Semantic Question Answering System over Linked Data using Relational Patterns

Sherzod Hakimov Hakan Tunc Marlen Akimaliev Erdogan Dogdu

TOBB University of Economics and Technology, Ankara, Turkey
{hakimov, hakantunc, makimaliev, edogdu}@etu.edu.tr

ABSTRACT

Question answering is the task of answering questions in natural language. Linked Data project and Semantic Web community made it possible for us to query structured knowledge bases like DBpedia and YAGO. Only expert users, however, with the knowledge of RDF and ontology definitions can build correct SPARQL queries for querying knowledge bases formally. In this paper, we present a method for mapping natural language questions to ontology-based structured queries to retrieve direct answers from open knowledge bases (linked data). Our tool is based on translating natural language questions into RDF triple patterns using the dependency tree of the question text. In addition, our method uses relational patterns extracted from the Web. We tested our tool using questions from QALD-2, Question Answering over Linked Data challenge track and found promising preliminary results.

Categories and Subject Descriptors

H.5.2 [**Information Systems**]: User interfaces-natural language, Theory and methods

I.2.1 [**Artificial Intelligence**]: Natural language interfaces.

General Terms

Algorithms, Performance, Design, Experimentation.

Keywords

Question Answering, Pattern Extraction, Linked Data, Semantic Web.

1. INTRODUCTION

Recently, question answering on the web gained momentum due to the large structured knowledge bases such as DBpedia, Freebase, YAGO that regularly collect information from open and ever expanding knowledge resources such as Wikipedia.

Web of Data and Linked Data

Linked data refers to the Web of data in contrast to the Web of documents. Linked data extends the current Web that consists of documents and the links between documents. In the case of linked data or the Web of data, meaningful links with types between data elements exist unlike the links in the Web of documents where links are only untyped references in the form of hyperlinks. Linked data is therefore more structured and machine processable; applications can traverse this Web of data, easily find useful data and pinpoint the right information [12]. In the Web of documents, or the current web, searching and finding information is by way of parsing documents and looking for useful information by matching keywords and terms or using some natural language processing techniques; thus it is a dummy search. Instead, users can query structured databases like DBpedia[1], YAGO[2] or Freebase[3] to find the exact information such as who the president of the USA, or the population of Italy, etc. However, querying these knowledge bases requires the knowledge of RDF data model and the related ontologies. For example the above facts are presented in DBpedia in the form of RDF triples as follows: <United_States, leaderName, Barack_Obama> and <Italy, populationCencus, "59.464.644">. But someone to find this information, that person has to know leaderName and populationCencus properties and the regarding entities to formulate the queries.

If search engines or question answering systems can make use of linked data and translate the natural language questions to linked data queries automatically, users can find the intended information much faster.

DBpedia

DBpedia is one of the central linked data datasets in Linked Open Data[4] project [13]. It is created by converting infobox information of Wikipedia articles to RDF data model. Latest version of DBpedia contains more than 3.77 million things, including 764.000 persons, 573.000 places, 112.000 music albums, 72.000 films and 18.000 video games, 192.000 organizations, etc. in 111 different languages.

In this paper, we present a method to translate natural language questions to linked data queries. We developed a tool that takes natural language questions, converts them to SPARQL[5] queries for DBpedia and answer the questions automatically.

We explain our approach in detail in Section 2. In Section 3 we present the evaluation results and show that the method is capable of translating natural language questions to SPARQL queries and provide an answer. We present the related work in section 4 and

EDBT/ICDT '13, March 18 - 22, 2013, Genoa, Italy.

[1] http://dbpedia.org
[2] http://yago-knowledge.org
[3] http://freebase.com
[4] http://linkeddata.org
[5] http://www.w3.org/TR/rdf-sparql-query

some discussion on the topic in section 5, and conclude in Section 5.

2. PROPOSED APPROACH

The question answering system in our approach consists of the following 3 main steps:

(1) Triple pattern extraction: Candidate RDF triples are extracted from natural language questions using the dependency graph and POS tags of the questions.
(2) Entity and property extraction: Using RDF triples from the previous step, all subject, predicate and objects are mapped to DBpedia entities, classes or properties.
(3) Answer extraction: Candidate triples are queried over DBPedia and all the answers matching the expected type of a question are ranked and the top result is given as an answer.

