Svega: Answering Natural Language Questions over Knowledge Base with Semantic Matching

Gaofeng Li, Pingpeng Yuan, and Hai Jin
Services Computing Technology and System Lab. / Cluster and Grid Computing Lab. / Big Data Technology and System Lab.
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
Email: {gaofengli, ppyuan, hjin}@hust.edu.cn

***Abstract*—Nowadays, more and more large scale knowledge bases are available for public access. Although these knowledge bases have their inherent access interfaces, such as SPARQL, they are generally unfriendly to end users. An intuitive way to bridge the gap between users and knowledge bases is to enable users to ask questions with natural language interface and return desired answers directly. Here the challenge is how to discover the query intention of users. Another challenge is how to obtain accurate answers from knowledge bases. In this paper, we model the query intention with a graph based on an entity-driven method. Consequently, the core problem of natural language question answering can be treated as subgraph matching over knowledge bases. For a query graph, there is a huge number of candidate mappings in a knowledge base, including ambiguities. Thus, a semantic vector is proposed to address disambiguation by evaluating the semantic similarity between edges in a query graph and paths in a knowledge base. By this way, our system can extract accurate answers directly without any offline work. Extensive experiments over the series of QALD challenges show the effectiveness of our system Svega in terms of recall and precision against other state-of-the-art systems.**

I. Introduction

Nowadays, many knowledge bases, such as DBpedia [1], Yago [2], are available for public access. Distinct from document bases of information retrieval systems, knowledge bases integrate substantial small facts, known as triples (subject-predicate-object). To access these knowledge bases, SPARQL, a SQL-like query language, is provided as a standard interface. However, SPARQL is unfriendly to general users because of its complex syntax. Although keyword search, commonly applied in information retrieval, is simple and convenient for end users, it may not work properly when employing keyword search to retrieve answers from knowledge bases, because it is difficult for keywords to express the query intention completely. For example, the keywords of the question *"Who starred in the films that were directed by Stanley Kubrick?"* may be *"star"*, *"film"*, and *"Stanley Kubrick"*. However, if only taking into account these keywords, the real query intention, *actors of the films*, will be missing. Compared with SPARQL and keyword, natural language is not only intuitive to end users, but also able to express users' query intention accurately. Thus it is important to bridge the gap between unstructured natural language and structured knowledge bases.

DOI reference number: 10.18293/SEKE2018-119

One way to bridge the gap is to translate a natural language question to a structured query and then extract answers from the mapping results of the query. However, it is difficult to convert a natural language question to a structured query. One reason is that most structured queries need to specify the query statement accurately, while some query statements have ambiguities between natural language and knowledge bases. Considering the above example, the phrase *"Stanley Kubrick"* may refer to <Stanley_Kburick> or <Stanley_Kubrick_Archive> in knowledge bases. The exact meaning of phrases depends on the context of questions and knowledge bases. In order to clarify the meaning of phrases and obtain correct mappings from knowledge bases, some QA (*Question Answering*) systems provide candidates for users to interactively choose [3], which actually is a controlled natural language. The above example question can be converted into *"Who dbp:starring the dbo:Film that were-dbp:director res:Stanley_Kubrick?"* by replacing the phrases with the words of a knowledge base. This approach facilitates QA systems, but it requires users to do much. Other QA systems, such as gAnswer [4], automatically map phrases in the natural language into words in knowledge bases based on a dictionary, which is generally built manually or using machine learning approach. However, it is impossible for the offline-built dictionary to indicate comprehensive mapping relationships between phrases in the natural language and knowledge bases.

Here we present an online natural language question answering system Svega. An entity-driven method is proposed to translate a natural language question into a query graph. Consequently, the problem of natural language question answering is converted to subgraph matching over knowledge bases. To address disambiguation, we first search isomorphism subgraphs of the query graph from a knowledge base based on vertex mapping. Then a semantic vector is proposed to filter ambiguities by evaluating semantic similarity. Our contributions are as follows.

1) An online question answering system Svega is presented. Svega provides a natural language interface for general users to retrieve desired answers from knowledge bases directly, without any offline work.
2) Query graph is constructed to model the query intention of a natural language question by adopting an entity-

driven approach. In this approach, entities are identified firstly and then the query intention is explored by extracting predicates based on grammatical relationships.
3) A semantic vector is proposed to represent the semantics of paths (namely edges in query graph and paths in knowledge base). Thus, the problem of disambiguation is addressed by evaluating the semantic similarity between a query graph and mapping subgraphs online.
4) Extensive experiments on the standards of QALD series competitions are conducted and the experimental results demonstrate that Svega has a competitive performance in terms of recall and precision.

