Natural language question answering over RDF – A graph data driven approach

Article · June 2014

DOI: 10.1145/2588555.2610525

Lei Zou
Peking University
91 PUBLICATIONS **1,517** CITATIONS
SEE PROFILE

Ruizhe Huang
Peking University
4 PUBLICATIONS **151** CITATIONS
SEE PROFILE

Haixun Wang
Microsoft
261 PUBLICATIONS **9,647** CITATIONS
SEE PROFILE

Jeffrey Xu Yu
The Chinese University of Hong Kong
298 PUBLICATIONS **5,664** CITATIONS
SEE PROFILE

Natural Language Question Answering over RDF — A Graph Data Driven Approach

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

Ruizhe Huang
Peking University
Beijing, China
huangruizhe@pku.edu.cn

Haixun Wang [*]
Microsoft Research Asia
Beijing, China
haixun@gmail.com

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

Wenqiang He
Peking University
Beijing, China
hewenqiang@pku.edu.cn

Dongyan Zhao
Peking University
Beijing, China
zhaody@pku.edu.cn

ABSTRACT

RDF question/answering (Q/A) allows users to ask questions in natural languages over a knowledge base represented by RDF. To answer a national language question, the existing work takes a two-stage approach: question understanding and query evaluation. Their focus is on question understanding to deal with the disambiguation of the natural language phrases. The most common technique is the joint disambiguation, which has the exponential search space. In this paper, we propose a systematic framework to answer natural language questions over RDF repository (RDF Q/A) from a graph data-driven perspective. We propose a semantic query graph to model the query intention in the natural language question in a structural way, based on which, RDF Q/A is reduced to subgraph matching problem. More importantly, we resolve the ambiguity of natural language questions at the time when matches of query are found. The cost of disambiguation is saved if there are no matching found. We compare our method with some state-of-the-art RDF Q/A systems in the benchmark dataset. Extensive experiments confirm that our method not only improves the precision but also speeds up query performance greatly.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*RDF, Graph Database, Question Answering*

1. INTRODUCTION

As more and more structured data become available on the web, the question of how end users can access this body of knowledge becomes of crucial importance. As a de facto standard of a knowledge base, RDF (Resource Description Framework) repository is a collection of triples, denoted as ⟨subject, predicate, object⟩, and can be represented as a graph, where subjects and objects are vertices and predicates are edge labels. Although SPARQL is a standard way to access RDF data, it remains tedious and difficult for end users because of the complexity of the SPARQL syntax and the RDF schema. An ideal system should allow end users to profit from the expressive power of Semantic Web standards (such as RDF and SPARQLs) while at the same time hiding their complexity behind an intuitive and easy-to-use interface [13]. Therefore, RDF question/answering (Q/A) systems have received wide attention in both NLP (natural language processing) [29, 2] and database areas [30].

[*]Haixun Wang is currently with Google Research, Mountain View, CA.

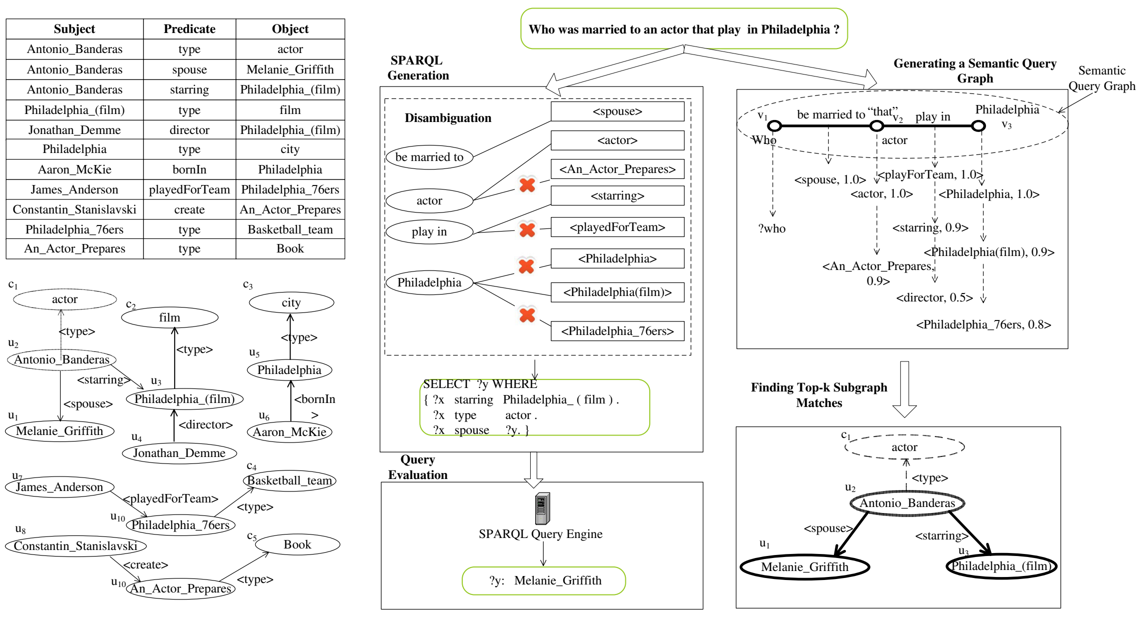
1.1 Motivation

There are two stages in RDF Q/A systems: *question understanding* and *query evaluation*. Existing systems in the first stage translate a natural language question N into SPARQLs [29, 6, 13], and in the second stage evaluate all SPARQLs translated in the first stage. The focus of the existing solutions is on query understanding. Let us consider a running example in Figure 1. The RDF dataset is given in Figure 1(a). For the natural language question N "Who was married to an actor that played in Philadelphia ? ", Figure 1(b) illustrates the two stages done in the existing solutions.

The inherent hardness in RDF Q/A is the ambiguity of natural language. In order to translate N into SPARQLs, each phrase in N should map to a semantic item (i.e, an entity or a class or a predicate) in RDF graph G. However, some phrases have ambiguities. For example, phrase "Philadelphia" may refer to entity ⟨Philadelphia(film)⟩ or ⟨Philadelphia_76ers⟩. Similarly, phrase "play in" also maps to predicates ⟨starring⟩ or ⟨playForTeam⟩. Although it is easy for humans to know the mapping from phrase "Philadelphia" (in question N) to ⟨Philadelphia_76ers⟩ is wrong, it is uneasy for machines. Disambiguating one phrase in N can influence the mapping of other phrases. The most common technique is the joint disambiguation [29]. Existing disambiguation methods only consider the semantics of a question sentence N. They have high cost in the query understanding stage, thus, it is most likely to result in slow response time in online RDF Q/A processing.

In this paper, we deal with the disambiguation in RDF Q/A from a different perspective. We do not resolve the disambiguation problem in understanding question sentence N, i.e., the first stage. We take a lazy approach and push down the disambiguation to the query evaluation stage. The main advantage of our method is it can avoid the expensive disambiguation process in the question understanding stage, and speed up the whole performance.

Our disambiguation process is integrated with query evaluation stage. More specifically, we allow that phrases (in N) correspond



Subject	Predicate	Object
Antonio_Banderas	type	actor
Antonio_Banderas	spouse	Melanie_Griffith
Antonio_Banderas	starring	Philadelphia_(film)
Philadelphia_(film)	type	film
Jonathan_Demme	director	Philadelphia_(film)
Philadelphia	type	city
Aaron_McKie	bornIn	Philadelphia
James_Anderson	playedForTeam	Philadelphia_76ers
Constantin_Stanislavski	create	An_Actor_Prepares
Philadelphia_76ers	type	Basketball_team
An_Actor_Prepares	type	Book

```
SELECT ?y WHERE
{ ?x starring Philadelphia_( film ) .
  ?x type actor .
  ?x spouse ?y. }
```

(a) RDF Dataset and RDF Graph

(b) SPARQL Generation-and-Query Framework

(c) Our Framework

Figure 1: Question Answering Over RDF Dataset

to multiple semantic items (e.g., subjects, objects and predicates) in RDF graph G in the question understanding stage, and resolve the ambiguity at the time when matches of the query are found. The cost of disambiguation is saved if there are no matching found. In our problem, the key problem is how to define a "match" of question N in RDF graph G and how to find matches efficiently. Intuitively, a match is a subgraph (of RDF graph G) that can fit the semantics of question N. The formal definition of the match is given in Definition 3 (Section 2).

We illustrate the intuition of our method by an example. Consider a subgraph of graph G in Figure 1(a) (the subgraph induced by vertices u_1, u_2, u_3 and c_1). Edge $\overrightarrow{u_2c_1}$ says that "Antonio Banderas is an actor". Edge $\overrightarrow{u_2u_1}$ says that "Melanie Griffith is married to Antonio Banderas". Edge $\overrightarrow{u_2u_3}$ says that "Antonio Banderas starred in a film ⟨Philadelphia(film)⟩". The natural language question N is "Who was married to an actor that played in Philadelphia". Obviously, the subgraph formed by edges $\overrightarrow{u_2c_1}$, $\overrightarrow{u_2u_1}$ and $\overrightarrow{u_2u_3}$ is a *match* of N. "Melanie Griffith" is a correct answer. On the other hand, we cannot find a match (of N) containing ⟨Philadelphia _76ers⟩ in RDF graph G. Therefore, the phrase "Philadelphia" (in N) cannot map to ⟨Philadelphia_76ers⟩. This is the basic idea of our data-driven approach. Different from traditional approaches, we resolve the ambiguity problem in the query evaluation stage.

A challenge of our method is how to define a "match" between a subgraph of G and a natural language question N. Because N is unstructured data and G is graph structure data, we should fill the gap between two kinds of data. Therefore, we propose a semantic query graph Q^S to represent the question semantics of N. We formally define Q^S in Definition 2. An example of Q^S is given in Figure 1(c), which represents the semantic of the question N. Each edge in Q^S denotes a semantic relation. For example, edge $\overline{v_1v_2}$ denotes that "who was married to an actor". Intuitively, a match of question N over RDF graph G is a subgraph match of Q^S over G (formally defined in Definition 3).