These steps are explained below in detail:

2.1 Triple Pattern Extraction

In order to delve into the linked data ocean we have to talk with its own language that is not English or any other natural language. Just like words' combination build up a sentence, linked data is the building block of the Semantic Web and it is based on uniform triples. Thus, we need to translate natural language words into triples' elements and sentences into triples.

To extract triples from the natural language sentences, we are using Stanford CoreNLP[6] (SCNLP) [2] [3] library which provides a number of natural language analysis tools. SCNLP basically generates the tree structure of the English language text. It helps us analyze the parts of speech (POS), tags of phrases and word dependencies. By utilizing this information, our triple pattern extraction tool determines the triple patterns and the relations between the triples.

We provide a sentence to SCNLP and receive dependency tree as the one shown in Figure 1. The tree includes the POS tags and the relation of children phrases with its parent. The relations include a wide range of tags such as subject, object and their types, participial modifier, prepositional modifier and copula. Starting from the root of the tree we examine each node with its children. We treat a node and its children as a subtree and by looking their POS tags, relation tags and children's own triples, we decide if they make up any triple. We apply the same procedure to the children of the node. Then, we consider if the triples should be combined by one common element of the triples. We traverse the whole tree recursively.

For each query sentence, we build a triple bucket. Initially the bucket is empty and in the end it must have all the essential extractable triples from the sentence. In our method, we treat the triple including root as the main triple and derive other triples as dependent on it through its elements as subjects or objects.

So far, we have covered the basic and intermediate grammar structures; there is still a great deal of sentence structure though. Verbs are the central elements in the decision process.

[6] http://nlp.stanford.edu/software/corenlp.shtml

Dependency graph received from the SCNLP for the question "Which book is written by Orhan Pamuk" is given below.

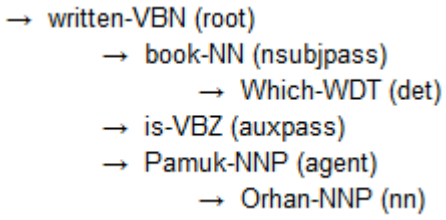

Figure 1 Dependency graph

This is the structure we receive from the library. From this structure, through the POS tags of the words and relations between them, we define triples. The above example results with two triples:

[Subject: ?x] [Predicate: rdf:type] [Object: book]
[Subject: ?x] [Predicate: written] [Object: Orhan Pamuk]

Since the entrance point is 'written', the first triple candidate is between 'written' and its children 'book', 'is', and 'Pamuk'. The verbs are the most likely predicates of triples. As we can find out which child the questioned element is by traversing the tree and locating the question word, we extract our first triple from the sentence: [Subject: ?x] [Predicate: written] [Object: Orhan Pamuk]. Then each child will be evaluated recursively.

Next step is going through other words treating them as roots. The only other triple will be derived from 'book' and its child 'which'. Since 'which' is the questioned element and it determines the word 'book', we resolve the questioned element as a book, thus: [Subject: ?x] [Predicate: rdf:type] [Object: book].

Although there are different cases for extracting triples, the above example shows the basic structure.

2.2 Entity and property extraction

Let $T=\{t_1 ,.., t_n\}$ be the set of triple patterns produced for a question in the previous triple pattern extraction step. For each triple pattern t_i in T we take the subject and the object of the triple and find the matching DBpedia entities if they exist. Then we find a set of possible properties found in DBpedia for each predicate t_i. Let this property set be $Pt_i=\{pi_1 \dots p(i_m)\}$. Therefore, we have cardinality of Pti possible triples for each ti. By using all members of T, we can build 'n' piece of triples. By using all possibilities we can build $\prod$ (cardinality of Pt_n) (from 1 to n). For example if T has three members each has 2, 5 and 3 possible predicates consecutively. Then there will be 30 possible triple query list. Let $Q=\{q_1 \dots q_{\text{product of P(t1 to tn)}}\}$ be the set of candidate queries constructed using T.

