Answering Natural Language Questions by Subgraph Matching over Knowledge Graphs

Sen Hu[1], Lei Zou[1], Jeffrey Xu Yu[2], Haixun Wang[3], Dongyan Zhao[1]
[1]*Peking University, China;* [2]*The Chinese University of Hong Kong, China;* [3]*Facebook, USA*
[1]{husen,zoulei,zhaody}@pku.edu.cn, [2]yu@se.cuhk.edu.hk, [3]haixun@google.com

Abstract—RDF question/answering (Q/A) allows users to ask questions in natural languages over a knowledge base represented by RDF. To answer a natural 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. More specifically, we propose two different frameworks to build the semantic query graph, one is relation (edge)-first and the other one is node-first. 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.

Index Terms—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 [**?**]. Therefore, RDF question/answering (Q/A) systems have received wide attention in both NLP (natural language processing) [**?**], [**?**] and database areas [**?**].

Generally, 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 [**?**], and in the second stage evaluate all SPARQLs translated in the first stage. The focus of the existing solutions is on question understanding. Let us consider a running example in Figure **??**. The RDF dataset is given in Figure **??**(a). Given a natural language question N_1="What is the budget of the film directed by Paul Anderson?", it is first interpreted as a SPARQL query that is evaluated to get the answers (as shown in Figure **??**(b)).

1.1 Motivation

The inherent hardness of RDF Q/A lies in the *ambiguity* of *unstructured* natural language question sentences. Generally, there are two main challenges.

Phrase Linking. A natural language phrase ws_i may have several meanings, i.e., ws_i correspond to several semantic items in RDF graph G. As shown in Figure **??**(b), the entity phrase "Paul Anderson" can map to three persons ⟨Paul_Anderson_(actor)⟩, ⟨Paul_S._Anderson⟩ and ⟨Paul_W._S._Anderson⟩. For a relation phrase, "directed by" also refers to two possible predicates ⟨director⟩ and ⟨writer⟩. Sometimes a phrase needs to be mapped to a non-atomic structure in knowledge graph. For example, "uncle of" refers to a predicate path (see Table **??**). In RDF Q/A systems, we should eliminate "the ambiguity of phrase linking".

Composition. The task of composition is to construct corresponding query or query graph by assembling the identified phrases. In the running example, we know the predicate ⟨director⟩ is to connect subject ⟨film⟩ and object ⟨Paul_W._S._Anderson⟩; consequently, we generate a triple ⟨?film, director, Paul_W._S._Anderson⟩. However, in some cases, it is difficult to determine the correct subject and object for a given predicate, or there may exist several possible query graph structures for a given question sentence. We call it "the ambiguity of query graph structure".

In this paper, we focus on how to address the two challenges. Different from existing solutions that try to solve ambiguity in the question understanding stage, we propose to combine disambiguation (for both phrase linking and query graph construction) and query evaluation together. Specifically, 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 is no match

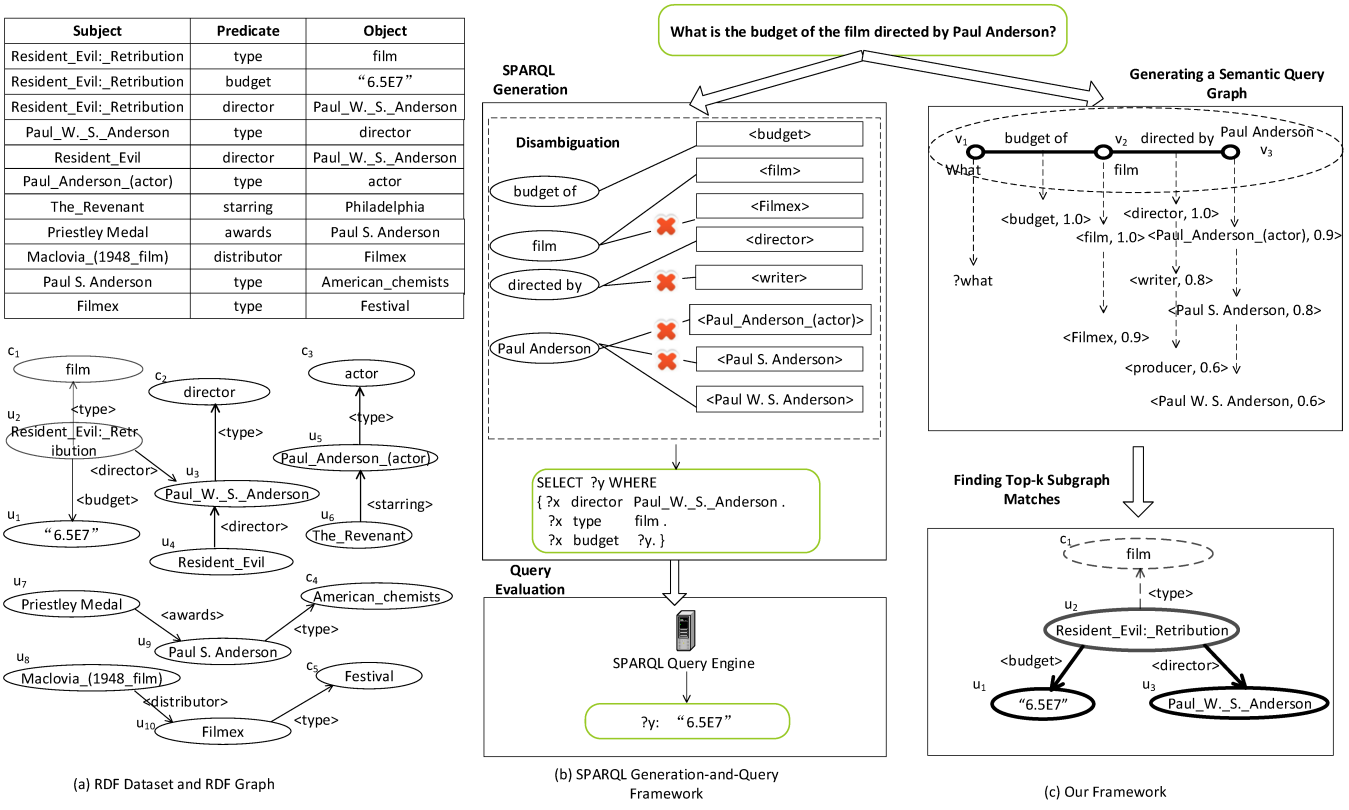


Fig. 1. Question Answering Over RDF Dataset

found. We call this as the *graph data-driven* approach for RDF Q/A. We illustrate the intuition of our method by an example.

Example 1. Consider a subgraph of graph G in Figure **??**(a) (the subgraph induced by vertices u_1, u_2, u_3 and c_1). Edge $\overrightarrow{u_2c_1}$ says that "Resident Evil: Retribution is a film". Edge $\overrightarrow{u_2u_1}$ says that "The budget of Resident Evil: Retribution is $ 65 million". Edge $\overrightarrow{u_2u_3}$ says that "Paul W. S. Anderson directed the film Resident Evil: Retribution". The natural language question N_1 is "What is the budget of the film directed by Paul Anderson". Obviously, the subgraph formed by edges $\overrightarrow{u_2c_1}$, $\overrightarrow{u_2u_1}$ and $\overrightarrow{u_2u_3}$ is a *match* of N_1. "6.5E7" is a correct answer. On the other hand, we cannot find a match (of N_1) containing $\langle$Paul_Anderson_(actor)$\rangle$ inc G, i.e., the phrase "Paul Anderson" (in N_1) cannot map to $\langle$Paul_Anderson_(actor)$\rangle$. Therefore, we address the ambiguity issue of phrase linking when the matches are found. We can also resolve the ambiguity of query graph structure following the same idea. More details will be discussed in Section **??**.

The above example illustrates the intuition of our graph data-driven approach. A fundamental issue in 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 them. Therefore, we propose a semantic query graph Q^S (defined in Definition **??**) to represent the question semantics of N. An example of Q^S is given in Figure **??**(c), which represents the semantic of the question N. Answering natural language question equals to finding matches of Q^S over the underlying RDF graph G. To build Q^S, we propose two different frameworks: relation (edge)-first and node-first.

1.2 Our Approach

Although there are still two stages "question understanding" and "query evaluation" in our method, we do not generate SPARQL at the question understanding step as existing solutions do. As we know, a SPARQL query can also be represented as a query graph, which does not include any ambiguity. Instead, our method builds a query graph that represents users' query intention, but it allows for the ambiguity at the question understanding stage, such as the ambiguity of phrase linking and query graph structure. We resolve the ambiguity when the matches are found at the query evaluation.