1.2 Our Approach

Although there are still two stages "question understanding" and "query evaluation" in our method, we do not adopt the existing framework, i.e., SPARQL generation-and-evaluation. We propose a graph data-driven solution to answer a natural language question N. The coarse-grained framework is given in Figure 1(c).

In the question understanding stage, we interpret a natural language question N as a *semantic query graph* Q^S (see Definition 2). Each edge in Q^S denotes a semantic relation extracted from N. A *semantic relation* is a triple $\langle rel, arg1, arg2\rangle$, where rel is a relation phrase, and $arg1$ and $arg2$ are its associated arguments. For example, ⟨"play in","actor","Philadelphia"⟩ is a semantic relation. The edge label is the relation phrase and the vertex labels are the associated arguments. In Q^S, two edges share one common endpoint if the two corresponding relations share one common argument. For example, there are two extracted semantic relations in N, thus, we have two edges in Q^S. Although they do not share any argument, arguments "actor" and "that" refer to the same thing. This phenomenon is known as "coreference resolution" [25]. The phrases in edges and vertices of Q^S can map to multiple semantic items (such as entities, classes and predicates) in RDF graph G. We allow the ambiguity in this stage. For example, the relation phrase "play in" (in edge $\overline{v_2v_3}$) corresponds to three different predicates. The argument "Philadelphia" in v_3 also maps to three different entities, as shown in Figure 1(c).

In the query evaluation stage, we find subgraph matches of Q^S over RDF graph G. For each subgraph match, we define its matching score (see Definition 6) that is based on the semantic similarity of the matching vertices and edge in Q^S and the subgraph match in G. We find the top-k subgraph matches with the largest scores. For example, the subgraph induced by u_1, u_2 and u_3 matches query Q^S, as shown in Figure 1(c). u_2 matches v_2 ("actor"), since u_2 (⟨Antonio_Banderas⟩) is a type-constraint entity and u_2's type is ⟨actor⟩. u_3 (⟨Philadelphia(film)⟩) matches v_3 ("Philadelphia") and u_1 (⟨Melanie_Griffith⟩) matches v_1 ("who"). The result to question N is ⟨Melanie_Griffith⟩. Also based on the subgraph match query, we cannot find a subgraph containing u_{10}(⟨Philadelphia_76ers⟩) to match Q^S. It means that the mapping from "Philadelphia" to u_{10} is a false alarm. We deal with disambiguation in query evaluation based on the matching result.

Pushing down disambiguation to the query evaluation stage not only improves the precision but also speeds up the whole query response time. Take the up-to-date DEANNA [20] as an example. DEANNA [29] proposes a joint disambiguation technique. It mod-

Table 1: Notations

Notation	Definition and Description
$G(V,E)$	RDF graph and vertex and edge sets
N	A natural language question
Q	A SPARQL query
Y	The dependency tree of q_{NL}
D	The paraphrase dictionary
T	A relation phrase dictionary
rel	A relation phrase
v_i/u_i	A vertex in query graph/RDF graph
C_{v_i}/$C_{\overline{v_iv_j}}$	Candidate mappings of vertex v_i/edge $\overline{v_iv_j}$
d	Candidate mappings of vertex v_i/edge $\overline{v_iv_j}$

els the disambiguation as an ILP (integer liner programming) problem, which is an NP-hard problem. To enable the disambiguation, DEANNA needs to build a disambiguation graph. Some phrases in the natural language question map to some candidate entities or predicates in RDF graph as vertices. In order to introduce the edges in the disambiguation graph, DEANNA needs to compute the pairwise similarity and semantic coherence between every two candidates on the fly. It is very costly. However, our method avoids the complex disambiguation algorithms, and combines the query evaluation and the disambiguation in a single step. We can speed up the whole performance greatly.

In a nutshell, we make the following contributions in this paper.

1. We propose a systematic framework (see Section 2) to answer natural language questions over RDF repositories from a graph *data-driven* perspective. To address the ambiguity issue, different from existing methods, we combine the query evaluation and the disambiguation in a single step, which not only improves the precision but also speed up query processing time greatly.

2. In the offline processing, we propose a graph mining algorithm to map natural language phrases to top-k possible predicates (in a RDF dataset) to form a paraphrase dictionary D, which is used for question understanding in RDF Q/A.

3. In the online processing, we adopt two-stage approach. In the query understanding stage, we propose a *semantic query graph* Q^S to represent the users' query intention and allow the ambiguity of phrases. Then, we reduce RDF Q/A into finding subgraph matches of Q^S over RDF graph G in the query evaluation stage. We resolve the ambiguity at the time when matches of the query are found. The cost of disambiguation is saved if there are no matching found.

4. We conduct extensive experiments over several real RDF datasets (including QALD benchmark) and compare our system with some state-of-the-art systems. Experiment results show that our solution is not only more effective but also more efficient.

2. FRAMEWORK

The problem to be addressed in this paper is to find the answers to a natural language question N over an RDF graph G. Table 1 lists the notations used throughout this paper.

There are two key challenges in this problem. The first one is how to represent the query intention of the natural language question N in a structural way. The underlying RDF repository is a graph structured data, but, the natural language question N is unstructured data. To enable query processing, we need a graph representation of N. The second one is how to address the ambiguity of natural language phrases in N. In the running example, "Philadelphia" in the question N may refer to different entities, such as

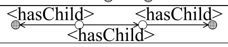
Relation Phrases	Predicates or Predicate Paths	Confidence Probability
"be married to"	<spouse>	1.0
"play in"	<starring>	0.9
"play in"	<director>	0.5
"uncle of"		0.8
… …	… …	… …

Figure 3: **Paraphrase Dictionary** D

$\langle$Philadelphia(film)$\rangle$ and $\langle$Philadelphia_76ers$\rangle$. We need to know which one is users' concern.

In order to address the first challenge, we extract the semantic relations (Definition 1) implied by the question N, based on which, we build a semantic query graph Q^S (Definition 2) to model the query intention in N.

DEFINITION 1. ***(Semantic Relation)****. A* semantic relation *is a triple* $\langle rel, arg1, arg2\rangle$*, where* rel *is a relation phrase in the paraphrase dictionary* D*,* $arg1$ *and* $arg2$ *are the two argument phrases.*

In the running example, $\langle$"be married to", "who","actor"$\rangle$ is a semantic relation, in which "be married to" is a relation phrase, "who" and "actor" are its associated arguments. We can also find another semantic relation $\langle$"play in", "that","Philadelphia"$\rangle$ in N.

DEFINITION 2. **(Semantic Query Graph)** *A* semantic query graph *is denoted as* Q^S*, in which each vertex* v_i *is associated with an argument and each edge* $\overline{v_iv_j}$ *is associated with a relation phrase,* $1 \leq i, j \leq |V(Q^S)|$.

Actually, each edge in Q^S together with the two endpoints represents a semantic relation. We build a semantic query graph Q^S as follows. We extract all semantic relations in N, each of which corresponds to an edge in Q^S. If the two semantic relations have one common argument, they share one endpoint in Q^S. In the running example, we get two semantic relations, i.e., $\langle$"be married to", "who","actor"$\rangle$ and $\langle$"play in", "that","Philadelphia"$\rangle$, as shown in Figure 2. Although they do not share any argument, arguments "actor" and "that" refer to the same thing. This phenomenon is known as "coreference resolution" [25]. Therefore, the two edges also share one common vertex in Q^S (see Figure 2(c)). We will discuss more technical issues in Section 4.1.

To deal with the ambiguity issue (the second challenge), we propose a data-driven approach. The basic idea is: for a candidate mapping from a phrase in N to an entity (i.e., vertex) in RDF graph G, if we can find the subgraph containing the entity that *fits* the query intention in N, the candidate mapping is correct; otherwise, this is a false positive mapping. To enable this, we combine the disambiguation with the query evaluation in a single step. For example, although "Philadelphia" can map three different entities, in the query evaluation stage, we can only find a subgraph containing $\langle$Philadelphia_film $\rangle$ that *matches* the semantic query graph Q^S. Note that Q^S is a structural representation of the query intention in N. The *match* is based on the subgraph isomorphism between Q^S and RDF graph G. The formal definition of *match* is given in Definition 3. For the running example, we cannot find any subgraph match containing $\langle$Philadelphia$\rangle$ or $\langle$Philadelphia_76ers$\rangle$ of Q^S. The answer to question N is "Melanie_Griffith" according to the resulting subgraph match.

Generally speaking, there are offline and online phases in our solution.

