# CASIA@V2: A MLN-based Question Answering System over Linked Data

Shizhu He, Yuanzhe Zhang, Kang Liu, and Jun Zhao

National Laboratory of Pattern Recognition,
Institute of Automation, Chinese Academy of Sciences.
Zhongguancun East Road 95#, Beijing, 100084, China.
{shizhu.he,yzzhang,kliu,jzhao}@nlpr.ia.ac.cn

**Abstract.** We present a question answering system (CASIA@V2) over Linked Data (DBpedia), which translates natural language questions into structured queries automatically. Existing systems usually adopt a pipeline framework, which contains four major steps: 1) Decomposing the question and detecting candidate phrases; 2) mapping the detected phrases into semantic items of Linked Data; 3) grouping the mapped semantic items into semantic triples; and 4) generating the rightful SPARQL query. We present a jointly learning framework using Markov Logic Network(MLN) for phrase detection, phrases mapping to semantic items and semantic items grouping. We formulate the knowledge for resolving the ambiguities in three steps of QALD as first-order logic clauses in a MLN. We evaluate our approach on QALD-4 test dataset and achieve an F-measure score of 0.36, an average precision of 0.32 and an average recall of 0.40 over 50 questions.

**Keywords:** Question Answering, Linked Data, Markov Logic Network

## 1 Introduction

With the rapid development of the Web of Data, there are many RDF datasets published as Linked Data [1], such as DBpedia [2], Freebase [3] and YAGO [4]. The growing Linked Data contain a wealth of knowledge including entities, classes and properties. Moreover, these linked data usually have complex structures and are highly heterogeneous. However, there are the gaps between the users and the Linked Data. On the one hand, the users, even expert programmers, need a lot of practices to handle standard structured query languages like SPARQL. On the other hand, due to the diversity

In: Cappellato, L., Ferro, N., Halvey, M., and Kraaij, W., editors (2014). CLEF 2014 Labs and Workshops, Notebook Papers. CEUR Workshop Proceedings (CEUR-WS.org), ISSN 1613-0073, http://ceur-ws.org/ Vol-1180/.

Thanks to Prof. Hang Li, Dr. Yibo Zhang and Dr. Jie Zhang at Huawei Noah'S Ark Lab and the anonymous reviewers for their insightful advices and and suggestions. This work was sponsored by the National Natural Science Foundation of China (No. 61272332 and No. 61202329) and CCF-Tencent Open Research Fund. This work was supported in part by Noah's Ark Lab of Huawei Tech. Ltm.

and high heterogeneity of the Linked Data, it is difficult for humans to select relevant resources and discover useful information. Thus, developing user-friendly interface for accessing those linked data become increasing important.

Question answering over Linked Data [5] is aimed at eliminating those gaps, which attempts to allow the users to access those structured data with natural language. For example, with respect to the question: *"Which software has been developed by organizations founded in California, USA?"*, the aim is to automatically convert this utterance into a SPARQL query which contains the following *subject-property-object (SPO)* triple format: ⟨**?url rdf:type dbo:Software, ?url dbo:developer ?x1, ?x1 rdf:type dbo:Company, ?x1 dbo:foundationPlace dbr:California**⟩[1].

Lopez et al. [6] have given a comprehensive survey in this research area. The authors develop *PowerAqua* system [7] to answer questions on large, heterogeneous datasets. For questions containing quantifiers, comparatives or superlatives, Unger et al. [8] translate natural language(*NL*) questions to formal language(*FL*) structured query using several SPARQL templates and a set of heuristic rules mapping phrase to semantic items. And *DEANNA* [9] jointly disambiguates the following tasks based on integer linear programming: segmenting question, mapping phrases to semantic items and constructing SPARQL triple patterns.

This paper proposes a novel algorithm based on a learning framework, Markov Logic Network (MLN) [10], to learn a joint model for constructing structured queries from natural language utterances. MLN is a statistical relational learning framework that combines first-order logic and Markov networks. The appealing property of MLN is that it is readily interpretable by humans and is natural to perform joint learning under the Markov logic framework. We formulate the knowledge for resolving the ambiguities in three steps of QALD (phrase detection, phrase-to-semantic-item mapping and semantic items grouping) as first-order logic clauses in a MLN. In the framework of MLN, all clauses will make interacted effects which combines resolving all problems into a unified process. In this way, the result in each step can be globally optimized. Moreover, different from previous methods, we adopt a learning strategy to automatically learn the patterns for semantic items grouping. We also formulate the learned patterns as first-order clauses in MLN. The model will learn the weights of each clause to determine the most effective patterns for semantic triples construction. In this way, our approach can cover more semantic expressions and answer more questions than previous method based on manually designed patterns.

