Answering Natural Language Questions via Phrasal Semantic Parsing*

Kun Xu, Sheng Zhang, Yansong Feng**, and Dongyan Zhao

Peking University, Beijing, China
{xukun,evancheung,fengyansong,zhaodongyan}@pku.edu.cn

Abstract. Understanding natural language questions and converting them into structured queries have been considered as a crucial way to help users access large scale structured knowledge bases. However, the task usually involves two main challenges: recognizing users' query intention and mapping the involved semantic items against a given knowledge base (KB). In this paper, we propose an efficient pipeline framework to model a user's query intention as a phrase level dependency DAG which is then instantiated regarding a specific KB to construct the final structured query. Our model benefits from the efficiency of linear structured prediction models and the separation of KB-independent and KB-related modelings. We evaluate our model on two datasets, and the experimental results showed that our method outperforms the state-of-the-art methods on the Free917 dataset, and, with limited training data from Free917, our model can smoothly adapt to new challenging dataset, WebQuestion, without extra training efforts while maintaining promising performances.

1 Introduction

As very large structured knowledge bases have become available, e.g.,YAGO [2], DBpedia [3] and Freebase[4], answering natural language questions over structured knowledge facts has attracted increasing research efforts. Different from keyword based information retrieval, the structure of query intentions embedded in a user's question can be represented by a set of predicate-argument structures, e.g., $<$*subject, predicate, object*$>$ triples, and effectively retrieved by a database search engine. Generally, the main challenge of understanding the query intention in a structural form is to solve two tasks: recognizing the predicate-argument structures and then instantiating these structures regarding a given KB.

Considering the example question shown in Figure 1, the structure of the query intention consists of multiple predicate-argument pairs, involving an named entity *france* mapping to a KB entity "France", a word *country* mapping to a KB type "Country" and a verb *colonise* possibly indicating a KB relation *a country* ***"/colonise"*** *another country*. Intuitively, the two subtasks would be solved in a joint framework, e.g., [14]

* This work was supported by the National High Technology R&D Program of China (Grant No. 2012AA011101, 2014AA015102), National Natural Science Foundation of China (Grant No. 61272344, 61202233, 61370055) and the joint project with IBM Research.

** Corresponding author.

C. Zong et al. (Eds.): NLPCC 2014, CCIS 496, pp. 333–344, 2014.

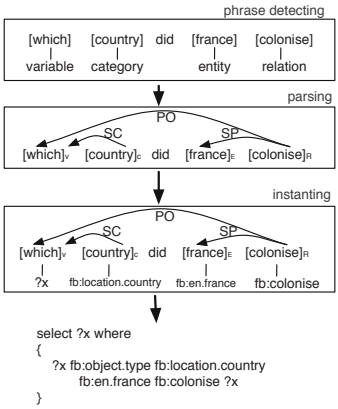

Fig. 1. An example of converting a natural language question into a structured query via phrasal semantic parsing

proposed a PCFG-based semantic parser to simultaneously learn the combination rules among words or phrases and the mappings to specific KB components. However, given the size of existing KBs (usually thousands of predicates, millions of entities and billions of knowledge facts), it makes difficult to jointly train such a PCFG-based parser (the model of [14] takes several days to train with 3,000 sentences), and even more difficult to adapt to other KBs, let alone retrieving multiple KBs within one query, e.g., some queries in the QALD task[6] are mixed with predicates from both DBpedia and Yago. In contrast, we find that recognizing the query intention structure is usually KB-independent. Take Figure 1 as an example, without grounding to a knowledge base, we can still guess that a location called *france* has some relationship, indicated by the verb *"colonise"*, with some *countries*, (the queried objects), which can be learned directly without reliance on a specified KB. On the other hand, the task of mapping semantic phrases from the intention structures to items in a given KB and producing the final structured queries is KB-dependent, since one has to solve these mappings according to the schema of a specified KB.

Given the observations above, we thus assume that the structure of a question's query intention can be learned independent from a specific knowledge base, while grounding and converting a query intention into a structured query is dependent on a knowledge base. Our assumption will naturally lead to a pipeline paradigm to translating a natural language question into a structured query, which can then be directly retrieved by a structured database query engine, e.g., Virtuoso[1].