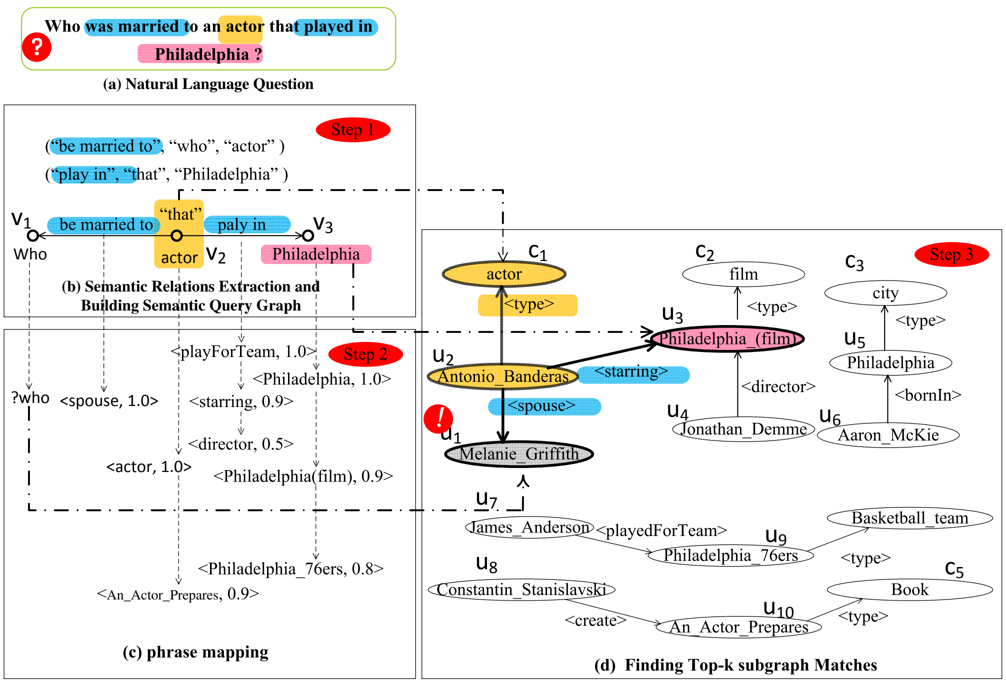


Figure 2: Natural Language Question Answering over Large RDF Graphs

2.1 Offline

To enable the semantic relation extraction from N, we build a *paraphrase dictionary* D, which records the semantic equivalence between relation phrases and predicates. For example, in the running example, natural language phrases "be married to" and "play in" have the similar semantics with predicates $\langle$spouse$\rangle$ and $\langle$starring$\rangle$, respectively. Some existing systems, such as Patty [18] and ReVerb [10], provide a rich relation phrase dataset. For each relation phrase, they also provide a support set with entity pairs, such as ($\langle$Antonio_Banderas$\rangle$, $\langle$Philadelphia(film)$\rangle$) for the relation phrase "play in". Table 2 shows two sample relation phrases and their supporting entity pairs. The intuition of our method is as follows: for each relation phrase rel_i, let $Sup(rel_i)$ denotes a set of supporting entity pairs. We assume that these entity pairs also occur in RDF graph. Experiments show that more than 67% entity pairs in the Patty relation phrase dataset occur in DBpedia RDF graph. The frequent predicates (or predicate paths) connecting the entity pairs in $Sup(rel_i)$ have the semantic equivalence with the relation phrase rel_i. Based on this idea, we propose a graph mining algorithm to find the semantic equivalence between relation phrases and predicates (or predicate paths).

2.2 Online

There are two stages in RDF Q/A: *question understanding* and *query evaluation*.

1) Question Understanding. The goal of the question understanding in our method is to build a semantic query graph Q^S for representing users' query intention in N. We first apply Stanford Parser to N to obtain the dependency tree Y of N. Then, we extract the semantic relations from Y based on the paraphrase dictionary D. The basic idea is to find a minimum subtree (of Y) that contains all words of rel, where rel is a relation phrase in D. The subtree is called an embedding of rel in Y. Based on the embedding position in Y, we also find the associated arguments according to some linguistics rules. The relation phrase rel together with the two associated arguments form a *semantic relation*, denoted as a triple $\langle rel, arg1, arg2\rangle$. Finally, we build a semantic query graph Q^S by connecting these semantic relations. We will discuss more technical issues in Section 4.1.

2) Query Evaluation. As mentioned earlier, a semantic query graph Q^S is a structural representation of N. In order to answer N, we need to find a subgraph (in RDF graph G) that *matches* Q^S. The match is defined according to the subgraph isomorphism (formally defined in Definition 3)

First, each argument in vertex v_i of Q^S is mapped to some entities or classes in the RDF graph. Given an argument arg_i (in vertex v_i of Q^S) and an RDF graph G, entity linking [31] is to retrieve all entities and classes (in G) that possibly correspond to arg_i, denoted as C_{v_i}. Each item in C_{v_i} is associated with a confidence probability. In Figure 2, argument "Philadelphia" is mapped to three different entities $\langle$Philadelphia$\rangle$, $\langle$Philadelphia(film)$\rangle$ and $\langle$Philadelphia_76ers$\rangle$, while argument "actor" is mapped to a class $\langle$Actor$\rangle$ and an entity $\langle$An_ Actor_Prepares$\rangle$. We can distinguish a class vertex and an entity vertex according to RDF's syntax. If a vertex has an incoming adjacent edge with predicate $\langle$rdf:type$\rangle$ or $\langle$rdf:subclass$\rangle$, it is a class vertex; otherwise, it is an entity vertex. Furthermore, if arg is a wh-word, we assume that it can match all entities and classes in G. Therefore, for each vertex v_i in Q^S, it also has a ranked list C_{v_i} containing candidate entities or classes.

Each relation phrase $rel_{\overline{v_iv_j}}$ (in edge $\overline{v_iv_j}$ of Q^S) is mapped to a list of candidate predicates and predicate paths. This list is denoted as $C_{\overline{v_iv_j}}$. The candidates in the list are ranked by the confidence probabilities.

It is important to note that we do not resolve the ambiguity issue in this step. For example, we allow that "Philadelphia" maps to three possible entities, $\langle$Philadelphia_76ers$\rangle$, $\langle$Philadelphia$\rangle$ and $\langle$Philadelphia(film)$\rangle$. We push down the disambiguation to the query evaluation step.

Second, if a subgraph in RDF graph can match Q^S if and only if the structure (of the subgraph) is isomorphism to Q^S. We have the following definition about match.

DEFINITION 3. **(Match)** *Consider a semantic query graph Q^S with n vertices $\{v_1,...,v_n\}$. Each vertex v_i has a candidate list C_{v_i}, $i = 1,...,n$. Each edge $\overline{v_iv_j}$ also has a candidate list of $C_{\overline{v_iv_j}}$,*

Table 2: Relation Phrases and Supporting Entity Pairs

Relation Phrase	Supporting Entity Pairs
"play in"	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩), (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),......
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),......

where $1 \leq i \neq j \leq n$. A subgraph M containing n vertices $\{u_1,...,u_n\}$ in RDF graph G is a match of Q^S if and only if the following conditions hold:

1. *If v_i is mapping to an entity u_i, $i = 1,...,n$, u_i must be in list C_{v_i}; and*
2. *If v_i is mapping to a class c_i , $i = 1,...,n$, u_i is an entity whose type is c_i (i.e., there is a triple $\langle u_i$ rdf:type $c_i\rangle$ in RDF graph) and c_i must be in C_{v_i}; and*
3. *$\forall \overrightarrow{v_iv_j} \in Q^S; \overrightarrow{u_iu_j} \in G \vee \overrightarrow{u_ju_i} \in G$. Furthermore, the predicate P_{ij} associated with $\overrightarrow{u_iu_j}$ (or $\overrightarrow{u_ju_i}$) is in $C_{\overrightarrow{v_iv_j}}$, $1 \leq i, j \leq n$.*

Each subgraph match has a score, which is derived from the probability confidences of each edge and vertex mapping. Definition 6 defines the score, which we will discuss later. Our goal is to find all subgraph matches with the top-k scores. A TA-style algorithm [11] is proposed in Section 4.2.2 to address this issue. Each subgraph match of Q^S implies an answer to the natural language question N, meanwhile, the ambiguity is resolved.

For example, in Figure 2, although "Philadelphia" can map three different entities, in the query evaluation stage, we can only find a subgraph (included by vertices u_1, u_2, u_3 and c_1 in G) containing ⟨Philadelphia_film ⟩ that *matches* the semantic query graph Q^S. According to the subgraph graph, we know that the result is "Melanie_Griffith", meanwhile, the ambiguity is resolved. Mapping phrases "Philadelphia" to ⟨Philadelphia⟩ or ⟨Philadelphia_76ers⟩ of Q^S is false positive for the question N, since there is no data to support that.

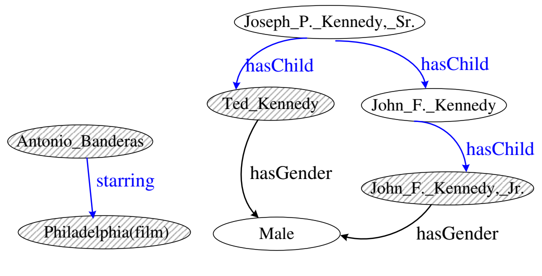
3. OFFLINE

The semantic relation extraction relies on a paraphrase dictionary D. A relation phrase is a surface string that occurs between a pair of entities in a sentence [17], such as "be married to" and "play in" in the running example. We need to build a paraphrase dictionary D, such as Figure 3, to map relation phrases to some candidate predicates or predicate paths. Table 2 shows two sample relation phrases and their supporting entity pairs.

In this paper, we do not discuss how to extract relation phrases along with their corresponding entity pairs. Lots of NLP literature about relation extraction study this problem, such as Patty [18] and ReVerb [10]. For example, Patty [18] utilizes the dependency structure in sentences and ReVerb [10] adopts the n-gram to find relation phrases and the corresponding support set. In this work, we assume that the relation phrases and their support sets are given.

The task in the offline processing is to find the semantic equivalence between relation phrases and the corresponding predicates (and predicate paths) in RDF graphs, i.e., building a paraphrase dictionary D like Figure 3.

Suppose that we have a dictionary $T = \{rel_1, ..., rel_n\}$, where each rel_i is a relation phrase, $i = 1, ..., n$. Each rel_i has a support set of entity pairs that occur in RDF graph, i.e., Sup (rel_i) $= \{ (v_i^1, v_i'^1), ..., (v_i^m, v_i'^m)\}$. For each rel_i, $i = 1, ..., n$, the goal is to mine top-k possible predicates or predicate paths formed by consecutive predicate edges in RDF graph, which have sematic equivalence with relation phrase rel_i.



(a) "play in" (b) "uncle of"

Figure 4: Mapping Relation Phrases to Predicates or Predicate Paths

Although mapping these relation phrases into canonicalized representations is the core challenge in relation extraction [17], none of the prior approaches consider mapping a relation phrase to a sequence of consecutive predicate edges in RDF graph. Patty demo [17] only finds the equivalence between a relation phrase and a single predicate. However, some relation phrases cannot be interpreted as a single predicate. For example, "uncle of" corresponds to a length-3 predicate path in RDF graph G, as shown in Figure 3. In order to address this issue, we propose the following approach.

