QA^3: a Natural Language Approach to Question Answering over RDF Data Cubes

Editor(s): Stamatia Dasiopoulou, Pompeu Fabra University, Spain; Georgios Meditskos, Centre for Research and Technology Hellas, Greece; Leo Wanner, ICREA and Pompeu Fabra University, Spain; Stefanos Vrochidis, Centre for Research and Technology Hellas, Greece; Philipp Cimiano, Bielefeld University, Germany

Maurizio Atzori [a], Giuseppe M. Mazzeo [b] and Carlo Zaniolo [b]

[a] *Department of Maths and Computer Science, University of Cagliari, V. Ospedale 72, 09124 Cagliari, Italy*
E-mail: atzori@unica.it

[b] *Computer Science Department, UCLA, 3532D Boelter Hall Los Angeles, CA 90095-1596, CA, USA*
E-mail: mazzeo@cs.ucla.edu, zaniolo@cs.ucla.edu

Abstract. In this paper we present QA^3, a question answering (QA) system over RDF data cubes. The system first tags chunks of text with elements of the knowledge base, and then leverages the well-defined structure of data cubes to create a SPARQL query from the tags. For each class of questions with the same structure a SPARQL template is defined, to be filled in with SPARQL fragments obtained by the interpretation of the question. The correct template is chosen by using an original set of regex-like patterns, based on both syntactical and semantic features of the tokens extracted from the question. Preliminary results obtained using a limited set of templates are encouraging and suggest a number of improvements. QA^3 can currently provide a correct answer to 27 of the 50 questions of the test set of the task 3 of QALD-6 challenge, remarkably improving the state of the art in natural language question answering over data cubes.

Keywords: question answering, RDF data cube, statistical queries, free natural language

1. Introduction

Governments of several countries have recently begun to publish information about public expenses, using the RDF data model [4] 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, i.e., the RDF data cube model [5], whose first draft was proposed by W3C in 2012, and its current version was published in January 2014. The availability of data of public interest from different sources has fostered the creation of projects, such as LinkedSpending [1], that collect statistical data from several organizations, making them available as an RDF knowledge base, following the Linked Data principles. However, while RDF data can be efficiently queried using the powerful SPARQL language, only technical users can effectively extract 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. All the proposed solutions provide some systems which translate simple actions and statements entered by the user into a SPARQL query, whose result represents the answer to the intention that the user expressed through its operations. Among these systems we find those based on *exploratory browsing*, *faceted search* and *by-example structured query* are based on a tailored user interface, that require user training and provide limited expressivity statistical queries (for instance limited or no support for inner select, group-by or other aggregate operators). Seeking to overcome these problems, recent work has focused on *natural language interfaces* which let the user type any question in natural language and translate it into a SPARQL query. While NL QA can be very user friendly and potentially very expressive it poses many difficult research issues. In fact, many systems only accept a *controlled natural language* (CNL), that is a language which is generated by a restricted grammar and vocabulary, and can be efficiently interpreted by machines, more recent work focus on *free* natural language question answering over RDF data.

The current state-of-the-art system for (free) question answering over Wikipedia/DBpedia is Xser [24], which was able to yield an F-score equal to 0.72 and 0.63 in 2014 and 2015 QALD challenges [2] the respectively. Although its accuracy is far from being very high, Xser largely won over the other participating systems. This suggests that translating natural language questions into SPARQL queries is a really hard task.

In particular, the problem of creating NL QA interfaces for RDF data cubes is one that has received much recent attention and proven particularly challenging. Statistical question answering (SQA), meaning question answering over data cubes, is a very recent research area born thanks to the success of projects such as OpenSpending[1], that collects datasets on public financial information, and LinkedSpending [1,16], that focuses on representing OpenSpending data through the use of RDF and a W3C standardized vocabulary for datacubes [5]. The intersection between RDF data cubes and SQA is sometimes referred to as RDCQA [15]. In [15] the RDCQA benchmark of 100 questions and 50 datasets used for Task 3 (“Statistical question answering over RDF datacubes") of the 6th Question Answering on Linked Data (QALD-6) Challenge is also described.

In this paper we propose QA^3 (pronounced as *QA cube*), a free natural language question answering system tailored for RDF cubes. It is based on a general tagging system and an original regex-like pattern language to describe flexible SPARQL partial templates. Their use is interleaved, that is, tagging helps the matching and filling in the correct template, but it can also be driven by the results of the matching. A question only partially tagged, or a chosen template which is incompletely filled in are signs that the question was interpreted to a limited extent or wrongly interpreted. This makes the system very flexible and tunable in order to trade off precision vs recall, depending on the applications. As we will see in the followings, QA^3 participated at set of the task 3 of QALD-6 challenge, remarkably improving the state of the art in free natural language question answering over data cubes.

The paper is organized as follows. First, in Section 2 the model used to represent data cubes in RDF is described. Then, the approach used by QA^3 is presented in Section 3, and preliminary experimental results are reported in Section 4. Section 5 reviews the related work in the area of Statistical Question Answering over RDF data. A discussion on the specific problem of answering OLAP-related questions through natural languages and lesson learnt in this work, including possible improvements, are presented in Section 6. Finally, Section 7 concludes the paper, describing the ongoing work.

2. The RDF data cube model

The RDF Data Cube Vocabulary [5] has been recently proposed as a W3C Recommendation for representing multi-dimensional statistical data using the RDF format. The model underlying this vocabulary is based on the SDMX model [3], an ISO standard for representing statistical data that can be shared among organizations.