Our goal is to find all possible DBpedia entities, classes, object and data properties for each t_i in T using string similarity and relational patterns. First, all subject and objects are mapped to DBpedia entities or classes. Then, all predicates are mapped to DBpedia object or data properties.

2.2.1 Find object properties

Predicates with POS tag value that are verbs are searched among DBpedia object properties using a string similarity score.

For example, predicate "written" is searched for properties in DBpedia property list that have the greatest common

subsequence. The score is calculated by the length of greatest common subsequence over the length of the word. For instance, the property 'taxiDriver' encapsulates the word 'river'. With this scoring scheme, we eliminate these kinds of miscalculations. The object property dbont:writer[7] is the most similar object property for the predicate "written".

We constructed a list of all possible pairs of object properties from DBpedia with similar meanings. For each item we have calculated the similarity score by using Lin and Wu & Palmer metrics in WordNet::Similarity [14]. If the metrics are higher than the assigned threshold (0.75 for Lin, 0.85 for Wu & Palmer) then both properties are regarded as properties with similar meanings. By doing so, we get a list of object properties with their similar meanings. For the sample question, **dbont:writer** has similar meaning with **dbont:author**; the full list can be obtained from the project homepage.

For each t_n in T object properties are added to the list of possible predicates for t_n as Pt_1("written") = {dbont:writer, dbont:author}. The predicate rdf:type is already a DBpedia object property, which is found in Triple Extraction step.

2.2.2 Find data properties

Predicates with POS tag value noun or adjective is searched for candidate DBpedia data properties using score of string similarity.

For the question "What is the height of Michael Jordan?" the resulting triple will be:
[Subject: Michael Jordan] [Predicate: height] [Object: ?x].

Using common subsequence of the predicate, "height" is mapped to **dbont:height**.

We constructed a list of adjectives for all data properties defined by DBpedia ontology using API WordNet Searching (JAWS)[8]. By doing so, we get a list of data properties with their adjective meanings.

For the question "How tall is Michael Jordan?" the resulting triple will be :
[Subject: Michael Jordan] [Predicate: tall] [Object: ?x].
Using adjective list the predicate "tall" is mapped to **dbont:height**.

For each t_n in T, object properties are added to the list of possible predicates for tn as Pt_1("height") = {dbont:height} or Pt1("tall") = {dbont:height}.

2.2.3 Find properties using relational patterns

Question answering systems need patterns that denote relations between entities. There would be cases where the intended questions can be phrased in numerous ways. For instance, a question about the birth place of Michael Jackson can be built as "Where was Michael Jackson born in?" or "Where was Michael Jackson born at?". However, in order to query DBpedia we will need the object property of "birthPlace". For this reason, we have to construct a "birthPlace" relationship with "Michael Jackson" and Gary, Indiana, where he was born in, for both questions. Here, "born in" and "born at" are text phrases and we need a tool that maps such phrases into the same object property.

Nakashole et al. [6] proposed a method that constructs a large resource for textual patterns that denotes binary relations between entities, organizing patterns into a set of synonymous patterns similar in spirit to WordNet [7]. The patterns are semantically typed and organized into a subsumption taxonomy called PATTY. Extracting textual patterns from a corpus follows two steps:

Firstly, relational patterns from a corpus are compiled and then a semantically typed structure is imposed on them.

After extracting the textual patterns from a corpus, this work arranges the patterns in a semantic taxonomy. A prefix-tree for frequent patterns is used to determine inclusion, mutual inclusion, or independence. The prefix-tree stores support sets of patterns. An algorithm is developed for obtaining set intersections from prefix-tree. Although this work effectively extracts patterns from a corpus and arranges these patterns in a semantic taxonomy, it has some drawbacks.

Firstly, the PATTY extraction and mining algorithms were run on two different input corpora: the New York Times archive and the English edition of Wikipedia. PATTY taxonomy is restricted to a number of relational patterns because these corpora include a certain number of textual patterns that denote a relation between entity pairs. It does not make use of a universal corpus that can include nearly all available facts about entities or textual phrases combining entity pairs.

Next, PATTY includes some noisy data that is not related with the actual pattern; conversely, it harnesses patterns that have an opposite meaning against the actual pattern. For instance, the "deathPlace" relation name includes a textual pattern "born in" which is opposed to "died at" and has a relatively strong confidence. The cause of this PATTY's behavior is related to the noisy data existing in corpora.