Given a relation phrase rel_i, its corresponding support set containing entity pairs that occurs in RDF graph is denoted as $Sup(rel_i)$ $= \{ (v_i^1, v_i'^1), ..., (v_i^m, v_i'^m)\}$. Considering each pair $(v_i^j, v_i'^j)$, $j = 1, ..., m$, we find all simple paths between v_i^j and $v_i'^j$ in RDF graph G, denoted as $Path(v_i^j, v_i'^j)$. Let $PS(rel_i) = \bigcup_{j=1,...,m}$ $Path(v_i^j, v_i'^j)$. For example, given an entity pair (⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩), we locate them at RDF graph G and find simple pathes between them (as shown in Figure 4). If a path L is frequent in PS("uncle of"), L is a good candidate to represent the semantic of relation phrase "uncle of".

For efficiency considerations, we only find simple paths with no longer than a threshold[1]. We adopt a bi-directional BFS (breath-first-search) search from vertices v_i^j and $v_i'^j$ to find $Path(v_i^j, v_i'^j)$. Note that we ignore edge directions (in RDF graph) in a BFS process. For each relation phrase rel_i with m supporting entity pairs, we have a collection of all path sets $Path(v_i^j, v_i'^j)$, denoted as $PS(rel_i) = \bigcup_{j=1,...,m} Path(v_i^j, v_i'^j)$. Intuitively, if a predicate path is frequent in $PS(rel_i)$, it is a good candidate that has semantic equivalence with relation phrase rel_i.

However, the above simple intuition may introduce noises. For example, we find that (hasGender , hasGender) is the most frequent predicate path in PS("uncle of") (as shown in Figure 4). Obviously, it is not a good predicate path to represent the sematic of relation phrase "uncle of". In order to eliminate noises, we borrow the intuition of tf-idf measure [15]. Although (hasGender ,hasGender) is frequent in PS("uncle of"), it is also frequent in the path sets of other relation phrases, such as PS("is parent of"), PS("is advisor of") and so on. Thus, (hasGender ,hasGender) is not an important *feature* for PS("uncle of"). It is exactly the same with measuring the importance of a word w with regard to a document. For example, if a word w is frequent in lots of documents in a corpus, it is not a good feature. A word has a high *tf-idf*, a numerical statistic in measuring how important a word is to a document in a corpus, if it occurs in a document frequently, but the frequency of the word in the whole corpus is small. In our problem, for each relation phrase rel_i, $i = 1, ..., n$, we deem $PS(rel_i)$ as a virtual document. All predicate paths in $PS(rel_i)$ are regarded as virtual words. The corpus contains all $PS(rel_i)$, $i = 1, ..., n$. Formally, we define tf-idf value of a predicate path L in the following definition. Note that if L is a length-1 predicate path, L is a predicate P.

[1]We set the threshold as 4 in our experiments. More details about the parameter setting will be discussed in Section 6.

DEFINITION 4. *Given a predicate path L , the* tf-value *of L in $PS(rel_i)$ is defined as follows:*

$$tf(L, PS(rel_i)) = |\{Path(v_i^j, v_i'^j) | L \in Path(v_i^j, v_i'^j)\}|$$

The idf-value *of L over the whole relation phrase dictionary $T = \{rel_1, ..., rel_n\}$ is defined as follows:*

$$idf(L, T) = \log \frac{|T|}{|\{rel_i \in T | L \in PS(rel_i)\}| + 1}$$

The tf-idf value *of L is defined as follows:*

$$\text{tf}-\text{idf}(L, PS(rel_i), T) = tf(L, PS(rel_i)) \times idf(L, T)$$

We define the confidence probability of mapping relation phrase rel to predicate or predicate path L as follows.

$$\delta(rel, L) = \text{tf}-\text{idf}(L, PS(rel_i), T) \quad (1)$$

Algorithm 1 Finding Predicate Path Patterns with Sematic Equivalence with Textual Patterns

```
Require: Input: A relation phrase dictionary $T = \{rel_1, ..., rel_n\}$ and
    each textural pattern $rel_i$ , $i = 1, ..., n$, has a support set $Sup(rel_i)$
    = { $(v_i^1, v_i'^1), ..., (v_i^m, v_i'^m)$} and an RDF graph $G$.
    Output: Each relation phrase $rel_i$ ($i = 1, ..., n$) has the top-k possible
    predicate path patterns $\{L_{i_1}, ..., L_{i_k}\}$ with the sematic equivalence
    with $rel_i$.
1: for each relation phrase $rel_i$, $i = 1, ..., n$ in $T$ do
2:     for each entity pair $(v_i^j, v_i'^j)$ in $Sup(rel_i)$ do
3:         Find all simple predicate path patterns (with length less than
           a predefined parameter $\theta$) between $v_i^j$ and $v_i'^j$, denoted as
           $Path(v_i^j, v_i'^j)$.
4:     $PS(t_i) = \bigcup_{j=1,...,m} Path(v_i^j, v_i'^j)$
5: for each relation phrase $rel_i$ do
6:     for each predicate path pattern $L$ in $PS(t_i)$ do
7:         Compute tf-idf value of $L$ (according to Definition 4)
8:     for relation phrase $rel_i$, record the $k$ predicate path patterns with the
       top-k highest tf-idf values.
```

Algorithm 1 shows the details of finding top-k predicate paths for each relation phrase. All relation phrases and their corresponding k predicate paths including tf-idf values are collected to form a paraphrase dictionary D. Note that the tf-idf is a probability value to evaluate the mapping (from relation phrase to predicate/predicate paths) confidence.

To maintain the dictionary D, we can just re-mine the mappings for newly introduced predicates, or delete all mappings for the predicates when they are removed from the dataset. More details about the maintenance issue can be found at the full version of this paper [1].

THEOREM 1. *The time complexity of Algorithm 1 is $O(|T| \times |V|^2 \times d^2)$, where $|T|$ is the number of relation phrases in T, $|V|$ is the number of vertices in RDF graph G, and d is the maximal vertex degree.*

We omit the proof due to space constraints.

4. ONLINE

The online processing consists of two stages: question understanding and query evaluation. In the former one, we need to translate a natural language question N to a semantic query graph Q^S. Then, in the second stage, we find subgraph matches of Q^S. We resolve the ambiguity of natural language phrases at the time when the matches are found.

4.1 Question Understanding

This subsection discusses how to identify semantic relations in a natural language question N, based on which, we build a semantic query graph Q^S to represent the query intention in N.

In order to extract the semantic relations in N, we need to identify the relation phrases in question N. Obviously, we can simply regard the natural language question N as a sequence of words. The problem is to find which relation phrases (also regarded as a sequence of words) are subsequences of N. However, the ordering of words in a natural language sentence is not fixed, such as inverted sentences and fronting. For example, let us consider a question "*In which movies did Antonio Banderas star* ?". Obviously, "star in" indicates the semantic relation between "Antonio Banderas" and "which movie", but, "star in" is not a subsequence of the question language due to *preposition fronting*. The phenomenon is known as "long-distance dependency". Some NLP (natural language processing) literature suggest that the dependency structure is more stable for the relation extraction [18]. The dependencies are grammatical relations between a *governor* (also known as a regent or a head) and a *dependent*. Usually, we can map straightforwardly these dependencies into a tree, called *dependency tree* [8]. Specifically, words in the sentence are corresponding to nodes in a dependency tree and the edge labels describe the grammatical relations. Note that "*<u>In</u> which movies did Antonio Banderas <u>star</u>*?" and "*Which movies did Antonio Banderas <u>star in</u>* ?" have the same dependency tree, even though the former has the preposition fronting phenomenon.

Therefore, in our work, we first apply Stanford Parser [8] to N to obtain the dependency tree Y. Let us recall the running example. Figure 5 shows the dependency tree of N (in the running example), denoted as Y. The next question is to find which relation phrases occurring in Y.

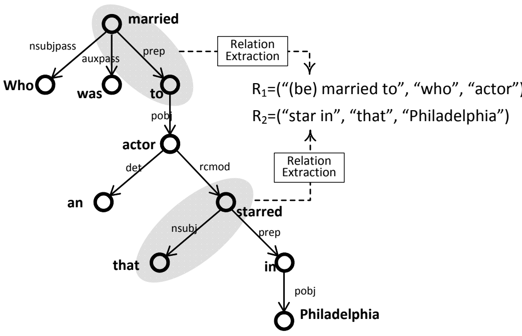


Figure 5: Relationship Extraction

DEFINITION 5. *Let us consider a dependency tree Y of a natural language question N and a relation phrase rel. We say that rel* occurs *in Y if and only if there exists a connected subtree y (of Y) satisfying the following conditions:*

1. *Each node in y contains one word in rel and y includes all words in rel.*

2. *We cannot find a subtree y' of Y, where y' also satisfies the first condition and y is a subtree of y'.*

In this case, y is an embedding *of relation phrase rel in Y.*

Given a dependency tree Y of a natural language question N and a relation phrase dictionary $T = \{rel_1, ..., rel_n\}$, we need to find which relation phrases (in T) are occurring in Y.

