ISOFT at QALD-4: Semantic similarity-based question answering system over linked data

Seonyeong Park, Hyosup Shim, and Gary Geunbae Lee

Department of Computer Science and Engineering,
Pohang University of Science and Technology, Pohang, Gyungbuk, South Korea

{sypark322, hyosupshim, gblee}@postech.ac.kr

Abstract. We present a question answering system over linked data. We use natural language processing tools to extract slots and SPARQL templates from the question. Then, we use semantic similarity to map a natural language question to a SPARQL query. We combine important words to avoid loss of meaning, and compare combined words with uniform resource identifiers (URIs) from a knowledgebase (KB). This process is more powerful than comparing each word individually. Using our method, the problem of mapping a phrase of a user question to URIs from a KB can be more easily solved than without our method; this method improves the F-measure of the system.

Keywords: Question answering, SPARQL, Semantic similarity

1 Introduction

Question answering (QA) systems (QASs) find the answers to natural language (NL) questions by analyzing huge corpora. In the big data era, this property of QA is increasingly necessary. Two popular types of QAS are those based on information retrieval (IR) and those based on knowledgebases (KBs).

Most IR-based QASs use the following general strategy [1, 2, 3]: 1) analyze the question and translate the questions into the queries for IR, 2) retrieve a small passages including answers of the document collection by using IR technology, and 3) extract answer candidates and select the final answer. Typically, the question analysis module involves NL processing (NLP) such as part-of-speech (POS) tagging and dependency parsing.

Recently, very large, structured, and semantically-rich KBs have become available; examples include Yago [4], DBpedia [5], and Freebase [6]. As an increasing quantity of resource description framework (RDF) data are published in linked form, intuitive ways of accessing the data are becoming increasingly necessary. DBpedia forms the nucleus of the web of linked data [7]; it inter-connects large-scale-RDF data sources with 2.46 billion subject-predicate-object triples. Several QASs use RDF data; examples include Aqualog [8], **F**eedback, **R**efinement and **E**xtended Vocabular**Y** **A**ggregation (FREyA) [9], Template-Based SPARQL Learner (TBSL) [10] and the ontology-

based QAS Pythia [11]. Most KB-based QASs use the following general strategy [8,9,10,11,12,14]: 1) analyze the question, 2) map the entity and predicate from user question to the words in a KB, 3) formulate and select queries, and 4) search queries and extract answers. Strategies of KB-based QASs are not much different from those of IR-based QASs, but require that the user's phrase be translated to words that exist in a KB.

In FALCON [1], an IR-based QAS, the question is parsed to extract the expected answer type and an ordered list of keywords that are used to retrieve relevant text passages. However, the keyword search lacks a clear specification of the relations among entities.

Translating NL questions into SPARQL query requires performing both a disambiguation task and a mapping task. **DE**ep **A**nswers for ma**N**y **N**aturally **A**sked questions (DEANNA) [12] is based on an integer linear program to solve several disambiguation tasks jointly: segmentation of questions into phrases; mapping of phrases to semantic entities, class and relations; and construction of SPARQL triple patterns.

In this paper, we aim to solve the problem of translating the NL question into a SPARQL query to search for the answer in the KB. To improve the F-measure of the system, we use linguistic information about the user question, and semantic similarity for translating the NL words into the words in the KB.

We used semantic similarity based on Explicit Semantic Analysis (ESA) [13] for mapping predicates in the user NL question to predicate uniform resource identifiers (URIs) in the KB. ESA converts target strings to semantic vectors that can convey their explicit meaning as weighted vectors of Wikipedia concepts, so calculating the similarity of two vectors reveals the semantic relatedness of the two strings from which the vectors were generated. We also increased the effectiveness of mapping the NL words to URIs by concatenating additional-information to the predicate.

However, previous work did not use our approach in this problem of using semantic vectors from Wikipedia. FREyA uses a method that generates suggestions with string similarity and hierarchy defined in an ontology, scores them again using string similarity algorithms, then maps them to high-scoring concepts in the ontology. PowerAqua [14] includes a linguistic component part, an element mapping component (PowerMap), a triple mapping component, and a merging and ranking component. PowerMap provides automatic mapping for inter-ontology concepts and semantic relevance analysis; it calculates semantic relatedness as the distance between corresponding senses in Wordnet's graph.