In the first framework, we first extract semantic relations based on the dependency tree structure of question sentences to build a semantic query graph Q^S. A *semantic relation* is a triple $\langle rel,arg1,arg2\rangle$, where rel is a relation phrase, and $arg1$ and $arg2$ are its associated node phrases. For instance, $\langle$"directed by","film","Paul Anderson"$\rangle$ is a semantic relation. In Q^S, two edges share one common endpoint if the two corresponding relations share one common node phrase. Each node (entity/class mention) and edge (relation mention) in Q^S may have multiple candidates. The first framework addresses the ambiguity of phrase linking when the *matches* (see Definition **??**) of Q^S are found. Note that the first framework does not address the ambiguity of query graph's structure and assumes that the query graph can be *uniquely* fixed at the question understanding step.

The second framework takes another perspective. When there exist some implicit or uncertain relations in N, the relation-first framework often fails to extract such relations. Therefore, the second framework starts with extracting nodes from the question sentence N and connects these nodes to form a query graph. Furthermore, different from the relation-first framework, the node-first framework allows for the ambiguity of query graph structure

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 (of N)
Q^S	The Semantic Query Graph (of N)
Q^U	The Super Semantic Query Graph (of N)
Y	The dependency tree (of N)
D^E/D^R	The entity/relation mention dictionary
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 $\overline{v_iv_j}$

at the beginning. It does not intend to build Q^S in the question understanding step. Instead, it builds a super graph Q^U of Q^S that includes uncertain edges. To match Q^U over the underlying RDF graph G, we allow for mismatching some edges in Q^U, i.e., *approximate match* (Definition **??**). We resolve the ambiguity of phrase linking and query graph structure together when the approximate matches are found. Actually, the approximate matching position (in RDF graph G) defines the semantic query graph Q^S that we aim to build. In other words, we push down resolving the ambiguity of Q^S's structure to the query evaluation stage.

In a nutshell, we make the following contribution.

1) We propose two graph data-driven frameworks for RDF Q/A task, different from exiting solutions, in which the disambiguation and query evaluation are combined together. In the first framework, we address ambiguity of phrase linking at the query evaluation; while in the second framework, the ambiguity of phrase linking and query graph's structure are both resolved. The graph data-driven frameworks not only improve the precision but also speed up query processing time greatly.
2) In the offline processing, we propose a graph mining algorithm to build a relation mention dictionary, i.e., mapping natural language phrases to possible predicates, which is used for question understanding in RDF Q/A.
3) In the online processing, in order to speed up query evaluation, we propose efficient top-k (approximate) graph matching algorithms of matching Q^S and Q^U over RDF graph.
4) We conduct extensive experiments over several real RDF datasets (including QALD benchmark and WebQuestions benchmark) and compare our system with some state-of-the-art systems. The performance of our approach beat the other systems on QALD benchmark while close to the best on the WebQuestions benchmark.

2 Overview

The problem of this paper is to find the answers to a natural language question N over a RDF graph G. Table **??** lists the notations used throughout this paper.

There are two key issues in RDF Q/A problem. The first one is how to represent the query intention of the natural language question N in a structural way. The second one is how to address the ambiguity of natural language N. In this paper, we focus on the ambiguity of phrase linking and query graph structure (composition) that are mentioned in Section **??**.

2.1 Semantic Query Graph

We define a semantic query graph (Definition **??**) to represent the query intention of the question N in a graph structured way.

Definition 1. **(Semantic Query Graph)**. A *semantic query graph* (denoted as Q^S) is a graph, in which each vertex v_i is associated with an entity phrase, class phrase or wild-cards in the question sentence N; and each edge $\overline{v_iv_j}$ is associated with a relation phrase in the question sentence N, $1 \leq i, j \leq |V(Q^S)|$.

Given the question sentences N_1, the corresponding semantic query graphs Q_1^S is given in Figure **??**(b). In Q_1^S, nodes v_1, v_2 and v_3 are associated with "what" (wild-card), "film" (a class phrase) and "Paul Anderson" (an entity phrase), respectively. The relation phrase "(be) budget of " denotes the relation between v_1 and v_2, as well as the relation phrase "directed by" between v_2 and v_3.

As mentioned in the introduction, we want to find a "match" of the semantic query graph Q^S over RDF graph G. When the matches are found, we resolve the ambiguity of natural language question sentence; meanwhile we find the answers to the question. Generally, a "match" is defined based on subgraph isomorphism. Given a node v_i in a semantic query graph Q^S, if v_i is an entity phrase or a class phrase, we can use entity linking algorithm [**?**] to retrieve all entity/class (in RDF graph G) that possibly correspond to v_i, denoted as $C(v_i)$; if v_i is a wild-card (such as wh-word), we assume that $C(v_i)$ contains all vertices in RDF graph G. Analogously, each edge $\overline{v_iv_j}$ in Q^S also maps to a list of candidate predicates, denoted as $C_{\overline{v_iv_j}}$. Consider the semantic query graph Q^S in Figure **??**(b). We also visualizes the candidates for each vertex and edge in Q^S in Figure **??**(c). For example, v_3 ("Paul Anderson") corresponds to ⟨Paul_Anderson_(actor)⟩, ⟨Paul_S._Anderson⟩ and ⟨Paul_W._S._Anderson⟩; and edge "$\overline{v_2v_3}$" maps to ⟨director⟩, ⟨writer⟩ and ⟨producer⟩. Formally, we define the match as follows.

Definition 2. (Match) Consider a semantic query graph Q^S with n nodes $\{v_1,...,v_n\}$. Each node v_i has a candidate list C_{v_i}, $i = 1, ..., n$. Each edge $\overline{v_iv_j}$ also has a candidate list $C_{\overline{v_iv_j}}$, 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 ⟨u_i rdf:type c_i⟩ in RDF graph) and c_i must be in C_{v_i}; and
3) $\forall \overline{v_iv_j} \in Q^S \Leftrightarrow \overrightarrow{u_iu_j} \in M \vee \overrightarrow{u_ju_i} \in M$. Furthermore, the predicate P_{ij} associated with $\overrightarrow{u_iu_j}$ (or $\overrightarrow{u_ju_i}$) is in $C_{\overline{v_iv_j}}$, $1 \leq i, j \leq n$.

Let us see Figure **??**. The subgraph included by vertices c_1, u_1, u_2 and u_3 (in RDF graph G) denotes a *match* of semantic query graph Q^S in Figure **??**(b). When the matches are found, we resolve the ambiguity, e.g., "Paul Anderson" should refer to ⟨Paul_W._S._Anderson⟩ rather than others., meanwhile that we find the answers to the question, i.e., "6.5E7"[1] is the film budget.

The core of our graph data-driven solution lies in two aspects: one is how to build a semantic query graph Q^S *accurately* and the other one is how to find matches *efficiently*. In order to address the above issues, we propose two different frameworks. The first

