

[16] R. Delmonte. Hybrid systems for information extraction and question answering. In *Proceedings of the Workshop on How Can Computational Linguistics Improve Information Retrieval*, pages 9–16, 2006.

[17] D. Deng, G. Li, H. Wen, and J. Feng. An efficient partition based method for exact set similarity joins. *PVLDB*, 9(4):360–371, 2015.

[18] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[19] D. Diefenbach, V. Lopez, K. Singh, and P. Maret. Core techniques of question answering systems over knowledge bases: a survey. *Knowledge Information Systems*, (2):1–41, 2017.

[20] L. Dong, J. Mallinson, S. Reddy, and M. Lapata. Learning to paraphrase for question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 886–897, 2017.

[21] L. Dong, F. Wei, M. Zhou, and K. Xu. Question answering over freebase with multi-column convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pages 260–269, 2015.

[22] G. Durrett, D. L. W. Hall, and D. Klein. Decentralized entity-level modeling for coreference resolution. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 114–124, 2013.

[23] A. Fader, L. Zettlemoyer, and O. Etzioni. Open question answering over curated and extracted knowledge bases. In *SIGKDD*, pages 1156–1165, 2014.

[24] J. R. Finkel, T. Grenager, and C. D. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL*, pages 363–370, 2005.

[25] E. Kaufmann and A. Bernstein. Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. *J. Web Sem.*, 8(4):377–393, 2010.

[26] J. Kim and H. Lee. Efficient exact similarity searches using multiple token orderings. In *IEEE 28th International Conference on Data Engineering*, pages 822–833, 2012.

[27] T. Kwiatkowski, E. Choi, Y. Artzi, and L. S. Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *EMNLP*, pages 1545–1556, 2013.

[28] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1):73–84, 2014.

[29] P. N. Mendes, M. Jakob, and C. Bizer. Dbpedia: A multilingual cross-domain knowledge base. In *Proceedings of the Eighth International Conference on*

Language Resources and Evaluation, pages 1813–1817, 2012.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781:1–12, 2013.

[31] D. I. Moldovan and M. Surdeanu. On the role of information retrieval and information extraction in question answering systems. In *Information Extraction in the Web Era: Natural Language Communication for Knowledge Acquisition and Intelligent Information Agents*, pages 129–147, 2002.

[32] N. Nakashole, T. Tylenda, and G. Weikum. Fine-grained semantic typing of emerging entities. In *ACL*, pages 1488–1497, 2013.

[33] L. Niu, J. Wu, and Y. Shi. Entity disambiguation with textual and connection information. In *Proceedings of the International Conference on Computational Science*, pages 1249–1255, 2012.

[34] M. Recasens, M. Can, and D. Jurafsky. Same referent, different words: Unsupervised mining of opaque coreferent mentions. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics*, pages 897–906, 2013.

[35] S. Reddy, O. Täckström, M. Collins, T. Kwiatkowski, D. Das, M. Steedman, and M. Lapata. Transforming dependency structures to logical forms for semantic parsing. *TACL*, 4:127–140, 2016.

[36] S. Ruseti, A. Mirea, T. Rebedea, and S. Trausan-Matu. Qanswer - enhanced entity matching for question answering over linked data. In *Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum*, pages 1–12, 2015.

[37] W. Tao, D. Deng, and M. Stonebraker. Approximate string joins with abbreviations. *PVLDB*, 11(1):53–65, 2017.

[38] C. Unger, L. Bühmann, J. Lehmann, A.-C. N. Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over rdf data. In *WWW*, pages 639–648, 2012.

[39] C. Unger, C. Forascu, V. López, A. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter. Question answering over linked data (QALD-5). In *Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum*, pages 1–12, 2015.

[40] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, pages 85–96, 2012.

[41] C. Xiao, W. Wang, X. Lin, and H. Shang. Top-k set similarity joins. In *Proceedings of the 25th International Conference on Data Engineering*, pages 916–927.

[42] K. Xu, Y. Feng, S. Huang, and D. Zhao. Question answering via phrasal semantic parsing. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction - 6th International Conference of the CLEF Association*, pages 414–426, 2015.

[43] K. Xu, Y. Feng, and D. Zhao. Answering natural language questions via phrasal semantic parsing. In *Working Notes for CLEF*, pages 1260–1274, 2014.

[44] K. Xu, S. Reddy, Y. Feng, S. Huang, and D. Zhao. Question answering on freebase via relation extraction and textual evidence. In *ACL*, pages 2326–2336, 2016.

[45] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL*, pages 379–390, 2012.

[46] X. Yao. Lean question answering over freebase from scratch. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 66–70, 2015.

[47] X. Yao and B. V. Durme. Information extraction over structured data: Question answering with freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 956–966, 2014.

[48] W. Yih, M. Chang, X. He, and J. Gao. 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*, pages 1321–1331, 2015.

[49] J. Yin, W. X. Zhao, and X. Li. Type-aware question answering over knowledge base with attention-based tree-structured neural networks. *J. Comput. Sci. Technol.*, 32(4):805–813, 2017.

[50] M. Yu, G. Li, D. Deng, and J. Feng. String similarity search and join: a survey. *Frontiers of Computer Science*, pages 1–19, 2015.

[51] M. Yu, G. Li, D. Deng, and J. Feng. String similarity search and join: a survey. *Frontiers of Computer Science*, 10(3):399–417, 2016.

[52] Y. Zhang, X. Li, J. Wang, Y. Zhang, C. Xing, and X. Yuan. An efficient framework for exact set similarity search using tree structure indexes. In *33rd IEEE International Conference on Data Engineering*, pages 759–770, 2017.

[53] W. Zheng, L. Zou, X. Lian, J. X. Yu, S. Song, and D. Zhao. How to build templates for RDF question/answering: An uncertain graph similarity join approach. In *SIGMOD*, pages 1809–1824, 2015.

[54] 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 *SIGMOD Conference*, pages 47–57, 2014.