In the following, we briefly describe the RDF data cube model, focusing our attention on the features depicted in Fig. 1, that are the most relevant for the question answering task. A data cube, called *dataset*, is an instance of the class `qb:Dataset`[2]. The cells of the data cube, called *observations*, are instances of the class `qb:Observation`. Each observation can have one or more *attribute*, *dimension*, and *measure* properties. The properties that define measures are of type `qb:MeasureProperty`. These properties associate the observations with quantitative values (e.g., an amount of money), having a numerical range. Attributes and dimensions are of type `qb:AttributeProperty` and `qb:DimensionProperty`, respectively. They value can be both a literal or an object. Dimensions represent the context of the observation (e.g., city in which the measured spend happened), whereas attributes provide additional information about the measure, such as the unit of measure (e.g., the currency used for representing the spend). However, attribute values they are also used (e.g., in

[1]`https://openspending.org/`

[2]`qb` is the prefix for the namespace `http://purl.org/linked-data/cube#`

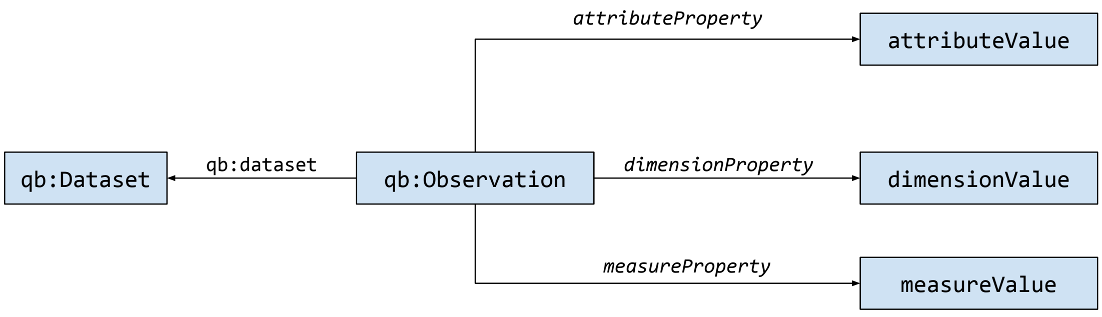


Fig. 1. The RDF data cube model simplified

several datasets of LinkedSpending [1]) to represent the temporal context of the observation (e.g., the year).

Fig. 2 depicts a set of triples that use the RDF data cube vocabulary. Assuming that the prefix `ex` denote an example namespace, the first four triples define the metadata of the dataset, in particular: (i) the dataset object with URI `ex:expenses`, (ii) a dimension with URI `ex:department`, (iii) an attribute with URI `ex:refYear`, and (iv) a measure with URI `ex:amount`. The fifth triple defines an observation with URI `ex:obs1`, and the sixth triple states that this observation belongs to the `ex:expenses` dataset. The last three triples specify the department (`ex:PublicWorks`), the year (`2015`), and the amount (`5432.1`) of the observation `ex:obs1`. In a typical RDF cube several observations would be defined (here, we limit our example to one observation for sake of brevity), each associated with the measures, dimensions, and attributes defined by the metadata. Typical queries on such data combine aggregation with simple constraints defined on the attributes/measures of the observation. For instance, a user might be interested in the total amount spent by the department of Public Works in 2015, which can be found using the following SPARQL query:

```
SELECT sum(xsd:decimal(?measure)) {
  ?observation qb:dataset ex:expenses.
  ?observation ex:amount ?measure.
  ?observation ex:department
         ex:PublicWorks.
  ?observation ex:refYear
         "2015"^^xsd:gYear.
}
```

3. An overview of QA^3

QA^3 is a system that translates questions posed in natural language into SPARQL queries, assuming that the answer to the questions can be provided using the knowledge base (KB) "known" by the system. In particular, the KB contains a set of data cubes stored using the RDF data cube model described in Section 2. 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 tokenizer and POS tagger [18], and the tags obtained at this step are augmented using the tags obtained at the previous step. The sequence of tokens is then matched against some regular expressions, 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.

3.1. Tagging the questions with elements of the KB

The purpose of the first step of the QA^3 system is two-fold:

1. finding the correct dataset to be queried, and
2. finding references in the question to concepts and values in that dataset, to generate the SPARQL query in the following steps.

It must be noted that missing the correct dataset will definitely lead to a completely wrong answer, therefore accuracy in this step is of paramount importance

```
ex:expenses          rdf:type             qb:Dataset
ex:department        rdf:type             qb:DimensionProperty
ex:refYear           rdf:type             qb:AttributeProperty
ex:amount            rdf:type             qb:MeasureProperty
ex:obs1              rdf:type             qb:Observation
ex:obs1              qb:dataset           ex:expenses
ex:obs1              ex:department        ex:PublicWorks
ex:obs1              ex:refYear           "2015"^^xsd:gYear
ex:obs1              ex:amount            "5432.1"^^xsd:double
```

Fig. 2. Example of possible triples using the RDF Data Cube Vocabulary

Algorithm 1 QA^3*tagger* pseudocode

```
Require: Question $Q$ (a string)
// this is performed only once and cached
for all $D \in$ datasets do
    normalize literals in $D$ (see details in Sect. 3.1)
    create index $I_D$
end for

// this is performed for every given question Q
for all $D \in$ datasets do
    $\text{score}_D \Leftarrow |Q|$
    for $n \in 7 \ldots 1$ do
        for all $P =$ n-gram $\in Q$ do
            if phrase $P \in I_D$ then
                in $Q$ tag $P$ with its URL from $I_D$
                $\text{score}_D \Leftarrow \text{score}_D - n$
            else if phrase $P \in$ title or description of $D$ then
                $\text{score}_D \Leftarrow \text{score}_D - 1.1 \cdot n$
            end if
        end for
    end for
end for

// compute the correct dataset
return $\text{argmin}_{D \in \text{datasets}}(\text{score}_D)$
```

to keep final performance acceptable. Furthermore, the problem of finding the right dataset is peculiar of the statistical question answering over linked data, since most of the other general-purpose question answering systems assume only one given KB, either a single one or the union of different KBs. In statistical QA, the typical desiderata is to have a system that can answer questions over hundreds of publicly available datasets about government spendings. For instance, the project *LinkedSpending* [1] currently counts about 1K datasets. In order to avoid ambiguities and non-sense, such as aggregating a measure defined in a datasets over observations defined in a different dataset, the creation of a unique big dataset from the union of all given datasets is impractical.

