Querying RDF Data Cubes through Natural Language
[Discussion Paper]

Maurizio Atzori[1], Giuseppe M. Mazzeo[2⋆], and Carlo Zaniolo[3]

[1] Math/CS Department
University of Cagliari, Italy
atzori@unica.it
[2] Facebook Inc.
Menlo Park, USA
mazzeo@cs.ucla.edu
[3] University of California
Los Angeles, USA
zaniolo@cs.ucla.edu

Abstract. In this discussion paper we present QA^3, a question answering (QA) system over RDF cubes. The system first tags chunks of text with elements of the knowledge base (KB), and then leverages the well-defined structure of data cubes to create the SPARQL query from the tags. For each class of questions with the same structure a SPARQL template is defined. The correct template is chosen by using a set of regex-like patterns, based on both syntactical and semantic features of the tokens extracted from the question. Preliminary results are encouraging and suggest a number of improvements. Over the 50 questions of the QALD-6 challenge, QA^3 can process 44 questions, with 0.59 precision and and 0.62 recall, remarkably improving the state of the art in natural language question answering over data cubes.

1 Introduction

Governments of several countries recently started to publish information about the public expenses using the RDF data model [2], in order to improve transparency. The need to publish statistical data, which concerns not only Governments but also many other organizations, has pushed the definition of a specific RDF-based model, namely, the RDF data cube model [4], whose current version was published in January 2014. The availability of data of public interest from different sources has thus favored the creation of projects, such as LinkedSpending [1], that collect statistical data from several organizations, making

⋆ This work has been done when Dr. Mazzeo was working at University of California, Los Angeles (USA)

SEBD 2018, June 24-27, 2018, Castellaneta Marina, Italy.

them available as an RDF KB, according to the Linked Data principles. However, while RDF data can be efficiently queried using the powerful SPARQL language, only technical users can benefit from their potential by extracting human-understandable information. The problem of providing a user-friendly interface that enables non-technical users to query RDF knowledge bases has been widely investigated during the last few years. Some of the existing approaches are the following. *Exploratory browsing* enables users to navigate the RDF graph by starting from an entity (node) and then moving to other entities by following properties (edges). Although users do not need to know beforehand the names of properties, this approach is effective only for exploring the graph in the proximity of the initial entity, and it is not suitable for aggregate queries. *Faceted search* supports top-down searches: starting with the whole dataset as potential results of the search, the user can progressively restrict the results by adding constraints on the properties of the current result set [10]. This approach was recently applied to the RDF data cubes [15], following the long tradition of graphical user interfaces for OLAP analysis, based on charts representing different kind of aggregations of the underlying data. Another user-friendly system for querying RDF data is SWiPE [6], which enables users to type the constraints directly in the fields of the infobox of WikiPedia pages. While this paradigm is very effective for finding list of entities with specific properties, its generalization to the RDF data cubes is not trivial. *Natural language interfaces* let the user type any question in natural language and translate it into a SPARQL query. The current state-of-the-art system is Xser [18], which was able to yield F-scores equal to 0.72 and 0.63 in the 2014 and 2015 QALD challenges [3], respectively. Although its accuracy is not very high, Xser largely won over the other participating systems. This witnesses the fact that translating natural language questions into SPARQL queries is a really hard task. The difficulty of this task can be reduced by using a *controlled natural language* (CNL), i.e., a language which is is generated by a restricted grammar [9,14,16]. While the accuracy of these systems is very high, some effort is required for following the language that can be recognized. None of the previously proposed systems, however, has been specialized for question answering over RDF data cubes.

Question answering over RDF cubes is, indeed, a brand new challenge, which raises issues different from those of question answering on a "general" KBs [11]. In fact, questions in this context are very specific, i.e., oriented towards extracting statistical information. For these questions a very accurate interpretation of specific features of the typical OLAP queries, such as different kind of aggregations or constraints on dimensions, is crucial, and misinterpretations can not be tolerated. In fact, while a partial interpretation of a general question might yield an acceptable answer, in an aggregation query misinterpreting a constraint is likely to result in a totally wrong answer.

In this discussion paper we present QA^3 (pronounced as *QA cube*), a question answering system for RDF data cubes [5]. An overview of QA^3 is presented in Section 2, and preliminary experimental results are reported in Section 3. We finally conclude the paper with Section 4.

2 An overview of QA^3

QA^3 translates questions posed in natural language into SPARQL queries, assuming that the answer to the questions can be provided using one of the RDF data cubes "known" by the system. Given a KB containing datasets stored using the RDF data cube model [4], QA^3 works in three steps:

- the question is tagged with elements of the KB, belonging to the same dataset. This step also detects which dataset is the most appropriate to answer the question;
- the question is tokenized, using Stanford parser [13], and the annotations are augmented using the tags obtained from the previous step. The sequence of tokens is then matched against some regex-like patterns, each associated with a SPARQL template;
- the chosen template is filled with the actual clauses (constraints, filters, etc.) by using the tags and the structure of dataset.