In this paper, we deal with the task of understanding natural language questions in a pipeline paradigm, involving mainly two steps: recognizing the query intention structure inherent in the natural language questions, and then instantiating the query intention structures by mapping the involved semantic items into existing KBs. In the first phase, we build a phrase detector to detect possible semantic phrases, e.g., variables, entity phrases, category phrases and relation phrases. We then develop a semantic parser to

[1] http://www.virtuoso.com

predict the predicate-argument structures among phrases to represent the structure of query intentions. In the second phase, given the intention structures, we are then able to adopt a structured perceptron model to jointly solve the mappings between semantic phrases and KB items. By taking a two-phase format, our proposed model can benefit from the separation of KB related components and KB independent steps, and recognize the intention structures more efficiently while making the KB-related component flexible, e.g., we can only retrain the second phase when adapting to new KBs, which is similar in sprite with [13], who rely on a CCG parser to produce an ontological-independent logical representation to express users' intention. We evaluate our model on three datasets, and show that our model can effectively learn the structures of query intentions, and outperform the state-of-the-art methods in terms of question-answering accuracy. Specifically, by testing on a new dataset with large and broad-coverage KB predicates, our model can still perform comparably to the state of the arts without any extra training on the new datasets. By just adjusting the KB related components, our model can maintain a promising results on a new KB.

This rest of the paper is organized as follows. We first briefly describe related work in Section 2. Our Task definition is introduced in Section 3. Section 4 and 5 describe the two steps of our framework: recognizing the structure of query intention and instantiating query intention regarding KB. Our experiments are presented and discussed in Section 6. We finally conclude this paper in Section 7.

2 Related Work

Question answering is a long-standing problem in the field of natural language processing and artificial intelligence. Previous research is mainly dominated by keyword matching based approaches, while recent advancements in the development of structured KBs and structured query engines have demanded the research of translating natural language questions into structured queries, which can then be retrieved using a structured query engine. Existing methods can be roughly categorized into two streams, pattern/template-based models [8–10] and semantic parsing-based models [11–15].

[8] use lexical-conceptual templates for query generation but do not address the disambiguation of constituents in the question. [16] rely on a manually created ontology-driven grammar to directly map questions onto the underlying ontology, where the grammars are hard to adapt or generalize to other large scale knowledge bases. They further develop a template-based approach to map natural language questions into structured queries[10]. [9] collect the mapping between natural language expressions and Yago2 predicates using a set of predefined patterns over dependency parses, and find an optimal mapping assignments for all possible fragments in the questions using an ILP model. Those methods are mainly reply on a set of manually created templates or patterns to collect lexicons or represent the structure of query intentions, therefore are difficult to scale in practice due to the manual efforts involved.

[11] use distant supervision to collect training sentences as well as manual rules to construct CCG lexicons from dependency parses in order to train a semantic parser. [12] develop a probabilistic CCG-based semantic parser, FreeParser, where questions are automatically mapped to logical forms grounded in the symbols of certain fixed ontology

or relational database. They take a similar distant supervision approach to automatically construct CCG lexicon and induce combination rules [17], though with inadequate coverage, for example, their parser will fail if any phrase in the question is not included in the lexicon of the PCCG parser. [14] develop a PCFG-based semantic parser, where a *bridge* operation is proposed to improve coverage and they utilize a set of manual combination rules as well as feature-simulated *soft rules* to combine predicates and produce logical forms.

To handle the *mismatch* between language and the KB, [13] develop a PCCG parser to build an ontology-independent logical representation, and employ an ontology matching model to adapt the output logical forms for each target ontology. [15] first generate candidate canonical utterances for logical forms, then utilize paraphrase models to choose the canonical utterance that best paraphrases the candidate utterance, and thereby the logical form that generated it.

In contrast, we focus on translating natural language questions into structured queries by separating the KB independent components from the KB-related mapping phase. Like [12], our model takes question-phrase dependency DAG pairs as input for our structure recognition phase, but relies far less training data than [12] towards a open domain parser, since we do not learn KB related mappings during structured predictions. We then learn a joint mapping model to instantiate the phrase dependency DAG with a given KB. Our model is simple in structure but efficient in terms of training, since we have a much smaller search space during structure prediction with respect to the query intention, and still hold the promise for further improvement, for example, taking question-answer pairs as training data after initializing with some question-DAG training samples.

3 The Task

We define the task of using a KB to answer natural language questions as follows: given a natural language question q_{NL} and a knowledge base KB, our goal is to translate q_{NL} into a structured query in certain structured query language, e.g., SPARQL, which consists of multiple triples: a conjunction of <subject, predicate, object> search conditions.