1. 65 million

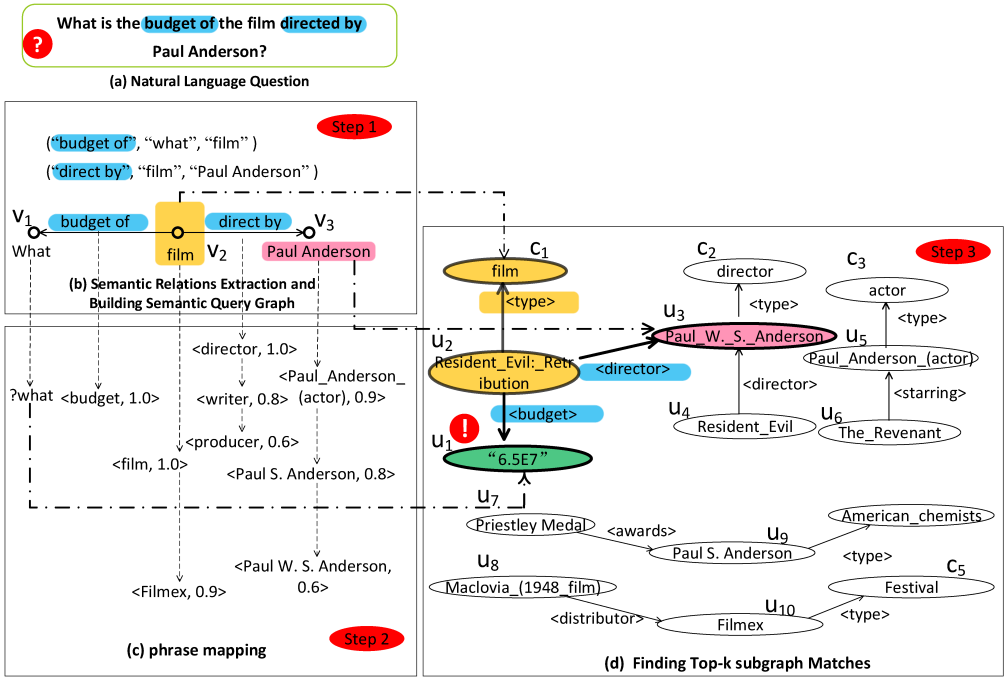


Fig. 2. Question Answering with Semantic Query Graph in Relation-first framework

one is called "relation (edge)-first". It means that we always extract relations from the natural language question sentence N and represent them as edges. Then, we assemble these edges to form a semantic query graph. The second framework takes another perspective, called "node-first". It starts with finding nodes (entity/class phrases and wild-cards) and try to introduce edges to connect them to form a semantic query graph Q^S. Furthermore, another major difference between the two frameworks is that the node-first framework defines a super graph (called Q^U) of Q^S when there exist some implicit or uncertain relations in the question sentence. In other words, the node-first framework is not to fix the Q^S's structure before subgraph matching evaluation as the relation-first framework does.

2.2 Relation-first Framework

Given a natural language question sentence N, the relation-first framework begins with extracting *semantic relations* (edge together with two end points) from N.

Definition 3. **(Semantic Relation)**. A *semantic relation* is a triple $\langle rel, arg1, arg2\rangle$, where rel is a relation mention, $arg1$ and $arg2$ are the two node phrases.

In the running example, ⟨"directed by", "film","Paul Anderson"⟩ is a semantic relation, in which "directed by" is a relation mention (phrase), "who" and "actor" are its associated node phrases. We can also find another semantic relation ⟨"budget of", "what","film"⟩ from the question sentence N_1.

2.2.1 Question Understanding

The goal of the question understanding in the first framework is to build a semantic query graph Q^S for representing users' query intention in N. Specifically, we first extract all semantic relations in N, each of which corresponds to an edge in Q^S. The semantic relation extraction is based on the dependency tree of users' question sentence and a relation mention dictionary (see more details in Section **??**). If the two semantic relations have one common node, they share one endpoint in Q^S. In the running example, we get two semantic relations, i.e., ⟨"directed by", "film","Paul Anderson"⟩ and ⟨"budget of", "what","film"⟩, as shown in Figure **??**. They can be combined through the common node phrase "film" as showed in Figure **??**(c). In addition, if two node phrases refer to same thing after "coreference resolution" [**?**], we also combine the corresponding two semantic relations.

2.2.2 Query Executing

As mentioned earlier, a semantic query graph Q^S is a structural representation of N. In order to answer N, we need to find subgraphs of RDF graph G that *match* Q^S. The match is defined according to the subgraph isomorphism (see Definition **??**)

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

2.3 Node-first Framework

The relation-first framework has two main obstacles. The first is that some relations are difficult to be extracted. If the relation does not explicitly appeared in the question sentence, it is difficult to extract such semantic relations, since our relation extraction relies on the relation mention in the relation mention dictionary. Let us consider two examples "show me all films started by a Chinese actor", "show me all films stared by an actor who was born in China". Obviously, the latter question has one explicit relation mention "(be) born in", where the relation in the former one is implicitly mentioned. Therefore, it is difficult to extract these implicit relations. Secondly, in the relation-first framework, semantic relation extraction relies on the syntactic dependency tree of users' question sentence and heuristic linguistic rules. If the syntactic

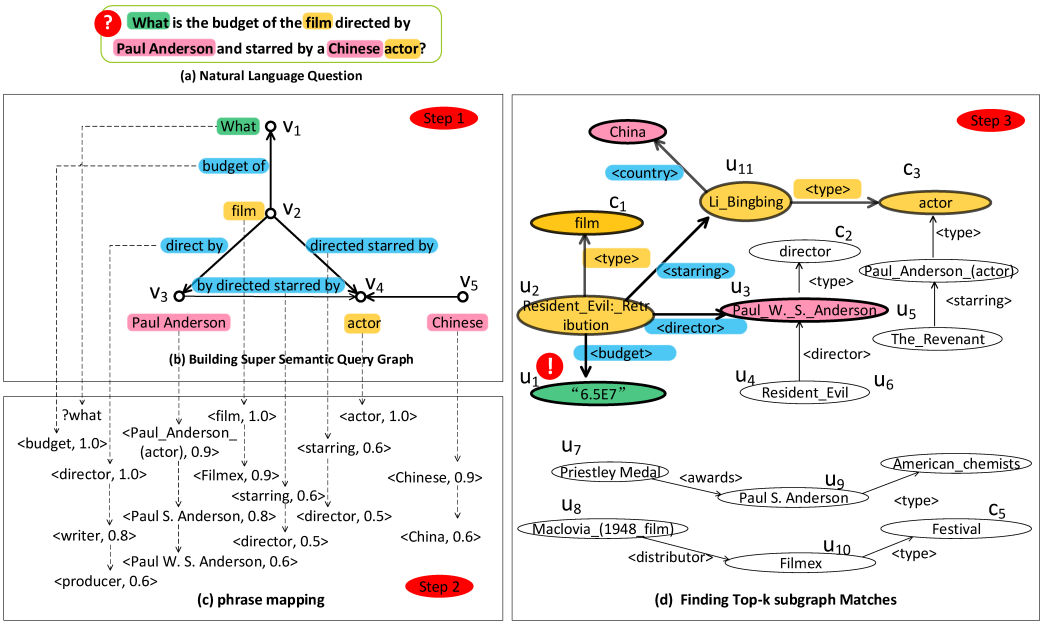


Fig. 3. Question Answering with Super Semantic Query Graph in Node-first Framework

dependency tree has some mistakes, it inevitably leads to wrong semantic query graph Q^S's structure and wrong answers.

Considering the above two obstacles, we design a *robust* framework even in the presence of implicit relations and mistakes in the dependency parse tree. There are two key points in the second framework:

1) The first step is to extract node phrases (such as entity phrase, class phrase and wh-words) from the question sentence N, instead of relation extraction in the first framework.
2) We do not intend to build a semantic query graph Q^S at the question understanding step. Instead, we build a super semantic query graph Q^U, which possibly has some uncertain or implicit relations (i.e. edges). In other words, we allows the structure ambiguity of query graph in the question understanding step, which will be resolved at the query evaluation step.

A *super semantic query graph* Q^U is analogue to Q^S (see Definition **??**), but allows for explicit or uncertain relations (edges).

Definition 4. **(Super Semantic Query Graph).** A *super semantic query graph* (denoted as Q^U) is a graph, in which each vertex v_i is associated with an entity phrase, class phrase or wild-card in the question sentence N; and each edge $\overline{v_i v_j}$ is associated with a relation in N, $1 \leq i, j \leq |V(Q^U)|$. If the relation is explicit, the edge label is the relation mention occurring in N; otherwise, the edge label is empty when the relation is implicit.

The following example illustrates the intuition of the second framework.

Example 2. Consider N_2 in Figure **??**. "What is the budget of the film directed by Paul Anderson and starred by a Chinese actor?". The correct SPARQL query of N_2 has two additional triples than N_1, which are t_1=⟨?film,starring,?actor⟩ and t_2=⟨?actor,country,China⟩. The relation-first framework cannot generate t_2 because the predicate "country" has no explicit relation mention in N_2. In the node-first framework, we introduce an edge between v_4 ("actor") and v_5 ("Chinese") in Figure **??**(b), whose edge label is empty. For detected relation mention "starred by", it is difficult to determine its corresponding two nodes. There are three candidate nodes: "Paul Anderson", "film", and "actor". In Q^U, we introduce two edges between "film" and "actor"; and "Paul Anderson" and "actor". In the query evaluation step, we perform the *approximate match* (defined in Definition **??**) to match Q^U with RDF graph G, i.e., finding the occurrences of Q^U in RDF graph G with (possible) mismatching edges. In this example, the final match is denoted using bold lines in Figure **??**, in which the edge between "Paul Anderson" and "actor" (in Q^U) is not matched.

It is easy to infer that an *approximate match* of Q^U equals to an exact *match* of a connected *spanning subgraph*[2] of Q^U, where the spanning subgraph is the semantic query graph Q^S that we aim to build. Therefore, in the second framework, we fix the semantic query graph Q^S when the matches are found; meanwhile the answers to the question have been found. In other words, we resolve the "structure ambiguity" of query graph at the time the matches are found. We also briefly discuss the two steps of the node-first framework as follows. More technical details are given in Section **??**.