2 System Description

2.1 Overall system architecture

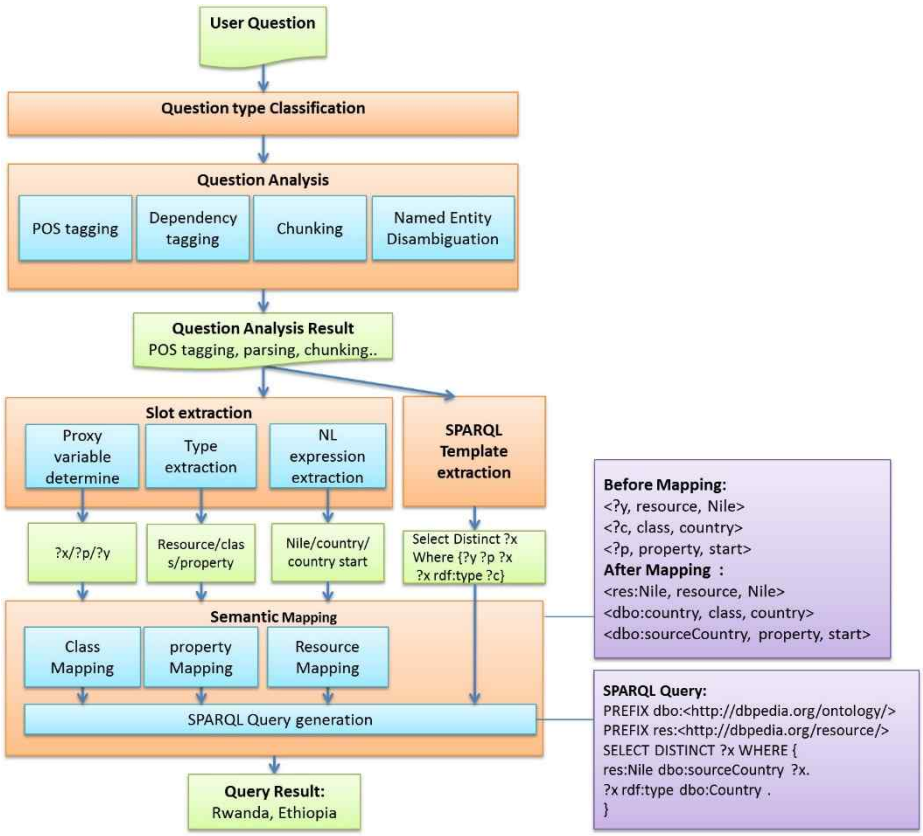

Fig. 1. Overall proposed QA system: processes are described in the text

We built our system by extending TBSL [10]. We added more question analysis techniques and ESA for measuring semantic similarity. The user question was translated into a SPARQL query in several steps (**Fig. 1**). We must extract the slots and the SPARQL template. The SPARQL templates correspond directly to the internal structure of the question [10]; they specify the query's "select" or "ask" clause, its filter and aggregation functions, and the number and forms of its triples. Each slot has three components: a proxy variable; the type of intended URI (class, property or resource); and the NL expression. The proxy variable is replaced with the appropriate URI after it is identified. The translation steps with examples are:

1. *NL question:*
   In which country does the Nile start?

2. *Slot* :
   <?x, resource, Nile> <?c, class, country> <?p, property, start >
3. *SPARQL Template:*
   SELECT ?x WHERE {?x ?p ?y . ?x rdf:type ?c}
4. *Slot with URI* :
   < res:Nile, resource, Nile> <dbo:country, class, country> <dbo:sourceCountry, property, start>
5. *SPARQL query:*