We evaluate our approach on QALD-4 test dataset and achieve an F-measure score of 0.36, an average precision of 0.32 and an average recall of 0.40 over 50 questions.

## 2 System Description

The current version of our QA system is not designed to answer the questions which contain numbers, date comparisons and aggregation operations such as *group by* or *order by*. We also do not consider the questions which contain filter condition. Figure

[1] prefixes in semantic items indicate the source of its vocabularies, **dbr** indicate entity and **dbo** indicate class and property defined in the DBpedia ontology (http://wiki.dbpedia.org/Ontology39).

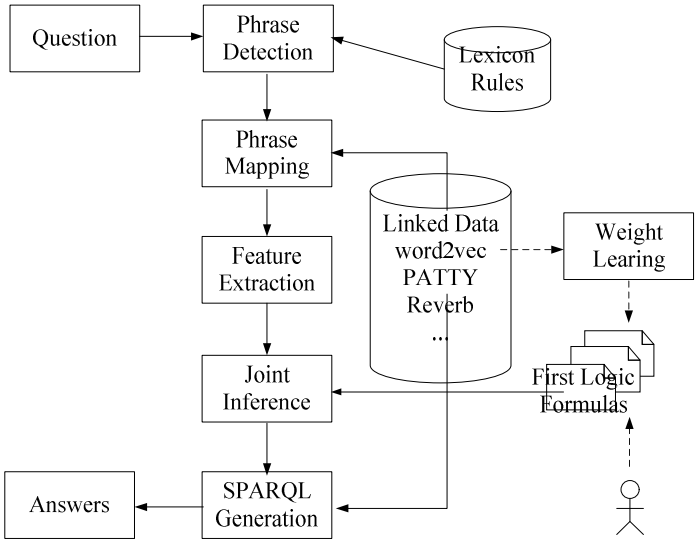


**Fig. 1.** The architecture of CASIA@V2

1 shows the architecture of our system to translate a question into a formal SPARQL query.

At first, sequence of tokens(phrase) are detected that probably indicate semantic items, such as *software*, *developed by* and *California*. This step detects full potentially phrases, and leaves the decision for phrases selecting in later steps.

Next, the phrases are mapping to semantic items. Phrases can denote entities, classes and properties. For example, the phrase *software* can denote class **dbo:Software** and property **dbo:developer**, and the phrase *developed by* can denote entity **dbr:videogamedeveloper**, class **dbo:BritishRoyalty** and property **dbo:developer**. This step purely constructs a candidate space for every possible mapping, and leaves the decision for select which semantic items in the next step.

Then, we should make the decisions for choosing phrases, mapping the chosen phrases to suitable semantic items and determine the relations of selected semantic item. We formulate the joint decisions as an generalized inference task. We employ rich features and constraints (including hard and soft constraints) to infer the joint decisions using a MLN.

Finally, with the inference results, the last step constructs a semantic item query graph, and generates an executable SPARQL query with the question type.

We will give a detailed description of each component and give a step by step explanation with the following example, Figure 2 shows the intermediate results of every steps:

*Which software has been developed by organizations founded in California, USA?.*