II. Related Work

There are many QA systems available. The traditional QA systems can only return texts related to keywords. Although some extending QA systems, such as [5], consider semantics of results, they still can not retrieve answers directly. The development of knowledge bases, storing fine-grained facts, makes it possible to directly return desired answers [6], [7], [8], [9]. Since the inherent interfaces, such as SPARQL, provided by knowledge bases are too complex for general users, current QA systems allow users to employ natural language to access knowledge bases. According to the restriction on natural language, these systems can be broadly classified into two categories: controlled natural languages, which restrict the grammar and vocabulary in order to reduce ambiguity and complexity, and un-controlled natural language.

Controlled natural language. These systems using this approach provide some candidate entities and predicates for users' choice [3], [7]. Then users choose words from candidates to indicate their query intention. For instance, the example can be transferred to *"Who dbp:starring the dbo:Film that were-dbp:director res:Stanley_Kubrick?"*. In the question, *"dbp:starring"*, *"dbo:Film"*, *"dbp:director"*, and *"res:Stanley_Kubrick"* are not natural language words, but entities and predicates from a knowledge graph. Thus, this way limits the expressiveness and usability of QA systems [3], but improves the correctness to answer questions.

Un-controlled natural language. Different from the approaches with controlled natural languages, users can answer questions using any words in thees systems. Then the QA systems usually translate natural language questions into structured queries, such as SPARQL. In this approach, the key is how to identify entities and map entities into words of knowledge bases. For example, Xser [8] maps phrases into words of knowledge bases using an ad-hoc lexicon. CASIA [9] detects phrases by grammatical token and then uses Markov Logic Network to resolve ambiguities. gAnswer [4] builds a paraphrase dictionary offline to achieve the mapping of predicate. However, the dictionary built offline can not indicate the exact meaning of each predicate because language is live. All in all, these systems address disambiguation only by taking into account the semantic of single words instead of context of the words.

III. Preliminary and Overview of Svega

A. Preliminary

The core issues of answering natural language questions over knowledge bases is to bridge the gap between structured knowledge bases and unstructured natural language question. Regrading this problem, we first model the query intention of a natural language question with query graph *Q*.

Definition 1: **(Query Graph).** A query graph is denoted as *Q = (V, E)*, where *V* is a set of vertices, corresponding to entities or variables. Specially, the query intention or variable is represented as *"?"*. The edge between vertex v_i, v_j can be denoted as *triple T =* $< v_i,\ r,\ v_j >$, where *r* represents the relation between v_i, v_j.

Except variable vertices, there are two kinds of vertices in query graph. We name a vertex with an entity label as a *Key Vertex* (KV). The other vertices are not assigned a label because they are not indicated in questions, which are named as *Hidden Vertex* (HV) since they are implicit. For example, Fig.1 shows the query graph of the running question. <*Stanley_Kubrick, directed, HV*> is a triple, in which <*Stanley_Kubrick*> is a *KV*. The graph also has a *HV*, which is the set of films that were directed by *Stanley Kubrick*.

Now, answering natural language question is actually to find matches of a query graph over knowledge bases. When mapping a query graph to a knowledge base, ambiguities will be introduced. For a candidate mapping of an entity in *Q*, if there is no isomorphism subgraph containing it, it must be an ambiguity. For example, the vertex <Stanley_Kubrick_Archive> in Fig.1 is a ambiguity. Based on this idea, we address disambiguation with subgraph matching, which considers the schema of knowledge bases.

Definition 2: **(Match).** A subgraph *S* in a knowledge base is a match of query graph *Q* if and only if the following conditions hold:

1) $\forall v \in V$ of Q, $\exists u \in S$ where the vertex u is a mapping of the vertex v;
2) $\forall e : (v_i, v_j) \in E$ of Q, $\exists (u_i, u_j) \in S$, associated with e, and where the two vertices u_i and u_j are mappings of v_i and v_j respectively.

The mappings of vertex *"?"* in *Q* will be answers of the query. In Fig.1, vertex <Walter_Cartier> and <Stadmueller> are the matches of vertex *"?"*. Thus, they are two answers of the running question.