The idea behind the solution for this step adopted by QA^3 is to choose 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. This way, we can reduce the problem of finding the correct dataset to the problem of tagging the question. This is accomplished by the steps shown in Algorithm 1.

First, an in-memory index of all literals (labels, comments and values) contained in each dataset is created, associating each of them to the corresponding URI in the RDF cube graph. As a simple optimization, we index multiple occurrences of the same literal only once. Although lossy, since different URIs may be associated to the same literal, this trivial compression reduces of orders of magnitudes (depending on the dataset) the footprint of the index, scaling well even for large datasets (where chances of multiple occurrences are much higher).

All textual elements, that is keys in the indexes and the question itself, are normalized, by removing the stop words and performing lemmatization by means of the WordNet lemmatizer. For maximum flexibility and resilience, the system also handles optional synsets, such as those provided by WordNet, although in our experiments we only used the following two equivalence classes: $\{$*earning*, *revenue*$\}$ and $\{$*expenditure*, *spending*$\}$.

After normalization, for each dataset the question is tagged with the corresponding index. Given a dataset, we are able to tag the question with linear time complexity, thanks to the following trick: we lookup the index for every n-grams from the question, with n varying from 7 to 1. This way if our normalized question is composed of k words, we perform $O(7k)$ operations, and we are able to match all phrases made of up to 7 words. By using a negative step (as we said, we start using 7-grams up to single words) we automatically give higher priority to matchings involving longer phrases, something desirable to avoid fragmentation in the tagging and maximize the chances of a correct tagging.

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 (or score), which is intuitively based on the weighted percentage of Q that is covered by tagged chunks. We tried different heuristics and weights, but we found that the following score function has very good performance in terms of dataset recognition:

$$\text{\# untagged words } -0.1 \cdot \text{\# words matching the dataset descr.}$$

The lower the score, the better the tagging. In fact, this measure corresponds to the number of words that remained untagged (potentially related to wrong or insufficient tagging) except for a reward of an extra 10% given for any occurrence of the dataset name or description. This measure enables the system to choose the dataset which most likely has to be used to provide an answer to the question. Depending on the setting (e.g., in order to improve recall), the top-k datasets can be used as candidates, and the final choice is made in the following steps performed by the system. In case of a high score (that is, bad tagging), QA^3 is able to autonomously understand that it cannot compute the correct answer, therefore refusing to answer and eventually increasing precision over processed answers.

3.2. Finding the SPARQL query template

The domain in which QA^3 operates is quite structured, especially if we compare it to that in which general purpose question answering systems are required to work. 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, following the running example in Sect. 2, 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:

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

where `<dataset>` and `<measure>` must be replaced with the correct URIs, and `<constraints>` must be replaced with a set of triples specifying the constraints for the variable `?observation`, representing the observations. These constraints are extracted from the question and represented as *SPARQL fragments*, replacing the `<constraints>` placeholder in the main template.

Another possibility is asking which department has spent the most in 2015. In this case, the SPARQL query will select the value of the dimension that represents the department, and will provide the one associated with the largest sum of spent amount of money. Even this question, like most of the questions that can be asked in this domain, can be answered using a SPARQL query which has a structure that can be used to answer many similar questions.

In order to leverage the typical homogeneity of the structures of these questions, we implemented a system which allows the definition of a set of SPARQL templates and to automatically detect the one that should 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 [18], which tokenizes the question and annotates each token with its lemma, its POS (part of speech) tag, and its NER (named entity recognition) tag. We augment the annotations with the elements of the knowledge base (dataset, measure, dimension, attribute, literal) provided by the previous step, and with a tag representing a possible aggregate function. For the latter, we associated each aggregation function with words that users typically use to denote the function, as reported in Figure 3.

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

Aggregation function	Words
sum	sum, total
avg	average, avg
max	max, maximum, highest, largest
min	min, minimum, lowest, smallest

Fig. 3. Words used to recognize aggregate functions in questions

5. the KB tag (S: dataset, M: measure, D: dimension, A: attribute, E: entity, L: literal, O: none)
6. the aggregation tag (S: sum, A: average, M: max, N: min, O: none)

A *generalized-token* is defined as a 6-tuple, where the i-th element represents the possible set of values for the i-th feature of the actual tokens (the features are assumed to follow the order in which they are listed above). For instance, the generalized-token `WP|WDT,_,_,_,!E` matches against actual tokens that have `WP` or `WDT` as POS tag, any lemma, word, and NER tag (_), and the token must not be annotated as entity (`!E`). The features that are not explicitly specified in the generalized-token (in this case the aggregation tag), are allowed to take any value (i.e., _ is implicitly assumed).

A generalized-token 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 generalized-token. The generalized-tokens can be used to build more complex regular expressions. Fig. 4 reports all the 7 templates currently used by QA^3, with the corresponding regular expression.