Let us consider Figure 5. The dependency tree Y of the natural language in the running example is given in Figure 5. We find two

Algorithm 2 Finding Relation Phrases Occurring in a Natural Language Questions N

```
Require: Input: A dependency tree $Y$ and an inverted index over the relation phrase dictionary $T$.
    Output: All embeddings of relation phrases (in $T$) occurring in $Y$.
 1: for each node $w_i$ in $Y$ do
 2:     Find a list of relation phrases $PL_i$ occurring in $T$ by the inverted list.
 3: for each node $w_i$ in $Y$ do
 4:     Set $PL = PL_i$
 5:     for each relation phrase $rel \in PL$ do
 6:         Set $rel[w_i] = 1$ // indicating the appearance of word $w_i$ in $rel$.
 7:     Call Probe($w_i$, $PL$)
 8:     for each relation phrase $rel$ in $PL_i$ do
 9:         if all words $w$ of $rel$ have $rel[w] = 1$ then
10:             $rel$ is an occurring relation phrase in $Y$
11:             Return $rel$ and a subtree rooted at $w_i$ includes (and only includes) all words in $rel$.
Probe($w$, $PL'$)
 1: for each child $w'$ of $w$ do
 2:     $PL'' = PL' \cap PL_i$.
 3:     if $PL'' == \phi$ then
 4:         return
 5:     else
 6:         for each relation phrase $rel \in PL''$ do
 7:             Set $rel[w'] = 1$ // indicating the appearance of word $w'$ in $t$.
 8:         Call Probe($w'$, $PL''$)
```

relation phrases rel_1 ="be married to" and rel_2="star in" occurring in Y, where rel_1 and rel_2 are in a paraphrase dictionary T. We also find their associated arguments of rel_1 and rel_2, respectively. Finally, we can find two relations R_1 and R_2 in this example, which are denoted as the triple representation (as shown in Figure 5). Now, we discuss how to find relation phrase embeddings and the associated arguments as follows.

4.1.1 Finding Relation Phrase Embeddings

Given a natural language question N, we propose an algorithm (Algorithm 2) to identify all relation phrases in N. In the offline phase, we build an inverted index over all relation phrases in the paraphrase dictionary D. Specifically, for each word, it links to a list of relation phrases containing the word. At run time, we first get the dependency tree Y of a natural language question N. In order to find relation phrases occurring in Y, we propose Algorithm 2. The basic idea is as follows: For each node (i.e., a word) w_i in Y, we find the candidate pattern list PL_i (Steps 1-2 in Algorithm 2). Then, for each node w_i, we check whether there exists a subtree rooted at w_i including all words of some relation phrases in PL_i. In order to address this issue, we propose a depth-first search strategy. We probe each path rooted at w_i (Step 3). The search branch stops at a node w', where there does not exists a relation phrase including w' and all words along the path between w' and w_i (Note that, w' is a descendant node of w_i.)(Lines 3-4 in Probe function.) We utilize $rel[w]$ to indicate the presence of word w of rel in the subtree rooted at w_i (Line 6). When we finish all search branches, if $rel[w] = 1$ for all words w in relation phrase rel, it means that we have found a relation phrase rel occurring in Y and the embedding subtree is rooted at w_i (Lines 8-11). We can find the exact embedding (i.e., the subtree) by probing the paths rooted at w_i. We omit the trivial details due to the space limit.

THEOREM 2. *The time complexity of Algorithm 2 is $O(|Y|^2)$.*

4.1.2 Finding Associated Arguments

After finding a relation phrase in Y, we then look for the two associated arguments. Generally, the arguments arg1 and arg2 are recognized mainly based on the grammatical subject-like and object-like relations around the embedding, which are listed as follow [2]:

1. *subject-like relations*: subj, nsubj, nsubjpass, csubj, csubjpass, xsubj, poss;
2. *object-like relations*: obj, pobj, dobj, iobj

Assume that we find an embedding subtree y of a relation phrase rel. We recognize arg1 by checking for each node w in y whether there exists the above subject-like relations (by checking the edge labels in the dependency tree) between w and one of its children (note that, the child is not in the embedding subtree). If a subject-like relationship exists, we add the child to arg1. Likewise, arg2 is recognized by the object-like relations. When there are still more than one candidates for each argument, we choose the nearest one to rel.

On the other hand, when arg1/arg2 is empty after this step, we introduce several heuristic rules (based some computational linguistics knowledge) to increase the recall for finding arguments. The heuristic rules are applied until arg1/arg2 becomes none empty.

- Rule 1: Extend the embedding t with some light words, such as prepositions, auxiliaries. Recognize subject/object-like relations for the newly added tree node.
- Rule 2: If the root node of t has subject/object-like relations with its parent node in Y, add the root node to arg1.
- Rule 3: If the parent of the root node of t has subject-like relations with its child, add the child to arg1.
- Rule 4: If one of arg1/arg2 is empty, add the nearest wh-word (such as what, who and which) or the first noun phrase in t to arg1/argu2.

If we still cannot find arguments arg1/arg2 after applying the above heuristical rules, we just discard the relation phrase rel in the further consideration. Finally, we can find all relation phrases occurring in N together with their embeddings and their arguments arg1/arg2.

Let us recall dependency tree Y in Figure 5. There is a "nsubjpass" dependency between "who" and a relation phrase "(be) married to". Thus, the first argument is "who". Analogously, we can find another argument "actor" based on the "pobj" dependency. Therefore, the first semantic relation is ⟨"(be) married to", "who", "actor"⟩. Likewise, we can also find another semantic relation ⟨"play in", "that", "Philadelphia"⟩.

4.1.3 Building Semantic Query Graph

After obtaining all semantic relations in a natural language N, we need to build a semantic query graph Q^S. Actually, a semantic query graph is a structural representation of users' query intention. Figure 2(b) shows an example of Q^S. In order to build a semantic query graph Q^S, the method is as follows. We represent each semantic relation $\langle rel, arg1, arg2\rangle$ as an edge. Two edges share one common endpoint if their corresponding semantic relations have one common argument. The formal definition of a query semantic graph has been given in Definition 2.

Let us recall the running example. Although two relations ⟨"be married to", "who", "actor"⟩ and ⟨"play in", "that", "Philadelphia"⟩ do not share any argument, "actor" and "that" actually refer to the same thing. This phenomenon is known as "coreference resolution" [25]. Therefore, the two corresponding edges share an endpoint ("actor" and "that") in Q^S, as shown in Figure 2(b). Note that

[2]These grammatical relationships (called *dependencies*) that are defined in [8]. For example, "nsubj" refers to a nominal subject. It is a noun phrase which is the syntactic subject of a clause. Interested reader can refer to [8] for more details.

we can utilize the existing solutions to address the "coreference resolution". This problem has been well studied in natural language processing [25].

4.2 Query Evaluation

4.2.1 Phrases Mapping

The semantic query graph Q^S plays an important role in answering the natural language question N. Actually, Q^S bridges the gap between users' un-structured query intention of a natural language question N and the structured RDF data G. There are two different items in Q^S. One is the relation phrase in each edge of Q^S. The other one is the argument in each vertex of Q^S. In this subsection, we discuss how to map the relation phrases and arguments to candidate predicates/predicate paths and entities/classes, respectively.

Mapping Edges of Q^S. Each edge $\overline{v_iv_j}$ in Q^S has a relation phrase $rel_{\overline{v_iv_j}}$. According to the paraphrase dictionary D (see Section 3), it is straightforward to map $rel_{\overline{v_iv_j}}$ to some predicates P or predicate paths L. The list is denoted as $C_{\overline{v_iv_j}}$. As mentioned earlier, P is a special case when L's length is one. For simplicity of notations, we use L in the following discussion. Each mapping is associated with a confidence probability $\delta(rel, L)$ (defined in Equation 1). For example, edge $\overline{v_2v_3}$ has a relation phrase $rel_{\overline{v_2v_3}}$="play in". Its candidate list $C_{\overline{v_2v_3}}$ contains three candidates, ⟨playForTeam⟩, ⟨starring⟩, and ⟨director⟩, as shown in Figure 2(c). We also record the confidence probability $\delta(rel, L)$ in $C_{\overline{v_2v_3}}$. Note that we allow the ambiguity in the mapping in this stage.

Mapping Vertices of Q^S. Let us consider any vertex v in Q^S. The argument associated with v is arg. If arg is a wh-word, it can be mapped to all entities and classes in RDF graph G. Otherwise, given an argument arg, we should return a list of corresponding entities or classes. This is an entity linking problem [16, 21]. We use notation C_v to denote all candidates with regard to vertex v in Q^S. For example, argument "Philadelphia" in v_3 (in Figure 2) is linked to three different entities in RDF graph, i.e., ⟨Philadelphia⟩, ⟨Philadelphia(film)⟩ and ⟨Philadelphia_76ers⟩. "actor" in v_2 can be linked to a class node ⟨actor⟩ or an entity node "An_Actor_Prepares". The entity linking problem is well studied in the literature, such as Wikify [16] and Wikifier [21]. In this work, we use an existing system, DBpedia Lookup [4] [3], to return a list of corresponding entities and classes for an argument. If arg is mapped to an entity u, we use $\delta(arg, u)$ to denote the confidence probability; if arg is mapped to a class c, we use $\delta(arg, c)$ to denote the confidence probability. As mentioned earlier, in our method, we do not address the disambiguation issue in this stage.

Graph Data-driven Disambiguation. Obviously, there are some ambiguity problems in the above mapping process. For example, "Philadelphia" in the natural language question N can be mapped to three different entities, i.e., ⟨Phila delphia(film)⟩, ⟨Philadelphia⟩ and ⟨Philadelphia_76ers⟩. In this work, we regard the disambiguation from another perspective—graph data-driven solution. We allow the ambiguity in the phrase mapping step and push down the disambiguation into the query evaluation stage. Assume that a vertex v in Q^S maps to lots of candidate vertices in RDF graph G. If a candidate is in a *match* (see Definition 3) of Q^S in RDF graph G, it is a reasonable candidate. Furthermore, the subgraph is an answer to the natural language question N. Otherwise, the candidate is a false positive.