In this work, we used relational patterns extracted by PATTY.

Considering the question "Where did Abraham Lincoln die?"
the resulting triple is :
[Subject: Abraham Lincoln] [Predicate: die] [Object: ?x] .
Using relational patterns the predicate "die" is mapped to DBpedia properties like **dbont:deathPlace**, **dbont:birthPlace**, **dbont:residence**.

The word "die" may occur in many forms in pattern texts. We count all occurrences of the word and assign it as a frequency value to the relative property. Thus, **dbont:birthPlace, dbont:residence, dbont:deathPlace** contain the word "die". In this case, we rank properties based on frequency values. Frequency of the predicate "die" is higher for **dbont:deathPlace** than **dbont:birthPlace** and **dbont:residence**. Frequency of a pattern determines the ranking score of the predicate. We use this ranking score in Answer E step to assign priorities between queries.

For each t_n in T object properties are added to the list of possible predicates for tn as Pt_1("die") = {dbont:deathPlace, dbont:birthPlace, dbont:residence}.

[7] dbont: http://dbpedia.org/ontology/

[8] http://lyle.smu.edu/~tspell/jaws/

2.2.4 *Find entity classes*

For each t in T if t_i contains **rdf:type**[9] predicate then the object of t_n is regarded as DBpedia entity class. For the question "Which book is written by Orhan Pamuk?" the word "book" is mapped to **dbont:Book** using label properties of all entity classes.

2.2.5 *Find named entities and disambiguate*

For each t in T if t_i contains named entity it is disambiguated using method described in [15]. The disambiguation method is based on page links between all spotted named entities. Additionally, we assign score of string similarity between spotted entities and named entity, which needs to be disambiguated.

For the question "Which book is written by Orhan Pamuk?" the named entity "Orhan Pamuk" in **[Subject: ?x] [Predicate: written] [Object: Orhan Pamuk]** is disambiguated to **res:Orhan_Pamuk**[10].

Finally, for each t in T we have a list of candidate properties denoted with P. For each p in P of any t we add candidate query q to Q with ranking score of property based on ranking method explained in 2.3.1.

2.3 Answer Extraction

In this step, we construct all possible queries using object and data properties extracted from step 2.2 and execute on knowledge base.

For the question "Which book is written by Orhan Pamuk?" the extracted triples are given below.

[Subject: ?x] [Predicate: rdf:type] [Object: book]
[Subject: ?x] [Predicate: written] [Object: ?Orhan Pamuk]

predicate "written" is mapped to **dbont:writer** and **dbont:author**,
object "book" is mapped to **dbont:Book** class,
subject "Orhan Pamuk" is mapped to **res:Orhan_Pamuk**.

So, candidate queries for our sample question are given below.

Query1:

```
SELECT ?x WHERE {
        ?x   rdf:type        dbont:Book.
        ?x   dbont:writer    res:Orhan_Pamuk.
}
```

Query2:

```
SELECT ?x WHERE {
        ?x   rdf:type         dbont:Book.
        ?x   dbont:author     res:Orhan_Pamuk.
}
```

2.3.1 *Ranking of Answers*

We run all queries against the knowledge base and we assign ranking score for each triple in a query. The ranking score of a query is calculated by multiplying frequencies of all used triples' predicates. Extracted answers are ranked based on ranking scores.

2.3.2 *Expected Type Checking*

Depending on questions, expected answer type is determined for each question type. The list is given below.

Table 1 Expected answer types for questions

Question Type	**Expected answer type**
Who	Person, Organization, Company
Where	Place
When	Date
How many	Numeric

'Which' questions are usually followed by a noun that is extracted by using step in 2.1and there is no need to check answer type.

Finally, we execute all constructed queries. Candidate answers matching expected type of a question are ranked based on ranking score. The candidate with the highest-ranking score is given as an answer.

3. EVALUATION

We evaluated our system based on DBpedia questions that are used in QALD-2[11] workshop open challenge (Question answering over linked data workshop as part of ESWC 2012 Extended Semantic Web Conference). The test set contains 100 questions.