We describe in details one of them, namely the number 4, which is the most complex:

$$\underbrace{\{\_\}*}_{1}\ \underbrace{\texttt{WP|WDT}}_{2}\ \underbrace{\{\_,\_,\_,\_,\texttt{O},\texttt{O}\}*}_{3}\ \underbrace{\_,\_,\_,\_,\texttt{O},!\texttt{O}\#1}_{4}\ \underbrace{\{\_,\_,\_,\_,\texttt{M|S}\#2\}*}_{5}\ \underbrace{\{\_\}*}_{6}$$

Each of the 6 generalized-token of the expression above must match subsequent chunks of the whole question. A question, seen as a sequence of tokens, matches the expression above if it consists of contiguous sub-sequences of tokens with the following properties:

1. any sequence of tokens (`{_}*`);
2. a token with the POS tag `WP` or `WDT`;
3. 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`);
4. 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`);
5. 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 practical case, it will be none). These tokens are assigned the label *2* (`#2`);
6. 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, questions matching this expression ask 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*. This expression also considers the possibility that the measure is implicitly denoted by the name of the dataset (this can happen, for instance, when the dataset is about a specific measure of a set of observations - *expenditures of town of Cary*), or that the measure is not explicitly mentioned. In this case, we assume that a default measure is defined for each dataset. The default measure is obvious for datasets with one only measure. In case of multiple measures, our system need to manually define the default measure for the dataset. This operation is performed just once, when a new dataset is added to the whole KB.

The questions of this kind can be translated into a SPARQL query with the following structure (template):

```
select <groupbyvar> <aggrFunct##1>(?measure)
where {
  ?obs qb:dataSet <dataset> .
  ?obs <measure##2> ?measure .
  <constraints>
}
<groupby>
```

where `<aggrFunct##1>` has to be replaced with 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* (or with the default measure, if no token has been la-

1	RegEx	`{_}* _,how _,many {_,_,_,_,O,O}* {_,_,_,_,A|D#1}+ {_}*`
	Template	`select <groupbyvar> count(distinct ?ans) where { ?obs qb:dataSet <dataset> . ?obs <property##1> ?ans . <constraints> } <groupby>`
	Matching Query	How many programs were done under the class of General Government in the expenditure of the Town of Cary?
2	RegEx	`{_}* _,how _,much|high {_}*`
	Template	`select <groupbyvar> sum(?measure) where { ?obs qb:dataSet <dataset> . ?obs <measure> ?measure . <constraints> } <groupby>`
	Matching Query	How much was spent on public safety by the Town of Cary in 2010?
3	RegEx	`_,when {_}*`
	Template	`select distinct ?ans where { ?obs qb:dataSet <dataset> . ?obs <http://linkedspending.aksw.org/ontology/refDate> ?ans . <constraints> }`
	Matching Query	When was the upgrade of the Parks-Baba Park paid?
4	RegEx	`{_}* WP|WDT {_,_,_,_,O,O}* _,_,_,_,O,!O#1 {_,_,_,_,M|S#2}* {_}*`
	Template	`select <groupbyvar> <aggrFunct##1>(?measure) where { ?obs qb:dataSet <dataset> . ?obs <measure##2> ?measure . <constraints> } <groupby>`
	Matching Query	What was the highest single expenditure amount proposed by the Maldives Broadcasting Corporation?
5	RegEx	`{_}* WP|WDT {_,_,_,_,O,O}* {_,_,_,_,A|D#1}+ {_}*`
	Template	`select distinct ?ans where { ?obs qb:dataSet <dataset> . ?obs <measure> ?measure . ?obs <property##1> ?ans . <constraints> } <orderbysummeasure>`
	Matching Query	Which class achieved the highest revenue for the Town of Cary?
6	RegEx	`{_}* WP|WDT {_,_,_,_,O,O}* {_,_,_,_,M|S#2}+ {_}*`
	Template	`select <groupbyvar> sum(?measure) where { ?obs qb:dataSet <dataset> . ?obs <measure##2> ?measure . <constraints> } <groupby>`
	Matching Query	What was the frontex staff budget in 2005?
7	RegEx	`_,have|be _,_,there {_}*`
	Template	`ask { ?obs qb:dataSet <dataset> . <constraints> }`
	Matching Query	Has there been a big lottery fund grant to Stanbury Court Social Club?

Fig. 4. Pattern/Templates used by QA^3 and an example of matching query for each template

beled). Finally, `<constraint>`, `<groupbyvar>` and `<groupby>` must be replaced with the actual variable/clauses (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 (for instance, none of the 7 patterns can match the QALD query "Top 3 IW Council Spending service areas?"), as we show in Section 4 we empirically found that few expressions are enough to cover most of the questions posed in a typical form.

3.3. *Filling the query template using the annotations*

The input to this step is the output of the previous two steps, i.e., the SPARQL query template, the dataset to be used, and the tokens with the annotations based on the KB tags. In particular, given the SPARQL template, some portions of the template must be replaced in order to obtain the final query, by using the result of the previous KB tagging. For instance, as stated above, `<dataset>` is replaced with the URI of the dataset found in the first step, `<measure##p>` (where p is a label identifier) is replaced with the actual measure described by tokens with label p (or with the default measure), `<aggregationFunction##q>` has to be replaced with the actual aggregation function, which can be derived using the token annotated with label q. The most interesting part is the construction of the constraints and and the group-by clauses. In order to construct the constraints, we observe that

- if a literal is tagged, then it must be connected to the observation through an attribute, which is also reported in the annotation;
- if an entity is tagged, then it must be connected to the observation through a dimension. The dimension could be be explicitly tagged in the question, but 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 SPARQL fragment representing the RDF triple pattern 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 `?groupbyvar`, and `?observation` is connected to `?groupbyvar` using the URI of X, and the `<groupby>` placeholder is replaced with `group by ?groupbyvar`. If no such attribute/dimension X can be found, both `<groupbyvar>` and `<groupby>` are replaced with the empty string.