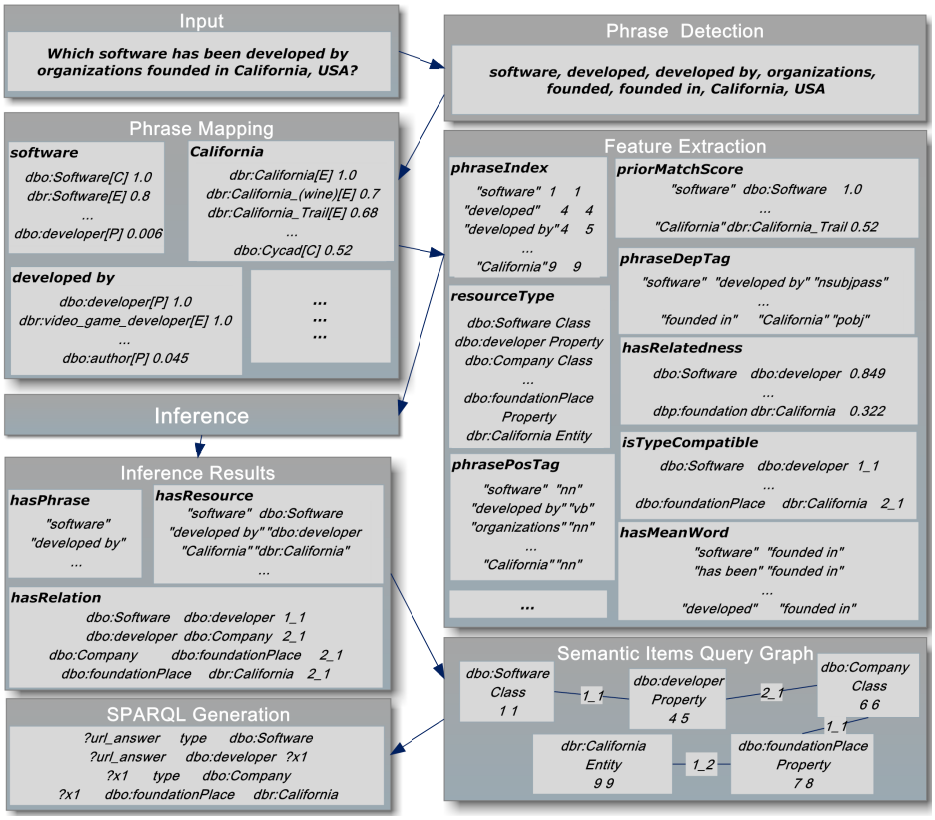


**Fig. 2.** Framework of our system.

### 2.1 System Pipleline

**1) Phrase detection**. Sequences of tokens (phrases) that probably indicate semantic items are detected. To this end, we do not use named entity recognizer (NER) because of its low coverage. [2] To avoid missing useful phrases, we retain all n-grams as candidate phrases, and then use some rules to filter them. The rules include: the length of tokens span must be less than 4 (excepting all contiguous tokens are capitalizations); the POS tag of the start token must be *jj*, *nn*, *rb* and *vb*; all contiguous capitalization tokens must not be split, and so on. For instance, *software*, *developed by*, *organizations*, *founded in* and *California* are detected in the example.

**2) Mapping phrase to semantic item**. After phrases are detected, each phrase may be mapped to the semantic items in KB (entities, classes and properties). For example, *software* is mapped to **dbo:Software**, **dbo:developer**, etc.; *California* is mapped to

[2] We perform testing in two commonly used question corpus (QALD-3 Training data and free917) using Stanford CRF-based NER tool(http://nlp.stanford.edu/software/CRF-NER.shtml). The results demonstrate that merely 51.5% and 23.8% right NEs can be recognized, respectively.

**dbr:California**, **dbr:California_(wine)**, etc. We use different techniques and resources to map phrases to different types of semantic items. For mapping phrases to entities, considering the entities in DBpeida are curated from Wikipeida, we employ anchors, redirections and disambiguations information from Wikipedia. For mapping phrases to classes, considering that classes have lexical variation, especially synonyms, e.g., **dbo:Film** could be mapped from *film*, *movie* and *show*, we use *word2vec* tool[3] to convert every word (phrase) into a vector and compute the similarity between the phrase and the class in KB. The similarity scoring methods are introduced in Section 3.2. Then, the top-N most similar classes for each phrase are returned. For mapping phrases to properties, we employ the resources from PATTY [11] and ReVerb [12]. Specifically, we first compute the associations between the semantic properties in DBpedia and relation patterns in PATTY and ReVerb through instance alignments as same as [13]. Next, if a detected phrase is matched to some relation pattern, the corresponding properties in DBpedia will be returned as the candidates. This step purely constructs a candidate space for every possible mapping, and the decision of selecting the best fitting semantic item is made in the next step.