B. Overview of Svega

Our approach mainly consists two parts: *Entity-driven Query Graph Construction* and *Disambiguation with Query Graph Mapping*. Fig.1 shows the framework of our system.

Entity-driven Query Graph Construction. In order to model the query intention of a natural language question with *Q*, we extract *triple*s (see Definition 1) from natural language question firstly, because *triple* has a simpler structure. Based on this idea, we propose an entity-driven approach to extract *triple*s and construct *Q* by joining the same vertex of *triple*s. The details will be described in Sect. IV.

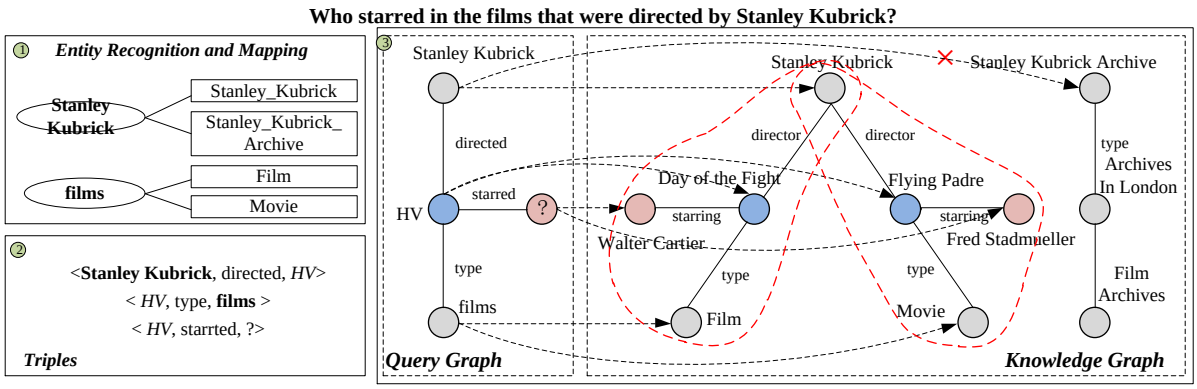


Fig. 1: System Overview

Disambiguation with Query Graph Mapping. Naturally, a query graph matching mainly includes vertices and edges mapping, which will introduce ambiguities. To address disambiguation, we first search isomorphism subgraphs of Q based on vertices mapping. Then a semantic vector is proposed to prune ambiguities by evaluating the semantic similarity. The details will be described in Sect. V.

IV. Entity-driven Query Graph Construction

In a query graph, any edge with two endpoints can be represented as a triple. Thus, we first extract *triple*s from a natural language question. Each *triple* is composed by two entities (vertices in a graph) and the relation between them (edge between two vertices). We then construct the query graph of the questions from the *triple*s. To extract triples, the system needs to identify entities and relationship between entities indicated in natural language questions. The words to show relationship are various in natural language while the form of entity is relatively simple. They can be verb and adjective phrases etc. Here, we first identify entities and then propose an entity-driven approach to extract relational phrases based on the grammatical relationships between words.

A. Entity Identification

Here, we employ DBpedia Spotlight [10] and Stanford Parser [11] to analyze a sentence and generate a *dependency tree* to indicate grammatical relationships between words. The output is generally a dependency tree. For example, Fig.2 is the tree of the example question. There are many auxiliary words ("in", "that" etc) in a dependency tree while a query graph only contains key entities and relationships between them. So a dependency tree is still far from a query graph. In order to construct Q, we need to know both the key entities and theirs semantics. For instance, the category words (e.g. *"film"* in the running example) generally indicate 'is-a' relationship. In the case, we will add some extra nodes and edges to show this. As shown in Fig.2, we will add a node and an edge labeled *"type"* which represents the fact that *"film"* is a *"type"*.

Some questions may contain hidden information. The example question *"films that were directed by Stanley Kubrick"* has a hidden entity *"that"* which represents all the films directed by *Stanley Kubrick*. Here, we denote it by a *HV*. *HV*s can be identified based on the structural feature of vertices in a query graph. If a vertex is adjacent to only one vertex, it is a *KV*, otherwise it is a *HV*. So identifying *HV*s needs more information. In the running example, *"film"* and *"Stanley Kubrick"* can be recognized in this stage, and *HV*s will be inferred in the following stage.

