Machine Learning of SPARQL Templates for Question Answering over LinkedSpending

Roberto Cocco
Department of Math/CS (DMI)
University of Cagliari
Cagliari, Italy
robertococco@outlook.com

Maurizio Atzori
Department of Math/CS (DMI)
University of Cagliari
Cagliari, Italy
atzori@unica.it

Carlo Zaniolo
Computer Science Department
University of California, Los Angeles
Los Angeles (CA), USA
zaniolo@cs.ucla.edu

***Abstract*—We present a Question Answering system aimed to answer natural language questions over the open RDF spending data provided by LinkedSpeding. We propose an original machine-learning approach to learn generalized SPARQL templates from an existing training set of *(NL question, SPARQL query)* pairs. In our approach, the generalized SPARQL templates are fed to an instance-based classifier that associates a given user-provided question to an existing pair that is used to answer the user question. We employ an external tagger, delegating the Named-Entity Recognition (NER) task to a service developed for the domain we want to query. The problem is particularly challenging due to the small training set size available, counting only 100 questions/SPARQL queries.**

We illustrate the results of our new approach using data provided by the Question Answering over Linked Data challenge (QALD-6) task 3, showing that we can provide a correct answer to 14 of the 50 questions of the test set. These results are then compared to existing systems, including our previous system, QA^3, where templates were provided by an expert rather than being generated automatically from a training set.

***Index Terms*—Question Answering, SPARQL, Semantic Web, Machine Learning**

I. Introduction

In the recent years, we have seen a steady growth of structured data being made available to the public, with governments contributing to this trend by publishing information about public expenses. At the same pace, the need has grown to make this data available to non-technical users. In this paper, we propose a tool capable of answering questions posed in natural language by users who have no expertise on RDF datasets and SPARQL queries. This work is a follow-up to our previous template-based question-answering system called QA^3, with the main difference being the way templates are obtained. In fact, while QA^3 requires the templates to be generated by an experienced user, in this paper we present a system that is able to learn new templates from datasets fed to an intelligent template generator. Our system works by processing a dataset containing *(question, SPARQL query)* pairs in input. Specific references contained in each question and its associated SPARQL query are automatically linked by our system, and treated as "variables" that may vary in different user questions. In other words, both questions and SPARQL queries are generalized into templates that can be filled in with different values w.r.t. the instances available in the dataset.

At query answering time, the user question is subject to a similar generalization process, whereby it is matched to a list of generalized templates containing the same tags and ordered by the Jaccard Index (also known as *intersection over union*). The best matching templates, and their associated queries, are then filled in with the data so obtained by tagging the user question. Due to the kind of approach used, more than one query per template is reconstructed in some cases. Then, we show the user the template question filled in with the highest Jaccard Index and ask the user if this has the same meaning as the original one. This allow our system to exploit user's feedback to further refine the initial training set. Furthermore, the system keeps presenting the user with questions until he/she returns a positive response or no more questions are left to show. This results in a very flexible system, requiring the user to input only non-technical data as long as an initial small training set of question/answer pairs.

II. Background

We introduce the reader to the basics concepts covered in this paper.

A. RDF

The Resource Description Framework[1] is a set of specifications used to represent data in a tabular notation. It provides us with a general method to decompose knowledge into triples, composed of a $subject$, a $predicate$ and an $object$, usually denoted by Uniform Resource Identifiers (URIs) with prefixes. For instance, triple `dbr:Capri rdf:type dbr:Island` states that *Capri* is an island.

B. Linked Spending

With more and more governments providing spending data to the public, Linked Spending[2] took on the task of making open spending data available via RDF datacubes, a W3C standard [4]. It now provides more than 2 million planned or carried out financial transactions.

[1]https://www.xml.com/pub/a/2001/01/24/rdf.html
[2]http://linkedspending.aksw.org/

C. Question answering

Question answering systems focus on correctly answering questions posed in natural language, instead of retrieving information on the basis of user-generated keywords as traditional search engines do [12] or SQL-like query languages like SPARQL. This problem can be approached in many different ways, but the main challenges can be summarized as:

- Extracting the relevant information from the questions and mapping it to the datacube,
- Dealing with term ambiguity,
- Working with more than one dataset,
- Data quality and heterogeneity.