**3) Feature extraction and joint inference.** There are ambiguities in phrase detection and mapping-phrase-to-semantic-item. This step consists in the resolution of these ambiguities and determine the relations among the mapped semantic items. It is the core contribution of this paper, which performs disambiguation in a unified manner. First, feature extraction is performed to prepare rich features from the question and the knowledge base. Next, the disambiguation is performed in a joint fashion with a Markov Logic Network (MLN). The detailed information will be presented in the next Section.

**4) SPARQL generation.** Based on the inference results, we construct a query graph. The vertex contains the following information: the phrase, token span indexs of the phrase, the mapped semantic item and its type. The edge indicates the relation between two semantic items. For example, we use **1_2** to indicate that the first argument of an item matches the second argument of another item[4]. The right bottom in Figure 2 shows an example of it. The relation in the query graph is paired data merely, but the SPARQL queries need the grouped triples of semantic items. Thus, we convert a query graph into multiple joined semantic triples. Three interconnected semantic items, which must ensure the middle item is a ***property***, are converted into a semantic triple. For example, the query graph $\lceil$**dbo:Book[Class]** $\underset{\leftarrow\rightarrow}{1\_2}$ **dbo:author[Property]** $\underset{\leftarrow\rightarrow}{1\_1}$ **dbr:Danielle_Steel[Entity]**$\rceil$ is converted into $\langle$**?x rdf:type dbo:Book, dbr:Danielle dbo:author ?x**$\rangle$, and $\lceil$**dbo:populationTotal[Property]** $\underset{\leftarrow\rightarrow}{1\_2}$ **dbo:capital[Property]** $\underset{\leftarrow\rightarrow}{1\_1}$ **dbr:Australia[Entity]**$\rceil$[5] is converted into $\langle$**?x1 dbo:populationTotal ?answer, ?x1 dbo:capital dbr:Australia**$\rangle$. If the query graph only contains one vertex which indicates a class ***ClassURI***, we generate $\langle$ **?x rdf:type ClassURI**$\rangle$. If the query graph contains two connected vertexes, we append a variable to bind the missing match argument of the semantic item.

The final SPARQL query is constructed by joining the semantic item triples and combining them with the corresponding SPARQL template. We divide the questions into three types: **Yes/No**, **Number** and **Normal**. Yes/No questions use the ***ASK WHERE***

[3] https://code.google.com/p/word2vec/

[4] The other marks will be introduced in Section 3.1.

[5] corresponding the question "*How many people live in the capital of Australia?*"

template. With respect to number questions, we use ***SELECT COUNT(?url) WHERE*** template, if it cannot obtain a fitting SPARQL query(the query result are not a number), we then use normal question template to generate a query again. Normal questions use the ***SELECT ?url WHERE*** template. For instance, we construct the SPARQL query ***SELECT(?url) WHERE*** $\{$ ***?url rdf:type dbo:Software. ?url dbo:developer ?x1. ?x1 rdf:type dbo:Company. ?x1 dbo:foundationPlace dbr:California.*** $\}$ toward the example.

## 3 Feature extraction & joint inference

In this section, we will first briefly describe Markov Logic Networks. Then, we present the predicates(features) and the first-order logic formulas for joint inference.

### 3.1 Markov Logic Networks

Markov logic networks combine Markov networks with first-order logic in probabilistic framework[10,14]. A Markov logic network consists of a set of first-order formulas. Each formula consists of a set of first-order predicates, logical connectors and variables. However, differently to first-order logic where a formula represents a hard constraints, these logic formulas are softened and can be violated with some penalty (the weight of formula) in MLN.

An MLN $M$ is consists of several weighted formulas $\{(\phi_i, w_i)\}_i$, where $\phi_i$ is a first order formula and $w_i$ is the penalty(the formula's weight). These weighted formulas define a probability distribution over sets of possible worlds. Let $y$ denote a possible world, then $p(\mathbf{y})$ is defined as follows:

$$p(\mathbf{y}) = \frac{1}{Z} exp\left(\sum_{(\phi_i, w_i)\in M} w_i \sum_{\mathbf{c}\in C^{n_{\phi_i}}} f_{\mathbf{c}}^{\phi_i}(\mathbf{y})\right), \tag{1}$$

where each $c$ is a binding of free variables in $\phi_i$ to constants; $f_{\mathbf{c}}^{\phi_i}$ is a binary feature function that returns 1 if the ground formula we get through replacing the free variables in $\phi_i$ with the constants in **c** under the given possible world **y** is true, and 0 otherwise; $C^{n_{\phi_i}}$ is the set of all possible bindings for the free variables in $\phi_i$. $Z$ is a normalization constant. The Markov Network corresponds to this distribution, where nodes represent ground atoms and factors represent ground formulas.