In the following each step is described in more detail.

2.1 Tagging questions with elements of the KB

Given a string Q representing a question, for each dataset D we try to match the literals in D against chunks of Q. Before performing the matching, the strings are normalized, by removing the stop words and performing lemmatization. The result of the matching between Q and the dataset D can be represented as set of pairs $\langle C, T \rangle$, where C is a chunk of Q and T is a set of triples in D. Each matching is associated with a quality measure, which is based on the percentage of Q that is covered by tagged chunks. Some other advanced ranking methods (such as [7]) could be used to further improvements in case of multiple candidates. We notice that related work [8] extracts connected subgraphs given keywords instead of triples, although we connect them later constraining them to a sparql template (see later). The quality measure (% of Q) enables the system to choose the dataset which most likely has to be used to provide an answer to the question.

2.2 Finding the SPARQL query template

The domain in which QA^3 operates is quite structured, especially compared to that of general question answering (e.g., DBpedia). As a consequence, the meaningful questions (i.e., questions that can be answered using the available KB) are likely to have a SPARQL translation which follow a limited set of well defined *templates*. For instance, if the user wants to know how much his city spent for public works in 2015, the question has to contain all the elements needed to detect the dataset to be used, the measure to aggregate and the aggregation function, and the constraints to be applied to restrict the computation on the observations in which the user is interested. This question, like a wide range of similar questions, can be answered by using the following SPARQL query template, based on the RDF cube dictionary [4]:

```
SELECT sum(xsd:decimal(?measure)) {
   ?observation qb:dataset <dataset> .
   ?observation <measure> ?measure .
   <constraints>
}
```

where `<dataset>` and `<measure>` have to be replaced with the correct URIs, and `<constraints>` has to be replaced with a set of triples specifying the constraints for the variable `?observation`, representing the observations.

In order to leverage the typical homogeneity of the structures of these questions, we implemented a system, working of a set of SPARQL templates, that automatically detects the template to be used to provide an answer. To this end, each template is associated with one (or possibly more) regular expressions built on the tokens of the questions. The tokens are obtained using the Stanford parser [13], which tokenizes the question and annotates each token with its lemma, its POS (part of speech) tag, and its NER (named entity recognition) tag. The annotations are augmented with the elements of the knowledge base (dataset, measure, dimension, attribute, literal) obtained at the previous step, and with a tag representing a possible aggregate function. For the latter, a specific (small) dictionary has been defined, using the words that are used most often to name aggregation functions (e.g., sum, total, average, maximum, etc.).

Thus, for each token we have the following features: (1) the POS tag, (2) the lemma, (3) the word (i.e., the original text), (4) the NER tag, (5) the KB tag (S: dataset, M: measure, D: dimension, A: attribute, E: entity, L: literal, O: none), and (6) the aggregation tag (S: sum, A: average, M: max, N: min, O: none).

The patterns used to match the tokens are defined as 7-tuples, where the i-th element represents the possible set of values for the i-th feature of the token (the features are assumed to follow the order in which they are listed above). For instance, the pattern `WP|WDT,_,_,_,!E` matches with tokens having `WP` or `WDT` as POS tag, any lemma, word, NER tag (`_`), and the token must not be annotated as entity (`E`). The features that are not explicitly specified in the pattern (in this case the aggregation tag), are allowed to take any value (`_` is implicitly assumed).

Patterns can be followed by a `#` symbol and by a string that represents a label name, that can be used to get the actual token/s that matched the pattern. These simple patterns can be used to build more complex regular expressions. We describe them by means of an example:

```
{_}* WP|WDT {_,_,_,_,O,O}* _,_,_,_,O,!O#1 {_,_,_,_,M|S#2}* {_}*
```

Each pattern of the expression above must match subsequent chunks of the

whole question. The interpretation of the patterns appearing in the expression is the following:

- any sequence of tokens ({_}*);
- a token with the POS tag WP or WDT;
- any sequence of tokens, whatever their POS tag, lemma, word, and NER are (_), without any specific KB annotation and without any specific aggregation function (O);
- a token with any POS tag, any lemma, any word, and any NER, without any specific KB annotation (O) and with a specific aggregation function (!O). This token is assigned the label *1* (#1);
- any sequence of tokens with any POS tag, any lemma, any word, and any NER, with a KB annotation that can be a measure (M) or a dataset (S). The type of tag for the aggregation function is not specified, which means it can be anything (in practice, it will be none). These tokens are assigned the label *2* (#2);
- any sequence of tokens ({_}*).