2.3.1 Question Understanding

Given a natural language question sentence N, we first extract all constant nodes from N by applying entity extraction algorithms, which are referred to entities or classes. We also extract all wh-words (such as who, what and which et al.) from N as variable nodes. Then, to build Q^U, we need to introduce an edge between two nodes if there is a semantic relation between them. A naive solution is to introduce an edge between any two nodes. Obviously, this method introduces more noises and ambiguity for the query graph's structure. On the other hand, the *approximate match* in the node-first framework allows mis-matching one or

2. a spanning subgraph for graph Q is a subgraph of Q which contains every vertex of Q.

more edges in Q^U. The naive solution leads to $O(2^n)$ possible matching structures in the final evaluation step, where n is the number of nodes in Q^U. This is quite costly.

To eliminate more noises and reduce the search space, we propose a simple yet effective assumption:

Assumption 1. Two nodes v_1 and v_2 has a semantic relation if and only if there exists *no* other node v^* that occurs in the simple path between v_1 and v_2 of the dependency parse tree of question sentence N.

Although this method also depends on the dependency parse tree, it is not like the first framework, in which, extracting semantic relations and node phrases (to build Q^S) *heavily* depend on the parse tree's structure, POS tag[3] and dependency relation (such as subj, obj and et al.) [4] In other words, the node-first framework (i.e., the second framework) is more robust to dependency errors.

Let us recall Example 2. We first extract five nodes: "what", "film", "Paul Anderson", "Chinese", "actor" from the question N_2. Figure **??** illustrates the dependency parse tree $Y(N_2)$ of question sentence N_2. According to the assumption, we introduce an edge $\overline{v_1v_2}$ between two nodes v_1 and v_2 if there is *no* other node v^* in the simple path between v_1 and v_2 over $Y(N_2)$. The words along the simple path between v_1 and v_2 form the edge label of $\overline{v_1v_2}$. For example, the edge label between nodes "what" and "film" is " (be) budget of". The edge label between nodes "Chinese" and "node" is empty, which is the *implicit* relation. For nodes "Paul Anderson" and "actor", there is no other nodes along the simple path between them. According to Assumption 1, we introduce an edge between them and the edge label is "directed by started by". Due to the same reason, there is another edge between nodes "film" and "actor". Finally, we obtain the super semantic query graph Q^U as shown in Figure **??**(b).

2.3.2 Query Executing

First, we find candidates for each node and edge in Q^U, which is analogue to the query evaluation of Q^S in the first framework. According to the entity mention dictionary D^E, for each node, we can obtain a list of candidate entities, classes. If it is a wh-word, we assume that it can map all vertices in RDF graph G. For each edge label (i.e., the relation mention $rel_{\overline{v_1v_2}}$), we also map it to all possible candidate predicates based on the relation mention dictionary D^R. If the edge label $rel_{\overline{v_1v_2}}$ is empty, e.g., the edge label between nodes "Chinese" and "actor" is empty, we generate candidate predicates by applying a data mining method on G. Section **??** gives more technical details.

Then, based on the data-driven's idea, we try to *match* Q^U over RDF graph G. Different from the *exact* match of Q^S, in the node-first framework, we define the approximate match (allowing dis-matching edges) of super semantic query graph Q^U as follows:

Definition 5. **(Approximate Match)** Consider a super semantic query graph Q^U 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 $C_{\overline{v_iv_j}}$, where $1 \leq i \neq j \leq n$. A subgraph M containing n vertices $u_1, ..., u_n$ in RDF graph G is an *approximate match* of Q^U 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 $\langle u_i rdf : typec_i \rangle$ in RDF graph) and c_i must be in C_{v_i}; and
3. $\forall \overrightarrow{u_iu_j} \in M \Rightarrow \overline{v_iv_j} \in Q^U$. Furthermore, the predicate P_{ij} associated with $\overrightarrow{u_iu_j}$ is in $C_{\overline{v_iv_j}}$, $1 \leq i, j \leq n$.

The only difference between the *approximate match* and *match* is item (3) of Definition **??** and **??**: some edges of Q^U may not be matched. Let us recall Example 2. The final approximate match is denoted by the bold lines in Figure **??**(d). The edge between node "Paul Anderson" and "actor" (in Q^U) is not matched. The approximate match is used to address the ambiguity of the query graph's structure.

TABLE 2
Entity Mention Dictionary D^E

Entity Mention	Referring Entity	Confidence Probability
"Paul Anderson"	⟨Paul_S._Anderson⟩	0.8
"Paul Anderson"	⟨Paul_W._S._Anderson⟩	0.6
"USA"	⟨United_States⟩	1.0
"America"	⟨United_States⟩	1.0
.....	.....	.....

3 Offline Phase

In the offline phase, we build two dictionaries, which are entity mention dictionary D^E and relation mention dictionary D^R. They will be used to extract entities and relations from users' question sentences in the online phase. Note that both D^E and D^R are used in our two frameworks (relation-first framework and node-first framework).

3.1 Build Entity Mention Dictionary

An entity mention is a surface string that refers to entities. For example, "Paul Anderson" could refer to the person ⟨Paul_W._S._Anderson⟩ or ⟨Paul_S._Anderson⟩. We need to build an entity mention dictionary D^E, such as Table **??**, to map entity mentions to some candidate entities with confidence probabilities. There are lots of existing work about entity-mention dictionary construction [**?**], [**?**] and the dictionary-based entity linking [**?**], [**?**]. A popular way to build such a dictionary D^E is by crawling Web pages and aggregating anchor links that point to Wikipedia entity pages. The frequency with which a mention (anchor text), m, links to a particular entity (anchor link), c, allows one to estimate the conditional probability $p(c|m)$ [**?**]. Entity-mention dictionary construction is not our technical contribution, in this paper, we adopt CrossWikis dictionary [**?**], which was computed from a Google crawler of the Web. The dictionary contains more than 175 million unique strings with the entities they may represent.

3.2 Build Relation Mention Dictionary

A relation mention is a surface string that occurs between a pair of entities in a sentence [**?**], such as "be directed by" and "budget of" in the running example. We need to build a relation mention

3. It is called part-of-speech tag, also grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, such as nouns, verbs, adjectives, adverbs, etc.

4. These grammatical relationships (called *dependencies*) that are defined in [**?**]. For example, "nsubj" refers to a nominal subject. It is a noun phrase which is the syntactic subject of a clause.

TABLE 3
Relation Mentions and Supporting Entity Pairs

Relation Mention	Supporting Entity Pairs
"directed by"	(⟨ Resident_Evil ⟩, ⟨ Paul_W._S._Anderson ⟩), (⟨ Roman_Holiday ⟩, ⟨ William_Wyler ⟩),.....
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),.....

TABLE 4
Relation Mention Dictionary D^R

Relation Mention	Predicate or Predicate Path	Confidence Probability
"directed by"	<director>	1.0
"starred by"	<starring>	0.9
"budget of"	<budget>	0.8
"uncle of"	<hasChild> <hasChild> <hasChild>	0.8
… …	… …	… …

dictionary D^R, such as Table **??**, to map relation mentions to some candidate predicates or predicate paths.

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

Suppose that we have a mention set $T = \{rel_1, ..., rel_n\}$, where each rel_i is a relation mention, $i = 1, ..., n$. 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 semantic equivalence with relation mention rel_i.

Given a relation mention rel_i, considering each pair $(v_i^j, v_i'^j)$ in $Sup\ (rel_i)$, we find all simple paths between v_i^j and $v_i'^j$ in RDF graph G, denoted as $Paths(v_i^j\ , v_i'^j)$. Let $PS(rel_i) = \{Paths(v_i^j, v_i'^j) | 1 \leq j \leq m\}$.

For efficiency considerations, we only find simple paths with no longer than a threshold[5]. We adopt a bi-directional BFS (breath-first-search) search from vertices v_i^j and $v_i'^j$ to find $Paths(v_i^j\ , v_i'^j)$. Note that we ignore edge directions (in RDF graph) in a BFS process.

Intuitively, if a predicate path is frequent in $PS(rel_i)$, it is a good candidate that has semantic equivalence with relation mention 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"). Obviously, it is not a good predicate path to represent the semantic of relation mention "uncle of". In order to eliminate noises, we borrow the intuition of tf-idf measure [**?**]. Although (hasGender ,hasGender) is frequent in $PS($"uncle of"), it is also frequent in the path sets of other relation mentions, 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"). 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.