4. Experimental results

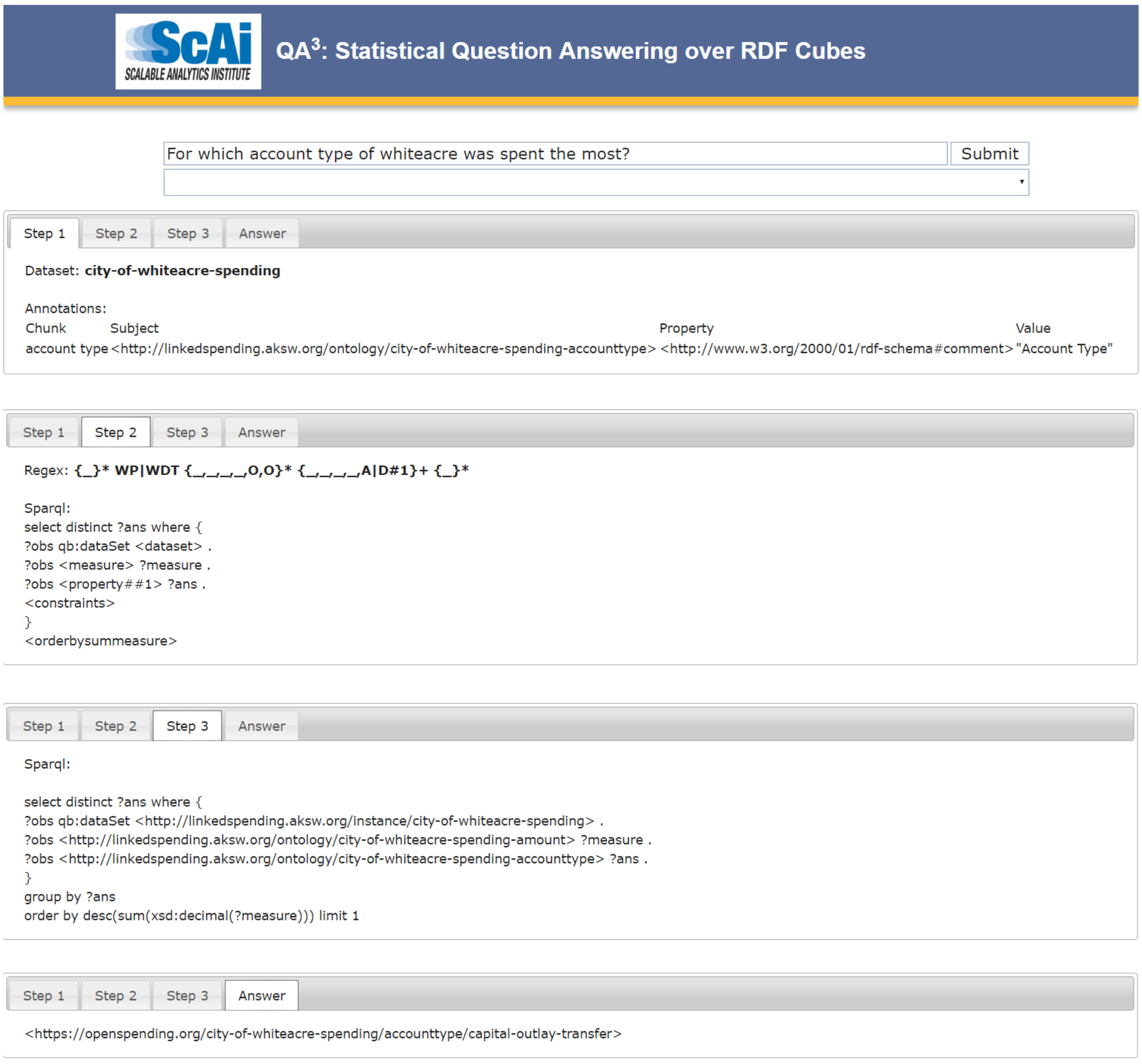
We implemented the backend of QA^3 [6] using Python and Java[3] and a web interface is available at `http://qa3.link/`. A screenshot of the system is available in Fig. 5, where the question *For which account type of whiteacre was spent the most?* is posed to QA^3 and the results of each intermediate step are shown. The system finds the correct dataset (`city-of-whiteacre-spending`), executes a group-by query against the SPARQL endpoint and finally provides *capital-outlay-transfer* as the answer (which turns to be correct).

The online system allows to freely type questions, but it also provides the user with a quick selection of the questions of the training and test sets of the task 3 of QALD-6 challenge [2], for which the experimental outcomes are described next.

Fig. 6 reports the number of total questions (column "N"), and the results obtained by QA^3 [6] over the two set of questions. Besides the average recall, precision and F1-score computed over all the questions, it is interesting to analyze the accuracy of the internal steps of QA^3. In particular, for the training set, over 100 available questions, the correct dataset (column "Correct DS") is found for 81 questions, the correct set of tags is found for 56 questions, and the correct query is generated for 47 questions. QA^3 performed even better on the test set[4]: over the 50 available questions the correct dataset was found for 42 questions, the correct set of

[3]The Python code implementing the initial tagging step is available at `https://github.com/atzori/qa3tagger`, the Java code implementing the following template-related steps can be found at `https://github.com/gmmazzeo/qa3`, the code of the demo web application is available at `https://github.com/gmmazzeo/qa3demo`.

[4]Note that this results are not the official ones: they are slightly different since we did some minor improvements on the prototype after the challenge


SCAI
SCALABLE ANALYTICS INSTITUTE
QA³: Statistical Question Answering over RDF Cubes
Submit
Step 1
Step 2
Step 3
Answer

For which account type of whiteacre was spent the most?