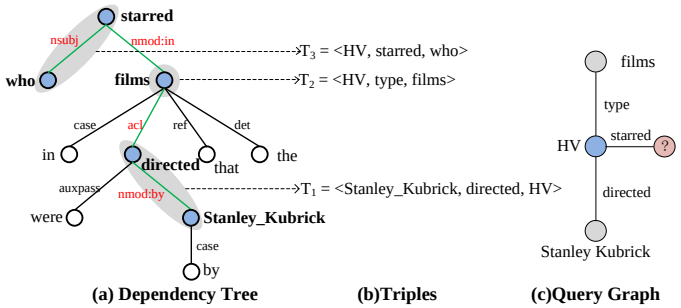


Fig. 2: Examples of Dependency Tree, Triple, and Query Graph

B. Recognizing Relation

Relation phrases generally co-occur with the corresponding entities. So they can be recognized from the words that have grammatical relationships with the obtained entities. Although a *dependency tree* [12] can reveal grammatical relationships between words in a sentence, the words connecting entities is not only the words that represent relations, but also auxiliary words. Here a relation priority is proposed to denote the possibility of a word to represent relation. The priority is computed based on the possibility of the grammatical relationships which are listed as follow in a non-ascending order [1]: *nsubj, subj, obj, dobj, nsubjpass, pobj, nmod:*, amod, prep, acl, auxpass, case, ref, det*.

In Fig.2, the word *"were"*, connecting to the word *"directed"*, will be pruned, because the edge label *"auxpass"*,

[1]These grammatical relationships are defined in [12]. For example, "nsubj" refers to a noun phrase which is the syntactic of a clause.

Algorithm 1: Extracting *T*s from a dependency tree

Input: Dependency tree, recognized entities set *RES*
Output: *T*s
Define a variable *var*;
for *each unvisited element in* RES **do**
  The element, *e*, is the member of a *T* and set *var* = *e*;
  Carry out traversal in the dependency tree from *var*;
  Select the edge with a highest-priority relationship;
  if *the* V *is a class word* **then**
    Insert a *HV* to current *T*;
    Build a new *T*, composed by *V*, *type* and a *HV*;
    Continue;
  else
    Insert *V* and *HV* to the current *T*;
    Build a new *T* and insert a *HV* to the new *T*;
    Set *var* = *V*;
    Continue;
  end
end

representing passive auxiliary, has a low priority. Simultaneously, *HV*s can be inferred according to the proposed structural feature of vertices.

Here, Algorithm 1 is used to extract *triples*. It starts from the entities recognized but not category words. For instance, the entity *"Stanley_Kubrick"* will be an element of T_1. Then, because the priority of *"nmod:by"* is higher than *"case"*, *"directed"* is selected as a relation. Then, the class word *"films"* will be traveled and it will introduce the relation *"type"*. So here a *HV* is needed because it will connect two edges with label *"directed"* and *"type"* respectively. So <*Stanley_Kubrick, directed, HV*> and <*HV, type, film*> are two triples extracted from the question. Next, *"starred"* and *"Who"* are travelled, and they are composed the T_3. Finally, the obtained three *T*s are composed together to build the *Q* by joining the same vertex.

V. Disambiguation with Query Graph Mapping

Once the query graph is built, we need to find subgraphs from a knowledge base, which match the semantics of the query graph. It is a NP hard problem to find the best mapping [4], and ambiguities will be introduced. Here we first filter ambiguities based on structural mapping, then a semantic vector is proposed to achieve semantic mapping.

A. Structural Mapping Based on Vertices

Query graph mapping includes vertices and edges mapping. Due to this reason, we first extract isomorphism subgraphs based on vertices mapping. The general way to map vertices, with entity label, from query graph to knowledge bases is by computing string similarity or edit distance. By this way, an entity in the natural language may correspond to several candidates in a knowledge base. For instance, the possible mappings of the entity *"Stanley Kubrick"* in the running question include <Stanley_Kubrick>, <Stanley_Kubrick_Arechive>

