# Natural Language Question/Answering: Let Users Talk With The Knowledge Graph

Weiguo Zheng
The Chinese University of Hong Kong
Shatin, Hong Kong
wgzheng@se.cuhk.edu.hk

Hong Cheng
The Chinese University of Hong Kong
Shatin, Hong Kong
hcheng@se.cuhk.edu.hk

Lei Zou
Peking University
Beijing, China
zoulei@pku.edu.cn

Jeffrey Xu Yu
The Chinese University of Hong Kong
Shatin, Hong Kong
yu@se.cuhk.edu.hk

Kangfei Zhao
The Chinese University of Hong Kong
Shatin, Hong Kong
kfzhao@se.cuhk.edu.hk

## ABSTRACT

The ever-increasing knowledge graphs impose an urgent demand of providing effective and easy-to-use query techniques for end users. Structured query languages, such as SPARQL, offer a powerful expression ability to query RDF datasets. However, they are difficult to use. Keywords are simple but have a very limited expression ability. Natural language question (NLQ) is promising on querying knowledge graphs. A huge challenge is how to understand the question clearly so as to translate the unstructured question into a structured query. In this paper, we present a data + oracle approach to answer NLQs over knowledge graphs. We let users verify the ambiguities during the query understanding. To reduce the interaction cost, we formalize an interaction problem and design an efficient strategy to solve the problem. We also propose a query prefetch technique by exploiting the latency in the interactions with users. Extensive experiments over the QALD dataset demonstrate that our proposed approach is effective as it outperforms state-of-the-art methods in terms of both precision and recall.

## KEYWORDS

Interactive Query; Natural Language Question and Answering; Knowledge Graph

## 1 INTRODUCTION

In recent years, knowledge graphs have received tremendous attention as more and more structured real-life data are available and maintained. However, providing an easy and effective access to such repositories for casual end users is important and challenging.

By supporting a simple way to retrieve the related web pages from millions of documents, keyword-based search engines (e.g., Google and Yahoo!) have gained a huge success. Nevertheless, keyword search may not be expressive enough to query structured data since only a few words are used to specify the query intention. In comparison, structured query languages, e.g., SPARQL [23], GraphQL [11] and PQL [17], are quite expressive but too complicated for casual end users. These query languages demand that users should be familiar with both the syntax and the back-end data structures, which is very difficult due to the schema-free nature of knowledge graphs.

Different from keyword and structured query language based methods, there have been lots of efforts to improve the usability by providing graphical query interfaces to explore knowledge graphs [4, 12, 14, 15, 22, 26, 27]. They allow users to construct query patterns by dragging the elementary components (e.g., vertex and edge). As pointed out in [16], these methods can be too complicated and laborious compared to systems allowing sentences. Specifically, during the query construction, users should be aware of the underlying schema (e.g., predicates and entities), and try to express her/his query intention through a structured query, which may put a heavy mental burden on end users.

Natural language question and answering interface provides a promising approach to accessing knowledge graphs. It offers users a familiar and intuitive way, rather than complicated structured query languages, to describe the query [10, 28–31, 36]. However, it needs to bridge the gap between unstructured natural language questions and structured knowledge graphs. In other words, it is required to study how to *understand the natural language question* and *generate the corresponding structured query over the knowledge graph*. Nevertheless, due to the linguistic variability and ambiguity [16], it is extremely difficult for a computer to understand some questions precisely. Let us consider an example as follows.

### 1.1 Motivating Example

Let us consider the input question in Figure 1. It is not easy to answer this question since there are three ambiguous phrases, i.e., "from", "played in", and "Philadelphia". For example, the first "in" may refer to "born in" or "live in"; "played in" may be "playingIn" or "starring"; and "Philadelphia" may be a film or a USA city or an NBA basketball team (Philadelphia_76ers). Hence, there are 12 possible interpretations with the combination of these ambiguous mappings. Using the existing techniques, e.g., joint disambiguation [24] and query-driven approach [36], we can screen out several false interpretations. For instance, the interpretation "which actor living in California is married to an actor that starred in Philadelphia_76ers" is false because the entity "Philadelphia_76ers" does not have the


*CIKM'17 , November 6–10, 2017, Singapore, Singapore*

ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3132977

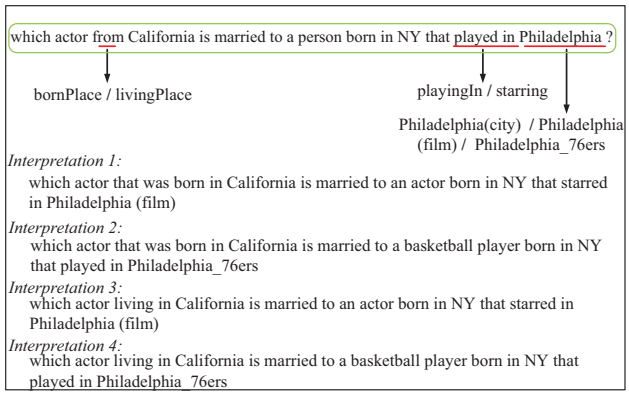



**Figure 1: An input question and its possible interpretations.**

incident predicate "starring" in the knowledge graph. However, as shown in Figure 1, there are still 4 interpretations (Interpretations 1-4) that cannot be screened out by using these techniques. The reason is that each of these interpretations may be correct. That is, there exist one or more subgraphs (in the knowledge graph) that match each interpretation. □

Therefore, it is very difficult to understand the input question clearly by using the existing disambiguation techniques. Furthermore, most of the existing work [18, 25, 36] heavily depend on the NLP tools to process natural language questions. However, these NLP tools are not guaranteed to always produce the correct results. The false results will inevitably result in more errors for understanding the query intention. For example, if a named entity in the question is not identified, it is unlikely to find the correct mapping entity in the knowledge graph, which will lead to false or empty answers.

In our daily communication, in case one person is not sure of the meaning of a question, she will try to make it clear by asking some questions back to the asker. Inspired by the natural interactive process, in this paper, we propose a novel approach that does not depend on any NLP tools. The basic principle is to *understand the question and generate the structured query through the talking between the data (i.e., the knowledge graph) and the user*. It benefits from two folds: (1) Since the user knows exactly what she wants, letting the user verify ambiguities can achieve better accuracy; (2) The knowledge graph specifies the structural relations among the mapping objects (entities/concepts/predicates) of the phrases in the question, which can help the query formulation.

## 1.2 Challenges and Contributions

In order to build an easy-to-use system that answers natural language questions over knowledge graphs for casual users, there are several challenges to be addressed.

***Challenge 1.*** *The gap between text question and structured knowledge graphs.* The question is given in natural language, which is unstructured. In contrast, the knowledge graph is structured. Hence, we need to bridge the gap by constructing a structured query for the question so as to perform the search. It is a challenging task because of two main reasons: 1) Due to the linguistic flexibility, there may be a lot of ambiguities in the process of constructing the structured query. Although some disambiguation techniques have been proposed, they cannot handle many ambiguities when the ambiguities have matches in the knowledge graph, e.g., the four possible interpretations as shown in the motivating example.

2) Schemaless nature of knowledge graphs. Unlike relational databases, knowledge graphs do not have strict schemas, which indicates that the techniques designed for relational databases, e.g., NaLIR [18] (it heavily depends on the underlying schema to construct the structured query), cannot be used to query knowledge graphs directly. Moreover, if a natural language question is parsed incorrectly, it is impossible to produce the correct structured query.