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
SELECT DISTINCT ?x WHERE{
res:Nile dbo:sourceCountry ?x .
?x rdf:type dbo:Country .}
```

2.2 Slot extraction

We use NL analysis to extract slots. Given NL question inputs, we classify questions to detect question-type keywords (e.g., "who", "what", "when", "where", and "how"). The heuristics to extract slots and templates differ slightly among question types. After the question is classified, the question is analyzed at the word, syntactic, and semantic levels. For these analyses, we used ClearNLP[1], which is available in github. Additionally, we used open NLP[2] for chunking. For named entity (i.e., resource) disambiguation, we used AIDA[3]. Additionally, we used keywords of questions provided by Question Answering over Linked Data (QALD-4[4]) organizers. Using the result of NL analysis, we developed several rules to extract slots.

First, we check all words in the NL question and find a word to maximize the appropriateness score (eq. 1), which will become a class (i.e., answer type). The class is a type of proxy variable *x* that the question seeks.

$$\textit{Appropriateness score}_{class}(\mathbf{w}_i) = \boldsymbol{wt}_{pos}(\boldsymbol{w}_i) + \boldsymbol{wt}_{depm\ v}(\boldsymbol{w}_i) + \boldsymbol{wt}_{hdqt}(\boldsymbol{w}_i), \quad (1)$$

where $\boldsymbol{w}_i$ is word *i* in the question, $\boldsymbol{wt}_{pos}(\boldsymbol{w}_i)$ is the pre-defined weight when the POS of $\boldsymbol{w}_i$ is *NN* or *NNS*; otherwise $\boldsymbol{wt}_{pos}(\boldsymbol{w}_i)$ is 0. $\boldsymbol{wt}_{depm\ v}(\boldsymbol{w}_i)$ is the pre-defined weight when $\boldsymbol{w}_i$ is a dependency of the main verb; otherwise $\boldsymbol{wt}_{depm\ v}(\boldsymbol{w}_i)$ is 0. $\boldsymbol{wt}_{hdqt}(\boldsymbol{w}_i)$ is the pre-defined weight when $\boldsymbol{w}_i$ is a head of question type; otherwise $\boldsymbol{wt}_{hdqt}(\boldsymbol{w}_i)$ is 0. We determine appropriateness scores by analyzing the answer type in QALD-4 training data and UIUC data [15]. The following heuristic is used to extract

[1] http://clearnlp.wikispaces.com/
[2] https://opennlp.apache.org/
[3] https://www.mpi-inf.mpg.de/yago-naga/aida/
[4] http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=task1&q=4

the class in all question types. If the class is not found, we regard pre-defined class (e.g., place) as type of proxy x followed by question-type keywords (e.g., "where").

Heuristics Class Extraction

Input ***S*** – Natural language question
Output ***c*** – Class

Initialize class c as null; the appropriateness score of null is 0; the number of words in the NL question is *n*
1: **For** all words in the NL question
  1: **Calculate** the appropriateness score of w_i and the appropriateness score of *current c*
  2: **Compare** the appropriateness score of w_i and the appropriateness score of *current c*
  3: **Update** current *c* as the lemma of w_i if the appropriateness score of w_i > *current c*
2:**Endfor**
3:**Return** *c*

When the named entities or noun phrases consist of more than one word, we use a dependency parser and a chunker to concatenate words in the noun phrase into one word before applying the heuristic rules. For example, first we use AIDA or given keywords to detect the named entity (i.e., resource). Named entities are usually subjects or objects in the NL questions. We used the dependency parser to concatenate "Walt" and "Disney" to "Walt Disney", which we regard as one word.