This expression can be matched against several questions, such as: "What is the total aid to the Anti Corruption Commission in the Maldives in 2015?". In general, that expression matches questions asking for the computation of an aggregation function, which is represented by the token with label *1* computed over a measure, which is represented by the token with label *2*. The measure could also be implicitly denoted by the name of the dataset (e.g., when the dataset is about a specific measure of a set of observations - *expenditures of town of Cary*). These questions can be translated into a SPARQL query with the following structure:

```
select <groupByVar>
   <aggregationFunction##1>(xsd:decimal(?measure)) {
      ?observation qb:dataSet <dataset> .
      ?observation <measure##2> ?measure .
      <constraints>
   } <groupByClause>
```

where <aggregationFunction##1> has to be replaced by the actual aggregation function, which can be derived using the token annotated with label *1*, <dataset> must be replaced with the URI of the dataset found in the previous step, <measure##2> must be replaced with the actual measure, using tokens labeled with *2*. Finally, <constraint>, <groupByVar> and <groupByClause> must be replaced with the actual variable/clause (possibly empty), that are derived using the KB tagging, as described in the following.

We remark that this strategy for deriving the SPARQL template is quite general and the definition of new templates is quite simple. Although capturing all the natural language questions is not possible through a finite set of patterns, we found that very few expressions are enough to cover most of the questions posed in a typical form (see Section 3).

2.3 Filling out contraints and group-by

The most interesting part is the construction of the constraints and the group-by clauses. In order to construct the constraints, we observe that (i) if a literal is tagged, then it must be connected to the observation through an attribute, which is also reported in the annotation, and (ii) if an entity is tagged, then it must be connected to the observation through a dimension. The dimension could be explicitly tagged in the question, but it can be also derived by maintaining an index that maps every entity e to the dimensions which can take e as value. The substring `<constraints>` of the template can be thus replaced with a string representing the triples obtained as described above. Regarding the group-by variable and clause, we observe that a question requiring their use has to contain an attribute or dimension which is not bound to a value (literal or entity, respectively). Therefore, we can try to find those tokens that are tagged with a dimension/attribute X to which no value is associated. We then replace `<groupByVar>` with a variable identifier, say `?gbvar`, and `?observation` is connected to `?gbvar` using the URI of X, and the `<groupByClause>` placeholder is replaced with `group by ?gbvar`. If no such attribute/dimension X can be found, both `<gbvar>` and `<groupby>` are replaced by the empty string.

3 Experimental results

QA^3 participated in the task 3 of QALD-6 challenge [3], where it was able to process 44 questions over the 50 available ones. The recall and precision over the 44 processed questions are 0.62 and 0.59 respectively, which correspond to F-score 0.6. The global F-score, assuming F-score 0 over the 6 unprocessed questions, is 0.53. In more details, over the 44 processed questions, QA^3 provides a correct answer (F-score 1) to 25 questions, a partial answer (F-score strictly between 0 and 1) to 3 questions, and a wrong answer (F-score 0) to 16 questions. The correct dataset can be found for 42 questions. For 30 of these questions the full set of correct annotations is found. Finally, for 25 of the questions with correct annotations QA^3 generates the SPARQL query that provides the correct results. A set of 6 pairs expression/template is currently being used, in order to detect the correct SPARQL template to be used for each question. These experimental results refer to our first version of QA^3, that we later improved and deeply discussed in [5] and made freely available.

A direct comparison against other systems has been independently performed by the QALD 6 Challenge [3], as reported in Fig. 1. The best performer in this comparison is a special version of the SPARKLIS system [9] tailored to statistical questions, a system that does not accept natural language questions. Instead, as explained in the next section it uses a faceted search approach, and its performance is, therefore, dependant on the level of expertise the user has (values for expert and beginner are shown). To the best of our knowledge, the

see the two modules `https://github.com/atzori/qa3tagger` (ad-hoc tagger) and `https://github.com/gmmazzeo/qa3` (template-based sparql generation)

System	Processed	Recall	Precision	F1 (processed)	F1 Global (overall)
SPARKLIS (used by an expert)	50	0.94	0.96	0.95	0.95
SPARKLIS (used by a beginner)	50	0.76	0.88	0.82	0.82
QA3 (initial tuning)	**44**	**0.62**	**0.59**	**0.60**	**0.53**
CubeQA	49	0.41	0.49	0.45	0.44

Fig. 1. Comparison against other QA systems, as reported by the QALD-6 independent competition

only two systems that answer free natural language questions over RDF cubes are CubeQA [12], described in the next section, and our QA^3. Compared to the state-of-the-art CubeQA, we get a remarkable improvement of $0.62/0.41 = 51\%$ in recall, 20% in precision. This is in part given by the ability of QA^3 to self-evaluate the confidence of a computed answer (thanks to the score measure of the tagger), and also by the good expressivity of the template/pattern system.