***Challenge 2.*** *Reducing interaction cost.* Although a system can understand the question better benefiting from interactions with end users (i.e., the system asks the user a question and the user gives an answer), it may degrade the user experience if there are many ambiguities to verify and no optimization is made. Hence, it is critical to schedule the interactions aiming at reducing the interaction cost. In other words, it is better to ask the user as few questions as possible. Nevertheless, it is a non-trivial task since the system does not know what the user wants.

***Challenge 3.*** *Graph match over large graphs.* To find answers to the input question, the corresponding structured query should be searched over the knowledge graph, which usually has millions of triples. Obviously, retrieving subgraphs (from a large graph) that match the query graph suffers from high computational hardness. Furthermore, the structured query is not given by the user directly. It has consumed much time to construct the structured query through interactions between the user and the system. Therefore, it is urgent to improve the time efficiency.

***Contributions.*** We propose a novel approach to construct structured queries without requiring any NLP tools (e.g., word segmentation and semantic parsing) and the schema of the knowledge graph. Specifically, we first enumerate all possible candidate phrases, and then find their corresponding candidate mappings in the knowledge graph. The phrase mappings are assembled to form complex query structures based on the input question and the underlying knowledge graph. If the system does not understand the question for some ambiguities in the whole process (generating candidate phrase mappings and query structures), it will resort to the user by presenting her with the ambiguous candidates and letting her make choice.

Actually, it may be unnecessary to verify all objects if they have some dependency relations. To reduce the interaction cost, we need to model the dependency relations and formalize an interaction problem. It has been proven to be NP-hard to devise the optimal interaction strategy in the paper. Hence, we design an efficient greedy approach to guide the interaction process.

Note that the system remains idle during the user verification. To improve the overall time efficiency, we interleave the query formulation and query evaluation by prefetch techniques. As soon as a query fragment is assembled and verified by the user, its matches over the knowledge graph are maintained. After the query formulation, it is easy to compute graph matches and generate final answers to the query based on the prestored candidate matches.

In summary, we make the following contributions in this paper:

(1) Incorporating interaction mechanisms, we present a systematic framework to enable users to query knowledge graphs by using natural language questions in an easy way.

**Table 1: Frequently-used Notations**

| Notation | Definition and Description |
|---|---|
| $G$ | a knowledge graph |
| $N$ | a natural language question (NLQ) |
| $N'$ | $N$ by removing stop words and independent phrases |
| $p$ | a phrase in the NLQ $N$ or $N'$ |
| $Ph(N')$ | the set of phrases in $N'$ |
| $EP(N')$ | the set of extended phrases |
| $S(p)$ | a set of meaning-equivalent phrases for the phrase $p$ |
| $ED(s_1, s_2)$ | the string edit distance between two strings |
| $\tau$ | the threshold for string edit distance |
| $C(p)$ | the candidate mappings for the phrase $p$ |
| $d(v)$ | the vertex degree of $v$ |
| $\mathcal{H}$ | an interaction graph |
| $\phi$ | vertex verifying order |
| $pw(\phi)$ | a possible world in the order $\phi$ |
| $\lambda(v_1, v_2)$ | the set of simple paths between $v_1$ and $v_2$ |
| $BGP$ | the basic graph pattern |

(2) We propose a novel approach to construct structured queries for natural language questions without resorting to the underlying schema and any NLP tools.
(3) Since we allow users to verify the ambiguities, enhancing the user experience is very important. To the best of our knowledge, we are the first to formalize an interaction problem and design effective strategies to guide the interaction.
(4) To improve the time efficiency, we design a prefetch technique that interleaves query formulation and query evaluation in the scenario of answering natural language questions over knowledge graphs.
(5) We demonstrate the usefulness of our method through extensive experiments over real datasets. Carefully designed user studies confirm that casual users are able to query knowledge graphs effectively in practice.

The rest of the paper is organized as follows. *Organization.* Section 2 presents the framework of our approach. In Section 3, we propose a data-driven approach to identify phrases in the natural language question and ground them to the knowledge graph. To deal with the ambiguities in the query formulation, we study how to interact with the user efficiently in Section 4. Section 5 presents the prefetch technique and generates answers to the input question.

## 2 OVERVIEW OF THE APPROACH

Table 1 lists the frequently-used notations throughout this paper. Given an input natural language (NLQ) $N$, the task is to find the answers to $N$ over the knowledge graph $G$. Figure 2 presents the overview of our approach.

### 2.1 Data-driven Question Analyzing

As shown in Figure 2, we first detect the candidate phrases and their mappings in $G$ by constructing a phrase dependency graph (Definition 3.2). After verified by the user, the mapping entities/-predicates are assembled to form a basic graph pattern (*BGP*) of the structured query based on their connections in $G$.

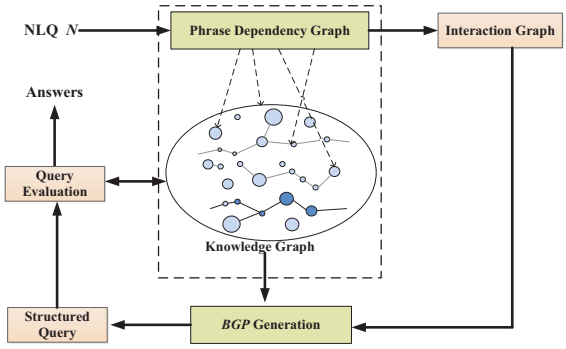



**Figure 2: Overview of the approach.**

Generally, a phrase $p$ may have multiple candidate mappings. Let $m_i$ denote the number of candidate mappings for $p$. Straightforwardly, there are $\prod m_i$ possible combinations. However, many of the combinations are not necessary to be considered since they have no relations in the knowledge graph. To find the meaningful combinations, we build a *phrase dependency graph* according to the candidate phrases and their relations in the knowledge graph, based on which we can reduce the cost of interacting with users.

After finding the mappings for phrases in the NLQ $N$, we obtain some small fragments. In general, there are many paths between two fragments. Thus, generating the structured query for the NLQ $N$ equals picking out the relevant paths. Without resorting to any semantic parser, we employ the connectivity between fragments to compute the candidate *BGP*.

### 2.2 Interacting With The User

During the construction of the structured query, it is very likely that the system does not understand the input question due to the ubiquitous ambiguities. For example, the four interpretations in Figure 1 are all correct, which are hard to determine by using the existing disambiguation techniques. To solve the problem, we propose to let the system interact with the user. Then the system could eliminate ambiguities based on the feedback from the user.

Interaction problem. Clearly, it will degrade the user experience if there are too many rounds of interactions. As discussed above, there exist dependency relations among the possible mappings. Different verifying orders of the ambiguous mappings require the distinct verifying cost. We formalize an interaction problem, which is proven to be NP-hard. Thus an efficient method is devised.

Verifying *BGP*. We try to construct the *BGP* by assembling the mapping entities/concepts/predicates. Generally, there are many paths between two fragments, each of which may belong to the *BGP*. In order to improve the efficiency, we let the user verify the candidate *BGP* at a time rather verifying the paths one by one.

## 3 DATA-DRIVEN QUESTION ANALYZING

Given a natural language question (NLQ) $N$, it should be understood by the system before retrieving the answers. We first should identify the phrases in NLQs and ground them to the specified knowledge graph (in Section 3.1). Then the mappings are assembled to form the basic graph pattern of the structured query (in Section 3.2).