In this paper, we leverage the award-winning results of QA[3], and focus on automating (through machine learning) the process of generating templates, which previously required a try-and-error approach by a human expert.

D. QALD

Question Answering over Linked Data[3] (QALD) is a series of evaluation campaigns that provides an up-to-date benchmark for assessing and comparing question answering systems. Quoting Unger et al. [12],

> "The key challenge for question answering over linked data is to translate the users' information needs into a form that enables their evaluation by standard Semantic Web query processing and inferencing techniques."

Over the years, QALD has generated a series of training/test sets. The relevant one for this work is Task 3 of QALD6, containing a training set of 100 instances of *(question, SPARQL query)* pairs for LinkedSpending, and another 50 pairs as test set.

E. QA[3] tagger

In this paper, we leverage the tagger developed for QA[3] and described in [2]. This tool takes a question as input, and returns (i) a structured answer containing the dataset that's most likely to contain the correct answer to the question, and (ii) the list of natural language expressions that found a match on said dataset with their associated links to the matching triple in LinkedSpending. Although this paper only shows the results obtained with the aid of the QA[3] tagger, our approach allows using any service that can annotate the text.

F. Template-based approaches

Template-based approaches to question answering work by constructing templates (or pseudo queries) from a linguistic analysis of the input question [12]. These templates are neutral, as they contain no reference to the dataset, and they stand in the middle between the natural language question and the query. Since these templates often reflect the linguistic structure of the questions, structural variations must be included, exponentially increasing the number of possible queries to build.

[3]http://qald.aksw.org/

III. Our Approach

The three main components that compose this template-based approach are the following:

- a *tagger*, that handles the Named-Entity Recognition task, and is provided by QA[3] [2];
- a *template generator*, that handles the induction of templates of both questions and SPARQL queries by processing a given training set of pairs;
- a *template matcher*, that first performs the ranking of templates for the question posed by the user, and then fills-in the best-matching template.

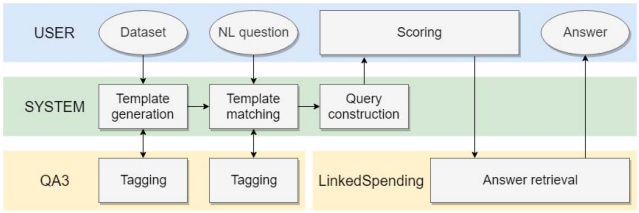

Fig. 1. Our System flow, including tools provided by QA^3 (lower layer) and inputs by the user (upper layer)

As shown in Figure 1, a dataset of *(NL question, SPARQL query)* pairs is fed to the *generator*, which, through the *tagger*, outputs a new dataset composed of a list of query templates paired with their respective questions' template. When asked a NL question, the system tags it while the *matcher* extract similar templates from the templates dataset. From these templates, initial queries and questions can be reconstructed if needed. The scoring of templates w.r.t. a given question is done by the system using semantic similarity, and the system presents the user with successive reconstructed questions until it receives a positive answer, or the list of questions is exhausted. Finally the query, paired with the question chosen by the user, is used to retrieve the answer from the LinkedSpending endpoint[4].

In the following we provide some additional information on each module of the system.

A. Tagger

The tagger gets the response from a NER service and makes it usable by the system. It sends the end-user question to a service specialized for the domain that is being questioned by the user. The service's response usually contains a reference to a datacube, the list of entities found on said datacube, along with the chunk from the question that matched the entity and, optionally, the entity's type.

B. Generator

The generator creates a template by processing a NL question, a SPARQL query, or both, and removing all the references to the input datacube. It initially gets the list of entities found on the input datacube and the datacube found by the *tagger*. If found in the query, the datacube reference is replaced with the tag (placeholder) `<DATASET>`. The prefixes

[4]http://linkedspending.aksw.org/sparql

are replaced by their expanded form, and aliases and the 'from' clause are removed. Next, the expressions and the supported types (numbers, years) found on both the question and the query are replaced with tags depending on the type of data they represent (e.g. <PROP> for properties, <VALUE> for values, <YEAR> for years, <NUM> for numbers ecc). Finally, the variables get streamlined by a simple naming convention (varA, varB, ...), and predicates for which no matching found in the datacube are replaced by blank nodes. During the filling of a template, the process is basically inverted. The generator still gets from the *tagger* a list of expressions, which in this case were derived from the English question asked by the end-user. Then the template's tags are replaced by the obtained results, based on their types.