We use AIDA and given keywords to extract the named entity. We use the chunker to extract the noun phrases and our heuristics to extract the class and then apply the chunker to obtain the predicate between the subject and object (**Fig. 2**). First, we extract entities as a subject and object, for example, "television show" and "Walt Disney". To extract the predicate between them, we use the dependency parser to detect the head of each word. We finally extract the predicate, which is a chunk that includes the two heads. If the heads are the same word, we regard the head as the predicate. If the two heads are not in the same chunk, we concatenate two chunks into the predicate.

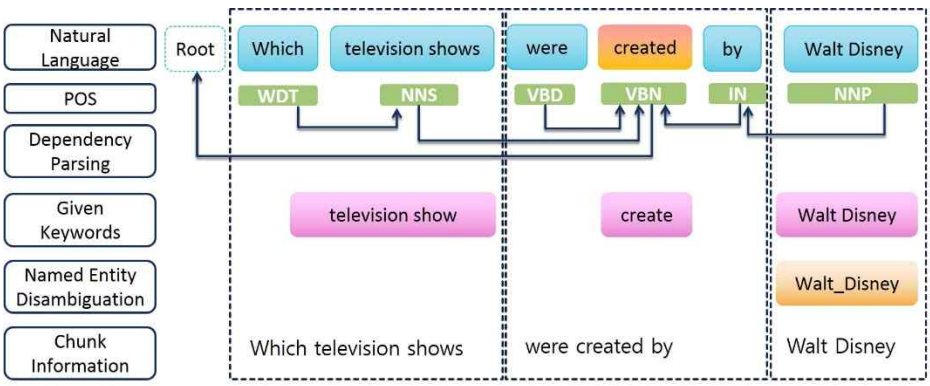

Fig. 2. Example of extracting a predicate: processes are described in the text

After the QALD-4 deadline passed, we added graph-based reduction rules (**Fig. 3**) to extract slots.

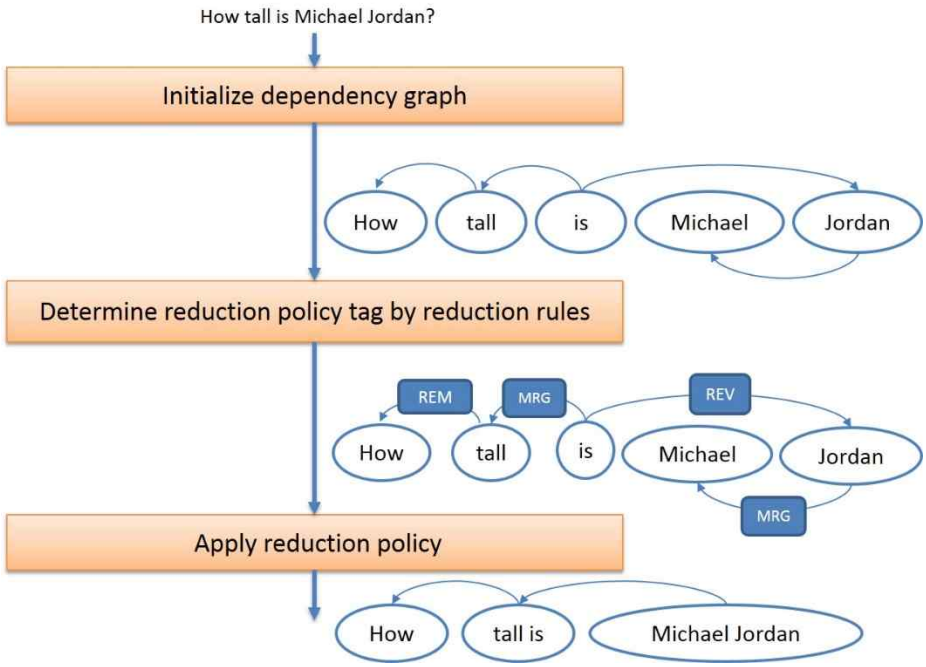

Fig. 3. Overall steps to map a graph-based reduction: processes are described in the text

1. *Initialize dependency graph*: We obtain a dependency graph from the result of the dependency parser. Each node has information such as word surface form and lemma form, and each edge represents a dependency relation.
2. *Determine reduction policy tag by reduction rules:* We give four policy tags, remove (OMT), merge (MRG), reverse (REV), and remain (REM) to each node followed by rules that check information such as the POS of the head, the dependent of the POS, the dependency-label (e.g., SUB or OBJ) and the word of the dependent and the head.
3. *Apply Reduction policy:* We apply four reduction policies to the initial dependency graph to reduce it.

These graph-based reduction rules improve our total system F-measure in the QALD-4 test dataset, but they still have limitations. First, devising rules to determine tags is a difficult task. Moreover, we determine each node as subject, predicate or object by the distance from "question-type keywords" (e.g., "who", "what", "when", "where", and "how"). For a question such as "Give me all movies with Tom Cruise.", the slots are difficult to extract.

2.3 SPARQL Template extraction

We used lexical information from each question to extract the appropriate SPARQL template.

- Template for a Boolean question: *ASK WHERE { ?x ?p ?y. }*

  Using lexical information, we detect whether the question is a "wh-question". If the question is a "yes/no question", it does not include "question-type keywords". We regard each question as a "yes/no question" if it contains neither question type words nor "list-question keywords" (e.g., "Give", "List").

- Template for a simple question: *SELECT DISTINCT ?x WHERE { ?x ?p ?y . (Option:?x rdf:type ?c.)}*