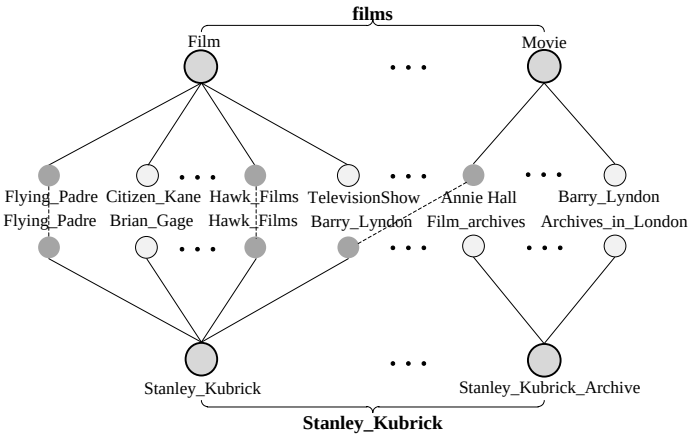


Fig. 3: Query Graph Mapping Based on Entities

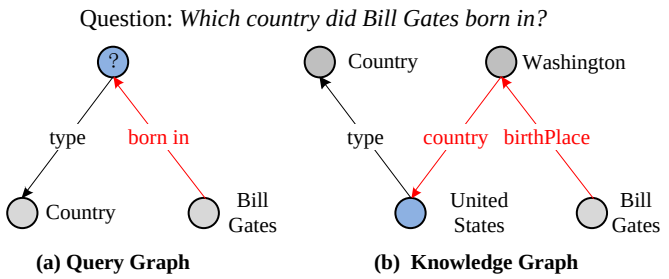
Algorithm 2: Pruning paths based on entities

Input: Knowledge graph *KG*, query graph *Q*
Output: mapping results of *Q*
for *each vertex* V_i *in the* Q **do**
  Obtain the mapping candidate set C_i of V_i;
  for *each element* c_j *in* C_i **do**
    Carry out BFS in *KG* from the vertex c_j;
  end
end
for *each edge between two vertex* V_i *and* V_j *in the* Q **do**
  Select two elements from the two candidate sets C_i and C_j respectively and find paths between them;
end

etc. Moreover, in order to ensure that all answers can be obtained, synonyms should also be considered, which is an entity linking problem. In our work, Lookup [13] is adopted to link an entity to candidate entities in a knowledge graph. Up to now, all vertices in a query graph will have some candidate results and the corresponding paths only occur between these candidate results, so we can prune the paths that have no connection with the entities mapping results. Based on this idea, we propose the Algorithm 2 to prune paths in the knowledge graph. Fig.3 shows temporary results of the running example, and some ambiguities, such as <Stanley_Kubrick_Archive>, will be filtered because it is isolation in the mapping results.

B. Semantic Mapping with Path Vector

After obtaining isomorphism subgraphs of a query graph, there are still ambiguities in candidates, because only the structure of a query graph is considered, while the semantics of a query graph is not taken into account. For example, the vertex <*Hawk_Films*>, included in the mapping results, is not a correct answer of the running question, and the predicate, between <*Hawk_Films*> and <*Stanley_Kubrick*> in the knowledge graph, is irrelevant to the corresponding relation *"directed"* in *Q*. Consequently, the pruning method should be executed according to the semantic confidence.



(a) Query Graph **(b) Knowledge Graph**

Fig. 4: Example of Edge and Mapped Path

An edge with a predicate label in a query graph may be mapped to a path in a knowledge graph, as shown in Fig.4. So, the key is how to represent the semantics of them and evaluate the similarity between them. Here, we propose semantic vector to quantify the semantic information of predicates and paths.

1) Path Vector: Generally, the semantic vector obtained by Glove [14] can only represent a single word. In knowledge bases, a path contains several edges, each of which has its labels. For example, in the path <*birthPlace, country*>, *"birthPlace"* and *"country"* will have a semantic vector representation respectively. In addition, semantics of edges in a knowledge graph between different vertices are irrelevant. So we propose the method that composing the semantic vectors together based on the synthesis of vector additions to compute the path vector α.

2) Predicate Vector: Generally, a predicate is a phrase. We also need a method to synthesize the predicate vector. Different from pathes in knowledge graph, the words in a phrase are relevant. For example, the words *"born"* and *"in"* are composed together to represent the relationship that someone is born in a place. The contribution of each word in the predicate is different. The word *"in"* occurs in many phrases, while the word *"born"* only appears in a few phrases. Thus, the word *"born"* is more important than *'in"*. However, we can not ignore *"in"* because it indicates a born place instead of time or anything else. Thereby, we use the *tf-idf* proposed firstly in information retrieval to measure the importance of a word to a predicate phrase.