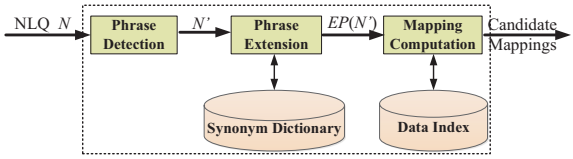



**Figure 3: Procedure of phrase mapping generation.**

## 3.1 Phrase Mapping Generation

*3.1.1 Candidate Phrase Mapping.* As presented in Figure 3, it requires three steps to generate candidate mappings.

*Phrase detection.* The phrases in NLQs can be categorized into two types: "independent phrases" and "dependent phrases". The independent phrases include variable phrases, aggregation phrases, operator phrases, modifier phrases and quantifier phrases. Its identification is independent of the knowledge graph. Similar to [18], we construct a phrase dictionary to identify independent phrases.

The dependent phrases include entity/concept/predicate phrases. Most existing methods [18, 36] exploit the NLP tools (e.g., Stanford Parser [6] and named entity recognition [5]) to chunk the NLQ to obtain a set of phrases. However, the chunking results may be incorrect, which will lead to false mappings or missing mappings.

Instead of using the existing phrase detection tools, we propose an $n$-gram based approach. We first remove stop words (e.g., "is", and "the"') and independent phrases from the original NLQ $N$, and obtain a simplified question $N'$. Then we generate a set of phrases, denoted as $Ph(N')$, by enumerating all possible phrases with length no larger than $\delta$, where $\delta$ is the predefined threshold. We discard the phrases that start with some preposition, such as "in, to, at".

*Example 3.1.* Consider the NLQ in Figure 1. After removing stop words and independent phrases, the simplified question $N'$ is "actor from California married to person born in NY played in Philadelphia". Assume $\delta = 2$, the set of phrases is $Ph(N') = \{$actor, actor from, California, California married, married, married to, person, person born, born, born in, NY, NY played, played, played in, Philadelphia$\}$.

*Phrase extension.* Because of the linguistic variability, an object can be described in diverse ways. For example, the string distance between "Secret Intelligence Service" and "MI6" is large, but both of them refer to the British intelligence agency. In other words, they have high semantic similarity. To capture the semantic similarity, we extend phrases according to a *synonym dictionary*, where each entry is a group of semantically equivalent phrases. Two well-known resources, i.e., WordNet [20] and Wikipedia, are used to build the synonym dictionary. There is a special link "redirected from" between Wikipedia articles, which indicates that the entities refer to the identical one. Moreover, we can also extract the synonym based on some rules (e.g., "commonly known as", "abbreviated"). Table 2 shows a fraction of the synonym dictionary. For each phrase $p \in Ph(N')$, we employ the synonym dictionary to find its meaning-equivalent phrases, denoted by $S(p)$. Thus, the extended phrases $EP(N')$ are obtained, i.e., $EP(N') = \{\langle p, S(p)\rangle | p \in Ph(N')\}$.

*Mapping computation.* Since the phrase extension above considers the semantic equivalence, we just need to map these extended phrases to the specified knowledge graph $G$ based on the string similarity. Here, we exploit a commonly used similarity measure for strings, i.e., the string edit distance. Actually, the mapping computation is a string similarity join problem. There have been a lot of methods proposed to improve the time efficiency (For more details, please refer to a survey [34]).

**Table 2: Synonym Dictionary**

| Group Id | Meaning-equivalent Phrases |
|---|---|
| 1 | Secret Intelligence Service, SIS, MI6 |
| 2 | car, auto, automobile, machine, motorcar |
| 3 | equip, fit, fit out, outfit |
| ... | ... |

*3.1.2 Phrase Dependency Graph.* Due to the linguistic ambiguity, a phrase may have multiple candidate mappings in the knowledge graph $G$. For instance, Philadelphia can map three different entities "Philadelphia (city)", "Philadelphia (film)" and "Philadelphia_76ers (basketball team)". Moreover, there may be some non-sense phrases in $Ph(N')$ since we enumerate all $n$-grams in $N'$. It is required to find the true phrases.

In order to understand the input NLQ, we let end users eliminate the ambiguities. Given a simplified NLQ $N'$, there are at most $(\delta|N'| - \frac{\delta^2-\delta}{2})$ phrases in $Ph(N')$, where $|N'|$ is the number of words in $N'$ and $\delta$ is the maximum gram length. For example, if $|N'| = 6$ and $\delta = 3$, the number of phrases is $|Ph(N')| = 15$, which requires 15 interactions with users in the worst case. It is very likely to degrade the user experience. To relieve user mental burden, we propose a phrase dependency graph, based on which the interaction cost can be reduced.

*Definition 3.2.* (Phrase Dependency Graph). Given a set of phrases $Ph(N')$ and the candidate mappings $C(p)$ of each phrase $p \in Ph(N')$, the corresponding phrase dependency graph, denoted by $PDG$ , contains two parts $PDG_1$ and $PDG_2$:

(1) $PDG_1$ is a graph consisting of phrases, where each vertex represents a phrase $p \in Ph(N')$, and two phrases (vertices) have an edge if they share at least one common word.
(2) $PDG_2$ is a subgraph consisting of the candidate mappings of phrases $Ph(N')$, where each vertex $c$ represents a candidate mapping (i.e., an entity/concept/predicate), and two vertices have an edge between if they are adjacent[1] in the knowledge graph $G$.
(3) There is an edge between each pair $\langle p, c\rangle$ if $c$ is a candidate mapping of $p$ (i.e., $c \in C(p)$), where $p \in PDG_1$ and $c \in PDG_2$.

Figure 4 presents a $PDG$ for the phrases of the NLQ in Figure 1. More importantly, there are two observations, based on which we can discover the dependencies in the phrase dependency graph.

Observation 1. *In the true phrase parsing result, each word in an input NLQ $N$ is just contained in one phrase at most.*

If a phrase $p$ is verified by users, we can conclude that the phrases that share at least one word with $p$ are not correct. Correspondingly, we have the first dependency on the phrase dependency graph.

Dependency 1. Inner Dependency. *If a vertex (phrase) $p \in PDG_1$ is verified to be true (a mapping is selected from $p$'s candidates) by users, its neighborhood vertices in $PDG_1$ can be removed.*

[1]We say a concept $t$ is adjacent to a predicate $r$ if there exists a triple containing both $e$ and $r$, where $e$ is an entity of type $t$.

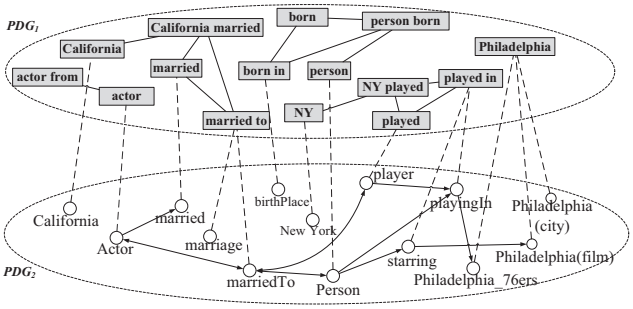



**Figure 4: Phrase dependency graph.**

Let us consider the *PDG* in Figure 4. If the phrase "played in" is verified to map "starring", the phrases "person played" and "played", and their candidate mappings can be removed.

Observation 2. *If a candidate mapping $c$ is confirmed to be true, the incident mappings of $c$ may not need to be verified.*

For instance, if "Philadelphia" maps the film "Philadelphia(film)", which is confirmed by users, the phrase "played in" should map "starring" without requiring users' verification. That is because "starring" is the only mapping that is incident to 'Philadelphia(film)".