4 Recognizing the Structure of Query Intention

Our framework first employs a pipeline of phrase detection and phrase dependency parsing to recognize the inherent structure of user's query intention, which is then instantiated regarding a specific KB.

4.1 Phrase Detection

We first detect phrases of interest that potentially correspond to semantic items, where a detected phrase is assigned with a label $l \in \{entity, relation, category, variable\}$. Entity phrases may correspond to entities of KB, relation phrases correspond to KB's

predicates and category phrases correspond to KB's categories. This problem can be casted as a sequence labeling problem, where our goal is to build a tagger to predict labels for a sentence. For example:

what	are	the	sub-types	of	coal
V-B	none	R-B	R-I	R-I	E-B

(Here, we use *B-I* scheme for each phrase label: *R-B* represents the beginning of a relation phrase, *R-I* represents the continuation of a relation phrase). We use structured perceptron[18] to build our phrase tagger. Structured perceptron is an extension to the standard linear perceptron for structured prediction. Given a question instance $x \in X$, which in our case is a sentence, the structured perceptron involves the following *decoding problem* which finds the best configuration $z \in Y$, which in our case is a label sequence, according to the current model **w**:

$$z = \arg \max_{y' \in Y(x)} w \cdot f(x, y')$$

where $f(x, y')$ represents the feature vector for instance x along with configuration y'. We use three types of features: lexical features, POS tag features and NER features. Table 1 summarizes the feature templates we used in the phrase detection.

Table 1. Set of feature templates for phrase detection

p = pos tag; n = ner tag; w = word; t = phrase type tag; i = current index

1	unigram of POS tag	p_i
2	bigram of POS tag	$p_i p_{i+1}$, $p_{i-1} p_i$
3	trigram of POS tag	$p_i p_{i+1} p_{i+2}$, $p_{i-1} p_i p_{i+1}$, $p_{i-2} p_{i-1} p_i$
4	unigram of NER tag	n_i
5	bigram of NER tag	$n_i n_{i+1}$, $n_{i-1} n_i$
6	trigram of NER tag	$n_i n_{i+1} n_{i+2}$, $n_{i-1} n_i n_{i+1}$, $n_{i-2} n_{i-1} n_i$
7	unigram of word	w_i
8	bigram of word	$w_i w_{i+1}$, $w_{i-1} w_i$
9	trigram of word	$w_i w_{i+1} w_{i+2}$, $w_{i-1} w_i w_{i+1}$, $w_{i-2} w_{i-1} w_i$
10	previous phrase type	t_{i-1}
11	conjunction of previous phrase type and current word	$t_{i-1} w_i$

4.2 Phrase Dependency Parsing with Multiple Heads

As shown in Figure 1, query intention can be represented by dependencies between "*country*", "*france*" and "*colonise*", forming a phrase dependency DAG, we thus introduce a transition-based DAG parsing algorithm to perform a structural prediction process and reveal the inherent structures.

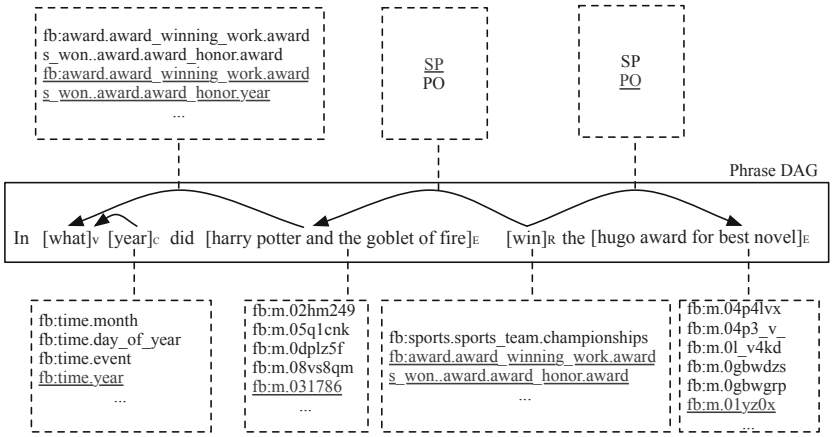

Fig. 2. An example of phrasal semantic DAG, where the dashed boxes list the mapping candidates for all phrases and the underlined are the gold-standard mappings.)