We excluded some of the questions that contain YAGO classes, YAGO entities and DBpedia RDF properties. We selected questions where query relies only on properties from DBpedia ontology without any links to other knowledge bases such as YAGO. The remaining set contains 55 questions. Results for each question is available at project homepage[12].

Table 2 Precision, Recall and F1 values

	Precision	Recall	F1
Our method	83 %	32 %	46 %

As results in Table 2 suggest, our tool performs well for finding correct answers to attempted questions with precision 83%. It indicates that the tool provides the correct answer most of the time if the question type is implemented. However, our tool can process only 32% of questions (18 questions out of 55) and provides an answer with 83% precision (15 correct answers out of 18) yet due to incomprehensiveness of Triple and Entity and property extraction methods.

4. RELATED WORK

Yahya et al. [5] present a method to answer questions by segmenting questions into phrases, mapping phrases to semantic

[9] http://www.w3.org/1999/02/22-rdf-syntax-ns#type

[10] http://dbpedia.org/resource/Orhan_Pamuk

[11] http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=challenge&q=2

[12] http://wis.etu.edu.tr/qas

entities, classes and relations, and disambiguating all semantic entities jointly by using integer linear programming. Unlike them, we solve two problems separately. First, we get candidate triples then disambiguate the entities and find DBPedia properties. Our methods differ especially in using external data such as relational patterns.

Unger et al. [4] present a system that uses templates to be filled by using parse trees of questions. After, each entity in queries is instantiated to knowledge base entity using statistical entity identification and predicate detection. In our method, however, we do not have any template queries. For each question we build a query bucket to question DBPedia.

Ferrucci et al. [1] demonstrated a new kind of deep question answering system called Watson. A key element in Ferrucci et al.'s approach is to decompose questions into several cues and sub-cues, with the aim to get answers for variations of cues.

Pythia et al. [9] present a system that relies on deep linguistic analysis of questions. They constructed manually vocabulary of meaning representations of a given ontology. The system can process linguistically complex questions using the constructed lexicon. We, on the other hand, constructed our own vocabulary (relation patterns) too, but extracted from large corpora using all relations in DBPedia.

FREyA [10] relies on user's feedback for the question to disambiguate the entity and select the most appropriate one. Their method is in need of supervision to train the system. In our system, the solution is fully automated and unsupervised.

PowerAqua [11] is a question answering system over based on data combined from different and distributed sources. The possible drawback of using different sources may result in incomplete, low quality or noisy data. Using heterogeneous data is important in the future of Semantic Web and Question Answering Systems. However, we only use DBPedia in our implementation to benefit from homogeneity of the data. In future projects, being able to use heterogeneous ontologies jointly is going to be crucial.

5. DISCUSSION

Our tool lacks the ability to map all questions to triples using method described in 2.1. For some questions, triple extraction step provides candidate triples for a question but finding appropriate object property by only using the list of DBpedia properties and relational patterns is not enough. For instance, the question "**Is Frank Herbert still alive?**" is mapped to **[Subject: Frank Herbert] [Predicate: is] [Object: alive]** from triple extraction step. The triple should be mapped to **[Subject: Frank Herbert] [Predicate: deathDate] [Object: ?x]**. Neither relational patterns contain the word "alive" nor the list of DBpedia properties. Thus, the new methods should be implemented to overcome this kind of issues.

Translating questions to candidate triples using triple extraction in 2.1 should be adapted to respond to more comprehensive questions. Even though our tool is not complete yet, the method based on dependency graph is promising. The dependency graph is giving all the necessary information for a text to build up the triples as long as the implementation covers all the subtree patterns.

Although the method described in [6] effectively extracts patterns from a corpus and arranges these patterns in a semantic taxonomy, it has some drawbacks.

Firstly, the PATTY extraction and mining algorithms were run on two different input corpora: the New York Times archive and the English edition of Wikipedia. PATTY taxonomy is restricted to a number of relational patterns because these corpora include a certain number of textual patterns that denote a relation between entity pairs. It does not make use of a universal corpus that can include nearly all available facts about entities or textual phrases combining entity pairs.

