Generating Quiz Questions from Knowledge Graphs

Dominic Seyler   Mohamed Yahya   Klaus Berberich
Max Planck Institute for Informatics
Saarbrücken, Germany
{dseyler, myahya, kberberi}@mpi-inf.mpg.de

ABSTRACT

We propose an approach to generate natural language questions from knowledge graphs such as DBpedia and YAGO. We stage this in the setting of a quiz game. Our approach, though, is general enough to be applicable in other settings. Given a topic of interest (e.g., Soccer) and a difficulty (e.g., *hard*), our approach selects a query answer, generates a SPARQL query having the answer as its sole result, before verbalizing the question.

Categories and Subject Descriptors
H.4.m [**Information Systems**]: Miscellaneous

Keywords
Knowledge Graphs, Natural Language Questions

1. INTRODUCTION

Knowledge graphs such as DBpedia, Freebase, and YAGO provide world knowledge as structured (RDF) data. Automatically answering natural language questions based on these resources has attracted ample attention, both from industry [1] and academia [7], in recent years.

In this work, we address the reverse problem of generating natural language questions from knowledge graphs. We stage this in the setting of a quiz game. Our objective is thus to come up with a natural language question belonging to a specific topic (e.g., Entertainment) and having a specific difficulty (e.g., *easy* or *hard*). However, we foresee that natural language question generation has more serious applications, for instance, in professional education settings such as training employees based on structured data about products, customers, or the company itself. The approach that we develop is general enough to be useful in such settings.

Challenges that we address along the way include estimating question difficulty and coming up with a question which has a unique answer in the knowledge graph. Our approach uses a SPARQL query as an intermediate representation, estimates question difficulty based on statistics from the knowledge graph as well as Wikipedia, and finally verbalizes the question using lexical resources.

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2742722.

2. RELATED WORK

We now put our work in context with existing research. Some of the relevant aspects have been looked at by the NLP community. Liu et al. [4] investigate how question difficulty can be estimated in the context of community question answering services. Sakaguchi et al. [6] focus on the problem of removing words from text to generate fill-in-the-blanks questions in language learning. Research in the DB and Semantic Web communities has investigated how structured queries formulated in SQL [3] or SPARQL [5] can be paraphrased in natural language.

3. QUESTION GENERATION

We next describe how we generate a natural language question given a topic of interest (e.g., Soccer) and a difficulty level. Our approach proceeds in three steps, moving from the question answer to the natural language question itself. First, a named entity `t` is selected as the answer of the question. Second, as an intermediate representation, a SPARQL query is generated, which has the target entity `t` as its sole result and consists of triple patterns `?t p o` and `s p ?t`. Third, the intermediate SPARQL query is verbalized, yielding a natural language question, which can then be posed to the user. Before providing details on each of these three steps, we describe how we estimate question difficulty.

3.1 Question Difficulty

We rely on statistics from the knowledge graph and Wikipedia to estimate the difficulty of natural language questions. We identified the following three factors:

Popularity of the target entity $p(\mathtt{t})$ measured as the fraction of links in Wikipedia which point to the target entity's article. This captures the intuition that questions about popular entities tend to be easier. Thus, a question about `Ronaldo` is arguably easier than one about `Stefan Kuntz`.

Selectivity of triple patterns in the intermediate SPARQL query. For a triple pattern `s p o` its selectivity $s(\mathtt{s\ p\ o})$ is measured as the reciprocal number of answer triples in the knowledge graph. For instance, the triple pattern `?t playsFor A.C._Milan` is more selective than `?t bornIn Rio_de_Janeiro`. Here, the intuition is that more selective triple patterns give more useful cues to the user.

Coherence of triples in the intermediate SPARQL query. For a triple `s p o` its coherence $c(\mathtt{s\ p\ o})$ is measured as the Jaccard coefficient of the sets of Wikipedia articles pointing to the entities `s` and `o`, respectively. Intuitively, the triple `David_Beckham playsFor Real_Madrid` is more coherent than `David_Beckham bornIn London` and thus provides a more useful cue.

Putting things together, we measure the difficulty of a question for target entity `t` via triple patterns $\mathtt{s_i}\ \mathtt{p_i}\ \mathtt{o_i}$ as

$$p(\mathtt{t}) + \frac{1}{n}\sum_{i=1}^{n} s(\mathtt{s_i}\ \mathtt{p_i}\ \mathtt{o_i}) + \frac{1}{n}\sum_{i=1}^{n} c(\mathtt{s_i}\ \mathtt{p_i}\ \mathtt{o_i})\ .$$

3.2 Answer Selection

Given a topic of interest, as a first step, we need to select a target entity `t` as the question's answer. To this end, we obtain the grouping of Wikipedia categories into high-level topics (e.g., Arts and Sports) from the Wikipedia Portal. For a given topic (e.g., Sports) we then identify all entities within any of the relevant Wikipedia categories (e.g., `Premier_League_Players`) and select our target entity `t` such that its popularity $p(\mathtt{t})$ fits the given difficulty level.