Step 1

```
Dataset: city-of-whiteacre-spending

Annotations:
Chunk        Subject                                                                  Property                                            Value
account type <http://linkedspending.aksw.org/ontology/city-of-whiteacre-spending-accounttype> <http://www.w3.org/2000/01/rdf-schema#comment> "Account Type"
```

Step 2

```
Regex: {_}* WP|WDT {_,_,_,_,O,O}* {_,_,_,_,A|D#1}+ {_}*

Sparql:
select distinct ?ans where {
?obs qb:dataSet <dataset> .
?obs <measure> ?measure .
?obs <property##1> ?ans .
<constraints>
}
<orderbysummeasure>
```

Step 3

```
Sparql:

select distinct ?ans where {
?obs qb:dataSet <http://linkedspending.aksw.org/instance/city-of-whiteacre-spending> .
?obs <http://linkedspending.aksw.org/ontology/city-of-whiteacre-spending-amount> ?measure .
?obs <http://linkedspending.aksw.org/ontology/city-of-whiteacre-spending-accounttype> ?ans .
}
group by ?ans
order by desc(sum(xsd:decimal(?measure))) limit 1
```

Answer

```
<https://openspending.org/city-of-whiteacre-spending/accounttype/capital-outlay-transfer>
```

Fig. 5. Screenshot of the QA^3 web interface, showing all the steps followed by the system and the final answer for an example question taken from QALD-6

Question Set	N	Correct DS	Correct tags	Correct queries	Recall	Precision	F1-score
Training	100	81	56	47	0.49	0.48	0.48
Test	50	42	31	27	0.57	0.56	0.56

Fig. 6. Results obtained by QA^3 (using multiple candidate datasets) on QALD-6 training set and test

System	Processed	Recall	Precision	F-1 (over processed questions)	F-1 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 (our system, initial tuning)	**44**	**0.62**	**0.59**	**0.60**	**0.53**
CubeQA	49	0.41	0.49	0.45	0.44

Fig. 7. Comparison against other QA systems, as reported by the QALD-6 independent competition

annotations for 31 questions and the correct query for 27 questions.

We believe the results are extremely encouraging because despite the small number of patterns (only the 7 ones described in Fig. 4), the system was able to provide the correct answer in the majority of the test questions. A direct comparison against the other systems described in Section 5 has been independently performed by the QALD 6 Challenge [2], as reported in Fig. 7. The most performer in this comparison is SPARKLIS[5], 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 are therefore dependant on the level of expertise the user has (values for expert and beginner are shown).

To the best of our knowledge, the only two systems that answers free natural language questions over RDF cubes are CubeQA [15], described in the next section, and our QA^3. Compared to the state-of-the-art CubeQA, we get an remarkable improvement of $0.62/0.41 = 51\%$ in recall, 20% in precision. This is in part given by the ability of QA^3 of self-evaluate the confidence of a computed answer (thanks to the score measure of the tagger), and also by the good expressivity and generality of the template/pattern system.

We also remark that F-1 and F-1 Global, i.e., the F-1 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. Thanks to the flexibility of the architecture, system settings and tuning may produce results biased on precision w.r.t. recall or viceversa, e.g. a larger number of candidates to keep or higher thresholds for their scores may lead to higher recall, while givin up answering a question in presence of low scores or ambiguity chosing the template would improve precision.

5. Related Work

From a general point of view, querying RDF data through user-friendly interfaces is an active area of research, with quite different approaches. To the best of our knowledge most of the proposed work can be categorized into two main approaches: *1)* based on ad-hoc user interfaces, and *2)* question answering systems, either accepting (free) natural language or constrained natural language. Among question answering systems, it is possible to distinguish those focusing on general RDF data, with the majority of them using the DBpedia ontology, and those called *Statistical Question Answering* systems, aiming at querying a large number of datasets containing RDF data cubes.

Our work falls into the *Statistical Question Answering* category, since QA^3 queries RDF data cubes by allowing users to write the query the way they like, while disambiguation and understanding is accomplished by the system.

In the following we discuss existing work on querying RDF data using the taxonomy discussed so far: *1)* based on ad-hoc user interfaces, *2)* question answering systems accepting natural language or CNL and *3)* statistical question answering systems, specifically addressing RDF data cubes.

5.1. Work on Interfaces to Query Linked Data

A number of very different interfaces to query RDF data in a user-friendly way have been proposed.

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 the users do not need to know beforehand the exact names of properties, this approach can be effectively used only 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 [13,10,9,12]. This approach was recently applied to the RDF data cubes [20], 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* [7,25], which makes the infobox of WikiPedia pages editable as if it was a fillable form. The user can type the constraints using the value fields of the infobox, according to the *By-Example Structured Query* (BEStQ) paradigm – a QBE-inspired interface to query DBpedia or similar RDF data. While

[5]In fact, it is a special version of the SPARKLIS system tailored to statistical questions, as described in the related work section.

this paradigm is very effective for finding lists of entities with specific properties and/or temporal constraints [25] starting from a Wikipedia page, its generalization to the RDF data cubes is not trivial.

5.2. *Work on Question Answering over Linked Data*

Natural language interfaces are the most challenging solutions to the problem of querying 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* [24], that works in two steps. In the first step, phrases are extracted from the question using a structured perceptron that can identify variables, entities, classes and relation phrases. By means of a semantic parser, the predicate-argument structure of phrases is derived, thus obtaining the structure of the query intention. In the second step, the semantic phrases are mapped against the elements of the knowledge base (specifically, DBpedia) by using WikipediaMiner [22] for entities, and an ad-hoc lexicon that maps classes and relation phrases

Xser was able to yield an F-score equal to 0.72 and 0.63 in the, respectively, 2014 and 2015 QALD challenges [2]. Although its accuracy is far from being 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 the task can be reduced by using a *controlled natural language* (CNL), i.e., a language which 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 [11] and GFmed [19].