[3]http://lookup.dbpedia.org/api/search.asmx

4.2.2 Finding top-k Subgraph Matches

Given a semantic query graph Q^S, we discuss how to find top-k subgraph matches over RDF graph G in this subsection. The formal definition of a subgraph match is given in Definition 3 (in Section 2.2). Each vertex v_i in Q^S has a list C_{v_i} of candidate vertices (i.e., entities and classes) in RDF graph G. Assume that v_i in Q^S has an argument arg_i. For any candidate vertex u_i in C_{v_i}, we use $\delta(arg_i, u_i)$ to denote the confidence probability. Analogously, each edge $\overline{v_iv_j}$ in Q^S also has a list $C_{\overline{v_iv_j}}$ of candidate predicates in G. Assume that edge $\overline{v_iv_j}$ has a relation phrase $rel_{\overline{v_iv_j}}$. For any candidate predicate P_{ij} in $C_{\overline{v_iv_j}}$, we use $\delta(rel_{\overline{v_iv_j}}, P_{ij})$ to denote the confidence probability. We assume that all candidate lists are ranked in the non-ascending order of the confidence probability. Figure 2(b) and (c) show an example of Q^S and the candidate lists, respectively. Each subgraph match of Q^S has a score. It is computed from the confidence probabilities of each edge and vertex mapping. The score is defined as follows.

Definition 6. *Given a semantic query graph Q^S with n vertices $\{v_1,...,v_n\}$, a subgraph M containing n vertices $\{u_1,...,u_n\}$ in RDF graph G is a* match *of Q^S. The match score is defined as follows:*

$$\begin{aligned}&Score(M) = \\ &\log(\prod_{v_i \in V(Q^S)} \delta(arg_i, u_i) \times \prod_{\overline{v_iv_j} \in E(Q^S)} \delta(rel_{\overline{v_iv_j}}, P_{ij})) \\ &= \sum_{v_i \in V(Q^S)} \log(\delta(arg_i, u_i)) + \sum_{\overline{v_iv_j} \in E(Q^S)} \log(\delta(rel_{\overline{v_iv_j}}, P_{ij}))\end{aligned} \quad (2)$$

where arg_i is the argument of vertex v_i, and u_i is an entity or a class in RDF graph G, and $rel_{\overline{v_iv_j}}$ is the relation phrase of edge $\overline{v_iv_j}$ and P_{ij} is a predicate of edge $\overrightarrow{u_iu_j}$ or $\overrightarrow{u_ju_i}$.

Given a semantic query graph Q^S, our goal is to find all subgraph matches of Q^S (over RDF graph G) with the top-k match scores [4]. This is an NP-hard problem. We provide the detailed analysis as follows.

Lemma 1. *Finding Top-1 subgraph match of Q^S over RDF graph G is an NP-hard problem.*

Proof. As we know, the decision problem of subgraph isomorphism is a classical NP-complete problem. We can reduce the decision problem of subgraph isomorphism to finding top-1 subgraph match of Q^S over G in a polynomial time, since the former can be solved by invoking the latter problem. It means finding the top-1 subgraph is at least as hard as subgraph isomorphism problem. Since subgraph isomorphism problem is an NP complete problem, it means that finding the top-1 subgraph is an NP-hard problem. □

Lemma 2. *Finding Top-k subgraph match of Q^S over RDF graph G is at least as hard as finding Top-1 subgraph match of Q^S over G.*

Proof. We can reduce finding top-1 subgraph match to finding top-k subgraph matches, since the former can be solved by invoking the latter problem. When we find top-k subgraph matches, we can find the top-1 in $O(k)$ time. It means that finding top-k subgraph match is at least as hard as finding top-1 subgraph match. □

Theorem 3. *Finding Top-k subgraph matches of Q^S over RDF graph G is an NP-hard problem.*

[4]Note that if more than one match have the identical score in the top-k results, they are only counted once. In other words, we may return more than k matches if some matches share the same score

PROOF. It can be derived from Lemmas 1 and 2. □

Since finding top-k subgraph matches is an NP-hard problem, hence, we concentrate on designing heuristic rules to reduce the search space. The first pruning method is to reduce the candidates of each list (i.e, C_{v_i} and $C_{\overline{v_i v_j}}$) as many as possible. If a vertex u_i in C_{v_i} cannot be in any subgraph match of Q^S, u_i can be filtered out directly. Let us recall Figure 2. Vertex u_5 is a candidate in C_{v_3}. However, u_5 does not have an adjacent predicate that is mapping to phrase "play in" in edge $\overline{v_2 v_3}$. It means that there exists no subgraph match of Q^S containing u_5. Therefore, u_5 can be pruned safely. This is called neighborhood-based pruning. It is often used in subgraph search problem, such as [32].

Algorithm 3 Generating Top-k SPARQL Queries

```
Require: Input: A semantic query graph $Q^S$ and a RDF $G$. Output:
    Top-k SPARQL Queries, i.e., the top-k matches from $Q^S$ to $G$.
1: for each candidate list $L_{r_i}$, $i = 1, ......, |E(Q^S)|$ do
2:     Sorting all candidate relations in $L_{r_i}$ in a non-ascending order
3: for each candidate list $L_{arg_j}$, $j = 1, ......, |V(Q^S)|$ do
4:     Sorting all candidate entities/classes (i.e., vertices in $G'$) in $L_{arg_j}$
       in a non-ascending order.
5: Set cursor $c_i$ to the head of $L_{r_i}$ and cursor $c_j$ to the head of $L_{arg_j}$,
   respectively.
6: Set the upper bound $Upbound(Q)$ according to Equation 3 and the
   threshold $\theta = -\infty$
7: while true do
8:     for each cursor $c_j$ in list $L_{arg_j}$, $j = 1, ......, |V(Q^S)|$ do
9:         Performance a exploration based subgraph isomorphism algo-
           rithm from cursor $c_j$, such as VF2, to find subgraph matches (of
           $Q^S$ over $G$), which contains $c_j$.
10:    Update the threshold $\theta$ to be the top-k match sore so far.
11:    Move all cursors $c_i$ and $c_j$ by one step forward in each list.
12:    Update the upper bound $Upbound(Q)$ according to Equation 3.
13:    if $\theta \geq Upbound(Q)$ then
14:        Break // TA-style stopping strategy
```

The second method is to stop the search process based on the top-k match score threshold as early as possible. We perform a TA-style algorithm to find the top-k matches. The pseudo codes are given in Algorithm 3. There is a cursor for each candidate list. For each vertex v_i in Q^S, we set cursor p_i point to list C_{v_i}. For each edge $\overline{v_i v_j}$ in Q^S, we also set cursor p_{ij} to list $C_{\overline{v_i v_j}}$. For ease of presentation, we also use p_i to denote the vertex (in G) that cursor p_i points to, when the context is clear. Initially, all cursors point to the heads of candidate lists. We maintain a threshold θ to be the k-th highest score so far. It is $-\infty$ at the beginning. For each p_i, $i = 1, ..., |V(Q^S)|$, we perform a subgraph isomorphism algorithm (such as VF2 algorithm [7]) from vertex p_i (in G) to find subgraph matches containing vertex p_i. We probe each p_i in a round-robin manner, $i = 1, ..., |V(Q^S)|$. Then, we update the current threshold θ according to the scores of these newly discovered matches. The upper bound for undiscovered matches is computed by the following equation.

$$Upbound = \sum_{p_i} \log(\delta(arg_i, p_i)) + \sum_{p_{ij}} \log(\delta(rel_{\overline{v_i v_j}}, p_{ij})) \quad (3)$$

where p_i denotes the current entity/class in the list C_{v_i} and p_{ij} denotes the current predicate in the list $C_{\overline{v_i v_j}}$.

If the threshold $\theta > Upbound$, it means that the score of all undiscovered matches cannot be higher than the current top-k match score. In other words, we have found the top-k matches. Thus, we can terminate the algorithm. Otherwise, we move all cursors one step forward and iterate the above steps.

Table 4: Statistics of RDF Graph

	DBpedia
Number of Entities	5.2 million
Number of Triples	60 million
Number of Predicates	1643
Size of RDF Graphs (in GB)	6.1

Table 5: Statistics of Relation Phrase Dataset

	wordnet-wikipedia	freebase-wikipedia
Number of Textual Patterns	350,568	1,631,530
Number of Entity Pairs	3,862,304	15,802,947
Average Entity Pair Number For Each Pattern	11	9

5. TIME COMPLEXITY ANALYSIS

In the offline phase, we need to build a paraphrase dictionary D by Algorithm 1. Therefore, the time complexity of the offline phase (Algorithm 1) is $O(|T| \times |V|^2 \times d)$, where $|T|$ is the number of relation phrases, $|V|$ is the number of vertices in RDF graph G, d is the largest vertex degree.

In the online phase, there are two stages, i.e., question understanding and query evaluation. The first stage consists of four steps. Given a natural language question N, we obtain a dependency tree Y by the dependency parser. As we know, the time complexity of the dependency parser is $O(|Y|^3)$ [5], such as Eisner's algorithm [9]. The second step is to find all relation embeddings by Algorithm 2, whose time complexity is $O(|Y|^2)$. The last two substeps (finding associated arguments and building semantic query graph) are linear with the number of relation phrases in N. It is straightforward to know there are at most $|Y|$ relation phrases, i.e., assuming each word in N is a relation phrase. Therefore, the whole time complexity of the question understanding in our method is $O(|Y|^3)$.

In the second stage, "query evaluation", our method needs to find subgraph matches of the semantic query graph over RDF graph G. It includes two sub steps, i.e., finding phrases mappings (Section 4.2.1) and finding subgraph matches (Section 4.2.2). Obviously, the time complexity of this stage is NP-hard due to the time complexity of finding subgraph matches (see Theorem 3).

Table 3 summarizes the time complexity of each step.

6. EXPERIMENTS

In this section, we compare our method with one state-of-the-art algorithm DEANNA [29] and all systems in QALD-3 competition on DBpedia RDF dataset. To build the paraphrase dictionary, we utilize relationship phrases in Patty [18] system. All experiments are implemented in a PC server with Intel Xeon CPU 2GB Hz, 32GB memory and 3T disk running Windows 2008. We also evaluate our method in other RDF repositories, such as Yago2. Due to the space limit, we only report the experiment results on DBpedia.