3.3 Query Generation

Having selected the target entity `t`, we next generate our intermediate SPARQL query. First, as building blocks, we retrieve all triples from the knowledge graph with the target entity as subject or object by issuing the queries `t ?p ?o` and `?s ?p t`. Some of these triples need to be filtered out, because they would spoil the query answer immediately. As a concrete example, consider the triple `David_Beckham marriedTo Victoria_Beckham` when asking for `David_Beckham`. Therefore, we filter out all triples which have non-stopword tokens in common with the target entity. As a sanity check, we turn the resulting set of triples into a SPARQL query by making `t` a variable `?t` – if this query returns more than one result, there is no hope of generating a question for this target entity and we have to start afresh. Otherwise, we next need to select a subset of the triples, so that the corresponding SPARQL query has `t` as its sole result. This is implemented using random search informed by our notion of query difficulty. More precisely, we randomly select a triple to be added/removed from the subset, taking into account its selectivity and coherence relative to the difficulty of the current query. Moreover, the search is constrained so that always at least one type triple (e.g., `David_Beckham type Premier_League_Players`) is selected. When we see a query which meets the desired difficulty level, we issue it and check whether it yields `t` as its sole result.

3.4 Question Verbalization

As a final step we need to verbalize the intermediate SPARQL query to turn it into a natural language question. By construction, as said above, our intermediate SPARQL query contains at least one type triple. We verbalize it following the simple pattern

$$\textit{This}\ \mathtt{type_1},\ldots,\ \textit{and}\ \mathtt{type_m}\ \mathtt{p_1}\ \mathtt{o_1},\ldots,\ \textit{and}\ \mathtt{p_n}\ \mathtt{o_n}\ .$$

Here, $\mathtt{type_i}$ are obtained as the objects of the type triples and $\mathtt{p_i}\ \mathtt{o_i}$ are the predicates and object entities of the remaining triples. For verbalization, types names are turned into singular and have their underscores removed. For object entities we use their canonical surface form as captured in the knowledge graph (e.g., *David Beckham*). To verbalize predicates, we manually constructed a dictionary containing two paraphrases for each predicate, one for the case when the target entity is the subject and another for when it is the object. Thus, the `playsFor` predicate can be verbalized as *plays for* or *has player* depending on whether our target entity is `David_Beckham` or `Real_Madrid`, respectively.

4. ANECDOTAL EXAMPLES

We now provide anecdotal example questions generated by our approach on the YAGO [2] knowledge graph. We consider the topics Soccer and Entertainment and fix the target entity as `Ronaldo` and `Elvis_Presley`, respectively. These are among the most popular entities within our topics. We generate an easy and a hard question for each entity.

```
Ronaldo type Brazilian_footballers .
Ronaldo type 2002_FIFA_World_Cup_players .
Ronaldo playsFor FC_Barcelona .
Ronaldo bornIn Rio_de_Janeiro .
```

This Brazilian footballer and 2002 FIFA World Cup player plays for FC Barcelona and was born in Rio de Janeiro. (**Easy**)

```
Ronaldo type Naturalised_citizens_of_Spain .
Ronaldo type Olympic_bronze_medalists_for_Brazil .
Ronaldo playsFor São_Cristóvão_de_Futebol_e_Regatas .
Ronaldo hasWonPrize Laureus_World_Sports_Awards .
```

This Olympic bronze medalist for Brazil and naturalised citizen of Spain plays for São Cristóvão de Futebol e Regatas and has won the Laureus World Sports Awards. (**Hard**)

```
Elvis_Presley type American_rock_singers .
Elvis_Presley diedIn Memphis,_Tennessee .
Elvis_Presley created Jailhouse_Rock .
Elvis_Presley created Heartbreak_Hotel .
```

This american rock singer died in Memphis, Tennessee and created Jailhouse Rock and Heartbreak Hotel. (**Easy**)

```
Elvis_Presley type entertainer .
Elvis_Presley type musician .
Elvis_Presley created It's_Easy_for_you .
Elvis_Presley created I_Got_Lucky .
```

This entertainer and musician created It's Easy for You and I Got Lucky. (**Hard**)

As can be seen from the examples, our approach reliably generates easy and hard questions. For instance, for `Ronaldo`, it selects the better known soccer club `FC_Barcelona` for the easy question – the corresponding triple has high selectivity and coherence. Likewise, for `Elvis_Presley`, the hard question includes less well-known songs.

5. SUMMARY & OUTLOOK

We put forward an initial approach to generate natural language questions from knowledge graphs. The approach is implemented in an end-to-end system available at:

`https://gate.d5.mpi-inf.mpg.de/q2g/`

6. REFERENCES

[1] D. A. Ferrucci. Introduction to "This is Watson". *IBM Journal of Research and Development*, 2012.

[2] J. Hoffart et al.: YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *AI*, 2013.

[3] G. Koutrika et al. Explaining structured queries in natural language. ICDE 2010

[4] J. Liu et al. Question difficulty estimation in community question answering services. EMNLP 2013

[5] A. N. Ngomo et al. Sorry, i don't speak SPARQL: translating SPARQL queries into natural language. WWW 2013

[6] K. Sakaguchi et al. Discriminative approach to fill-in-the-blank quiz generation for language learners. ACL 2013

[7] M. Yahya et al. Robust Question Answering over the Web of Linked Data. CIKM 2013