We also remark that F1 and F1 Global, i.e., the F1 measure computed over all questions (not only those for which the system provides an answer) are respectively 33% and 20% higher than those of CubeQA. These results show that QA^3 provides a sensible improvement over the state of the art.

In terms of running time, the most expensive part is performed by the search for the correct dataset. Our in-memory tagging algorithm takes about $100ms$ to annotate a question for a given dataset. Current version of QA^3 takes therefore $100ms*50 \approx 5s$ to check for the best candidate dataset and annotate the question with triples necessary to find and fill in the correct SPARQL query template (real times ranging from 2s to 7s). While reasonable, this exhaustive approach would not scale well in case of thousands of datasets. We consider current results very encouraging, and we plan to improve QA^3 by using 1) a heuristic-based approach for the dataset search, 2) word embeddings for semantic relatedness instead of lemma-based term matching, and 3) a machine-learning approach for SPARQL template generation.

4 Conclusions

In this discussion paper we have presented QA^3, a system that can answer natural language questions about statistical data by finding the appropriate Linked-Spending [1] dataset and computing a correct SPARQL query. It works on different kind of typical statistical questions. The system has been implemented, opensourced on GitHub and also freely available online at `http://qa3.link/`.

During the discussion, questions taken from the QALD-6 challenge [3] will be lively employed, but the attendees will be also welcome to suggest other questions. Future work will explore the use of machine learning to generate the regex-like patterns and related privacy issues [17] related with the use of statistical data.

Acknowledgments

This research was supported in part by a 2015 Google Faculty Research Award, NIH 1 U54GM 114833-01 (BD2K) and Sardegna Ricerche (*project OKgraph*, Capitale Umano Alta Qualificazione, CRP 120).

References

1. LinkedSpending project. `http://linkedspending.aksw.org/`.
2. Linking Open Data cloud diagram. `http://lod-cloud.net/`.
3. Question Answering over Linked Data (QALD). `http://qald.sebastianwalter.org/`.
4. The RDF Data Cube Vocabulary. `https://www.w3.org/TR/vocab-data-cube/`.
5. M. Atzori, G. M. Mazzeo, and C. Zaniolo. QA^3: a Natural Language Approach to Question Answering over RDF Data Cubes. *Semantic Web Journal*, 2018.
6. M. Atzori and C. Zaniolo. Swipe: searching wikipedia by example. In *Proc. of the 21st Int. World Wide Web Conf., WWW 2012 (Companion Volume)*, pages 309–312, 2012.
7. A. Dessi and M. Atzori. A machine-learning approach to ranking RDF properties. *Future Generation Computer Systems*, 54:366–377, 2016.
8. S. Elbassuoni and R. Blanco. Keyword search over rdf graphs. In *Proceedings of the 20th ACM CIKM '11*, pages 237–242. ACM, 2011.
9. S. Ferré. Sparklis: An expressive query builder for sparql endpoints with guidance in natural language. *Semantic Web*, 7(1):95–104, 2016.
10. R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted wikipedia search. In *Proc. of 13th Int. Conf. of Business Information Systems, BIS 2010*, pages 1–11, 2010.
11. K. Höffner and J. Lehmann. Towards question answering on statistical linked data. In *Proc. of the 10th Int. Conf. on Semantic Systems, SEMANTICS*, 2014.
12. K. Höffner, J. Lehmann, and R. Usbeck. Cubeqa - question answering on RDF data cubes. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, pages 325–340, 2016.
13. D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, 2003.
14. A. Marginean. Gfmed: Question answering over biomedical linked data with grammatical framework. In *Working Notes for CLEF 2014 Conference*, pages 1224–1235, 2014.
15. M. Martin, K. Abicht, C. Stadler, A. N. Ngomo, T. Soru, and S. Auer. Cubeviz: Exploration and visualization of statistical linked data. In *Proc. of the 24th Int. World Wide Web Conf., WWW 2015 (Companion Volume)*, pages 219–222, 2015.
16. G. M. Mazzeo and C. Zaniolo. Answering controlled natural language questions on RDF knowledge bases. In *Proc. of the 19th International Conference on Extending Database Technology, EDBT 2016*, pages 608–611, 2016.
17. D. Pedreschi, F. Bonchi, F. Turini, V. Verykios, M. Atzori, B. Malin, B. Moelans, and Y. Saygin. *Privacy protection: Regulations and technologies, opportunities and threats*. Springer, 2008.
18. K. Xu, Y. Feng, S. Huang, and D. Zhao. Question answering via phrasal semantic parsing. In *Proc. of 6th Int. Conf. of the CLEF Association, CLEF 2015*, pages 414–426, 2015.