6.1 Datasets

DBpedia RDF repository (http://blog.dbpedia.org/) is a community effort to extract structured information from Wikipedia and to make this information available on the Web [4]. The statistics of the RDF dataset are given in Table 4.

Patty relation phrase dataset [18] contains a large resource for textual patterns that denote binary relations between entities. We use two different relation phrase datasets, wordnet-wikipedia and freebase-wikipedia, in our experiments. The former has 350,568 relation phrases and the latter has 1,631,530 relation phrases. For each relation phrase, Patty also provides a set of supporting entity pairs. More details can be found at Table 5.

Table 3: Time Complexity of Each Step in Our Method

Offline	Online					
Building Paragraph Dictionary	Question Understanding				Query Evaluation	
	building dependency tree	finding relation embedding	finding arguments	build Q^S	Phrase mapping	Subgraph matching
$O(\|T\| \times \|V\|^2 \times d^2)$	$O(\|Y\|^3)$	$O(\|Y\|^2)$	$O(\|Y\|)$	$O(\|Y\|)$	$O(\|V\| + \|E\|)$	NP-hard

Table 6: A Sample of Textual Patterns and Predicates/Predicate Paths in DBpedia

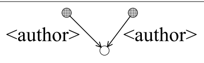
Relation Phrases	Predicates \Predicate Paths	Confidence Probability
"was married to"	<spouse>	1.00
"was born in"	<birthplace>	1.00
"mother of"	<parent>	0.95
"are located in"	<locatedInArea>	0.98
"is fed by"	<inflow>	1.00
"open in"	<locationCity>	1.00
"is coauthor of"	 <author> <author>	1.00

Table 7: Running Time of Offline Processing

	$\theta = 2$	$\theta = 4$
wordnet-wikipedia	17 mins	3.88 hrs
freebase-wikipedia	119 mins	30.33 hrs

6.2 Offline Performance

Exp 1. Precision of Paraphrase Dictionary. In this experiment, we evaluate the accuracy of our building paraphrase dictionary method. For each relation phrase, we output a list of predicates/predicate paths. They are ranked in the non-descending order of confidence probabilities. Table 6 shows a sample of outputs in DBpedia. Note that the confidence probabilities in Tables 6 are normalized.

In order to measure the accuracy, we perform the following experiments. We randomly select 1000 relation phrases from wordnet-wikipedia and freebase-wikipedia datasets, respectively. For each relation phrase, we output top-3 corresponding predicates/predicate paths. These results are shown to human judges. For each relation phrase and its corresponding predicate/predicate path, the judge has to decide a scale from 2 to 0. The result is correct and highly relevant(2), correct but less relevant (1), or irrelevant (0). We find the precision (P@3) is about 50% when the path length is 1. However, while increasing of path length (from 2 to 4), the precision goes down greatly. To guarantee the precision of the paraphrase dictionary for online process, the top-3 predicate paths (for each relation phrase) should go through a human verification process. To further improve the precision of mapping relation phrases to predicate paths, a supervised learning and constraint-based path finding is a possible technique to be explore, which we will study as our future work.

Exp 2. Running Time of Offline Processing. In this experiment, we evaluate the efficiency of our approach. Table 7 shows the total offline time. For example, when the path length threshold $\theta = 2$, the running time is 17 minutes using wordnet-wikipedia relation phrase dataset and DBpedia RDF graph. Obviously, with the increasing of path length, the running time is increasing as well. On the other hand, with the increasing of path length, the precision of the results is decreasing. As default, we set $\theta = 4$. The predicate paths with length longer than 4 will not be considered in our method.

6.3 Online Performance

QALD is the only benchmark for the RDF Q/A problem. It includes both the underling RDF knowledge base and the natural language questions. QALD is based on DBpedia knowledge base. To enable the comparison with other systems in QALD-3 competition, we report the experiment results in QALD-3 format in Table 8. We also compare with one state-of-the-art algorithm DEANNA [29] in our experiments.

In our experiments, we set $k = 10$ in finding top-k subgraph matches. If all top-10 results are correct, we say that we correctly answer the questions. The gold standards of QALD-3 are provided by the organizers of the QALD competition. Note that some questions have less than 10 results. In this case, if all results reported by our system are correct, we say that we correctly answer the questions.

Exp 3. (End-to-End Performance) We use all QALD-3 test queries (99 questions) in our experiments. We show the experiment results in QALD-3 result format (in Tables 8) to enable the comparison with all systems in QALD-3 (the experiment result of QALD-3 campaign is available at [5]). We also compare with DEANNA [29] and report the experiment results in Table 8. We report the query result (i.e., precision, recall, F-measure) of each question in the same format with QALD-3 result format in the full version of this paper [1].

Effectiveness Evaluation. Our method can answer 32 questions correctly. We report the 32 questions in Table 11 together with the total response time in our method. Furthermore, our system can answer 11 questions partially. DEANNA [29] can only answer 21 questions correctly. Our system can beat all systems in QALD-3 campaign in precision except for squall2sparql. Although the best system in QALD-3 campaign is squall2sparql, which can answer 77 questions, the input to squall2sparql is a *controlled English question* rather than a natural language question. Users need to specify the precise entities and predicates (denoted by URIs) in the question. For example, "Who is the dbp:father of res:Elizabeth II?" is an example input to squall2sparql system. In the question, "dbp:father" and "res:Elizabeth II" are URIs of the corresponding predicates and entities. Obviously, it is different from the traditional Q/A system. Therefore, we do not compare our method with squall2sparql. The second best system, CASIA, can answer 30 queries correctly [6]. More details are shown in Table 8.

Efficiency Evaluation. We compare the running time of our approach with DEANNA [29] and CASIA. Due to the space limit, we only report the comparison with DEANNA in Figure 6. We test all questions that can be answered by both DEANNA and our method. In the question understanding, DEANNA needs to generate SPARQLs, our system generates semantic query graph Q^s. The former has the exponential time complexity, but our method has the polynomial time complexity in the question understanding stage, as discussed in Section 5. The experiment results in Figure 6 confirm that generating SPARQLs is an expensive operation, since it needs to address the disambiguation issue. The question understanding of DEANNA often needs a few of seconds in our

[5]http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/3/qald3_results.pdf

Table 8: Evaluating QALD-3 Testing Questions (on DBpedia)

	Processed	Right	Partially	Recall	Precision	F-1
Our Method	76	32	11	0.40	0.40	0.40
squall2sparql	96	77	13	0.85	0.89	0.87
CASIA	52	29	8	0.36	0.35	0.36
Scalewelis	70	1	38	0.33	0.33	0.33
RTV	55	30	4	0.34	0.32	0.33
Intui2	99	28	4	0.32	0.32	0.32
SWIP	21	14	2	0.15	0.16	0.16
DEANNA	27	21	0	0.21	0.21	0.21

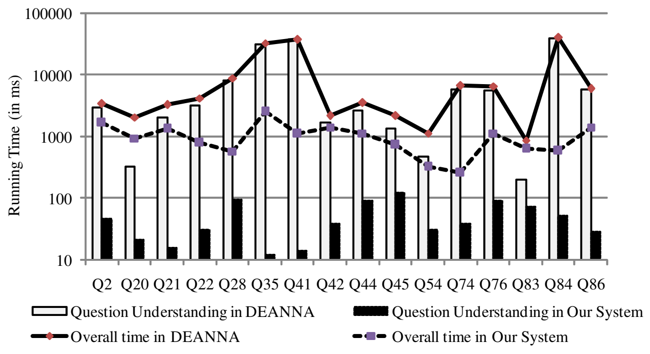


Figure 6: Online Running Time Comparison

experiments. However, our method only spends no more than 100 ms in the question understanding stage (i.e., generating semantic query graph). Furthermore, the total response time of our method is faster than DEANNA by 2-68 times.

Exp 4. (Evaluating Heuristic Rules For Finding Associated Arguments) In this experiment, we evaluate the effectiveness of the four heuristic rules in Section 4.1.2. In Table 9, we show the effects of the four heuristic rules on finding associated arguments. We also report the number of questions that can be answered correctly in our method using the four rules or without the four rules in Table 9. Table 9 shows that we can find the associated arguments in 48 questions correctly if using the four rules, but, the number is 32 without using the four rules. Table 9 also shows that we can answer more questions correctly using the four rules than without using that. Thus, the four heuristic rules can help improve the precision of our method.

Exp 5. (Failure Analysis) We provide the failure analysis of our method. There are three key reasons for the failure of some questions in our method. The first reason is the named entity linking problem. For example, "In which UK city are the headquarters of the MI6 ". We fail in linking MI6 to the corresponding entity. The second one is the failure of the semantic relation extraction. The third one is that our method cannot answer some aggregation (such as Max/Min) questions. They should be translated to SPARQLs with FILTER, like "ORDER BY DESC(?x) OFFSET 0 LIMIT 1". We give the ratio of each reason in Table 10. Also, we give an example of each reason in Table 10. The failure analysis will help the improvement of our RDF Q/A system's precision.

7. RELATED WORK

Q/A (natural language question answering) has a quite long history since the seventies of last century [24]. Compared to keyword search, users can imply semantic relationships between the keywords using a whole sentence. Generally, there are three different categories of QA systems: (1) Text-based QA systems [20, 22] first retrieve a set of documents that are most relevant to the question, and then extract the answers from these documents. (2) Collaboration-based QA systems [28] exploit answers from the similar questions which have been answered by users on collaborative QA platforms, such as Yahoo! Answer and Quora. (3) Structured-data-based QA systems find answers by searching the database instead of the corpus, where the natural language questions are usually translated into some structural queries, such as SQL [19], SPARQL [29, 27] and others [3]. [27] pareses the questions to SPARQL templates, and instantiate the templates by entity/predicate mapping. DEANNA [29] builds a disambiguation graph and reduces disambiguation as an integer linear programming problem. Our work belongs to the third category and differs from previous work in three points.