  We define the basic query template as above. Optionally, we used ?c (replaced with class) if the class is exactly correct. Our class detections module works well for a "which" question or a "who" question. We usually include only one triple. We use lexical information (conjunction such as "and" and relative pronoun such as "who") and a dependency parser to extract *n* triples, but this remains a difficult task. We must extract more than one triple to successfully translate the NL question into a SPARQL query in some cases. However, we cannot extract *n* triples in many cases such as "Give me all films produced by Steven Spielberg with a budget of at least $80 million.", because it does not include a conjunction or relative pronoun explicitly; to extract *n* triples is a difficult task to complete by only using heuristics.

- Template for including an aggregation function question: We used the "aggregation" attribute, which indicates whether any operations beyond triple pattern matching are required to answer the question. We define "aggregation" functions as "count", filter" and "order", and define the modifier target as a proxy that is the target of the aggregation function. We used three types of functions, for example:

  — *COUNT: SELECT COUNT (DISTINCT ?x) WHERE { ?x ?p ?y. }*

  — *ORDER: SELECT DISTINCT ?x WHERE {?x ?p ?y. } ORDER BY DESC(?x) OFFSET 0 LIMIT n*

  — *FILTER: SELECT DISTINCT ?x WHERE {?x ?p ?y . FILTER (?y <1950)}*

  Some words indicate the types of functions to use. We define these words as "aggregation indicators". For example, if the sentence contains "How many" we extract COUNT template and extract the target, i.e., the head of the "many". If the question contains a superlative, we infer that the question requires an ORDER operation. If the question contains a comparative, we infer that the question requires a FILTER operation. We use the head-dependency relation of an "aggregation indicator" to detect the targets of these types of operations and the constant (e.g., 1950) that the filter uses.

2.4 *Mapping the proxy variable to the URI*

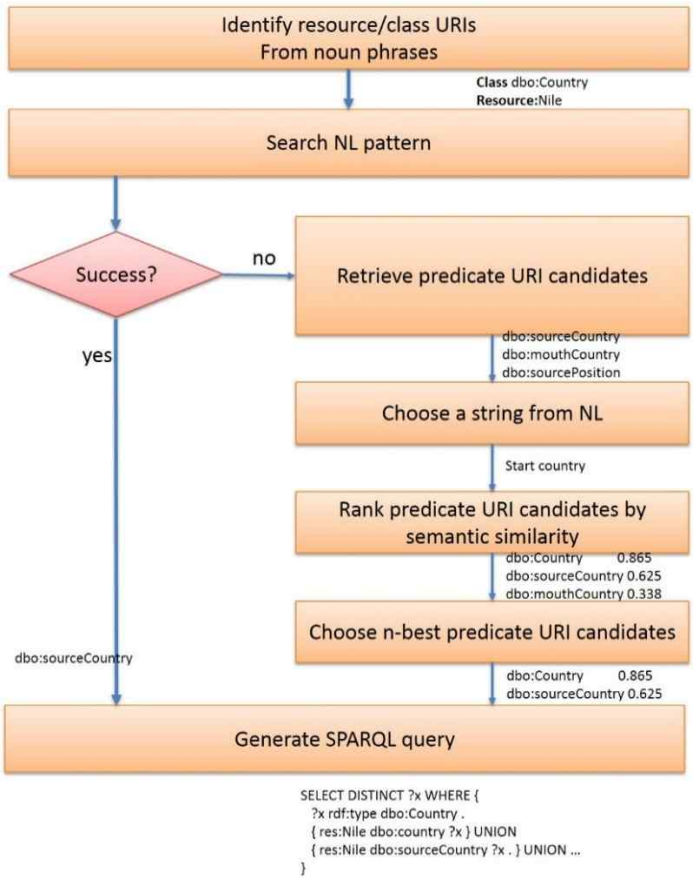

Fig. 4. Overall steps to map the Proxy to the URI: processes are described in the text

The process (**Fig. 4**) for mapping proxy variables to URIs entails the following seven steps:

1. *Identify resource/class URIs from noun Phrases*: We use noun phrases delivered from the slot extraction module to identify URIs from the KB; normal noun phrases are regarded as class type URIs, and proper noun phrases are regarded as resource-type URIs. We built an inverted index of all resource/class type URIs from DBpedia; consequently, the actual mapping is performed by searching the inverted index with variant forms of NPs, which were generated by heuristics in capitalization and white spacing; this process works well. To map quickly, we used lucene[5].

[5] http://lucene.apache.org/

2. *Search the NL pattern:* We used PATTY[6] to map NL predicates to the words in the KB. However, in most cases, the PATTY pattern repository failed to map the predicates from the questions into the appropriate words in KB.
3. *Retrieve predicate URI candidates:* We determine that the predicate URI is a word from the user's phrase; this word is translated to words extracted from the KB. We collect predicate URI candidates occurring in triples that have a resource or class that was previously identified as a subject or object. Queries to retrieve predicates are structured as follows:

<*SPARQL Query for collecting predicate URI candidates from* ***resource***>