Definition 6. Given a predicate path L , the *tf-value* of L in $PS(rel_i)$ is defined as follows:

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

The *idf-value* of L over the whole relation mention set $T = \{rel_1, ..., rel_n\}$ 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 mention 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 in Appendix A shows the details of finding top-k predicate paths for each relation mention. All relation mentions and their corresponding k predicate paths including tf-idf values are collected to form a relation mention dictionary D^R.

4 Relation-First Framework

4.1 Building Semantic Query Graph

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 mentions in question N. Obviously, we can simply regard 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 *preposition fronting*. For example, consider a question "*In which movies did Li Bingbing star?*". Obviously, "star in" is a relation mention though it is not a subsequence of N. 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 [**?**].

Therefore, in our work, we first apply Stanford Parser [**?**] to N to obtain the dependency tree Y. Let us recall the running example. Figure **??** shows the dependency tree of N_1, denoted as $Y(N_1)$. The next question is to find relation mentions occurring in $Y(N_1)$.

Definition 7. Let us consider a dependency tree Y of a natural language question N and a relation mention 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 mention rel in Y.

5. We set the threshold as 4 in our experiments. More details about the parameter setting will be discussed in Appendix B.

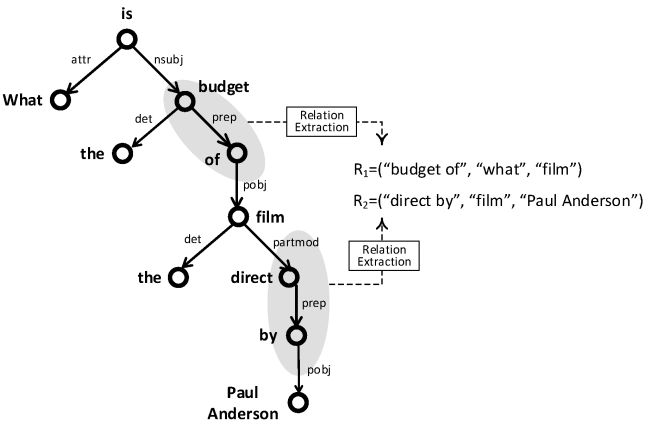


Fig. 4. Relationship Extraction in $Y(N_1)$

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

4.1.1 Relation Recognition

Given a natural language question N, we propose an algorithm (Algorithm 2 in Appendix A) to identify all relation mentions in N. In the offline phase, we build an inverted index over all relation mentions in the relation mention dictionary D^R. Specifically, for each word, it links to a list of relation mentions containing the word. The basic idea of Algorithm 2 is as follows: For each node (i.e., a word) w_i in Y, we find the candidate pattern list PL_i (Lines 1-2). Then, for each node w_i, we check whether there exists a subtree rooted at w_i including all words of some relation mentions in PL_i. In order to address this issue, we propose a depth-first search strategy. We probe each path rooted at w_i (Line 3). The search branch stops at a node w', where there does not exists a relation mention 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 mention rel, it means that we have found a relation mention 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. The time complexity of Algorithm 2 is $O(|Y|^2)$.

4.1.2 Finding Associated Nodes

After finding a relation mention in Y, we then look for the two associated nodes. If a phrase was recognized as entity/class mention , it is regarded as a node. Besides, the nodes are recognized also based on the grammatical subject-like and object-like relations around the embedding, which are listed as follow:

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

Assume that we find an embedding subtree y of a relation mention rel. We recognize arg1 by checking for each phrase w in y whether w is an entity/class mention or 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 node, 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 nodes. 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.
- Rule 2: If the root node of t has subject/object-like relations with its parent node in Y, add the parent node to arg1.
- Rule 3: If the parent of the root node of t has subject-like relations with its neighbors, add the child to arg1.
- Rule 4: If one of arg1/arg2 is empty, add the nearest wh-word or the first noun phrase in t to arg1/arg2.

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

Example 3. Let us recall dependency tree Y in Figure **??**. We get "what" as the first node of relation mention "budget of" by applying Rule 4. And we can find another node "film" as it is a class mention. Therefore, the first semantic relation is ⟨"budget of", "what", "film"⟩. Likewise, we can also find another semantic relation ⟨"direct by", "film", "Paul Anderson"⟩.

After obtaining all semantic relations in a natural language N, we need to build a semantic query graph Q^S. Figure **??**(b) shows an example of Q^S. In order to build a semantic query graph Q^S, 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 node phrase. The formal definition of a semantic query graph has been given in Definition **??**.

4.1.3 Phrases Mapping

In this subsection, we discuss how to map the relation mentions and node phrases to candidate predicates/predicate paths and entities/classes, respectively.

Mapping Edges of Q^S. Each edge $\overline{v_i v_j}$ in Q^S has a relation mention $rel_{\overline{v_i v_j}}$. According to the relation mention dictionary D^R (see Section **??**), it is straightforward to map $rel_{\overline{v_i v_j}}$ to some predicates P or predicate paths L. The list is denoted as $C_{\overline{v_i v_j}}$. 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 **??**). For example, edge $\overline{v_2 v_3}$ has a relation mention $rel_{\overline{v_2 v_3}}$="direct by". Its candidate list $C_{\overline{v_2 v_3}}$ contains three candidates, ⟨director⟩, ⟨writer⟩, and ⟨producer⟩, as shown in Figure **??**(c).

Mapping Vertices of Q^S. Let us consider any vertex v in Q^S. The phrase associated with v is arg. If arg is a wild-card (such as wh-word), it can be mapped to all vertices in RDF graph G. Otherwise, given an constant arg (entity/class mention), we adopt the dictionary-based entity linking approach [**?**] to find the

candidate entities or classes. We use notation C_v to denote all candidates with regard to vertex v in Q^S. For example, "film" in v_2 (in Figure **??**) can be linked to a class node $\langle$film$\rangle$ or an entity node $\langle$Filmex$\rangle$. If arg is mapped to an entity u or a class c, we use $\delta(arg, u)$ or $\delta(arg, c)$ to denote the confidence probability.

4.2 Query Executing

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 **??**. We assume that all candidate lists are ranked in the non-ascending order of the confidence probability. Figure **??**(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 8. Given a semantic query graph Q^S with n nodes $\{v_1,...,v_n\}$, 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:

$$Score(M) = \alpha \sum_{v_i \in V(Q^S)} \log(\delta(arg_i, u_i)) + (1-\alpha) \sum_{\overline{v_i v_j} \in E(Q^S)} \log(\delta(rel_{\overline{v_i v_j}}, P_{ij})) \quad (2)$$

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

The default value of weight α is 0.5, which means the entity score and relation score have equivalent status. If we have enough training data, α can be learned by some ranking models such as SVM-rank [**?**] . Details can be found in Section **??**.

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 [6]. To solve this problem, we designed an enumerative algorithm (Algorithm 3 in Appendix A) with two main pruning methods.

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 **??**. Vertex u_5 is a candidate in C_{v_3}. However, u_5 does not have an adjacent predicate that is mapping to phrase "direct by" 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 [**?**].

The second method is to stop the search process based on the top-k match score as early as possible. Obviously, enumerating all possible combination is inefficient. If we maintain an appropriate enumeration order so that the current matches are always better than undiscovered matches, we can terminate the search space as early as possible. The pseudo codes are given in Algorithm 3 in Appendix A. For ease of presentation, we use "candidate list" to symbol relation candidate list and entity/class candidate list together. Once we determine a candidate for each candidate list in Q^S, we obtain a "selection". The selection is expressed by a n-length vector, which n is the total number of candidate list (Line 2 in Algorithm 3). Initially the vector value is 0 which means we select the first candidate for each candidate list (Lines 3-4). Every time we get the best selection from the heap top of H. We can build a query graph Q^* by replacing all vertex/edge labels in Q^S using the selected candidates (Lines 5-6). Line 7 applies an existing subgraph isomorphism algorithm such as VF2 to find all subgraph matches of Q^* over G. Then we maintain the maximum heap H to guarantee each selection we get from H has the highest score among all untried selection as showed in Line 8-10. For each candidate list L_i, we add one at the i-th bit in current selection Γ to get a new selection Γ and put it into H. Thus we can early termination when we find k matches as showed in lines 11-12 in Algorithm 3.

[6]. 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

5 Node-First Framework

5.1 Building Super Semantic Query Graph

There are three steps in building Super Semantic Query graph Q^U: node recognition, query graph structure construction and phrase mapping.