Dependency 2. Outer Dependency. *Given the candidate mappings $C(p_1)$ and $C(p_2)$ for two phrases $p_1$ and $p_2$ (in $PDG_1$), respectively, if $p_1$ is verified to map $c_1 \in C(p_1)$ and $c_2$ is the only one in $C(p_2)$ that is adjacent to $c_1$ in $PDG_2$, the phrase $p_2$ maps $c_2$ without requiring users' verification.*

Actually, the phrase selection in the verifying process corresponds to choosing vertices in $PDG_1$. In order to reduce interactions, it is better to explore as few vertices as possible by employing the discovered dependencies. Formally, we propose a general optimization problem, i.e., the interaction problem, as shown in Section 4.

## 3.2 BGP Generation

In this section, we generate the *basic graph pattern* (Definition 3.3) based on the phrase mappings following the data-driven strategy.

*Definition 3.3.* (Basic Graph Pattern). The basic graph pattern (denoted as BGP) of a query graph refers to a subgraph of $G$, where each vertex and edge have been verified by users.

In the $PDG_2$ of a phrase dependency graph, some mappings (predicates/concepts/entities) in $G$ may be directly connected. These connected subgraphs are also called *query fragments*.

*Definition 3.4.* (Query Fragment). A query fragment, denoted by $F$, is a connected query subgraph in which vertices and edges have been mapped to the knowledge graph and verified by users.

Since the query graph is sketched by these fragments, we need to connect them appropriately to construct the structured query. Considering two vertices $v_1$ and $v_2$ in the knowledge graph $G$, let $\lambda(v_i, v_j)$ denote the set of simple paths between $v_i$ and $v_j$. It is clear that there may be multiple paths between two fragments. We need to select the paths $L$ that are most likely to be in the query graph for the natural language question $N$. Generally, the more relevant a path $L$ is to $N$, the larger the probability it belongs to the *BGP* of $N$.

**Relevance computation.** Assume that the path $L$ consists of $|L|$ labels (including predicates and entities on the path $L$) $l_1, l_2, \ldots, l_{|L|}$. Equation (1) computes the relatedness between $L$ and $N$ next.

$$Rel(L, N) = \frac{1}{|L|} \sum_{1}^{|L|} Rel(l_i, N') \tag{1}$$

where $N'$ is the simplified natural language question by removing stop words. $Rel(l_i, N')$ can be computed as the maximum relatedness between $l_i$ and each word $w_j$ in $N'$, i.e., $Rel(l_i, N') = \max_{w_j \in N'} Rel(l_i, w_j)$. There have been many methods to compute the relatedness between $l_i$ and $w_j$, such as Resnik similarity, Lin similarity [3], and word2vec [19]. If each fragment is treated as a special node, we can build a connecting graph (Definition 3.5).

*Definition 3.5.* (Connecting Graph). A connecting graph is a triple $G_C = (V_C, E_C, Rel)$: (1) Each node $F \in V_C$ represents a query fragment; (2) There may be multiple edges between every two nodes $F_1$ and $F_2$, denoted as $E_C(F_1, F_2)$, where $E_C(F_1, F_2) = \bigcup_{v_i \in F_1, v_j \in F_2} \lambda(v_i, v_j)$, and each edge $e \in E_C(F_1, F_2)$ represents a path $L$ in the knowledge graph; (3) Each edge $e$ is assigned with a relevance score $Rel(e)$ corresponding to path $L$, i.e., $Rel(e) = Rel(L, N)$.

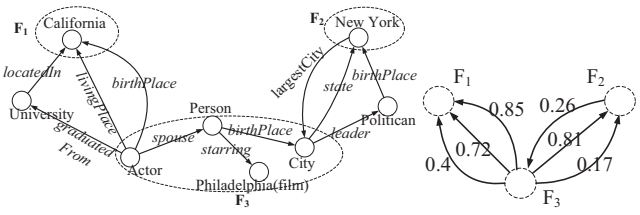
*Example 3.6.* Consider the input question in the motivating example. Figure 5(a) presents the subgraph pattern (in $G$) that contains the query fragments. Figure 5(b) is the corresponding connecting graph, where the scores on edges are fictitious.

We propose an adaptive approach to construct the *BGP*. The basic idea is to find a critical edge $e$ from the connecting graph and remove edges whose scores are less than $Rel(e)$.

*Definition 3.7.* (Critical Edge). Given an edge $e$ in $G_C$, $e$ is a critical edge if (1) Removing $e$ and all edges whose scores are less than $Rel(e)$ leads to a disconnected graph; (2) There is no other edge $e'$ with $Rel(e') < Rel(e)$ such that removing $e'$ and the edges whose scores are less than $Rel(e')$ leads to a disconnected graph.

We devise a simple but efficient solution to compute the critical edge for a connecting graph. Algorithm 1 outlines the process. Given the set of edges $E_C$, we first sort them according to their scores in the non-decreasing order (line 1). In each iteration, we try to remove the edge $e_i$ with the minimum score from $G'_C$, i.e., $G'_C/e_i$. If the left graph $G'_C/e_i$ is connected, the edge $e_i$ is removed from $G'_C$ (lines 4-5). The process repeats until $G'_C/e_i$ is disconnected.

***Time complexity***. Since it takes $O(|V_C| + |E_C|)$ to determine whether a graph is connected, the time complexity of lines 3-8 is $O(|E_C| \cdot (|V_C| + |E_C|))$. Hence, the overall time complexity of Algorithm 1 is $O(|E_C| \cdot (|V_C| + |E_C|))$.




(a) Query fragments (b) Connecting graph

**Figure 5: Query fragments and connecting graph.**

**Algorithm 1** CriticalEdge($G_C$)

**Input:** A connecting graph $G_C$;
**Output:**The critical edge.

```
1: e_0, e_1, ..., e_{|E_C|} ← sort edges e in G_C according to Rel(e) in the non-decreasing order
2: i ← 0, G'_C ← G_C
3: for i < |E_C| do
4:     if G'_C/e_i is connected then
5:         G'_C ← G'_C/e_i
6:     else
7:         return e_i
8:     i ← i + 1
```

## 4 ORACLE-BASED QUERY FORMULATION

In general, a successful system should not bring too much mental burden for users. We first formalize an interaction in Section 4.1 and then present how to verify *BGP* in Section 4.2.

### 4.1 Interaction Problem

It is very likely that there are some ambiguities during generating candidate phrase mappings and *BGP*. To make it clear, the system let end users refine the ambiguous mappings. In order to reduce the interaction cost, we formalize the interaction problem.

*Definition 4.1.* (Interaction Graph). An interaction graph is a 3-tuple $\mathcal{H} = (V, E, Pr)$, where (1) each vertex $v \in V$ represents an object; (2) each edge $(v_1, v_2) \in E$ represents the dependency relation between $v_1$ and $v_2$, which means if $v_1$ is verified to be true we do not need to verify $v_2$; (3) each $v$ takes "YES" (it can be interactively verified by users) with the probability of $Pr(v) \in [0, 1]$.

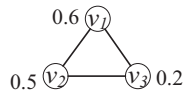
Given a vertex $v$ in the interaction graph, we ask a human (oracle) to verify whether $v$ takes "YES" or "NO". Let us consider the set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ in an interaction graph. In general, there are $n!$ possible verifying orders (i.e., the sequence of vertices to be verified). Since we do not know the actual status of each vertex, given a verifying order $\phi : v_1, v_2, \ldots, v_n$, we can define possible worlds for an interaction graph.

*Definition 4.2.* (Possible World w.r.t. $\phi$). Let $PW(\phi)$ denote the sets of all possible worlds for the interaction graph $\mathcal{H}$ in the order $\phi$. A possible world $pw(\phi) \in PW(\phi)$ is a deterministic status assignment (YES or NO) for the vertices following the order $\phi$.