```
SELECT DISTINCT ?p WHERE {
{<IDENTIFIED_RESOURCE_URI> ?p [] .}
UNION{[]?p<IDENTIFIED_RESOURCE_URI>.}
}
```

<*SPARQL Query for collecting predicate URI candidates from* ***class***>

```
SELECT DISTINCT ?p WHERE {
{?x ?p []. ?x rdf:type <IDENTIFIED_CLASS_URI>.} UNION
{[] ?p ?x. ?x rdf:type <IDENTIFIED_CLASS_URI>.}
}
```

4. *Choose a string from the NL against which to measure the semantic relatedness:* Pattern matching alone is insufficient to translate predicates in the NL question to predicate URIs. The slot extraction phase delivers several strings for measuring the semantic relatedness. Each string includes a different set of classes that restrict the meaning of the predicate. Restricting the meaning of the predicate helps ESA to choose the most relevant predicates. For example, the NL question "In which country does the Nile start?" requires the predicate URI "sourceCountry" for class type: Country. Measuring the semantic relatedness using the string "start" did not assign a high score to the desired predicate URI, but using the string "start country" did. Adding class information such as "start country" to the predicate is better than using only a predicate such as "start" to represent the user intention and semantic meaning of the question.
5. *Rank predicate URI candidates by semantic similarity:* Calculate the semantic relatedness between each predicate URI in the KB and the string from the question.
6. *Choose n-best predicate URI candidates:* Choose *n*-best similar predicate URIs from the KB. We experimentally determined *n* many times. In our cases, $n > 2$ was not helpful; it decreased the precision much more than it increased the recall.
7. *Query generation with n-best predicates:* Replace the slots with identified URIs and complete the query. This process identifies the proxy variables that the question requires.

[6] http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/patty/

3 Experiment

We used the QALD-4 test dataset for task 1 to evaluate the proposed method and used DBpedia 3.9[7] as the KB. We mainly compare two approaches. One (P) uses only pattern matching to identify resource/class/predicate URI, and the other (P+ESA) combines pattern matching to identify the resource/class URI and ESA to identify the predicate URI. The latter yielded a higher F-measure than did the former. Additionally, we added graph-based reduction rules to the latter, and this P+ESA+graph approach yielded the highest F-measure. We used the ESA open library[8]. The test dataset includes 31 simple questions, four yes/no questions and 15 questions with aggregation.

4 Results and Discussion

To evaluate the QA system, we used global precision, recall and F-measure (**Table 1**). Graph-based reduction reduces the parse tree by applying predefined reduction rules. P+ESA+graph yielded a higher F-measure than did P+ESA. Graph-based reduction is in initial development; we will continue to improve it. We do not have results for graph-based reduction in **Tables 2-4**. In this section, we mainly compare our semantic similarity approach and pattern matching approach. We usually used pattern matching (e.g., capitalization and white spacing) to map a class and resource. However, most predicates cannot be mapped using pattern matching, so we used ESA when mapping the NL predicate to the predicate URI in the KB. We show the number of questions that were answered successfully by predicate mapping using ESA (**Table 2**) and pattern matching (**Table 3**). The proposed method of computing semantic similarity achieved higher precision and recall than did the previous pattern matching approach (**Tables 2, 3**).

Table 1. Evaluation results for the QALD-4 test dataset.

Method	Total	SPARQL generation	Correct	Partially Correct	Global Recall	Global Precision	Global F-measure
P	50	0	0	0	0.00	0.00	0.00
P + ESA	50	28	10	3	0.26	0.21	0.23
P+ESA+graph	50	31	16	2	0.36	0.33	0.34