Squall2sparql [11], is a CNL system that translates queries written using its SQUALL language into SPARQL. The translation is based on about 100 rules of a Montague grammar. The chunks of the SQUALL sentence must be annotated by the user, and written in a form that enables the direct mapping to elements of the knowledge base. The language enables users to both query and update the knowledge base, and uses all the SPARQL features. Therefore, its Squall2sparql interface to RDF seems to push the CNL idea to its extreme, inasmuch as it achieves the greatest expressive power, but the need to manually annotate the chunks of the sentences severely limits its usability — a fact recognized by the author who proposes the use of a meta-level interface to guide the user in writing annotated questions.

GFMed [19] is a CNL system specialized for the biomedical domain. It is based on the Grammatical Framework [23], which enables to define grammars by means of an abstract syntax and one or more concrete syntaxes. The abstract syntax defines the concepts that can be expressed as non-terminal symbols and the rules for their composition. The concrete grammar defines how the trees specified through the abstract syntax are linearized into sentences of a specific language (e.g., English, SPARQL, etc.). The possibility of defining more concrete syntaxes allows GF to serve as a powerful tool for translating sentences from one language to another. The GFMed system consists of a GF program that defines a grammar allowing to pose questions over the knowledge bases such as DrugBank, Diseasome and SIDER. The GF program is completed with a post-processing procedure for handling literals, that can not be defined using the concrete syntax. GFMed proved to be very accurate on the biomedical questions of QALD-4. The main limitation of this approach is the need to write the grammar rules for all the concepts of the underlying the dataset, which can be a very hard task for large ontologies such as DBpedia.

Therefore, 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, and the latter has scalability problems, since the number of rules of its grammar depends on the size of the knowledge base.

Authors in [21] also proposed *CANaLI*, a system that overcomes the above 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.

None of the previously proposed systems, either based on full or controlled natural language, has been specialized for question answering over RDF data cubes. As also discussed next and in Section 6, RDF data cubes pose specific problems not arising in general question answering, therefore it is not known how to adapt the above systems to handle statistical question answering.

5.3. Work on Statistical Question Answering over RDF Data Cubes

Question answering over RDF cubes is a brand new challenge, which raises issues different from those of question answering on a “general” knowledge base [14]. In fact, questions in this context are likely to be very specific, i.e., oriented towards extracting statistical information. Thus, a system called to interpret these questions must be as accurate as possible in interpreting specific features of the typical OLAP queries, such as different kind of aggregations or constraints on dimensions. Furthermore, questions on statistical data are more sensitive to misinterpretations. For instance, while a partial interpretation of a general question might also yield an acceptable answer (maybe with non perfect precision), an aggregation query missing a constraint is likely to yield a totally wrong answer.

Although datasets, benchmarks and problem definition have been proposed only recently, there are already two other existing works that focused on Statistical Question Answering over Linked Data. The first system is called *CubeQA* [15], and reached an average 0.4 F-measure at Task3 of QALD-6 challenge, as described in the previous section. The system has a lightweight template module that matches the kind of question (what, how much, when, etc.) against a table that provides the expected result type (countable, uncountable, date, etc.). This improves the results provided by the main step, consisting of parsing the query and creating a tree. Elements of the tree are matched against a pre-computed index that allows to find the most probable dataset and dimensions and generate the corresponding SPARQL query.

In contrast, our approach is extremely general in the first steps, based on entity and property recognition within the query. We do not create a tree, trying instead to tag as much as possible numbers and words (either single or multiple words) in the question. We do this for each possible dataset, using an in-memory index. We then rank solutions according to a score, and the one with higher rank is choosen. Rank increases according to the portion of the question that has been interpreted/understood (tagged); also, ranking decreases on the number and size of recognized entities (multiple word tagging is preferred). We then use an original template system, with an ad-hoc language, that exploits NLP to express SPARQL templates based on the grammar structure of the query. We also propose partial templates, that (if matched) establish a portion of the SPARQL query (e.g. a new triple pattern, the use of group by over a given attribute, or the use of a specific aggregate function). The matching templates are then used together to generate the final SPARQL query. An interesting fact related to our approach is that each step we run provides insights on how well it has been performed. For instance, if a large portion of the question is not tagged, it means that most of it was not understood and, therefore, chances of obtaining the right answer in the following steps are low. In these cases, we can maximize precision over recall, not providing an answer. The same optimization can be applied for the template-based mechanism, since some templates are mutually exclusive (e.g. either SUM or COUNT can be used as a final aggregate function). Furthermore, the template mechanism is very simple and it easily allows new pattern to be input in order to answer more advanced questions. These techniques result in a sensible increment of the precision and recall of our QA^3 over CubeQA, as certified by the QALD-6.

A very different approach is presented in the work by Sébastien Ferré [10,9]. Based on the adaptation [12] of the general purpose Faceted Search system called SPARKLIS [10,9], the system allows users to incrementally add new constraints. In particular, disambiguation and most of the semantically difficult tasks are not done automatically, but left to the user. Although SPARKLIS takes care of also generating a natural language question that helps user to verify that the input constraints are leading to what he is willing to search for, this system cannot be considered a Natural Language approach to Question Answering. One peculiarity of the SPARKLIS system is therefore that achieved precision and recall strongly depend on the user which interacts with the system. According to the QALD-6, the system has been tested by two users with different skill level, and in both cases SPARKLIS achieved a higher F-measure than both CubeQA and our QA^3.

6. Discussion

In this section we discuss the lessons we learnt from implementing a system tailored for the specific task of statistical question answering.

First of all, as previous work already noticed [17, 14], there are good reasons why an analytical-aware representation of the data should be published instead of a flat one. In DBpedia, especially if we focus on the Yago categories, information needed to answer the question "What is the 2014 public transportation budget of Frankfurt?" may be found in triples like the following:

```
:Frankfurt :publicTransportBudget2014Eur
"5.6E7"^^xsd:decimal
```

Unfortunately, this representation may not be useful for statistical reasoning or analytics, e.g. "In which year Frankfurt had the higher public transportation budget" or "What is the average public transportation budget in Germany", because some data needed to filter or to aggregate is "hidden" within the property name.