Let us illustrate how MLN determine the relation of semantic items mapped by phrases. The following formulas indicates that if two semantic items have some dependency path tags[6], then they have some type of relations with some weights.

$$\begin{aligned}(\phi_1, w_1) : depPathTag(a, b, \text{“}pobj\text{”}) \wedge (a \neq b)\\ \Rightarrow relation(a, b, \text{“2\_1”}).\end{aligned} \tag{2}$$

$$\begin{aligned}(\phi_2, w_2) : depPathTag(a, b, \text{“}pobj\text{”}) \wedge (a \neq b)\\ \Rightarrow relation(a, b, \text{“1\_1”}).\end{aligned} \tag{3}$$

[6] Actually, the dependency path tags are extracted from the phrases which mapped to semantic items. Here is simplified to illustrate how MLN works.

Here, $a$ and $b$ are variables which represent any semantic item, *depPathTag* and *relation* are an observed predicate and a hidden predicate, respectively. The values of observed predicates are known from feature extraction, and the values of hidden predicates are infered. The values of two weights $w_1$ and $w_2$ affect the decision of choosing *relation* types between two semantic items.

There are a lot of methods to inference and learn the weights for MLN[14,10]. Several packages for MLN learning available online for free, such as Alchemy[7], Tuffy[8], thebeast[9].

### 3.2 Predicates

In MLN, we design several predicates to resolve the ambiguities in phrase detection, mapping phrases to semantic items and semantic items grouping. Specifically, we design a hidden predicate *hasPhrase(i)* to indicate that the *i*th candidate phrase has been chosen, predicate *hasResouce(i,j)* to indicate that the *i*th phrase is mapped to the *j*th semantic item, and predicate *hasRelation(j,k,rr)* to indicate that the *j*th semantic item and the *k*th semantic item can be grouped together with the relation type *rr*. Note that we define four relation types between two semantic items: *1_1*, *1_2*, *2_1* and *2_2*. Here, the relation type *t_s* means the *t*th argument of the first semantic item corresponds to the *s*th argument of the second semantic item. The detailed illustration is shown in Table 1.

**Table 1.** Examples of the relation types.

| Type | Example | Question |
|---|---|---|
| *1_1* | *dbo:height 1_1 dbr:Michael_Jordan* | *How tall is Michael Jordan?* |
| *1_2* | *dbo:River 1_2 dbo:crosses* | *Which river does the Brooklyn Bridge cross?* |
| *2_1* | *dbo:creator 2_1 dbr:Walt_Disney* | *Which spaceflights were launched from Baikonur?* |
| *2_2* | *dbo:birthPlace 2_2 dbo:capital* | *Which actors were born in the capital of American?* |

Moreover, we define a set of observed predicates to describe the properties of phrases, semantic items, relations between phrases and relations between semantic items. The observed predicates and descriptions are shown in Table 2.

Previous methods usually designed some heuristic patterns to group semantic items, which employ the human-designed syntactic path between two phrases to determine the relations between any two phrases. In contrast, we collect all the tokens in the dependency path between two phrases as possible patterns. The predicate *phraseDepTag* and *hasMeanWord* are designed to indicate the possible patterns. Note that if these tokens only contain POS tags *dt|in|wdt|to|cc|ex|pos|wp* or *stop words*, the predicate *hasMeanWord* is false, otherwise is true. In this way, our system is expected to cover more language expressions and answer more questions. Moreover, the SPARQL endpoint is used

[7] http://alchemy.cs.washington.edu
[8] http://hazy.cs.wisc.edu/hazy/tuffy/
[9] http://code.google.com/p/thebeast