Table 2. Precision (P) and recall (R) using ESA to map the NL predicate to the appropriate predicate URI in the KB. **Bold**: only answers found; *underlined italic*: several candidate answers found (some not appropriate); normal: no answers found.

	R = 1	0 < R < 1	R = 0
P = 1	**10**	*0*	0
0 < P < 1	*3*	*0*	0
P = 0	0	0	15

[7] http://dbpedia.org/

[8] http://ticcky.github.io/esalib/

Table 3. Precision (P) and recall (R) using pattern matching to map the predicate to the appropriate predicate URI in the KB. **Bold:** only answers found; *underlined italic*: several candidate answers found (some not appropriate); normal: no answers found.

	R = 1	0 < R < 1	R = 0
P = 1	**0**	*0*	0
0 < P < 1	*0*	*0*	0
P = 0	0	0	28

We successfully extracted slots and templates for 38 of the 50 questions. We finally generated 28 SPARQL queries using P+ESA and 31 SPARQL queries using P+ESA+graph. Among them, we used P+ESA to extract 10 correct answers and three partially correct answers (**Table 1**). We used P+ESA to map 13 NL predicates to appropriate predicate URIs in the KB without graph-based reduction (**Table 2**). The number of appropriate mapping results was the same as the result of the whole system. We used pattern matching to map three NL predicates to the appropriate predicate URIs in the KB (e.g., we can match "are spoken in" to "spokenIn"). We used both PATTY and the pattern matching (e.g., white spacing and capitalization). PATTY failed to map predicates in all cases. The pattern matching mapped only three appropriate predicate URIs but could not map classes and resources in the three questions, so we did not generate SPARQL queries from them and obtained no answer (**Tables 1, 3**).

Consequently, we focused on using P+ESA. We successfully extracted slots and templates for 38 questions and then generated 28 SPARQL queries. Many of the queries failed to map the proxy variables to the appropriate URIs for some of them; this is a difficult task. We analyzed error cases to learn ways to improve the global F-measure (**Table 4**).

Table 4. Analysis of Error Cases

Error Case	Example	Solution
Failed to detect entity in sentences and map to a correct resource/class URI in the KB.	How many James Bond movies are there? - "James Bond" was given by the analysis result, but "James Bond movies" was required to map. How deep is Lake Placid? - Several entities have the same name and no additional information was provided that could help to select the correct one.	Provide multiple analysis results and try them one by one if required.
Failed to map to a correct predicate URI in the KB.	Does the Isar River flow into a lake? - "flow into" was extracted from the sentence. However, it was mapped to "riverMouth", instead of "inflow" Where was Bach born? - NL predicate "was born place" was mapped to the predicate URI "deathPlace" in the KB.	These are cases that require certain inferences for mapping to the correct predicate URI. The semantic relatedness measure currently in use

		is based on TF/IDF and is not sophisticated enough. More sophisticated measuring similarity method may be useful.
Failed to extract correct triple patterns.	What was Brazil's lowest rank in the FIFA World Ranking? - "FIFA Word Ranking" should be treated as a predicate, but the cue gave a wrong hint that this should be treated as a resource. Give me the grandchildren of Bruce Lee. - Complex predicate "grandchildren" in the user question should be handled as a composite of the predicate "children".	A dictionary for implicit predicates or complex predicates may be useful.

5 Conclusions and future work

Using a dependency parser and other NLP, we extracted appropriate slots for SPARQL queries. We used prepositions to find URIs as well as nouns and adjectives, which are known as the most useful terms to find keywords. Additionally, we combined pattern matching with semantic similarity based on ESA to identify URIs. To improve the ESA result, we compared not only the predicate with predicate URIs in DBpedia but also concatenated the predicate with additional information to increase the concreteness of the meaning representation. This approach is more accurate than the pattern matching approach. By concatenating additional information to the predicate, we increased the effectiveness of mapping proxies to URIs. Our approach is domain-independent; consequently, it can be adapted easily to other KBs such as Yago2 and Freebase.