5.1.1 Node Recognition

The first step is to recognize all nodes from the question sentence N. Generally, we extract entities, classes and wild-cards as *nodes*. We adopt the dictionary-based entity linking approach [**?**] to find entities and classes. We collect all wh-words and nouns which could not map to any entities and classes as wild-cards. For example, given a question sentence N_2="What is the budget of the film directed by Paul Anderson and starred by a Chinese actor?", the node recognition result is illustrated in Figure **??**(a), i.e., "what", "film", "Paul Anderson", "Chinese", "actor".

5.1.2 Structure Construction

Given that all nodes have been recognized, the next step is to build a super semantic query graph Q^U. As mentioned in Section **??**, although our method still relies on the dependency tree of the question sentence, it is more robust to dependency errors compared with the relation-first framework.

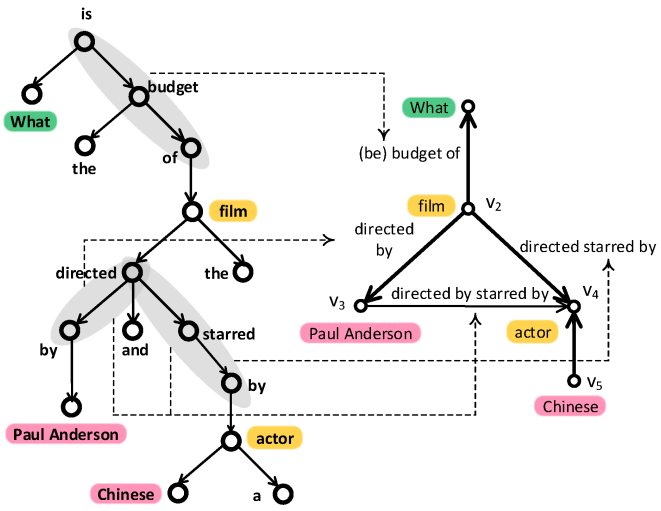


Fig. 5. Building Super Semantic Query Graph

Based on Assumption 1 (see Section **??**), we construct the super semantic query graph Q^U as follows: Given a node set V (which has been recognized in the first step) and a dependency

tree Y of question sentence, for any two nodes v_i and v_j ($\in V$), we introduce an edge between v_i and v_j if and only if the simple path between v_i and v_j does not contain other node in V. We propose a DFS based algorithm (see Algorithm 4 in Appendix A, with time complexity $O(|Y|)$) to find neighbors for each node and build the super semantic query graph Q^U.

For the question sentence N_2, the super semantic query graph Q^U is shown in Figure **??**. The node labels are those associated entity/class mentions or other phrases. The edge label of $\overline{v_iv_j}$ is the words along the simple path between v_i and v_j in the dependency tree $Y(N_2)$. For example, the path between "what" and "film" in the dependency tree contains three words: "is", "budget" and "of", thus, the edge label between v_1 and v_2 (in Q^U) is "(be) budget of ". If the simple path does not contain any word (such at the path between "actor" and "Chinese"), the edge label is empty.

5.1.3 Phrases Mapping

In this subsection, we discuss how to find candidate predicates and entities/classes for edges and nodes. The methods of mapping nodes and labeled edges are the same as phrases mapping of Q^S (see Section **??**). We only concentrate on how to map the unlabeled edges to predicates in RDF graph G.

Mapping unlabeled Edges of Q^U. For an unlabeled edge $\overline{v_iv_j}$, the relation between node v_i and v_j is implicit in given question. For example, edge $\overline{v_4v_5}$ denotes an implicit relation, the correspond word sequence in N_2 is "Chinese actor". We try to infer the implicit relation between the two given nodes v_i and v_j based on underlying knowledge graph. Firstly, we have the following assumptions:

1) Since there is an implicit relation between two nodes v_i and v_j, we assume that the distance between v_i and v_j in RDF graph G is short enough.
2) Assume that at least one node (v_i or v_j) is an entity or a class. It is impossible that two connected nodes are both wh-words.

Similar with the bridging operation in [**?**], we generate the candidate predicates as following. If two nodes are both constants (i.e., entities or classes), such as v_4 and v_5 in Figure **??**(b) (i.e., "Chinese actor"), we locate the two nodes at RDF graph G and find the predicate between them. If one node v_i is a wild-card and the other one v_j is an entity or class, we locate v_j in RDF graph G and select the most frequent adjacent predicates as the candidate predicates to match edge $\overline{v_iv_j}$.

5.2 Query Executing

Given a super semantic query graph Q^U, we discuss how to find approximate matches over RDF graph G with the top-k match scores, where the approximate match is defined in Definition **??** and the match score is analogue to Definition **??**. As mentioned in Definition **??**, some edges (in Q^U) are allowed dis-matching but all nodes should be matched. Consequently, the approximate match of Q^U is the same with the *exact* match (see Definition **??**) of one connected *spanning subgraph* of Q^U . Thus, a straightforward solution is to enumerate all spanning subgraphs S_i of Q^U. For each S_i, we call Algorithm 3 to find the top-k matches of S_i. Finally, we collect all top-k matches for each S_i to form answer set RS, and report k matches with the largest match scores in RS.

Obviously, the above solution is not efficient, since there are lots of common computations if two spanning subgraphs share common structures with each other. Therefore, we propose another bottom-up solution. The pseudo codes are given in Algorithm 5 in Appendix A. Different from the baseline algorithm, we do not decide the query graph at the beginning. Instead we try to construct the "correct" graph structure by expanding the current partial structure. Generally, in each step, we extend the current partial structure Q by expanding one more edge $\overline{v_ix}$, i.e., $Q = Q \cup \overline{v_ix}$ (Line 6 in Algorithm 5). Initially, Q only includes one starting vertex s in Q^U. We select the vertex with the smallest number of candidates as the starting vertex s. If the new expanded partial structure Q can find matches over RDF graph G (Lines 7-11), we continue the search branch. Furthermore, if Q has already been a spanning subgraph of Q^U (Lines 9-11), we record the matches of Q together with the match scores in answer set RS. We only keep the current top-k matches in RS and the current threshold δ. If Q cannot find matches over RDF graph G (Lines 12-13), we backtrack the search branch.

To improve the search performance, we can also perform threshold-based pruning (like A^*-style algorithm) and early terminate some search branches. For example, for a given partial structure Q, we estimate the upper bound of the match score if continually expanding Q. We can derive the upper bound assuming that all un-mached vertices and edges (of Q^U) can match the candidates with the largest score. If the upper bound is still smaller than the threshold δ, we can terminate the search branch. We do not discuss this tangential issue any further.

6 Experiments

We evaluate our system on DBpedia and Freebase with two benchmarks separately. For DBpedia, we use QALD[7] as the benchmark. As we know, QALD is a series of open-domain question answering campaigns, which mainly based on DBpedia. We compare our method with all systems in QALD-6 competition as well as DEANNA [**?**] and Aqqu [**?**]. For Freebase, we use WebQuestions [**?**] as the benchmark and compare our method with Sempre [**?**], ParaSempre [**?**], Aqqu [**?**], STAGG [**?**] and Yavuz et al. [**?**]. To build the relation mention dictionary, we utilize relation phrases in Patty dataset [**?**]. We also use the CrossWikis [**?**] as the entity mention dictionary. All experiments are implemented in a PC server with Intel Xeon CPU 2GB Hz, 64GB memory running Windows 2008. Our two frameworks (the relation-first framework and the node-first framework) are denoted as RFF and NFF, respectively.

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 [**?**]. We use the version of DBpedia 2014 and the statistics are given in Table **??**.
Freebase (https://developers.google.com/freebase/) is a collaboratively edited knowledge base. We use the version of Freebase 2013, which is same with [**?**]. The statistics are given in Table **??**.
Patty relation mention dataset [**?**] contains a large resource for textual patterns that denote binary relations between entities. We use two different relation mention datasets, wordnet-wikipedia and freebase-wikipedia. The statistics are given in Table **??**. The experiments of offline performance can be found in Appendix B.

7. http://qald.sebastianwalter.org/

TABLE 5
Statistics of RDF Graph

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

TABLE 6
Statistics of Relation Mention Dataset

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

6.2 Online Performance

Exp 1. (End-to-End Performance) We evaluate our system both on QALD benchmark and WebQuestions benchmark. For QALD dataset, we show the experiment results in the QALD competition report format to enable the comparison with all systems in QALD-6 (in Table **??**). We also reproduced DEANNA [**?**] and Aqqu [**?**] using the codes published by authors. For WebQuestions dataset, we show the average F1 to compare with previous works. We reproduced Aqqu [**?**] and report the results of other works in Table **??**. In Table **??**, "Processed" denotes the number of test questions that can be processed and "Right" refers to the number of questions that were answered correctly.