C. Matcher

The matcher, given a question template, returns a list of compatible query templates. This is done by querying a dataset processed by the generator for the tags in the question template. This list, ordered by the Jaccard Index (calculated between the question template, and the question templates associated to the queries in the dataset), will only contain query templates with an equal or lesser number of tags.

IV. Running Example

In this section, we detail the various steps performed by our prototype and how they work, using a running example associated with the following natural language question: *Which class achieved the highest revenue for the Town of Cary?*.

A. Data Preprocessing

Data preproccessing is done by feeding a dataset containing *(NL question, SPARQL query)* pairs to the *generator*. The result is a dataset containing a list of query templates associated with a list of question templates and a list of tags. Take a look at Table I for an example of such a pair.

Pairs that produce the same query template will be joined into an individual entity that references both question templates. Figure 2 shows a snippet from the resulting json dataset.

```
[{
  "query_template": "select <AGGRa> ?varA { ?obs
    <SUB0> ?varA . ?obs qb:dataSet <DATASET> . }",
  "questions": [
    { "question_template":
      "Which <PROP0> earned the <AGGRa> for the
        <VALUE1>?" }
  ],
  "tags": [ "AGGR", "PROP" ]
}]
```

Fig. 2. Dataset's structure

B. Question analysis & data matching

These tasks are done by the system's tagger with the aid of a domain-specific NER service, in our case the one provided by the QA^3 tagger. The tagger has the following two goals: (i) finding the correct dataset and (ii) matching question terms with the dataset terminology (NER). Missing the correct dataset or failing to find the correct terms will lead to an incorrect answer, so the NER service's accuracy needs to be very high for our system to work properly.

QA^3 proposed an approach based on choosing "the dataset that better *covers* the question, that is, the one that minimizes the portion of the question not referring to elements in the dataset" [2]. The question-to-dataset terms matching is done by first creating an in-memory index of all literals (labels, comments, and values) for each dataset, and normalizing the textual elements that are keys in the indexes and the questions by removing the stop words.
The response from the NER service usually consists of a dataset and a list of entities, as seen in Figure 3.
We then feed the question and the entities found by the tagger to the *generator* to obtain a template (e.g. *"Which <PROP0> achieved the <AGGRa> revenue for the <VALUE1>?"*).

```
"dataset": "town_of_cary_revenues",
"entities": [
  { "chunk": "class",
    "subject": "<http://linkedspending.aksw.org/
      ontology/town_of_cary_revenues-Class>",
    "property": "<http://purl.org/dc/terms/
      identifier>",
    "value": "Class"},
  { "chunk": "Town of Cary",
    "subject": "<http://linkedspending.aksw.org/
      ontology/town_of_cary>",
    "property": "<http://purl.org/dc/terms/
      identifier>",
    "value": "town_of_cary"},
]
```

Fig. 3. Response's structure

C. Query Construction

Inverting the data pre-processing task, the query construction takes a template as input, and returns a query. In this process, the templates are filled-in with the data obtained by the tagger from the NL question. In those cases where the same type of tag appears more than once, or we're working with a template with less tags than the NL question, the system creates a query for each combination, in order to cover different structural variation (e.g., the question *"Which class achieved the highest revenue for the Town of Cary, North Carolina?"* processed as *"Which <PROP0> achieved the <AGGRa> revenue for the <VALUE1>, <VALUE2>?"*, will result in the queries shown in Table II).
This outputs a total of $n!$ queries, where n is the number of times the tag appears in the template, unless templates have

TABLE I
AN EXAMPLE OF QUESTION/SPARQL PAIR

Question	SPARQL query
"Which program earned the most for the Town of Cary?"	`select max(?x) { ?obs qb:dataSet ls:town_of_cary_revenues. ?obs lso:town_of_cary_revenues-Program ?program. }`
Question Template	**SPARQL query Template**
"Which <PROP0> earned the <AGGRa> for the <VALUE1>?"	`select <AGGRa>(?varA) { ?obs qb:dataSet <SUB1>. ?obs <SUB0> ?varA. }`

fewer tags than the NL question. In this second situation, $n!/m$ templates are returned with m denoting the difference between the number of NL question's tags and the number of template's tags.