**Probability of a Possible World**. In actual, each possible world $pw(\phi)$ is a status assignment for vertices of the interaction graph in the order $\phi$. The probability of each possible world is computed as $Pr\{pw(\phi)\} = \prod_{v \in V} Pr(v|\phi)$, where $Pr(v|\phi)$ is the probability that $v$ takes the assigned status in the given order $\phi$.
**Oracle Calls of a Possible World**. Given a possible world $pw(\phi)$, we count the number of oracle calls, denoted as $Calls\{pw(\phi)\}$.

*Definition 4.3.* (Interaction Cost). The cost of a verifying order $\phi$ is the expected number of oracle calls that $\phi$ makes, i.e., $cost(\phi) = \sum_{pw(\phi) \in PW(\phi)} Pr\{pw(\phi)\} \cdot Calls\{pw(\phi)\}$.

*Example 4.4.* Figure 6(a) presents an example of interaction graph. Let us consider the order $\phi : v_1, v_3, v_2$. The corresponding possible worlds are listed in Figure 6(b), where "N" represents "NO",




(a) An interaction graph

| | $v_1$ | $v_3$ | $v_2$ | $Pr$ | $Calls$ |
|---|---|---|---|---|---|
| $pw_1$ | Y | × | × | 0.6 | 1 |
| $pw_2$ | N | Y | × | 0.08 | 2 |
| $pw_3$ | N | N | Y | 0.16 | 3 |
| $pw_4$ | N | N | N | 0.16 | 3 |

(b) Possible worlds

**Figure 6: An interaction graph and its possible worlds**

"Y" represents "YES", and "×" represents not requiring verification. The interaction cost of the order $\phi : v_1, v_3, v_2$ is computed as $cost(\phi) = 0.6 \cdot 1 + 0.08 \cdot 2 + 0.16 \cdot 3 + 0.16 \cdot 3 = 1.72$.

In order to reduce oracle calls, it is clear to exploit the order with the smallest interaction cost. We define the interaction problem as:

*Definition 4.5.* (Interaction Problem). Given an interaction graph $\mathcal{H} = (V, E, P)$, finding the verifying order with the smallest $cost(\phi)$.

Theorem 4.6. *The interaction problem is NP-hard.*

Proof. Let us consider a special case where each vertex takes *YES* with the probability 1. The proof is achieved by reducing from a known NP-hard problem, i.e., the minimum independent dominating set (shorted as MIDS) problem, to the special case above. The minimum independent dominating set problem is to select the fewest vertices $D$ from a graph $G = (V, E)$ such that every vertex not in $D$ is adjacent to at least one vertex of $D$ and no two vertices in $D$ are adjacent. Any instance of the MIDS problem (denoted by $I$) precisely corresponds to an instance of the special interaction problem above (denoted by $S$). Since each vertex takes *YES* with the probability 1, it indicates that each order has only one possible world and the corresponding probability is 1. Assume that we utilize an algorithm $A$ to find the solution to $S$. The set of verified vertices is just the solution to the problem $I$. If $S$ can be solved in polynomial time, so will be the problem $I$. Since the problem $I$ has been proven to be NP-hard [8], the problem $S$ is NP-hard as well. □

It is clear that the phrase verification task is an interaction problem. A phrase interaction graph $\mathcal{H}$ can be easily constructed by adding edges to $PDG_1$: If there is an edge between $c_1 \in C(p_1)$ and $c_2 \in C(p_2)$ in $PDG_2$, an edge is added between $p_1$ and $p_2$ in $PDG_1$.
**Probability estimation**. Intuitively, if a phrase is more similar to its candidates, it is very likely to be a correct phrase. Moreover, the candidate of a correct phrase tends to be adjacent to the candidates of other correct phrases. Hence, the probability of a phrase taking "YES" can be computed as:

$$Pr(p) = \frac{\sum_{c_i \in C(p)} sim(p, c_i)}{|C(p)|} \cdot \frac{\sum_{c_i \in C(p)} d(c_i)}{|E(PDG_2)|} \tag{2}$$

where $sim(p, c_i)$ is the similarity between $p$ and its candidate $c_i$, $|E(PDG_2)|$ is the number of edges in $PDG_2$, and $d(c_i)$ is the vertex degree of $c_i$ in $PDG_2$. The first part in Equation 2 is the average similarity between the phrase $p$ and its candidate mappings. The second part is the degree proportion of $p$'s candidate vertices in $PDG_2$. However, the interaction problem is NP-hard as shown in Theorem 4.6. In order to improve the time efficiency, we propose a greedy strategy to verify the phrases. Considering $Pr(p)$ and the pruning ability (the number of phrases that can be removed if a phrase is verified to be true, denoted by $d(p)$), we define the possible benefit $B(p)$ for a phrase $p$ as $B(p) = Pr(p) \cdot d(p)$.

**Algorithm 2** Greedy Interaction($\mathcal{H}$)

**Input:** The dependency graph $\mathcal{H}$;
**Output:**A sequence of verified phrases.

```
1: while |V(H)| > 0 do
2:   for each phrase p ∈ V(H) do
3:     compute the B(p)
4:   select the phrase p_i with the largest B(p_i)
5:   if p_i is verified to be true then
6:     remove the vertices adjacent to p_i from H
7:   remove the vertex p_i from H
```

In the verification, the vertex $p$ with the largest possible benefit is selected at each step. If $v$ is verified to be true, we delete the vertices adjacent to $v$. Algorithm 2 outlines the procedure.

***Time complexity.*** The update of $\mathcal{H}$ removes $d(p_i) + 1$ vertices at most. Hence, the overall time complexity of Algorithm 2 is $O(|V(\mathcal{H})|^2)$. Since the dependency graph is small in general, the computation process is very efficient.

## 4.2 Verifying BGP

There are also some ambiguities during the process of generating *BGP*. Generally, there are multiple paths between two query fragments in the knowledge graph, each of which may belong to the *BGP*. Actually, we can use the interaction problem model in Section 4.1 to refine the candidate *BGP*. Specifically, we can build an interaction graph by taking each path between two query fragments as a vertex. To further improve the efficiency, we propose to let the user verify the whole *BGP* at a time instead of refining the candidate paths one by one. As shown in Algorithm 1, once the critical edge is found, a candidate *BGP* can be obtained by retrieving paths in $G$ that correspond to edges in $G'_C$. Each time a candidate *BGP* is generated, we present it to the user. Users are allowed to pick out the incorrect edges from the candidate *BGP*. If an edge $e$ is verified to be incorrect (More precisely, an incorrect edge means it is irrelevant to the input question), it is required to remove $e$ from $G_C$ and recompute a new candidate *BGP* by invoking Algorithm 1. Otherwise, the score of the edge $e$ is set to be 1. In other words, the generation of the *BGP* proceeds under the supervision of the user. The process stops until there is no incorrect edge.

# 5 QUERY EVALUATION

If the query formulation and query evaluation are separated, the overall performance will degrade significantly since the system is idle when users conduct verification during the query formulation. Hence, we present a query prefetch technique in Section 5.1. The generation of final answers is described in Section 5.2.

## 5.1 Query Prefetch

Let us consider two fragments $F_1$ and $F_2$ to be joined. The joining space is $|M(F_1)| \cdot |M(F_2)|$, where $M(F_i)$ denotes the matches (the subgraphs that are isomorphic to a query) for the fragment $F_i$. The joining space is large if there are too many matches for $F_i$. Therefore, it is required to provide an effective mechanism to support the join operation efficiently.