For WebQuestions dataset, we use SVM-rank [**?**] to learn the weight α of aggregation function (see Definition **??**) as there are enough training data in WebQuestions. To train SVM-rank model, we generate several candidate query graphs with certain entities and relations for each training question. After matching these query graphs, we calculate the F1 score as their ranking score. The final α in our experiment is 0.136. As there are only 350 training questions of QALD-6, learning a perfect weight is hard. Therefore, we use the default value $\alpha = 0.5$ directly.

Effectiveness Evaluation. Our NFF method joined QALD-6 competition and won the second place at F-1 measure[8]. NFF can answer 68 questions correctly, while the relation-first framework (RFF) can answer 40 questions correctly. Generally, NFF can beat all systems in QALD-6 campaign in F-1 except for CANaLI [**?**]. Note that CANaLI aims to answer *controlled natural language questions*, in which, users need to specify the precise entities and predicates (denoted by URIs) in the question sentences. In other words, CANaLI asks users to do disambiguation task for phrase linking and CANaLI is not a fully natural language question answering system.

Table **??** shows the results on the test set of WebQuestions, which contains 2032 questions. Different from QALD benchmark, WebQuestions has low diversity and most questions are simple questions. The average F1 of our system (49.6%) is little less than the state-of-art work [**?**] (52.5%) and Yavuz et al. [**?**] (52.6%). Compared by [**?**] and [**?**], our approach performs not very well in relation extraction, which relies on the relation mention dictionary. Actually, the advantage of our approach lies in answering

8. The result of QALD-6 campaign is available at http://qald.sebastianwalter.org/6/documents/qald-6_results.pdf, and our team is named NbFramework.

TABLE 7
Evaluating QALD-6 Testing Questions (Total Question Number=100)

	Processed	Right	Recall	Precision	F-1
NFF	100	68	**0.70**	**0.89**	**0.78**
RFF	100	40	0.43	0.77	0.55
CANaLI	100	83	0.89	0.89	0.89
UTQA	100	63	0.69	0.82	0.75
KWGAnswer	100	52	0.59	0.85	0.70
SemGraphQA	100	20	0.25	0.70	0.37
UIQA1	44	21	0.63	0.54	0.25
UIQA2	36	14	0.53	0.43	0.17
DEANNA	100	20	0.21	0.74	0.33
Aqqu	100	36	0.37	0.39	0.38

TABLE 8
Evaluating WebQuestions Testing Questions

	Average F1
NFF	49.6%
RFF	31.2%
Sempre	35.7%
ParaSempre	39.9%
Aqqu	49.4%
STAGG	52.5%
Yavuz et al. (2016)	**52.6%**

complex questions (i.e., multi-hop relation questions), such as some questions in QALD benchmark. As the codes of [**?**] and [**?**] are not available to us, we compare our method with Aqqu [**?**] on QALD. Aqqu performs well on WebQuestions (49.4%) but has a poor performance on QALD benchmark (38% in Table **??**). It is because that the questions in WebQuestions are simpler than QALD and most of them could be translated into a "one-triple" query, i.e, have only one entity and one relation. Aqqu defines three query templates and try to match test questions to predefined templates. These three templates cover almost all of the questions in the WebQuestions benchmark [**?**]. However, when Aqqu meets some other questions which have different representation and could not be matched to predefined templates, it would get wrong answers. For instance, Aqqu could not answer "true-false" questions such as "Does Trump have any children?". However, those questions could be answered correctly by our system because we do not rely on particular dataset and do not use any predefined query templates.

Efficiency Evaluation. We compare the running time of our two frameworks with DEANNA [**?**] using QALD-6 dataset. We test all questions that can be answered correctly by both DEANNA and our methods. In the question understanding, DEANNA needs to generate SPARQLs, our systems generates semantic query graph Q^S or super semantic query graph Q^U. The former has the exponential time complexity, but our methods have the polynomial time complexity in the question understanding stage, as we reserved the ambiguity. The experiment results in Figure **??** confirm that generating SPARQLs is an expensive operation, since it needs to address the disambiguation issue. The question understanding of DEANNA often needs more than ten seconds in our experiments. However, our methods (both NFF and RFF) only spend about one second in the question understanding stage. The reason of NFF is faster than RFF is that RFF spends more time on relation extraction from a whole dependency tree Y. Actually, RFF spends $O(|Y|^2)$ time to extraction relations and build Q^S (see Algorithm 2 in Appendix) while NFF costs $O(|Y|)$ time to

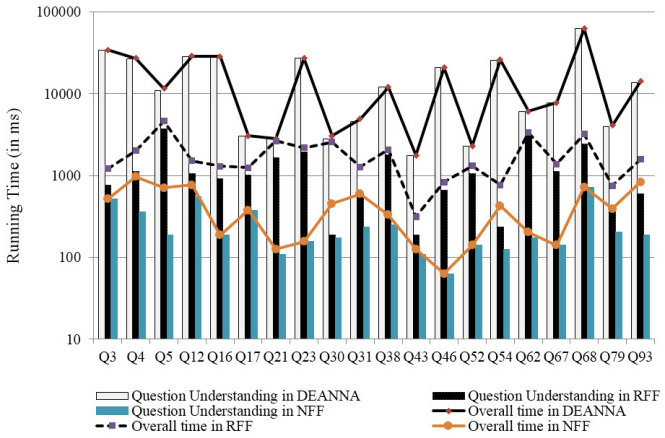


Fig. 6. Online Running Time Comparison

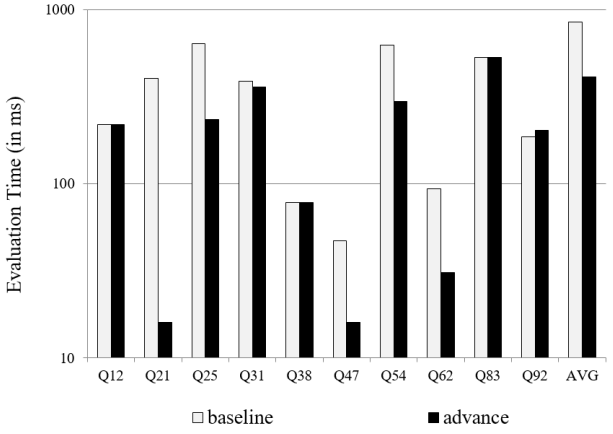


Fig. 7. Evaluation Methods Comparison

TABLE 9
Pipeline Accuracy

	Step1	Step2	Final
RFF	Relation Recognition: 0.65	Building Q^S: 0.54	0.40
NFF	Node Recognition: 0.92	Building Q^U: 0.92	0.68

build Q^U (see Algorithm 4 in Appendix).

Exp 2. (Pipeline Accuracy of two Frameworks) In this experiment, we evaluate the accuracies of main steps in both RFF and NFF using 100 test questions of QALD-6. QALD-6 competition report released the gold standard SPARQL statement for each question sentence in QALD-6. For each sentence N, we suppose that the generated semantic query graph is Q^S and super semantic query graph is Q^U and the correct SPARQL query is Q. In the relation-first framework (RFF), we say that "relation recognition" is correct if exists a correct one-to-one mapping from relation mentions (in Q^S) to predicate edges in SPARQL query graph Q. Furthermore, we say that Q^S is correct if Q^S is isomorphism to Q. Analogously, if there exists a one-to-one mapping from nodes in Q^U to vertices in Q, we say that the node recognition is correct. Q^U is correct if exists a connected spanning graph of Q^U that is isomorphism to Q.

By comparing the first step (i.e, relation recognition and node recognition) between RFF and NFF in 100 test questions, we can see that the node recognition (in NFF) is much more accurate than relation recognition (in RFF), where the former's accuracy is 0.92 and the latter is 0.65. This is the motivation of NFF framework. Furthermore, the accuracy of Q^S is 0.54, which means that 11 questions found wrong associated argument nodes after recognizing correct relations. On the contrary, the accuracy of Q^U is same as the node recognition (0.92), which means that once all nodes were recognized correctly, we can build a correct super semantic query graph Q^U. In other words, Assumption 1 (in Section **??**) of building Q^U is effective.

The final accuracy of RFF and NFF are 0.40 and 0.68, respectively, which means 40% and 68% of questions that can be answered correctly in the two frameworks. In some cases, even if we can generate a correct Q^S or Q^U, we may get the wrong answers. The reason of that is mainly because of out-of-dictionary entities/relations or complex aggregation operations that cannot be handled by our frameworks. The details of error analysis will be given in Exp 4.