Assume a predicate phrase p is composed of word w_i (i = *1, ..., n*). The *tf*-value of w_i is defined as follows:

$$tf(w_i, p) = |\{w_i \mid w_i \in p\}| \tag{1}$$

The *idf*-value of word w_i over the phrase dictionary d is defined as follows:

$$idf(w_i, d) = \log \frac{|d|}{|\{p \in d \mid w_i \in p\}| + 1} \tag{2}$$

Thus, the *tf-idf* value of w_i can be computed as following:

$$tf - idf(w_i, p, d) = tf(w_i, p) * idf(w_i, d) \tag{3}$$

According to the *tf-idf* value of each word, we define the importance weight of word w_i in predicate p as following:

$$\psi(w_i, p) = tf - idf(w_i, p, d) \tag{4}$$

Since each w_i has a vector v, we can compute the predicate vector β as follows:

$$\beta = \sum\nolimits_{w_i \in p} \psi(w_i, p) * v_i \tag{5}$$

C. Ranking Subgraph Matches

The subgraph matching mainly includes vertices and edges mapping. Consequently, semantic confidence of vertices and edges are composed together to measure the semantic similarity of a subgraph, which also reflect the confidence of the answer.

Definition 3: **(Answer Confidence)**. Given a query graph Q of a question and a mapping result M. Let $\phi(v, u)$ be the confidence between vertex $v \in Q$ and $u \in M$. And $\varphi(\overline{vw}, P(x, y))$ is the confidence between edge $\overline{vw}$ of Q and path *P(x, y)* of M. Thus, the answer confidence, *AC(Q, M)*, is defined as follows:

$$AC(Q, M) = \sum\nolimits_{v_i \in Q \ \&\& \ u_i \in M} \phi(v_i, u_i) + \sum\nolimits_{\overline{v_i v_j} \in Q \ \&\& \ P(u_i, u_j) \in M} \varphi(\overline{v_i v_j}, P(u_i, u_j)) \tag{6}$$

where

$$\varphi(\overline{v_i v_j}, P(u_i, u_j)) = \alpha(P(u_i, u_j)) \cdot \beta(\overline{v_i v_j}) \tag{7}$$

VI. Evaluation

In this section, we evaluate our system Svega against some existing popular natural language question answering systems using the QALD series benchmarks. Here, we do not choose squall2sparql [7] as a competitor because the input of squall2sparql is controlled language question rather than uncontrolled natural language.

A. Data Sets

QALD series competitions are one of important benchmarks to evaluate natural language question answering system. Here we choose QALD-3 and QALD-4 as many research did. According to the requirements of the QALD series competitions, DBpedia series knowledge bases are also used in the experiments and managed by TripleBit [15].

B. Effectiveness of Question Answering

We report the experimental results in Table I (QALD-3) and Table II (QALD-4). The experimental results of our competitors on QALD-3 and QALD-4 are available at the official website[2] of QALD, in which *Proceed* indicates the number of questions that can be return non-empty answers by these systems.

Table I shows that Svega is the best natural language question answering system on QALD-3, with highest *recall*, *precision*, and *F-measure*. Our system can answer 44 questions correctly and 9 questions partially, while CASIA [9] can only answer 29 questions all right.

In QALD-4, we can see that Svega outperforms all competitors on both *recall* and *precision* (Table II). Both the *recall* and

[2]https://qald.sebastianwalter.org/

TABLE I: QALD-3 on DBpedia 3.8

	Proceed	Right	Partially	Recall	Precision	F-measure
CASIA	52	29	8	0.36	0.35	0.36
Scalewelis	70	32	1	0.33	0.33	0.33
RTV	55	30	4	0.34	0.32	0.33
Intui2	99	28	4	0.32	0.32	0.32
SWIP	21	15	2	0.16	0.17	0.17
Svega	96	44	9	**0.52**	**0.52**	**0.52**

TABLE II: QALD-4 on DBpedia 3.9

	Proceed	Right	Partially	Recall	Precision	F-measure
Xser	40	34	6	0.71	0.72	0.72
gAnswer	25	16	4	0.37	0.37	0.37
CASIA	26	15	4	0.40	0.32	0.36
Intui3	33	10	4	0.25	0.23	0.24
ISOFT	28	10	3	0.26	0.21	0.23
RO_FII	50	6	0	0.12	0.12	0.12
Svega	48	35	6	**0.76**	**0.76**	**0.76**

precision of Svega are 0.76, while the *recall* and *precision* of the best competitors Xser [8] is 0.71 and 0.72 respectively. In addition, Xser needs to train a KB-independent model offline before it answers questions, while Svega does not need to train any model in advance.