Phrase Dependency DAG. We propose to use the predicate-argument dependencies to capture the query intention, that is, the arguments of a predicate are dependents of that predicate. Here, each predicate is either a unary predicate (characterize its only argument) or a binary predicate (represents the semantic relation between its two arguments). For example, in Figure 2, the category phrase "*year*" indicates the variable is one specific year, and the relation phrase "*win*" indicates that the award "*hugo award for best novel*" is won by "*harry potter and the goblet of fire*".

Phrase Dependency Parsing. Note that, in our setup, one phrase can have more than one head, as in Figure 2, variable node *what* has two heads in the resulting dependency DAG. We thus use the framework proposed by [19], i.e., extending traditional arc-eager shift-reduce parsing with multiple heads to find a DAG directly. Specifically, given a question with sequence of phrases, our parser uses a stack of partial DAGs, a queue of incoming phrases, and a series of actions to build a dependency DAG. We assume that each input phrase has been assigned a POS-tag and a semantic label.

Our semantic parser uses four actions: SHIFT, REDUCE, ARCRIGHT and ARCLEFT.

The SHIFT action follow the standard definitions that just pushes the next incoming phrase onto the stack.

The REDUCE action pops the stack top. Note that, the standard REDUCE action which is taken on the condition that the stack top has at least one head. This precondition ensures the dependency graph is a connected graph. However, our phrase dependency parser only concerns the predicate-argument structures, and we add a dependency only between the predicate and argument of our interest. In our case, the dependency graph can be a unconnected directed graph.

The ARCRIGHT action adds a dependency edge from the stack top to the first phrase of the incoming queue, where the phrase on the stack is the head and the phrase in the

Algorithm 1. The decoding algorithm for the phrase DAG parsing; K is the beam size

Require: sentence x
$agenda$: hold the K-best candidate items
Ensure: *candidate_output*
1: *agenda*.clear()
2: *agenda*.insert(GetStartItem(x))
3: *candidate_output* = NONE
4: **while** not *agenda*.empty() **do**
5: *list*.clear()
6: **for all** *item* $\in$ *agenda* **do**
7: **for all** *action* $\in getActions(actions, item)$ **do**
8: $item^{'}$ = *item*.apply(*action*)
9: **if** $item^{'}$.F == TRUE **then**
10: **if** *candidate_output* == NONE
or $item^{'}$.score $>$ *candidate_output*.score **then**
11: *candidate_output* = $item^{'}$
12: **end if**
13: **else**
14: *list*.append($item^{'}$)
15: **end if**
16: **end for**
17: **end for**
18: *agenda*.clear()
19: *agenda*.insert(*list*.best(K))
20: **end while**

queue is the dependent (the stack and queue are left untouched), as long as a left arc does not already exist between these two phrases.

The ARCLEFT action adds a dependency edge from the first phrase on the queue to the stack top, where the phrase in the queue is the head and the phrase on the stack is the dependent (again, the stack and queue are left untouched), as long as a right arc does not already exist between the two phrases.

The Decoding Algorithm for Phrase DAG Parsing. We apply the standard beam-search along with early-update to perform inexact decoding [20] during training. To formulate the decoding algorithm, we define a *candidate item* as a tuple <S,Q,F>, where S represents the stack with partial derivations that have been built, Q represents the queue of incoming phrases that have not been processed, and F is a boolean value that represents whether the candidate item has been finished. A candidate item is finished if and only if the queue is empty, and no more actions can be applied to a candidate item after it reaches the finished status. Given an input sentence x, we define the start item as the unfinished item with an empty stack and the whole input sentence as the incoming phrases(line 2). A derivation is built from the start item by repeated applications of actions (SHIFT, REDUCE, ARCLEFT and ARCRIGHT) until the item is finished.

To apply beam-search, an agenda is used to hold the K-best partial (unfinished) candidate items at each parsing step. A separate *candidate output* is used to record the

current best finished item that has been found, since candidate items can be finished at different steps. Initially the agenda contains only the start item, and the candidate output is set to none(line 3). At each step during parsing, each candidate item from the agenda is extended in all possible ways by applying one action according to the current status(line 7), and a number of new candidate items are generated(line 8). If a newly generated candidate is finished, it is compared with the current *candidate output*. If the candidate output is none or the score of the newly generated candidate is higher than the score of the *candidate output*, the *candidate output* is replaced with the newly generated item(line 11); otherwise the newly generated item is discarded (line 14). If the newly generated candidate is unfinished, it is appended to a list of newly generated partial candidates. After all candidate items from the agenda have been processed, the agenda is cleared(line 18) and the K-best items from the list are put on the agenda(line 19). Then the list is cleared and the parser moves on to the next step. This process repeats until the agenda is empty (which means that no new items have been generated in the previous step), and the candidate output is the final derivation. Pseudocode for the decoding algorithm is shown in Algorithm 1.