**Table 2.** Descriptions of observed predicates.

| | |
|---|---|
| **Describing the attributes of phrases and relation between two phrases** | |
| *phraseIndex(p, i, j)* | The start and end position of phrase *p* in question. |
| *phrasePosTag(p, pt)* | The POS tag of head word in phrase *p*. |
| *phraseDepTag(p, q, dt)* | The dependency path tags between phrase *p* and *q*. |
| *phraseDepOne (p, q)* | If there is only one tag in the dependency path, the predicate is true. |
| *hasMeanWord (p, q)* | If there is any one meaning word in the dependency path of two phrases, the predicate is true. |
| **Describing the attributes of semantic item and the mapping between phrase and semantic item** | |
| *resourceType(r, rt)* | The type of semantic item r. Types of semantic items include *Entity*, *Class* and *Property* |
| *priorMatchScore(p, r, s)* | The prior score of phrase *p* mapping to semantic item *r*. |
| **Describing the attributes of relation between two semantic items in knowledge base** | |
| *hasRelatedness(p, q, s)* | The semantic coherence of semantic items. |
| *isTypeCompatible(p, q, rr)* | If semantic items *p* is type-compatible with semantic items *q*, the predicate is true. |
| *hasQueryResult(s, p, o, rr1, rr2)* | If the triple pattern consisting of semantic items *s*, *p*, *o* and relation types *rr1*, *rr2* have query results, the predicate is true. |

to verify the type compatibility of two semantic items and whether or not one triple pattern can obtain query results.

The predicate *hasRelatedness* needs to computes the coherence score between two semantic items. Following Yahya et al. [9], we use the Jaccard coefficient based on the inlinks between two semantic items.

The predicate *priorMatchScore* assigns a score prior to mapping a phrase to a semantic item. We use different ways to compute this scores for different semantic item types. For entities, we use a normalized score based on the frequencies of a phrase referring to a entity. For classes and properties, we use different methods. At first, we define three similarity score metrics as follows: a) $s_1$: Levenshtein distance score between the labels of semantic item and phrase; b) $s_2$: word embedding score, which is the similarity between two phrases, is the maximum value of the cosine of words between two phrases; c) $s_3$: instance overlap score, which is computed using the Jaccard coefficient of instance overlap as a similarity score. The prior scores for mapping phrases to classes and properties are $\alpha s_1 + (1-\alpha)s_2$ and $\alpha s_1 + \beta s_2 + (1-\alpha-\beta)s_3$, respectively. The parameters are set with empirical values[10].

### 3.3 Formulas

We use two kinds of formulas for jointly decisions: *Boolean* and *Weighted* formulas. *Boolean* formulas are hard constraints which must be satisfied with the entire ground atoms in final inference results. *Weighted* formulas are soft constraints which could be violated with some penalties.

**1) Boolean Formulas (Hard Constraints)** Table 3 lists the boolean formulas used in this work. The "_" notation indicates an arbitrary constant. The "||" notation expresses the number of true grounded atoms in the formula. These formulas express the following constraints:

***hf1***: if a phrase is chosen, then it must have a mapped semantic item;

***hf2***: if a semantic item is chosen, then its mapped phrase must be chosen;

[10] Set $\alpha$ to 0.6 for class, set $\alpha$ and $\beta$ to 0.3 and 0.3 for property, respectively.

**Table 3.** Descriptions of boolean formulas.

| | |
|---|---|
| *hf1* | $hasPhrase(p) \Rightarrow hasResource(p, \_)$ |
| *hf2* | $hasResource(p, \_) \Rightarrow hasPhrase(p)$ |
| *hf3* | $\lvert hasResource(p, \_)\rvert \leq 1$ |
| *hf4* | $!hasPhrase(p) \Rightarrow !hasResource(p, r)$ |
| *hf5* | $hasResource(\_, r) \Rightarrow hasRelation(r, \_, \_) \vee hasRelation(\_, r, \_)$ |
| *hf6* | $\lvert hasRelation(r1, r2, \_)\rvert \leq 1$ |
| *hf7* | $hasRelation(r1, r2, \_) \Rightarrow hasResource(\_, r1) \wedge hasResource(\_, r2)$ |
| *hf8* | $phraseIndex(p1, s1, e1) \wedge phraseIndex(p2, s2, e2) \wedge overlap(s1, e1, s2, e2) \wedge hasPhrase(p1) \Rightarrow !hasPhrase(p2)$ |
| *hf9* | $resourceType(r, \text{“}Entity\text{”}) \Rightarrow !hasRelation(r, \_, \text{“}2\_1\text{”}) \wedge !hasRelation(r, \_, \text{“}2\_2\text{”})$ |
| *hf10* | $resourceType(r, \text{“}Entity\text{”}) \Rightarrow !hasRelation(\_, r, \text{“}2\_1\text{”}) \wedge !hasRelation(r, \_, \text{“}2\_2\text{”})$ |
| *hf11* | $resourceType(r, \text{“}Class\text{”}) \Rightarrow !hasRelation(r, \_, \text{“}2\_1\text{”}) \wedge !hasRelation(r, \_, \text{“}2\_2\text{”})$ |
| *hf12* | $resourceType(r, \text{“}Class\text{”}) \Rightarrow !hasRelation(\_, r, \text{“}2\_1\text{”}) \wedge !hasRelation(r, \_, \text{“}2\_2\text{”})$ |
| *hf13* | $!isTypeCompatible(r1, r2, rr) \Rightarrow !hasRelation(r1, r2, rr)$ |