Next, PATTY includes some noisy data that is not related with the actual pattern; conversely, it harnesses patterns that have an opposite meaning against the actual pattern. For instance, the "deathPlace" relation name includes a textual pattern "born in" which is opposed to "died at" and has a relatively strong confidence. The cause of this PATTY's behavior is related to the noisy data existing in corpora.

Extracted relational patterns in [6] consist of only object properties. There is a research gap for extracting relational pattern for data properties. Additionally, relational patterns should be extracted from large corpora to capture all possible expressions used in natural language.

6. CONCLUSION

We presented a tool for translating natural language questions to SPARQL queries. Our tool is based on extracting candidate triples using dependency graph and part of the speech tags of a question. Then, disambiguating and replacing triples with DBpedia property and entities using subject, predicate and object values in candidate queries. Finally, based on ranking score of queries and the expected answer type of the questions, the answer is presented to the user. Evaluation on the selected questions showed promising results and a room for improvement.

As for the future work, triple extraction method should be improved to handle a broad range of questions. Also, relational patterns for object and data properties can be extracted from large corpora, the web.

7. REFERENCES

[1] Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., ... & Welty, C. 2010. Building Watson: An overview of the DeepQA project. *AI Magazine*, 59–79.

[2] Toutanova, K., & Manning, C. D. 2000. Enriching the Knowledge Sources Used in a Maximum Entropy. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*, 63–70.

[3] Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. *Proceedings of HLT-NAACL* 2003, 252–259.

[4] Unger, C., Bühmann, L., Lehmann, J., Ngomo, A.-C. N., Gerber, D., & Cimiano, P. 2012. Template-based question answering over RDF data. *Proceedings of the 21st international conference on World Wide Web - WWW 2012 -*

Ontology Representation and Querying: RDF and SPARQL, 639–648.

[5] Yahya, M., Berberich, K., Elbassuoni, S., Ramanath, M., Tresp, V., & Weikum, G. 2012. Natural Language Questions for the Web of Data. *Empirical Methods in Natural Language Processing and Natural Language Learning (EMNLP 2012).*

[6] Nakashole, N., Weikum, G., & Suchanek, F. 2012. PATTY: A Taxonomy of Relational Patterns with Semantic Types. *Empirical Methods in Natural Language Processing and Natural Language Learning (EMNLP 2012).*

[7] M. M. Stark and R. F. Riesenfeld. Wordnet: An electronic lexical database. *In Proceedings of 11th Eurographics Workshop on Rendering. MIT Press*, 1998.

[8] M. De Marneffe, B. MacCartney, and C. Manning. 2006. Generating typed dependency parses from phrase structure parses. *Proceedings of LREC*, 6:449–454, 2006.

[9] C. Unger and P. Cimiano. 2011. Pythia: Compositional meaning construction for ontology-based question answering on the Semantic Web. *In Proceedings of the 16th International Conference on Applications of Natural Language to Information Systems (NLDB 2011).*

[10] Damljanovic, D., Agatonovic, M., & Cunningham, H. 2011. FREyA: An interactive way of querying Linked Data using natural language. *Proceedings of the 1st Workshop on Question Answering over Linked Data (QALD-1), ESWC 2011.*

[11] Lopez, V., Fernandez, M., Stieler, N., & Motta, E. 2011. PowerAqua : supporting users in querying and exploring the Semantic Web content. *Semantic Web Journal.*

[12] Bizer, C., Heath, T., Berners-Lee, T. 2009. Linked data-the story so far. Int. *Journal on Semantic Web and Information Systems, Special Issue on Linked Data*, 4(2), 1–22, 2009.

[13] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C.,Cyganiak, R., Hellmann, S. 2009. DBpedia - A crystallization point for the Web of Data. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3), 154-165, 2009.

[14] Pedersen, T., Patwardhan, S., & Michelizzi, J. 2004. WordNet:: Similarity: measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL 2004* (pp. 38-41). Association for Computational Linguistics.

[15] Hakimov, S., Oto, S. A., & Dogdu, E. 2012. Named Entity Recognition and Disambiguation using Linked Data and Graph-based Centrality Scoring. In *Proceedings of the 4th International Workshop on Semantic Web Information Management, SIGMOD 2012.*