D. Scoring

The scoring process takes place in two different steps. In the $matcher$, we sort the list by the Jaccard Index, for which we can also specify a threshold value. If none of the questions in the list has a value over the threshold, the system widens the search to include templates with fewer tags then the ones on the question. This is done in order to avoid matching unrelated templates, a situation that could occur when more entities are tagged than those that are actually needed. After the query construction, since both the $matcher$ and the $generator$ can return more than a query, our system ranks them by the Jaccard Index (higher scores first) and presents them one-by-one to the user, until the system gets a positive feedback (Figure 4), in which case the system will proceed to the next step. Otherwise it will terminate refusing to provide an answer.

```
Ask a question:
> Which class achieved the highest revenue for the
  Town of Cary?

Did you mean "Which class earned the most for the
Town of Cary?" [y, N]
> y

Answer:
['https://openspending.org/town_of_cary_revenues/
Class/1']
```

Fig. 4. User feedback exploitation (self-learning)

E. Answer retrieval & presentation

The reconstructed query associated to the question picked by the user is executed over the LinkedSpending datacube [8]. The answer obtained by running the SPARQL query computed from the SPARQL query template is run against the Linked-Spending endpoint. Unless the answer is empty, the results are returned to the user as the final answer to the processed question, as shown in Figure 4.

V. EXPERIMENTS

This section shows the results obtained on the QALD-6 challenge's datasets. We used a simple command line interface, i.e. the one shown in Figure 4, where the user poses a question to the system and is then presented with the questions found in the scoring process. Then, the system executes the query associated with the question chosen by the user. We assume that, in the scoring step, the user always picks the correct question from the proposed ones if available, or a wrong one if no correct questions are shown. This is the expected scenario where users are not acting as adversarial attackers. If each proposed question is associated with a query that returns an empty answer, we count the user question as not processed.

A. QALD-6 Dataset

We tested our system with the datasets of the QALD-6 challenge for statistical question answering over RDF datacubes. It consists in a training dataset of 100 questions, and a test dataset of 50. For each question, both datasets contain the correct answer and other metadata.

B. Results

The results are obtained by feeding the training dataset to the system generator, and then asking the questions from the training and test datasets respectively. The system scored a total of 14 answers on the test dataset, processing 30 out of 50 questions (precision 47% over processed questions). On the training dataset, it managed to process 59 out of 100 questions, and correctly answered to 41 (69% over processed questions). These results are reported in Table III.

Although precision and recall are lower than our award-winning QA^3 system, we believe these are very promising considering that no human expert contributed to this result. In fact, measures are reasonable when compared against existing approaches despite our approach being completely automatic. That is, the system will support new queries as long as at least one example is provided in the training set or the user provide a positive feedback. We believe this represents a significant improvement of the state-of-the-art in unsupervised QA on knowledge bases.

VI. RELATED WORK

All proposed solutions feature systems which translate "easy" operations specified by the user into a SPARQL query,

TABLE II
EXAMPLE OF A QUERY'S VARIATIONS

Query template	`select <AGGRa> (?varA) {?obs <SUB1> ?varA }`
Variation A	`select max (?varA) {?obs lso:town_of_cary ?varA }`
Variation B	`select max (?varA) {?obs lso:north_carolina ?varA }`

TABLE III
RESULTS ON TRAINING AND TEST DATASETS, OBTAINED BY FEEDING THE TRAINING DATASET TO THE SYSTEM

Dataset	N	Processed	Correctly answered
Training	100	59	41
Test	50	30	14

TABLE IV
COMPARISON AGAINST OTHER QA SYSTEMS, AS REPORTED BY THE QALD-6 INDEPENDENT COMPETITION

System	Processed	Recall	Precision	F-1
This approach	**30**	**0.41**	**0.47**	**0.44**
QA^3	44	0.62	0.59	0.60
CubeQA	49	0.41	0.49	0.45
SPARKLIS (expert user)	50	0.94	0.96	0.95
SPARKLIS (beginner user)	50	0.76	0.88	0.82

whose result represents the answer to the intention that the user expressed through those operations. Many approaches have been proposed in the literature, including the previously mentioned QA^3 system [2], and they are described next.