The proposed system in this paper is our initial system, and we will continue to develop it. In future work, we will develop our graph-based reduction rules, extract slots by reducing the number of nodes in the dependency parse tree and will use machine-learning algorithms. Furthermore, we will develop methods to select valid answers among the answer candidates.

Acknowledgments. This work was partly supported by the ICT R&D program of MSIP/IITP [10044508, Development of Non-Symbolic Approach-based Human-Like Self-Taught Learning Intelligence Technology] and the National Research Foundation of Korea (NRF) [NRF-2014R1A2A1A01003041, Development of multi-party anticipatory knowledge-intensive natural language dialog system].

References

1. Harabagiu, S. M., Moldovan, D. I., Pasca, M., Mihalcea, R., Surdeanu, M., Bunescu, R. C., ... & Morarescu, P. (2000, November). FALCON: Boosting Knowledge for Answer Engines. In TREC (Vol. 9, pp. 479-488).
2. Harabagiu, S. M., Maiorano, S. J., & Pasca, M. A. (2003). Open-domain textual question answering techniques. Natural Language Engineering, 9(3), 231-267.
3. Lee, G. G., Seo, J., Lee, S., Jung, H., Cho, B. H., Lee, C., ... & Kim, K. (2001, November). SiteQ: Engineering High Performance QA System Using Lexico-Semantic Pattern Matching and Shallow NLP. In TREC.
4. Suchanek, F. M., Kasneci, G., & Weikum, G. (2007, May). Yago: a core of semantic knowledge. In Proceedings of the 16th international conference on World Wide Web (pp. 697-706). ACM.
5. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., ... & Bizer, C. (2013). DBpedia-a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web Journal.
6. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., & Taylor, J. (2008, June). Freebase: a collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 1247-1250). ACM.
7. Heath, T., & Bizer, C. (2011). Linked data: Evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology, 1(1), 1-136.
8. Lopez, V., Uren, V., Motta, E., & Pasin, M. (2007). AquaLog: An ontology-driven question answering system for organizational semantic intranets. Web Semantics: Science, Services and Agents on the World Wide Web, 5(2), 72-105.
9. Damljanovic, D., Agatonovic, M., & Cunningham, H. (2012, January). FREyA: An interactive way of querying Linked Data using natural language. In The Semantic Web: ESWC 2011 Workshops (pp. 125-138). Springer Berlin Heidelberg.
10. Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A. C., Gerber, D., & Cimiano, P. (2012, April). Template-based question answering over RDF data. In Proceedings of the 21st international conference on World Wide Web (pp. 639-648). ACM.
11. Unger, C., & Cimiano, P. (2011). Pythia: Compositional meaning construction for ontology-based question answering on the Semantic Web. In Natural Language Processing and Information Systems (pp. 153-160). Springer Berlin Heidelberg.
12. Yahya, M., Berberich, K., Elbassuoni, S., Ramanath, M., Tresp, V., & Weikum, G. (2012, July). Natural language questions for the web of data. In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (pp. 379-390). Association for Computational Linguistics.
13. Gabrilovich, E., & Markovitch, S. (2007, January). Computing Semantic Relatedness Using Wikipedia-based Explicit Semantic Analysis. In IJCAI (Vol. 7, pp. 1606-1611).
14. Lopez, V., Fernández, M., Motta, E., & Stieler, N. (2012). PowerAqua: Supporting users in querying and exploring the Semantic Web. Semantic Web, 3(3), 249-265.
15. Xin, L., & Dan, R. (2002, August). Learning question classifier. In Proceedings of the 19th International Conference on Computational Linguistics, Taipei (pp. 556-562).