**Algorithm 3** Match Merging($F_1, F_2, F_3, MP(F_1), MP(F_2)$)

**Input:** $F_1$, $F_2$, $F_3$, $MP(F_1)$ and $MP(F_2)$;
**Output:**The matches $M(F_3)$ for $F_3$.

```
1: I_1 ← ∅, I_2 ← ∅
2: P ← the paths in F_3 that connect F_1 and F_2
3: for each path pt ∈ P do
4:   I_1 ← I_1 ∪ pt(F_1), I_2 ← I_2 ∪ pt(F_2)
5: for each mp_i ∈ MP(F_1) do
6:   if mp_i does not contain all edges in I_1 then
7:     remove mp_i from MP(F_1)
8: for each mp_j ∈ MP(F_2) do
9:   if mp_j does not contain all edges in I_2 then
10:    remove mp_j from MP(F_2)
11: for each m_i ∈ M(F_1) do
12:   for each m_j ∈ M(F_2) do
13:     for each path pt ∈ P do
14:       if no path connecting m_i, m_j corresponds to pt then
15:         go to line 12
16:     M(F_3) ← M(F_3) ∪ {m_i, m_j, pt}
17: return M(F_3)
```

*The basic idea*: We categorize the matches for each fragment according to the connectivity. To this end, we define the extended match and matching pattern as follows.

*Definition 5.1.* (Extended Match). An extended match for a query fragment $F$ is the subgraph of the knowledge graph that is obtained by adding all incident edges of each entity in a match $s$ for $F$ into $s$.

*Definition 5.2.* (Matching Pattern). A matching pattern for a query fragment $F$ is the graph formed by replacing each entity in the extended match (except the entity is specified explicitly in the natural language question) with the type of the entity.

If two query fragments $F_1$ and $F_2$ are merged to form a larger fragment $F_3$, it is easy to utilize matching patterns to compute matches for $F_3$. Algorithm 3 outlines the process. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ denote the edges in $\mathcal{P}$ that are incident to fragments $F_1$ and $F_2$, respectively, where $\mathcal{P}$ is the set of paths that connect $F_1$ and $F_2$. We first update $MP(F_1)$ (resp. $MP(F_2)$) by removing the matching patterns that do not contain edges in $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) as shown in lines 5-10. Then we check each pair of matches $m_i \in M(F_1)$ and $m_j \in M(F_2)$. If there are a set of paths connecting $m_i$ and $m_j$ that correspond to paths $\mathcal{P}$, a new match for the query fragment $F_3$ is found (lines 11-16). As soon as a new fragment is generated and verified by the user, the system computes its matching patterns and stores the corresponding entities. Obviously, the joining space, $|MP(F_1)| \cdot |MP(F_2)|$, is much smaller.

## 5.2 Answer Generation

As discussed above, the system computes matches for query fragments progressively in the process of query formulation. However, to obtain the final answer, we need to deal with the independent phrases (e.g., aggregation phrase) in the NLQ if any. If there is an aggregation phrase, we just need to conduct the corresponding computation, e.g., sum and average. Actually, the modifier phrase and operator phrase are some constraints on the matches. It is required to filter out the false matches according to the constraints.

**Table 3: Statistics of Synonym Dictionary**

| Synonym dictionary | *WordNet* | *Wikipedia* | *Sum* |
|---|---|---|---|
| ***Number of groups*** | 60,664 | 2,457,959 | 2,506,368 |

# 6 EXPERIMENTAL STUDY

We implemented an interactive natural language Q/A system to evaluate the effectiveness and efficiency of our proposed techniques.

## 6.1 Experimental Setup

*6.1.1 Datasets.* **Knowledge Graph. DBpedia**: DBpedia 2014 is an open-domain knowledge base, which contains 21,205,271 vertices (6,653,242 entities, 381,425 concepts, and 14,170,604 literals) and 155,385,713 triples (i.e., edges).

**Natural Language Queries. QALD-5**: We use QALD-5[2], a benchmark delivered in the fifth evaluation campaign answering over linked data, where each query is an NLQ. The corresponding SPARQL query and answers for each query are provided. The average number of words in questions is 8.07. The average number of vertices and edges in SPARQL queries are 3.45 and 3.02, respectively.

**Synonym Dictionary.** As discussed in Section 3.1.1, we construct a synonym dictionary based on two well-known resources. The statistics of the synonym dictionary are shown in Table 3.

*6.1.2 Metrics.* Effectiveness. We adopt two classical metrics, i.e., *precision* (the ratio of the correctly discovered answers over all returned answers, denoted by $P$) and *recall* (the ratio of the correctly discovered answers over all golden standard answers, denoted by $R$) to evaluate the effectiveness. For simplicity, we also utilize F1-measure to combine the precision and recall according to Equation 3.

$$F1 = \frac{2}{1/P + 1/R} \tag{3}$$

Usability. To evaluate the usability, we report both the response time consumed to answer a question (including the time taken by query formulation and query evaluation) and the number of interactions. We also allow users to rate the satisfaction on answering each question (on a scale of 1 to 5, where 5 denotes extremely easy to use). Then we average the rating for all questions.

The queries are randomly divided into 10 groups, each of which has 40 questions. Ten students are recruited to help evaluate the system. Each participant is assigned a group of queries. All experiments are conducted on an Intel(R) Xeon(R) CPU E5504 @ 2.00GHz and 30G RAM, on Windows Server 2008. All programs were implemented in C++.

## 6.2 Effectiveness Evaluation

*6.2.1 Effectiveness Results.* To evaluate the effectiveness, we compare our method with GAnswer [36] and QATemplate [35]. GAnswer is a state-of-the-art method that generates structured queries by employing the syntactic parser. QATemplate supports natural language question and answering by templates. It proposes an automatic approach to generate templates, based on which a natural language question is translated into a structured query.

[2]http://qald.sebastianwalter.org/index.php?x=home&q=5

**Table 4: Evaluating QALD-5 Questions**

| Method | Right | Partially | Precision | Recall | F1 |
|---|---|---|---|---|---|
| ***GAnswer*** | 98 | 53 | 0.43 | 0.41 | 0.42 |
| ***QATemplate*** | 197 | 64 | 0.62 | 0.60 | 0.61 |
| ***Our method*** | **256** | **62** | **0.84** | **0.78** | **0.81** |

**Table 5: Failure Analysis**

| Reason | #(Ratio) | Example |
|---|---|---|
| **Semantic coverage problem** | 45 (35%) | Q133. Give me all Australian nonprofit organizations |
| **False query formulation** | 49 (38%) | Q196. Give me all B-sides of the Ramones. |
| **Others** | 34 (27%) | Q74. Which capitals in Europe were host cities of the summer olympic games? |

Effectiveness results. Table 4 presents the results on QALD-5 queries, where Right specifies how many of these questions were answered with F1 measure of 1 and Partial specifies how many of the questions were answered with F1 measure strictly between 0 and 1.

Instead of dealing with the ambiguities, GAnswer tries to find all possible matches and then ranks the matches. Hence, its precision is not high enough. Benefiting from the generated templates, QATemplate can answer more questions than GAnswer. However, it requires query workloads as the training data. In comparison, our method can answer 256 questions correctly and 62 questions partially. Furthermore, we do not require any training data. It is very clear that our method in this paper is the most effective.