Exploratory browsing allows users to navigate through the triples of the RDF graph by starting from an entity (node) and then clicking on a property (edge), thus moving to another entity. Although users do not need to know beforehand the exact names of properties, this approach can only be used effectively for exploring the graph in the proximity of the initial entity. Furthermore, this approach is not suitable for aggregate queries, which are the real first-class citizens in the world of RDF data cubes.

Faceted search supports a top-down search of RDF graph, by starting with the whole dataset as potential results, and enabling the user to progressively restrict the results by defining some constraints on the properties of the current result set [6]. This approach was recently applied to the RDF data cubes [10], 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 [3], [14], which turns the infobox of WikiPedia pages into an active query interface, where the user can enter constraints using the value fields of the infobox, according to the *By-Example Structured Query* (BEStQ) paradigm. While this paradigm is very effective for finding list of entities with specific properties starting from a wikipage, its generalization to the RDF data cubes is far from trivial.

Natural language interfaces represent the most ambitious and challenging approach to question answering over RDF data. These systems 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 [13], which was able to yield an F-score equal to 0.72 and 0.63 in the, respectively, 2014 and 2015 QALD challenges [1]. Although its accuracy is far from being very high, Xser easily 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 the task can be reduced by using a *controlled natural language* (CNL), i.e., a language which is is generated by a restricted grammar and can be efficiently interpreted by machines, yet natural enough to be easily understood by non-technical users. Some of the CNL-based systems that participated in past QALD challenges are Squall2sparql [5] and GFmed [9]. While the accuracy of both systems is very high, the former suffers from usability issues, since the user needs to type non-natural phrases in the question, while the latter has scalability problems, since the number of rules of its grammar depends on the size of the knowledge base.

Authors in [11] present CANaLI, a system that overcomes issues, yet providing high accuracy. CANaLI relieves the usability issues derived from the use of a CNL by means of an interactive interface having no scalability issues, since it combines a restricted set of rules, which do not depend on the knowledge base, with the use of an index, queried at run-time to find the tokens that are consistent with the grammar of the CNL and the semantics of the knowledge base.

To the best of our knowledge, the only question answering systems based on full natural language specialized for question answering over RDF data cubes are CubeQA [7] and QA^3 [2], that we compared against in Section V. Our contribution w.r.t. these existing systems is the use of a machine-learning approach that make the need of a domain expert to construct SPARQL templates unnecessary, even when only small small training sets are available, as in the case of QALD.

VII. DISCUSSION ON OUR MACHINE LEARNING APPROACH

The most challenging issue we encountered in order to learn SPARQL templates was the size of the search space, that is, the set of all possible questions and corresponding SPARQL queries is huge. Therefore, an approach such as deep learning or standard machine learning would require a massive dataset, which is not available in many question answering applications and in particular for the Statistical Question Answering problem addressed in this paper. As described in the experiments, the QALD [1] workshop provided only 100 pairs *(question, sparql query)*, with very different kind of questions and structure in the SPARQL query. Another 50 pairs were later provided as test set. Most kind of questions are

unique, that is, there is no other pair from which you can learn how to answer that kind of question. To address this *small data* machine learning problem, we adopted two techniques: *semi-supervised learning* and *weak supervision*[5] (described next in this Section), producing templates for both questions and SPARQL queries. We then reduced the problem into that of instance-based classification, finding the most similar question, and therefore its associated template, and finally filling in SPARQL templates with actual values to query the endpoint and get the final answer.

A. Semi-Supervised Learning

This approach suggests the exploitation of structural assumptions to automatically leverage unlabeled data. We exploited some structure of the problem, in particular the fact that every SPARQL query needs to specify a dataset, and that `min`/`max`/`avg`/`count` are necessarily SPARQL aggregates associated to some specific text labels in the natural language question (e.g. "minimum", "smallest", etc. for `min`). Therefore, every pair in the training set containing at least one aggregate could be rewritten into other pairs by replacing it with other aggregates (both in the question and in the SPARQL query).