Table 2. The set of feature templates used in our phrase DAG parser

p = phrase; t = POS-tag; s = phrase type

Category	Description	templates
lexical features	stack top	$STpt$; STp; STt;
	current phrase	N_0pt; N_0p; N_0t
	next phrase	N_1pt; N_1p; N_1t;
	ST and N0	$STptN_0pt$; $STptN_0p$;
	POS bigram	N_0tN_1t
	POS trigrams	$N_0N_1tN_2t$;
	N0 phrase	$N_0pN_1tN_2t$;
semantic features	Conjunction of phrase label and pos tag	N_0s; N_0ts; N_0ps; N_1s;N_1ts;$STtN_0s$; $STsN_0t$; $STpN_0s$; $STtN_0t$; $STsN_0s$;
structural features	Indicates whether exists an arc between the stack top item and next input item, and if so what type of arc	ArcLeft(STs, N_0s); ArcRight(STs, N_0s)

Features. Features play an important role in transition-based parsing. Our parser takes three types of features: lexical, semantic and structure-related features. We summarize our feature templates in Table 2, where ST represents the top node in the stack, N_0, N_1, N_2 represent the three incoming phrases from the incoming queue, subscript t indicates POS tags, subscript p indicates lexical surface forms and subscript s represent the semantic label of the phrase (*entity,relation, category* and *variable*).

Lexical features include features used in traditional word level dependency parsing with some modifications: all co-occurrences are built on phrase nodes and the POS tag of a phrase is defined as the concatenation of each token's POS tag in the phrase.

Note that ARCLEFT and ARCRIGHT actions are considered only when the top phrase of the stack and the next phrase are variable, entity or relation phrases.To guide the ARCLEFT and ARCRIGHT actions, we introduce semantic features indicating the semantic label of a phrase.

Recall that, our phrase semantic DAG parser allows one phrase to have multiple heads. Therefore, we modify the ARCLEFT and ARCRIGHT actions so that they can create new dependency arcs without removing the dependent from further consideration for being a dependent of other heads. We thus introduce new structure-related features to indicate whether an arc already exists between the top phrase on the stack and the next phrase on the queue.

5 Instantiating Query Intention Regarding Existing KBs

Given the query intention represented in the phrase dependency DAG, we need to convert it into a structured query Q_{ind}, which can be then grounded to a *KB*-related database query Q_d via mapping the natural language phrases to the semantic items in the *KB*, e.g., Freebase. However, each phrase in Q_{ind} can be potentially mapped to multiple candidate items of Freebase, as shown in Figure2. Given a knowledge base *KB* and the Q_{ind} that consists of n triples, we will have:

$$Q_d^* = \arg\max P(Q_d|Q_{ind})$$

For the simplicity of computation, we made necessary independent assumptions, and approximate $P(Q_d|Q_{ind})$ as:

$$\overline{P}(Q_d|Q_{ind}) = \prod_{i=1}^{n} \overline{P}(s_{d_i}|s_{ind_i})\overline{P}(o_{d_i}|o_{ind_i})\overline{P}(p_{d_i}|p_{ind_i})$$

where the (s, p, o) corresponds to the three parts of a query triple: the subject s, predicate p and object o. In practice, we use the Freebase search API[2] to compute the probabilities of mapping the subject and object phrase. All subject and object phrases are sent to this API, which returns a ranked list of relevant items with their scores. We normalize the score by the sum of all candidate scores.

Inspired by ?, we apply the Naive Bayes model to compute the probability of mapping the relation phrase:

$$\begin{aligned}\overline{P}(p_d|p_{ind}) &= \overline{P}(p_{ind}|p_d)\overline{P}(p_d) \\ &= \prod_w \overline{P}(w|p_d)\overline{P}(p_d)\end{aligned}$$

[2] `https://developers.google.com/freebase/`

where the w is the word in the phrase p_{ind}. We used the dataset contributed by [21] additionally with type constraints to estimate $\overline{P}(p_d)$ and $\overline{P}(w|p_d)$. For instance, given the subject **/en/france** that has a Freebase type **/location/country** and the object **/en/French** that has a Freebase type **/language/human_language**, the target p_d should take a subject of type **/location/country** and an object of type **/language/human_language**, which are the type constraints in this case. In this case, the candidates of p_d only contains two properties: **/location/country/official_language** and **/location/country/languages_spoken**.