Effect of $\delta$ on effectiveness. During the mapping generation, we need to enumerate all phrases with length no larger than $\delta$ for an input question. To further study the effect of $\delta$, we vary $\delta$ from 1 to 6. Figure 8(a) illustrates the results. When $\delta$ is set to be 1, each word is treated as a phrase. However, a phrase may consist of two or more words. Such phrases cannot be identified if $\delta$ is 1, which will diminish both the precision and recall. As depicted in Figure 8(a), the precision and recall increase little when $\delta$ is larger than 4. The reason is that most phrases consist of no more than 5 words. Hence, $\delta$ is suggested to be 4 (also considering the efficiency, which is presented in Section 6.3.1) in our experiments.

*6.2.2 Failure Analysis.* Table 5 gives the ratio of each reason. The first reason is the semantic coverage problem, i.e., there may be multiple diverse structures that express the identical semantics. For example, consider the question "Give me all Australian nonprofit organizations". There are two different patterns corresponding to this query as shown in Figure 7. However, our method can only find one of the patterns. Hence, only partial answers can be found. The second problem is the false query formulation. Recall that we compute the BGP through the relevance $Rel(L, N)$ between each path and the question $N$. Since we resort to the semantic similarity to compute $Rel(L)$, which is not guaranteed to be accurate, it may result in false query graphs in the limited interactions with users. Some questions are not answered correctly due to lack of knowledge. Take the question "Give me all cosmonauts" as an example. We can only generate the query "?uri rdf:type dbo:Astronaut". Actually, there should be another constraint "?uri dbo:nationality res:Russia" or "?uri dbo:nationality res:Soviet_Union", which cannot be identified because there is no such knowledge at hand.

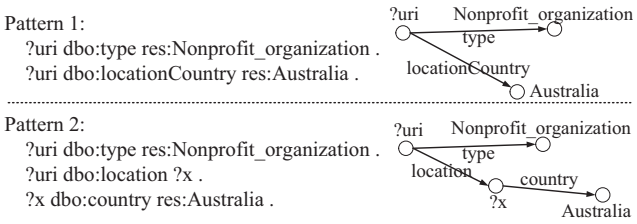



**Figure 7: Two different structure patterns for the query Q133.**

**Table 6: Response Time of Each Method (seconds)**

| Method | Question understanding | Query evaluation | Overall time |
|---|---|---|---|
| ***GAnswer*** | 0.103 | 2.431 | 2.534 |
| ***QATemplate*** | 1.216 | 1.932 | 3.148 |
| ***Our method w/o optimization*** | 37.035 | 1.186 | 38.221 |
| ***Our method*** | 26.433 | 0.074 | 26.057 |

## 6.3 Usability Evaluation

In this subsection, we evaluate the usability through the results on time efficiency and user satisfaction.

*6.3.1 Efficiency Results.* We first report the average response time (the time from receiving a question to delivering the final answers) of each method. As shown in Table 6, *GAnswer* and *QATemplate* do not need to interact with users. Hence, they are more efficient in the question understanding (i.e., mapping phrases in questions to the knowledge graph and generating the structured query). Instead of dealing with the ambiguities, *GAnswer* keeps all the candidate phrase mappings. Therefore, it is the most efficient in the first phase (question understanding). However, *GAnswer* must try to explore the knowledge graph to match the unrefined phrase mappings and then rank the candidate matches, which inevitably increases the search space and decreases the performance of query evaluation. In comparison, our method utilizes the interaction strategy to understand the meanings of questions, which consumes much more time. As presented in Table 6, the time cost increases greatly if we do not employ any optimization (i.e., the interaction strategy on choosing phrase mappings and verifying *BGP* and the prefetch technique). Moreover, our method is the most efficient on the query evaluation phase benefiting from the prefetch technique.

Effect of $\delta$ on efficiency. We also study the effect of $\delta$ (where $\delta$ is the maximum phrase length) on the time efficiency of phrase mapping. Let $T_{\delta=i}$ denote the time cost when $\delta$ is set to be $i$. Although there will be fewer candidate phrases when $\delta$ is 1, users may need to verify more candidate mappings so as to find the correct mappings. Hence, $T_{\delta=1}$ is larger than $T_{\delta=2}$ as depicted in Figure 8(b). When $\delta$ is larger than 4, there will be more candidate phrases, which may introduce more candidate mappings that need to be verified.

*6.3.2 Interaction Evaluation.* Since computing the phrase mapping and query formulation is crucial, we further study the user interactions. Figure 8(c) presents the number of interactions required to answer questions, where $|I|$ represents the number of interactions (including verifying both phrase mapping and *BGP*). As illustrated in the figure, most questions can be answered within 5 interactions with users, which indicates that our method is feasible and promising. Note that some questions can be answered without any interactions, such as "Is proinsulin a protein?".

To evaluate the user satisfaction, we ask users to rate our system on the ease of use and the ability of answering questions after accomplishing each query task. Then the ratings are averaged for all questions. The average satisfaction is 4.4 (5 denotes extremely easy to use and 1 denotes awkward to use), which indicates that the idea of answering questions in an interactive strategy is acceptable.

# 7 RELATED WORK

*Interactive visual query.* PRAGUE [14] interleaves visual query formulation and processing by exploiting GUI latency to prune false results. Modifications during the query formulation are supported. However, it is only designed for the graph database consisting of a large collection of small graphs. In order to handle large graphs, QUBLE [12] decomposes a large network into small data graphs, based on which the action-aware index for indexing the frequent fragments and the small infrequent fragments are built. Most of these methods above let the user construct a visual query by adding a vertex or edge at a time. It is time consuming to formulate a query with a large number of edges. More importantly, the user needs to know the underlying schema to draw queries precisely.

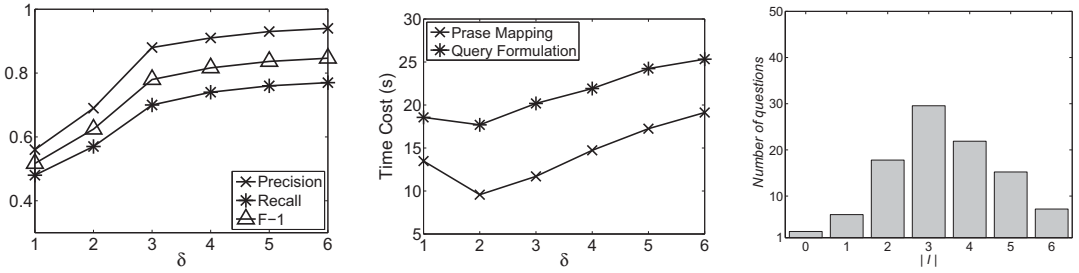
*Query by examples.* GQBE [13] allows users to query knowledge graphs by example entity tuples without writing complex queries. Different from GQBE, Mottin et al. propose a novel query paradigm exemplar queries, which indicates the type of elements that are expected to be in the results [21]. Yang et al. introduce a novel framework enabling schemaless and structureless graph querying based on a set of transformation functions [32]. Su et al. study how to improve graph query by relevance feedback [27].

*Natural language based methods.* Recently, there is a stream of research on querying knowledge graphs using natural language questions. A bunch of semantic parsing methods are proposed [1, 2, 7, 33]. They use a domain independent meaning representation derived from the combinatory categorial grammar (CCG) parses. However, they usually require many labeled corpus or pairs of questions and answers. SWSNL [9] incorporates a stochastic semantic analysis model into the question understanding stage. Yahya et al. propose an approach that allows questions to be partially translated into relaxed queries [31]. GAnswer [36] presolves the ambiguities at the time matches of the query are found. Finally, it returns the top-$k$ subgraph matches according to a match score. Zheng et al. present a template-based approach to answer natural language questions over knowledge graphs [35]. However, the method does not support aggregate-like questions.