**Table 4.** Descriptions of weighted formulas.

| | |
|---|---|
| *sf1* | $priorMatchScore(p, r, s) \Rightarrow hasPhrase(p)$ |
| *sf2* | $priorMatchScore(p, r, s) \Rightarrow hasResource(p)$ |
| *sf3* | $phrasePosTag(p, pt+) \wedge resourceType(r, rt+) \Rightarrow hasResource(p, r)$ |
| *sf4* | $phraseDepTag(p1, p2, dp+) \wedge hasResource(p1, r1) \wedge hasResource(p2, r2) \Rightarrow hasRelation(r1, r2, rr+)$ |
| *sf5* | $phraseDepTag(p1, p2, dp+) \wedge hasResource(p1, r1) \wedge hasResource(p2, r2) \wedge !hasMeanWord(p1, p2) \Rightarrow hasRelation(r1, r2, rr+)$ |
| *sf6* | $phraseDepTag(p1, p2, dp+) \wedge hasResource(p1, r1) \wedge hasResource(p2, r2) \wedge phraseDepOne(p1, p2) \Rightarrow hasRelation(r1, r2, rr+)$ |
| *sf7* | $hasRelatedness(r1, r2, s) \wedge hasResource(\_, r1) \wedge hasResource(\_, r2) \Rightarrow hasRelation(r1, r2, \_)$ |
| *sf8* | $hasQueryResult(r1, r2, r3, rr1, rr2) \Rightarrow hasRelation(r1, r2, rr1) \wedge hasRelation(r2, r3, rr2)$ |

***hf3***: a phrase can map to one semantic item at most;
***hf4***: if the phrase is not chosen, then its mapping semantic item should not be chosen;
***hf5***: if a semantic item is chosen, then it should have one relation with other semantic items at least;
***hf6***: two semantic items have one relation at most;
***hf7***: if a relation for two semantic items is chosen, then they must be chosen;
***hf8***: each two chosen phrases must not overlap;
***hf9*, *hf10*, *hf11*, *hf11***: the semantic item with type *Entity* and *Class* should not have second argument matching with others;
***hf12***: The chosen relation for two sematic items must be type-compatible.

**2) Weighted Formulas (Soft Constraints)** Table 4 lists the weighted formulas used in this work. The "+" notation in the formulas indicates that each constant of the logic variable should be weighted separately. Those formulas express the following properties in joint decisions:
***sf1*, *sf2***: The larger the score of phrase mapping to semantic item, the more likely the corresponding phrase and semantic item should been chosen;
***sf3***: there are some associations between POS tags of phase and types of mapped semantic item;
***sf4*, *sf5*, *sf6***: there are some associations between the dependency tags in the depen-

dency pattern path of two phases and the types of relation of two mapped semantic items;
***sh7***: the larger the relatedness of two semantic items, the more likely they have a relation;
***sf8***: if the triple pattern has query results, those semantic items should have corresponding relations.

## 4 Results and discussion

Stanford dependency parser [15] is used for extracting features from dependency parse trees. We use the toolkit *thebeast*[11] to learn the weights for the formulas and perform MAP inference. The inference algorithm uses a cutting plane approach. And for parameter learning, we set all initial weights to zeros and use online learning algorithm with MIRA update rules to update the weights of formulas. The numbers of iterations for training and testing are set to 10 and 200 epochs, respectively.