6 Experiments

6.1 Experimental Setup

The Free917 dataset [12] contains 917 questions annotated with logical forms grounded to Freebase. Note that in all of our experiments, we only use the training set of Free917 as our training data. To prepare the training data for our parser, we first parse these questions with CCG parser and accordingly replace the KB items in the gold-standard logical forms with natural language phrases, which we manually assign a semantic label to. Following [12], we held out 30% of the data for the final test, and perform 3 random 80%-20% splits of the training set for development.

The WebQuestions dataset [14] contains 5,810 question-answer pairs, with the same training/testing split with previous work. This dataset was created by crawling questions through the Google Suggest API, and then obtaining answers through Amazon Mechanical Turk. We directly employ the parser trained on the Free917 to test on the test set of WebQuestions and retrieve the answers by executing the queries against a copy of Freebase using the Virtuoso engine.

When evaluating on the Free917 dataset, we use the *parsing accuracy*, *instantiation accuracy* and *system accuracy* as the evaluation metrics. Specifically, the *parsing accuracy* evaluates our parsing phase, which converts the question into a KB-independent logical form. The *instantiation accuracy* evaluates the mappings from a KB-independent logical form to a KB-related database query. The *system accuracy* evaluates the whole pipeline approach, which converts a question into a KB-related database query. Note that we also evaluate gold answers for this dataset, since the queries with aggregation functions may differ from the gold queries. Considering the WebQuestions dataset only contains the gold answers, we use the F-measure as the system accuracy to evaluate our approach.

Table 3. Results on test sets of Free917 and WebQuestions

	Free917	WebQuestions
CY13	59.0%	-
BCFL13	62.0%	35.7%
KCAZ13	68.0%	-
BCFL14	68.5%	**39.9%**
Our work	**69.0%**	39.1%

6.2 Main Results

Table 3 represents results on the test set. Our main empirical result on the Free917 dataset is that our system obtains an answer accuracy of 69.0% on the test set, outperforming 59% reported by [12], 62% reported by [14]. and 68.5% reported by [15].

Note that, the Free917 dataset covers only 635 Freebase predicates and are only about Freebase. We thus evaluate our parser on a more natural dataset, WebQuestions, introduced by [14]. WebQuestions consists of 5,810 question-answer pairs involved more domains and relations.

We experiment on the original test set, directly using the parser trained on Free917 to predict the query intention regarding Freebase. Interestingly, our system is able to achieve a relative higher accuracy of 39.1% given the fact that the WebQuestions datasets covers a larger and broader set of predicates in Freebase, indicating that the KB-independent structured predictions can be learned separately from the KB-related mappings, while maintaining a comparable performance. In other words, these experiments show that regardless of the topics people are asking, their way of presenting the questions are still similar, which can be captured by learning the structures of phrases of semantic meanings.

6.3 Error Analysis

We analyzed the WebQuestions and the Free917 examples and found several main causes of errors: (i)Phrase detection also accounts for many errors, e.g., the entity phrase detected in "*What is the name of justin bieber brother*" is "*justin biber brother*" and not "*justin biber*". This detection error is propagated to phrasal semantic parsing (ii)the probabilistic mapping model may map a relation phrase to a wrong predicate, which is mainly due to the limited coverage of our mapping resources. (iii)our system is unable to handle temporal information, which causes errors in questions like *"what kind of government did the united states have after the revolution"* in WebQuestions, where we fail to recognize *"after the revolution"* as a temporal constraint, due to the fact that we never saw this syntax structures in our training data.

7 Conclusion and Future Work

In this paper, we propose a novel framework to translate natural language questions into structural queries, which can be effectively retrieved by structured query engines and return the answers according a KB. The novelty of our framework lies in modeling the task in a KB-independent and KB-related pipeline paradigm, where we use phrasal semantic DAG to represent users' query intention, and develop a KB-independent shift-reduce DAG parser to capture the structure of the query intentions, which are then grounded to a given KB via joint mappings. This gives the advantages to analyze the questions independent from a KB and easily adapt to new KBs without much human involvement. The experiments on two datasets showed that our model outperforms the state-of-the-art methods in Free917, and performs comparably on a new challenging dataset without any extra training or resources.