Table 9: Evaluating The Heuristic Rules

	without the four rules	**using the four rules**
finding arguments correctly	32	48
answering questions correctly	21	32

Table 10: Failure Analysis

Reason	#(Ratio)	Sample Example
Entity Linking Failure	17 (27%)	Q48: In which UK city are the headquarters of the MI6?
Relation Extraction Failure	14 (22%)	Q64. Give me all launch pads operated by NASA.
Aggregation Query	22 (35%)	Q13. Who is the youngest player in the Premier League?
Others	10 (16%)	Q37. Give me all sister cities of Brno.

First, different from [27], our method does not adopt any manually defined SPARQL templates.

Second, different from most existing RDF Q/A systems, such as [27] and DEANNA [29], we push down the disambiguation into the query evaluation stage. Existing solutions, like [27] and [29], generate the SPARQLs as the intermediate results in the question understanding stage. Obviously, they need to do disambiguation in this step. For example, DEANNA [29] proposes an integer linear programming (ILP)-based method to address the disambiguation issue. As we know, ILP is a classical NP-hard problem. Then, in the query evaluation stage, the existing methods need to answer these generated SPARQL queries. Answering SPARQL queries equals to finding subgraph matches of query graphs Q over RDF graph [33], which is also an NP-hard problem.

We compare our method with one state-of-the-art method DEANNA in [29] from the complexity viewpoint. Table 12 shows the time complexity of each stage in both DEANNA and our method. The time complexity of the question understanding stage in our method is polynomial, but it is NP-hard in [29]. Generally speaking, although both [29] and our method are NP-hard in the online phrase (since the query evaluation stage dominates the whole complexity), our method is much faster than [29] in practice. The complexity of our method lies in finding subgraph matches. However, lots of literatures discuss the optimization methods for subgraph search [32, 26]. Therefore, our method has better performance than existing work. The comparison experiment results have been given in Exp 4 (in Section 6.3) and Table 7.

Although the idea of combining disambiguation and query evaluation together has been considered recently, such as [12] for keyword search and [23] for entity search over the textural corpus, their problem definitions are different from ours. We focus on the natural language question answering on RDF repositories.

Third, our method can consider answer some questions that contains the semantic relationships which are represented by a *path* in the RDF graph, rather than a single edge. However, existing systems, such as [33] and DEANNA [29], only consider mapping the relation phrase to single predicates.

Furthermore, there are some work studying finding paths satisfying a regular expression between two vertices [14]. Note that this

Table 11: The QALD-3 Questions that can be Answered Correctly in Our System

ID	Questions	Response Time (in ms)
Q2	Who was the successor of John F. Kennedy?	1699
Q3	Who is the mayor of Berlin?	677
Q14	Give me all members of Prodigy?	811
Q17	Give me all cars that are produced in Germany ?	297
Q19	Give me all people that were born in Vienna and died in Berlin ?	2557
Q20	How tall is Michael Jordan ?	942
Q21	What is the capital of Canada ?	1342
Q22	Who is the governor of Wyoming ?	796
Q24	Who was the father of Queen Elizabeth II?	538
Q27	Sean Parnell is the governor of which U.S. state ?	1210
Q28	Give me all movies directed by Francis Ford Coppola.	577
Q30	What is the birth name of Angela Merkel ?	250
Q35	Who developed Minecraft ?.	2565
Q39	Give me all companies in Munich.	1312
Q41	Who founded Intel?	1105
Q42	Who is the husband of Amanda Palmer ?	1418
Q44	Which cities does the Weser flow through ?	1139
Q45	Which countries are connected by the Rhine ?	736
Q54	What are the nicknames of San Francisco ?	321
Q58	What is the time zone of Salt Lake City ?	316
Q63	Give me all Argentine films.	427
Q70	Is Michelle Obama the wife of Barack Obama ?	316
Q74	When did Michael Jackson die ?	258
Q76	List the children of Margaret Thatcher.	1139
Q77	Who was called Scarface?	719
Q81	Which books by Kerouac were published by Viking Press ?	796
Q83	How high is the Mount Everest ?	635
Q84	Who created the comic Captain America ?	589
Q86	What is the largest city in Australia ?	1419
Q89	In which city was the former Dutch queen Juliana buried ?	1700
Q98	Which country does the creator of Miffy come from ?	2121
Q100	Who produces Orangina ?	367

Table 12: Online Time Complexity Comparison

Different Approaches	Online	
	Question Understanding	Query Evaluation
Our Method	$O(\|V\|^3)$	NP-hard
DEANNA [28]	NP-hard (ILP problem)	NP-hard

is different from our path finding problem, since we focus on finding all simple paths between a pair of vertices under the path length constraint θ.

8. CONCLUSIONS

In this paper, we propose a whole graph data-driven framework to answer natural language questions over RDF graphs. Different from existing work, we allow the ambiguity in the question understanding stage. We push down the disambiguation into the query evaluation stage. Based on the query results over RDF graphs, we can address the ambiguity issue efficiently. In other words, we combine the disambiguation and query evaluation in an uniform process. Consequently, the graph data-driven framework not only improves the precision but also speeds up the whole performance of RDF Q/A system.

Acknowledgments. Lei Zou's work was supported by National Science Foundation of China (NSFC) under Grant No. 61370055, Beijing Higher Education Young Elite Teacher Project (YETP0016) and CCF-Tencent Open Research Fund. Jeffery Xu Yu was supported by grant of the RGC of Hong Kong SAR, No. CUHK 418512. Dongyan Zhao was supported by NSFC under Grant No. 61272344, 61202233 and China 863 Project under Grant No. 2012 AA011101. Authors thank Shuo Han (PhD student in Peking University) for some efforts in the experiments. Authors are also grateful to anonymous reviewers for their constructive comments.

9. REFERENCES

[1] Natural language question answering over rdf. In *technique report, omitted due to the double-blind reviewing. Some materials are provided in our response document*.
[2] I. Androutsopoulos and P. Malakasiotis. A survey of paraphrasing and textual entailment methods. *J. Artif. Intell. Res. (JAIR)*, 38:135–187, 2010.
[3] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544, 2013.
[4] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *J. Web Sem.*, 7(3):154–165, 2009.
[5] D. M. Cer, M.-C. de Marneffe, D. Jurafsky, and C. D. Manning. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In *LREC*, 2010.
[6] P. Cimiano, V. Lopez, C. Unger, E. Cabrio, A.-C. N. Ngomo, and S. Walter. Multilingual question answering over linked data (qald-3): Lab overview. In *CLEF*, pages 321–332, 2013.
[7] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372, 2004.
[8] M.-C. de Marneffe and C. D. Manning. Stanford typed dependencies manual.
[9] J. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *COLING*, pages 340–345, 1996.
[10] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *EMNLP*, pages 1535–1545, 2011.
[11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.
[12] G. Ladwig and T. Tran. Combining query translation with query answering for efficient keyword search. In *ESWC (2)*, pages 288–303, 2010.
[13] V. Lopez, C. Unger, P. Cimiano, and E. Motta. Evaluating question answering over linked data. *J. Web Sem.*, 21:3–13, 2013.
[14] K. Losemann and W. Martens. The complexity of regular expressions and property paths in sparql. *ACM Trans. Database Syst.*, 38(4):24, 2013.
[15] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, 2008.
[16] R. Mihalcea and A. Csomai. Wikify!: linking documents to encyclopedic knowledge. In *CIKM*, pages 233–242, 2007.
[17] N. Nakashole, G. Weikum, and F. M. Suchanek. Discovering and exploring relations on the web. *PVLDB*, 5(12):1982–1985, 2012.
[18] N. Nakashole, G. Weikum, and F. M. Suchanek. Patty: A taxonomy of relational patterns with semantic types. In *EMNLP-CoNLL*, pages 1135–1145, 2012.
[19] A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM, 2003.
[20] D. R. Radev, H. Qi, Z. Zheng, S. Blair-Goldensohn, Z. Zhang, W. Fan, and J. M. Prager. Mining the web for answers to natural language questions. In *CIKM*, pages 143–150, 2001.
[21] L.-A. Ratinov, D. Roth, D. Downey, and M. Anderson. Local and global algorithms for disambiguation to wikipedia. In *ACL*, pages 1375–1384, 2011.
[22] D. Ravichandran and E. Hovy. Learning surface text patterns for a question answering system. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 41–47, 2002.
[23] U. Sawant and S. Chakrabarti. Learning joint query interpretation and response ranking. In *WWW*, pages 1099–1110, 2013.
[24] R. F. Simmons. Natural language question-answering systems: 1969. *Commun. ACM*, 13(1):15–30, Jan. 1970.
[25] W. M. Soon, H. T. Ng, and D. C. Y. Lim. A machine learning approach to coreference resolution of noun phrases. *Comput. Linguist.*, 27(4):521–544, 2001.
[26] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *PVLDB*, 5(9):788–799, 2012.
[27] 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.
[28] Y. Wu, C. Hori, H. Kawai, and H. Kashioka. Answering complex questions via exploiting social q&a collection. In *IJCNLP*, pages 956–964, 2011.
[29] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural language questions for the web of data. In *EMNLP-CoNLL*, pages 379–390, 2012.
[30] M. Yahya, K. Berberich, S. Elbassuoni, and G. Weikum. Robust question answering over the web of linked data. In *CIKM*, pages 1107–1116, 2013.
[31] W. Zhang, J. Su, C. L. Tan, and W. Wang. Entity linking leveraging automatically generated annotation. In *COLING*, pages 1290–1298, 2010.
[32] P. Zhao and J. Han. On graph query optimization in large networks. *PVLDB*, 3(1):340–351, 2010.
[33] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao. gstore: Answering sparql queries via subgraph matching. *PVLDB*, 4(8):482–493, 2011.