Our system could learn the effectiveness patterns. We show the weights of the learned patterns corresponding with formula *sf3* in MLN, as shown in Table 5. From the table, we can see that *nn* is mapped to *Entity* more likely than *Class* and *Property*, and *vb* is most likely mapped to *Property*. It proves that our model can learn the effective and reasonable patterns for QALD.

**Table 5.** Sample weights of formulas, corresponding with formula *sf3*.

| POS tag of Phrase | type of mapped Item | Weight |
|---|---|---|
| *nn* | *Entity* | 2.11 |
| *nn* | *Class* | 0.243 |
| *nn* | *Property* | 0.335 |
| *vb* | *Property* | 0.517 |
| *wp* | *Class* | 0.143 |
| *wr* | *Class* | 0.025 |

Table 1 gives the evaluation results with average precision, average recall and F-measure. It shows the number of question our system can answer, the number of right and partially right answers among them.

**Table 6.** Evaluate results on QALD-4 test dataset.

| | Total | Processed | Right | Partially | Recall | Precision | F-measure |
|---|---|---|---|---|---|---|---|
| Xser | 50 | 40 | 34 | 6 | 0.71 | 0.72 | 0.72 |
| gAnswer | 50 | 25 | 16 | 4 | 0.37 | 0.37 | 0.37 |
| **CASIA** | 50 | 26 | 15 | 4 | 0.40 | 0.32 | 0.36 |
| Intui3 | 50 | 33 | 10 | 4 | 0.25 | 0.23 | 0.24 |
| ISOFT | 50 | 50 | 10 | 3 | 0.26 | 0.21 | 0.23 |
| RO_FII | 50 | 50 | 6 | 0 | 0.12 | 0.12 | 0.12 |

[11] http://code.google.com/p/thebeast

Among the factors that affect performance most are: 1) training set consisting of 110 questions is limited, we found some weights of grounded formula are zero; 2) the parameters are used for computing prior score mapping phrase to semantic item are difficult to tune, because we use different method to entities, classes and properties; 3) lack of global constraints, such as, it is hard to count the number of unmapped tokens in question.

## 5 Conclusion

In this paper, we present a question answering system over Linked Data which translates the natural language questions into the standard RDF data queries (SPARQL). We present a jointly learning framework for phrase detection, phrases mapping to semantic items and semantic items grouping. The novelty of our method lies in that we make joint inference and pattern learning for all subtasks in QALD by using first-order logic. Our benchmark results demonstrate the effectiveness of the proposed method.

## References


1. C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *International journal on semantic web and information systems*, vol. 5, no. 3, pp. 1–22, 2009.
2. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*. Springer, 2007, pp. 722–735.
3. K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *SIGMOD*, 2008.
4. F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *WWW*, 2007.
5. S. Walter, C. Unger, P. Cimiano, and D. Bär, "Evaluation of a layered approach to question answering over linked data," in *The Semantic Web–ISWC 2012*. Springer, 2012, pp. 362–374.
6. V. Lopez, V. Uren, M. Sabou, and E. Motta, "Is question answering fit for the semantic web?: a survey," *Semantic Web*, vol. 2, no. 2, pp. 125–155, 2011.
7. V. Lopez, E. Motta, and V. Uren, "Poweraqua: Fishing the semantic web," in *The Semantic Web: research and applications*. Springer, 2006, pp. 393–410.
8. C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and P. Cimiano, "Template-based question answering over rdf data," in *WWW*, 2012.
9. M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum, "Natural language questions for the web of data," in *EMNLP*, 2012.
10. M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.
11. N. Nakashole, G. Weikum, and F. Suchanek, "Patty: a taxonomy of relational patterns with semantic types," in *EMNLP*, 2012.
12. A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *EMNLP*, 2011.
13. J. Berant, A. Chou, R. Frostig, and P. Liang, "Semantic parsing on freebase from question-answer pairs," in *EMNLP*, 2013.
14. S. Riedel, "Improving the accuracy and efficiency of map inference for markov logic." UAI, 2008.
15. M.-C. De Marneffe, B. MacCartney, C. D. Manning *et al.*, "Generating typed dependency parses from phrase structure parses," in *LREC*, 2006.