C. Effectiveness of Query Graph Building and Mapping

We implement a system by replacing the semantic vector method of Svega with paraphrase dictionary method used in gAnswer. We name it as *ED+PD*.

The experimental results show that ED+PD is not better than Svega in all aspects (Table III, IV). It confirms that the similarity evaluating method of semantic vector is very effective, because the difference between ED+PD and Svega is only the similarity evaluating method. The results also show ED+PD outperforms gAnswer. It indicates that the entity-driven method of our system has more advantages on building query graph, because ED+PD and gAnswer use same dictionary, but different approach to build query graph.

TABLE III: Results on QALD-3

	Proceed	Right	Partially	Recall	Precision	F-measure
Svega	96	44	9	0.52	0.52	0.52
ED+PD	96	36	9	0.43	0.43	0.43
gAnswer	76	32	11	0.40	0.40	0.40

TABLE IV: Results on QALD-4

	Proceed	Right	Partially	Recall	Precision	F-measure
Svega	48	35	6	0.76	0.76	0.76
ED+PD	48	22	4	0.55	0.55	0.55
gAnswer	25	16	4	0.37	0.37	0.37

VII. Conclusions

In this paper, we present Svega - an online natural language question answering system over knowledge bases. Moreover, the query intention of a natural language question is modeled by a query graph based on an entity-driven method. As a result, the problem of natural language question answering over knowledge graph is converted to subgraph mapping. At last but not least, predicate vector and path vector are proposed to measure the semantic confidence between predicates and paths. Consequently, our approach is effective in the terms of recall and precision.

Acknowledgment

The research is supported by The National Key Basic Research Program (No. 2018YFB1004000002), NSFC (No. 61672255), Science and Technology Planning Project of Guangdong Province, China (No.2016B030306003 and 2016B030305002), and the Fundamental Research Funds for the Central Universities, HUST.

References

[1] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015.

[2] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *Proc. of WWW'07*. ACM, 2007, pp. 697–706.

[3] G. M. Mazzeo and C. Zaniolo, "Answering controlled natural language questions on RDF knowledge bases," in *Proc. of EDBT'16*, 2016, pp. 608–611.

[4] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao, "Natural language question answering over RDF: a graph data driven approach," in *Proc. of SIGMOD'14*. ACM, 2014, pp. 313–324.

[5] H. Bast and B. Buchhold, "An index for efficient semantic full-text search," in *Proc. of CIKM'13*. ACM, 2013, pp. 369–378.

[6] L. Shao, Y. Duan, X. Sun, H. Gao, D. Zhu, and W. Miao, "Answering who/when, what, how, why through constructing data graph, information graph, knowledge graph and wisdom graph," in *Proc. of SEKE'17*. KSI, 2017, pp. 1–6.

[7] S. Ferré, "squall2sparql: a translator from controlled english to full sparql 1.1," in *Proc. of Working Notes for CLEF'13*. Springer, 2013.

[8] K. Xu, Y. Feng, S. Huang, and D. Zhao, "Question answering via phrasal semantic parsing," in *Proc. of CLEF'15*. Springer, 2015, pp. 414–426.

[9] S. He, Y. Zhang, K. Liu, and J. Zhao, "Casia@v2: A mln-based question answering system over linked data," in *Proc. of CLEF'14*. Springer, 2014, pp. 1249–1259.

[10] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes, "Improving efficiency and accuracy in multilingual entity extraction," in *Proc. of I-Semantics'13*. ACM, 2013, pp. 121–124.

[11] S. Schuster and C. D. Manning, "Enhanced english universal dependencies: An improved representation for natural language understanding tasks," in *Proc. of LREC'16*, pp. 2371–2378.

[12] M. C. D. Marnee and C. D. Manning, "Stanford typed dependencies manual," Stanford University, Tech. Rep., 2008.

[13] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, R. Cyganiak, and S. Hellmann, "Dbpedia - a crystallization point for the web of data," *J. Web Sem.*, vol. 7, no. 3, pp. 154–165, 2009.

[14] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proc. of EMNLP'14*. ACL, 2014, pp. 1532–1543.

[15] P. Yuan, P. Liu, B. Wu, H. Jin, W. Zhang, and L. Liu, "Triplebit: a fast and compact system for large scale RDF data," *PVLDB*, vol. 6, no. 7, pp. 517–528, 2013.