Currently, our model requires question-DAG pairs as the training data, though we do not need to annotate new data for every datasets or KBs, it is still promising to extend

our model to train on question-answer pairs only, further saving human involvement. Secondly, in the pipeline of phrase detection and phrase dependency parsing, error propagated from the upstream to downstream. A joint model with multiple perceptrons may help to eliminate the error propagation.

References

1. Fader, A., Soderland, S., Etzioni, O.: Identifying Relations for Open Information Extraction. In: EMNLP, pp. 1535–1545 (2011)
2. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A core of semantic knowledge. In: WWW, pp. 697–706 (2007)
3. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: DBpedia: A Nucleus for a Web of Open Data. In: Aberer, K., et al. (eds.) ASWC/ISWC 2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007)
4. Bollacker, K.D., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: A collaboratively created graph database for structuring human knowledge. In: SIGMOD, pp. 1247–1250 (2008)
5. Berant, J., Chou, A., Frostig, R., Liang, P.: Semantic Parsing on Freebase from Question-Answer Pairs. In: EMNLP, pp. 1533–1544 (2013)
6. Cimiano, P., Lopez, V., Unger, C., Cabrio, E., Ngonga Ngomo, A.-C., Walter, S.: Multilingual Question Answering over Linked Data (QALD-3): Lab Overview. In: Forner, P., Müller, H., Paredes, R., Rosso, P., Stein, B. (eds.) CLEF 2013. LNCS, vol. 8138, pp. 321–332. Springer, Heidelberg (2013)
7. Kwiatkowski, T., Choi, E., Artzi, Y., Zettlemoyer, L.S.: Scaling Semantic Parsers with On-the-Fly Ontology Matching. In: EMNLP, pp. 1545–1556 (2013)
8. Frank, A., Krieger, H.-U., Xu, F., Uszkoreit, H., Crysmann, B., Jörg, B., Schäfer, U.: Question answering from structured knowledge sources. J. Applied Logic, 20–48 (2007)
9. Yahya, M., Berberich, K., Elbassuoni, S., Ramanath, M., Tresp, V., Weikum, G.: Natural Language Questions for the Web of Data. In: EMNLP-CoNLL, pp. 379–390 (2012)
10. Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.-C., Gerber, D., Cimiano, P.: Template-based question answering over RDF data. In: WWW, pp. 639–648 (2012)
11. Krishnamurthy, J., Mitchell, T.: Weakly Supervised Training of Semantic Parsers. In: EMNLP-CoNLL, pp. 754–765 (2012)
12. Cai, Q., Yates, A.: Semantic Parsing Freebase: Towards Open-domain Semantic Parsing. In: SEM (2013)
13. Kwiatkowski, T., Choi, E., Artzi, Y., Zettlemoyer, L.S.: Scaling Semantic Parsers with On-the-Fly Ontology Matching. In: EMNLP, pp. 1545–1556 (2013)
14. Berant, J., Chou, A., Frostig, R., Liang, P.: Semantic Parsing on Freebase from Question-Answer Pairs. In: EMNLP, pp. 1533–1544 (2013)
15. Berant, J., Liang, P.: Semantic Parsing via Paraphrasing. In: ACL (2014)
16. Unger, C., Cimiano, P.: Pythia: Compositional Meaning Construction for Ontology-based Question Answering on the Semantic Web. In: Muñoz, R., Montoyo, A., Métais, E. (eds.) NLDB 2011. LNCS, vol. 6716, pp. 153–160. Springer, Heidelberg (2011)
17. Kwiatkowski, T., Zettlemoyer, L.S., Goldwater, S., Steedman, M.: Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification. In: EMNLP, pp. 1223–1233 (2010)
18. Collins, M.: Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In: EMNLP (2002)
19. Sagae, K., Tsujii, J.: Shift-Reduce Dependency DAG Parsing. In: COLING, pp. 753–760 (2008)
20. Collins, M., Roark, B.: Incremental Parsing with the Perceptron Algorithm. In: ACL, pp. 111–118 (2004)
21. Yao, X., Van Durme, B.: Information Extraction over Structured Data: Question Answering with Freebase. In: Proceedings of ACL (2014)