B. Weak Supervision

We generalize the pairs, replacing every occurrence of known entities with placeholders. This way we shift the problem into one with a smaller search space (higher abstraction level) at the cost of introducing lower-quality pairs. For instance, the first row of Table I shows a pair *(question, SPARQL query)* as reported in the original QALD training set. We then rewrite this into the pair appearing in the second row, where both the phrases "program" and "Town of Cary" have been replaced with placeholders (resp. `<PROP0>` and `<VALUE1>`), and the corresponding references replaced also in the SPARQL query (second column). These templates are clearly more general than the original instances, with a smaller space, but these generated templates may not have the precision guaranteed in the original training set.

VIII. Conclusions and Future Works

The aim of this paper is to show that it is possible to implement a general-purpose template-based approach to question answering that doesn't require training and guidance by the experienced hand of domain experts. Although the f-measure of our current prototype does not outperform existing systems, results are very promising considering that in this case the generalization step is done automatically from the (few) instances available. In the following we list some possible way to further improve our system.

User feedback. A way to receive user feedback on correct answers or (NL question, SPARQL query) pairs, allowing the system to expand its own dataset. This might prove difficult, as the system would have no way to know that the user is providing correct data (trustiness should be addressed).

[5]more info at https://hazyresearch.github.io/snorkel/blog/ws_blog_post.html

Better UI. The command line interface is functional, but might look counter intuitive to an unexperienced user. This can be improved by implementing a desktop interface or, even better, by building a web application that will avoid headaches for the end-user.

Answer presentation. Another way to improve the user experience would be to show the answer in a more comprehensible way (e.g. as natural language or a graph), instead of presenting an URI to the user.

Acknowledgments

Authors are grateful to Dr. Massimo Mazzeo and Prof. Barbara Pes for early discussions and insightful comments. This work was supported in part by RAS and Sardegna Ricerche (project *OKgraph* and project *EmILIE*) and MIUR (project *HOPE: High quality Open data Publishing and Enrichment*, PRIN 2017).

References

[1] Question Answering over Linked Data (QALD6). `http://qald.aksw.org/index.php?x=challenge&q=6`.

[2] M. Atzori, G. M. Mazzeo, and C. Zaniolo. QA^3: a natural language approach to question answering over rdf data cubes. *Semantic Web Journal*, 10(3), 2019.

[3] M. Atzori and C. Zaniolo. SWiPE: searching wikipedia by example. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pages 309–312, 2012.

[4] R. Cyganiak and D. Reynolds. The RDF Data Cube Vocabulary (W3C Recommendation). `https://www.w3.org/TR/vocab-data-cube/`, Jan. 2014.

[5] S. Ferré. SQUALL: the expressiveness of SPARQL 1.1 made available as a controlled natural language. *Data Knowl. Eng.*, 94:163–188, 2014.

[6] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted wikipedia search. In *Business Information Systems, 13th International Conference, BIS 2010, Berlin, Germany, May 3-5, 2010. Proceedings*, pages 1–11, 2010.

[7] 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.

[8] K. Höffner, M. Martin, and J. Lehmann. Linkedspending: Openspending becomes linked open data. *Semantic Web*, 7(1):95–104, 2016.

[9] A. Marginean. Gfmed: Question answering over biomedical linked data with grammatical framework. In *Working Notes for CLEF 2014 Conference, Sheffield, UK, September 15-18, 2014.*, pages 1224–1235, 2014.

[10] M. Martin, K. Abicht, C. Stadler, A. N. Ngomo, T. Soru, and S. Auer. Cubeviz: Exploration and visualization of statistical linked data. In *Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015, Florence, Italy, May 18-22, 2015 - Companion Volume*, pages 219–222, 2015.

[11] G. M. Mazzeo and C. Zaniolo. Answering controlled natural language questions on RDF knowledge bases. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 608–611, 2016.

[12] C. Unger, A. Freitas, and P. Cimiano. An introduction to question answering over linked data. In *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*, pages 100–140, 2014.

[13] K. Xu, Y. Feng, S. Huang, and D. Zhao. Question answering via phrasal semantic parsing. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction - 6th International Conference of the CLEF Association, CLEF 2015, Toulouse, France, September 8-11, 2015, Proceedings*, pages 414–426, 2015.

[14] C. Zaniolo, S. Gao, M. Atzori, M. Chen, and J. Gu. User-friendly temporal queries on historical knowledge bases. *Inf. Comput.*, 259(3):444–459, 2018.