NaLIR [18] proposes an interactive natural language query interface for relational databases. Extensive user studies demonstrate that such a framework is good enough to be usable in practice [18]. However, we face the schema-free knowledge graphs rather than the relational database that has strict schema constraints. Hence, the techniques in NaLIR are oblivious to be adopted in our task.

# 8 CONCLUSIONS

Making it easier to query large knowledge graphs for casual end users is an important and challenging task. In this paper, we propose a systematic framework to answer natural language questions




(a) Effect of $\delta$ on effectiveness (b) Effect of $\delta$ on efficiency (c) The number of questions vs. $|I|$

**Figure 8: The effect of $\delta$ and $|I|$.**

in an interactive manner. To deal with ambiguities, we let the user verify the candidate mappings. In order to improve the efficiency and enhance user satisfaction, we formalize an interaction problem and carefully design an efficient greedy strategy. By exploiting the latency in the interactions with users, we also present a query prefetch technique. Extensive experiments over the QALD benchmark demonstrate that our proposed interactive natural language question answering over knowledge graphs is effective and promising.

## ACKNOWLEDGMENTS

The work described in this paper was substantially 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 The National Key Research and Development Program of China under grant 2016YFB1000603 and NSFC under grant 61622201 and 61532010.

## REFERENCES

[1] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *EMNLP*. 1533–1544.
[2] Jonathan Berant and Percy Liang. 2014. Semantic Parsing via Paraphrasing. In *ACL*. 1415–1425.
[3] Alexander Budanitsky and Graeme Hirst. 2006. Evaluating WordNet-based Measures of Lexical Semantic Relatedness. *Computational Linguistics* 32, 1 (2006), 13–47.
[4] Duen Horng Chau, Christos Faloutsos, Hanghang Tong, Jason I. Hong, Brian Gallagher, and Tina Eliassi-Rad. 2008. GRAPHITE: A Visual Query System for Large Graphs. In *ICDM*. 963–966.
[5] William W. Cohen and Sunita Sarawagi. 2004. Exploiting dictionaries in named entity extraction: combining semi-Markov extraction processes and data integration methods. In *SIGKDD*. 89–98.
[6] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In *LREC*. 449–454.
[7] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2014. Open question answering over curated and extracted knowledge bases. In *SIGKDD*. 1156–1165.
[8] Wayne Goddard and Michael A. Henning. 2013. Independent domination in graphs: A survey and recent results. *Discrete Mathematics* 313, 7 (2013), 839–854.
[9] Ivan Habernal and Miloslav Konopík. 2013. SWSNL: Semantic Web Search Using Natural Language. *Expert Syst. Appl.* 40, 9 (2013), 3649–3664.
[10] Andreas Harth. 2010. VisiNav: A system for visual search and navigation on web data. *J. Web Sem.* 8, 4 (2010), 348–354.
[11] Huahai He and Ambuj K. Singh. 2008. Graphs-at-a-time: query language and access methods for graph databases. In *SIGMOD*. 405–418.
[12] Ho Hoang Hung, Sourav S. Bhowmick, Ba Quan Truong, Byron Choi, and Shuigeng Zhou. 2014. QUBLE: towards blending interactive visual subgraph search queries on large networks. *VLDB J.* 23, 3 (2014), 401–426.
[13] Nandish Jayaram, Mahesh Gupta, Arijit Khan, Chengkai Li, Xifeng Yan, and Ramez Elmasri. 2014. GQBE: Querying knowledge graphs by example entity tuples. In *ICDE*. 1250–1253.
[14] Changjiu Jin, Sourav S. Bhowmick, Byron Choi, and Shuigeng Zhou. 2012. PRAGUE: Towards Blending Practical Visual Subgraph Query Formulation and Query Processing. In *ICDE*. 222–233.
[15] Changjiu Jin, Sourav S. Bhowmick, Xiaokui Xiao, James Cheng, and Byron Choi. 2010. GBLENDER: towards blending visual query formulation and query processing in graph databases. In *SIGMOD*. 111–122.
[16] Esther Kaufmann and Abraham Bernstein. 2010. Evaluating the usability of natural language query languages and interfaces to Semantic Web knowledge bases. *J. Web Sem.* 8, 4 (2010), 377–393.
[17] Ulf Leser. 2005. A query language for biological networks. In *Fourth European Conference on Computational Biology/Sixth Meeting of the Spanish Bioinformatics Network*. 39.
[18] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *PVLDB* 8, 1 (2014), 73–84.
[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).
[20] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (1995), 39–41.
[21] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. 2014. Exemplar Queries: Give me an Example of What You Need. *PVLDB* 7, 5 (2014), 365–376.
[22] Robert Pienta, Acar Tamersoy, Hanghang Tong, Alex Endert, and Duen Horng (Polo) Chau. 2015. Interactive Querying over Large Network Data: Scalability, Visualization, and Interaction Design. In *IUI*. 61–64.
[23] Eric Prud'hommeaux and Andy Seaborne. 2008. SPARQL Query Language for RDF. *W3C Recommendation* (2008), 468–471.
[24] Dragomir R. Radev, Hong Qi, Zhiping Zheng, Sasha Blair-Goldensohn, Zhu Zhang, Weiguo Fan, and John M. Prager. 2001. Mining the Web for Answers to Natural Language Questions. In *CIKM*. 143–150.
[25] Diptikalyan Saha, Avrilia Floratou, K. Sankaranarayanan, U. Farooq Minhas, A. R. Mittal, and F. Özcan. 2016. ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores. *PVLDB* 9, 12 (2016), 1209–1220.
[26] Ahmet Soylu, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ernesto Jiménez-Ruiz, Martin Giese, and Ian Horrocks. 2015. OptiqueVQS: Ontology-Based Visual Querying. In *VOILA/ISWC*. 91.
[27] Yu Su, Shengqi Yang, Huan Sun, Mudhakar Srivatsa, Sue Kase, Michelle Vanni, and Xifeng Yan. 2015. Exploiting Relevance Feedback in Knowledge Graph Search. In *SIGKDD*. 1135–1144.
[28] William Tunstall-Pedoe. 2010. True Knowledge: Open-Domain Question Answering Using Structured Knowledge and Inference. *AI Magazine* 31, 3 (2010).
[29] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. 2012. Template-based question answering over RDF data. In *WWW*. 639–648.
[30] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. 2012. Natural Language Questions for the Web of Data. In *EMNLP-CoNLL*. 379–390.
[31] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, and Gerhard Weikum. 2013. Robust question answering over the web of linked data. In *CIKM*. 1107–1116.
[32] Shengqi Yang, Yinghui Wu, Huan Sun, and Xifeng Yan. 2014. Schemaless and Structureless Graph Querying. *PVLDB* 7, 7 (2014), 565–576.
[33] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. In *ACL*. 1321–1331.
[34] Minghe Yu, Guoliang Li, Dong Deng, and Jianhua Feng. 2015. String similarity search and join: a survey. *Frontiers of Computer Science* (2015), 1–19.
[35] Weiguo Zheng, Lei Zou, Xiang Lian, Jeffrey Xu Yu, Shaoxu Song, and Dongyan Zhao. 2015. How to Build Templates for RDF Question/Answering: An Uncertain Graph Similarity Join Approach. In *SIGMOD*. 1809–1824.
[36] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffery Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural Language Question Answering over RDF —- A Graph Data Driven Approach. In *SIGMOD Conference*. 47–57.