Exp 3. (Efficiency of query evaluation in NFF framework) We evaluate the efficiency of the two approximate subgraph matching algorithms in section **??** using the 100 test questions in QALD-6. The results of 10 questions randomly selected and the average time of 100 questions are showed in Figure **??**. For half of the cases, the bottom-up algorithm (Algorithm 5) has obvious advantages, which verified our analysis in section **??**. In some cases, the performance gap is not clear, since Q^U of these questions is an acyclic graph, which means the problem of approximate matching Q^U is degenerated into matching Q^S. Generally, the average time of the bottom-up algorithm is faster than the baseline solution by twice.

Exp 4. (Failure Analysis) We provide the failure analysis of our two methods. We consider about QALD benchmark because it is harder than WebQuestions and more diversified. The failure analysis will help the improvement of our RDF Q/A system's precision.

There are four reasons for the failure of some questions in RFF. The first reason is the relation recognition problem, which accounted for 58%. That because many relations could not be captured by mentions, such as the question "Who was on the Apollo 11 mission". Some relations even be implicit such as "Czech movie". The second one is wrong nodes. For example, "In which countries do people speak Japanese?", the correct semantic relation is ⟨speak,Japanese,country⟩, however, we found the semantic relation ⟨speak,people,country⟩. The latter two reasons are same as the reasons in NFF. Notice that relation mapping failure is a part of relation recognition failure. We give the ratio of each reason with an example in Table **??**.

There are three reasons for the failure of some questions in NFF. The first reason is the node recognition problem. Some phrases were recognized as nodes by mistake. For example, "Who composed the soundtrack for Cameron's Titanic". We regarded the noun "soundtrack" as a variable node, however, it should be ignored. The second one is the failure of phrase mapping, which means we could not find the correct referred entity/relation for a given mention. For example, "What is Batman's real name". The correct relation ⟨alterEgo⟩ is not occurred in the candidate list of mention "real name" in our relation mention dictionary D^R. The third one is that our method cannot answer some complex aggregation questions. We give the ratio of each reason with an example in Table **??**.

TABLE 10
Failure Analysis of Relation-first Framework

Reason	#(Ratio)	Sample Example
Relation Failure	35 (58%)	Q4: Who was on the Apollo 11 mission?
Nodes Failure	11 (18%)	Q55: In which countries do people speak Japanese?
Entity Mapping	5 (9%)	Q32: Who developed Slack?
Complex Aggregation	9 (15%)	Q80: How short is the shortest active NBA player?

TABLE 11
Failure Analysis of Node-first Framework

Reason	#(Ratio)	Sample Example
Node Recognition Failure	8 (25%)	Q27: Who composed the soundtrack for Cameron's Titanic?
Entity Mapping	5 (16%)	Q22: Which computer scientist won an oscar?
Relation Mapping	10 (31%)	Q100: What is Batman's real name?
Complex Aggregation	9 (28%)	Q29: Show me all basketball players that are higher than 2 meters.

7 Related Work

Q/A (natural language question answering) has a quite long history since the seventies of last century [**?**]. Generally, the solutions of knowledge base QA can be mainly divided into two categories.

The first one is *semantic parsing-based*, where natural language questions are translated into logical forms, such as simple λ-DCS [**?**], [**?**], query graph [**?**], or executable queries such as SPARQL [**?**], [**?**], [**?**]. [**?**] defined a set of logical form templates. DEANNA [**?**] builds a disambiguation graph and reduces disambiguation as an integer linear programming problem.

The other category is *information retrieval-based*, where the systems are not intended to parse the question to a formal semantic interpretation. Instead, they select candidate answers first and then rank them by various methods [**?**], [**?**], [**?**], [**?**], [**?**]. [**?**] utilizes subgraph embedding to predict the confidence of candidate answers. [**?**] maximize the similarity between the distributed representation of a question and its answer candidates using Multi-Column Convolutional Neural Networks (MCCNN), while [**?**] aims to predicate the correct relations between topic entity and answer candidates with text evidence.

Our work belongs to the first category and differs from existing systems in three points. First, different from template-based works such as [**?**], [**?**], [**?**], [**?**], our method does not adopt any manually defined templates. To gap the mismatch between natural language and the knowledge base, [**?**] generates canonical utterances for each candidate logical form of the given question N, then it ranks the pairs of canonical utterance and logical form based on a paraphrase model. However, users should define logical form templates and the generation rules first. [**?**] mines millions of operators from unlabeled data, then learns to compose them to answer questions using evidence from multiple knowledge bases. It still uses predefined templates to map questions to queries. Similarly, [**?**] designs three query templates and try to match the given question N to those templates, then generates and ranks SPARQL queries of each matched template. However, both the templates and the generation rules are heavily relied on the particular dataset and could not handle some other questions. For example, none of above systems could answer true-false questions like "Does Trump have any children?". In contrast, our system does not rely on templates and could answer more kinds of questions. We evaluate [**?**] on the QALD-6 benchmark and the results could be found in Table **??**.

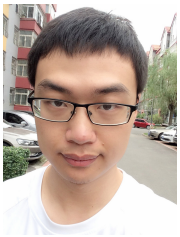
Second, different from most semantic parsing based systems, we push down the disambiguation into the query evaluation stage. Existing solutions, like [**?**] and [**?**], generate the SPARQLs as the intermediate results in the question understanding stage. Obviously, they need to do disambiguation in this step. For example, DEANNA [**?**] 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 [**?**], which is also an NP-hard problem.

Third, our approach have stronger representation power than most existing solutions. Information retrieval solutions like [**?**], find the *topic entity* and try to predicate the relationship between the answer and the topic entity which can only solve simple questions with one triple. For the questions have two entities, they utilize predefined patterns in dependency parse tree to decompose the complex question to two simple question. However, the precision of such system highly depends on the accuracy of the dependency parse tree, which is pretty low when the question is complex. In contrast, our work (especially NFF) is more robust to the errors of dependency parse trees.

Two recent semantic parsing methods [**?**] and [**?**] achieve the state-of-the-art precisions on WebQuestions benchmark. [**?**] builds query graphs from question sentence according to a state transition chain. It first recognizes a topic entity and an inference the relation between the topic entity and the answer. Further it allows other entities to restrict the answer node. The representation power of [**?**] is limited because the final query graph structure must be a tree with diameter less than 3. [**?**] improves semantic parsing via answer type inference. It transforms a question to a "subject, relation, object" order by dependency parse tree patterns and proposes a BLSTM model to predict the answer type. Finally the answer type can be used to prune the candidate logic forms generated by the semantic parsing baseline. This approach cannot tackle the questions uncovered by patterns or complex questions. Different from the above systems, our approach has stronger representation power as we do not restrict the query graph's structure.

8 Conclusions

In this paper, we propose a graph data-driven framework to answer natural language questions over RDF graphs. Different from existing work, we allow the ambiguity both of phrases and structure 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.



Sen Hu received his B.S. degree in Computer Science and Technology from Beijing University of Posts and Telecommunications in 2015. He is currently pursing the Ph.D. degree at Institute of Computer Science and Technology of Peking University, focusing on knowledge graph and question answering.

Lei Zou received his B.S. degree and Ph.D. degree in Computer Science at Huazhong University of Science and Technology (HUST) in 2003 and 2009, respectively. Now, he is an associate professor in Institute of Computer Science and Technology of Peking University. His research interests include graph database and semantic data management.

Jeffrey Yu Xu has held teaching positions in the Institute of Information Sciences an Electronics, University of Tsukuba, and in the Department of Computer Science, Australian National University, Australia. Currently, he is professor in the Department of Systems Engineering and Engineering Management, Chinese University of Hong Kong, Hong Kong. His current research interests include graph database, graph mining, and social network analysis.

Haixun Wang received the bachelors and masters degrees in computer science from Shanghai Jiao Tong University in 1994 and 1996, respectively, and the PhD degree in computer science from the University of California, LA, in 2000. He joined Google Research, Mountain View, CA, in 2013. His research interests include text analytics, natural language processing, knowledge base, semantic network, artificial intelligence, graph data management, etc.

Dongyan Zhao received the B.S. degree, M.S. degree and Ph.D. degree from Peking University in 1991, 1994 and 2000, respectively. Now, he is a professor in Institute of Computer Science and Technology of Peking University. His research interest is on information processing and knowledge management, including computer network, graph database, and intelligent agent.