The adopted solution in literature is an ad-hoc form of reification where a specific OLAP ontology is used [5]. This data model must be taken into account by any question answering system, as the matching between the question and the data follow a different path. In particular, this time answers need to be derived from statistical observations, selecting the appropriate dimensions among many, and values often need to be further processed, e.g., by applying aggregate functions.

In our work instead of adapting an existing QA system, we started from scratch. In our opinion, one of the most important decisions to take while designing a statistical QA system is how to handle the multitude of datasets. Questions in the task 3 of QALD-6 testbed all refers to one dataset per query. In fact, more complex query (such as comparisons among countries or cities) could be answered by two different subqueries, each using one dataset only. Pointing to the right dataset is of paramount importance because the result will be definitely wrong otherwise. We learnt that the first phases of the system, that recognize references to properties and entities within the question, are very important to exclude a vast number of datasets unrelated to the question. Still, given the similarities among some datasets contents (e.g., between `town_of_cary_revenues` and `town_of_cary_expenditures`) the search for the right dataset will contain more than one solution in the first phases of the QA system. Later steps, which actually try to match tagged elements with existing patters and/or generating the SPARQL query, are useful to further disambiguate. Therefore, in our experience finding the right dataset is something that may involve the whole system, until the last step of generating the results. In fact, getting no results may suggest a wrong interpretation of the question and a different interpretation leading to another SPARQL query could be then assumed in order to improve the recall.

In terms of time performance, our implementation scales linearly with the number of datasets. With in-memory indexes it is acceptable for the 50 datasets used at QALD, but both time and memory resources required for, e.g., thousands of datasets, can be demanding. Anyway, QA^3 can be adapted to scale to tens of thousands of datasets by substituting the *for all* $D \in$ *datasets* loops in Algorithm 1 and the many indexes I_d with a single trie (prefix tree) with keys from all the datasets. In this case, the structure can stay offline and, for each n-gram, multiple taggings for different datasets can be done by a single access to the trie.

Another aspect is the trade off between precision and recall. Every step in QA^3 is associated with an internal score that represents our confidence in that specific decision/interpretation. Going further with low scores leads to higher recall but potentially lower precision, and the viceversa is also true. The system can be therefore tailored for a specific application and/or coupled with a user interface that interacts with the user asking for more (potentially wrong) results. In order to improve recall, another improvement would be to "guess" correct references to constraints, properties and measures required by a template but missing, by exploiting tagged entities/observations. This can be done, e.g., by ranking features associated to each entity [8].

But perhaps the most important lesson we learnt is that patterns are at the same time very important (if the question contains "average of ...", this must be taken into account) but also that specific patterns may support only few questions. One naive approach may be to generate all possible patterns and generate its natural language representation, then trying the best matching with the current question. The exponential explosion in the search space needs to be addressed, perhaps with original compact representations. As described in the paper, we instead developed small, possibly overlapping patterns that may occur together in the same question. These can be seen as "features" in the question, each associated with a fragment of SPARQL. The combination of the features found in the question leads to the complete SPARQL query, unless the features cannot combined together (in this case another alternative is taken).

We believe that having a good, comprehensive set of these features, that is, question patterns associated with SPARQL fragments, can lead to high precision *and* recall. Our future work is to extend and improve our manually-crafted set of patterns using machine learning techniques. In particular, the QALD-6 testbed, containing both the natural language questions and the SPARQL queries, can be used as input to statistical learners that can learn, e.g., that "average of ..." can be associated to the SPARQL fragment `AVG(...)`, leading to new automatically-learned patterns.

7. Conclusions

In this paper we have presented QA^3, a system supporting statistical question answering over RDF data cubes. The system combines the syntactical tags obtained using the Stanford tokenizer and POS tagger with semantic tags derived by QA^3 from the available knowledge base. An extensible set of regular expressions, defined over the tokens, is used to detect the query template to be used. The template is then filled in by using the previously found KB tags. We remark that this approach is effective in the context of RDF data cubes because the meaningful questions naturally tend to follow structured patterns, while most of the effectiveness would be lost in a more general context (e.g., questions posed on DBpedia KB), where the variability of the structure of meaningful question is much higher. The preliminary results are encouraging, and we are currently working on improving all the three steps of the translation process used by QA^3. In particular, since the wrong annotations often do not enable to fill in all placeholders in the query template, the performances are likely to be improved by considering additional datasets and tagging results, whenever the choice of one dataset does not yield a valid SPARQL query. This is likely to provide a correct answer to some of the questions for which the correct dataset is assigned the same quality measure as other wrong datasets, and even to some questions for which a wrong dataset is initially detected as the best candidate. Low scores in the first step must be appropriately weighted with the higher scores obtained in the later steps.

The tagging system could be also improved by allowing an approximate matching. This possibility, however, is not obvious since while the approximate matching will improve the recall of the tagging, it could also worsen the precision. A more sophisticated handling of synonyms, taking care of a fuzzy semantic relativity could also improve the answers of questions expressed with different terms w.r.t. the one found in the dataset.

Finally, we are also working on improving the set and internal language of SPARQL templates, in order to enable the use of more advanced SPARQL features (es., the `HAVING` clause, the sub-queries, etc.).

Acknowledgements

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

References

[1] LinkedSpending project. `http://linkedspending.aksw.org/`.
[2] Question Answering over Linked Data (QALD). `http://www.sc.cit-ec.uni-bielefeld.de/qald`.
[3] Statistical Data and Metadata eXchange. `https://sdmx.org/`.
[4] The Linking Open Data cloud diagram. `http://lod-cloud.net/`.
[5] The RDF Data Cube Vocabulary. `https://www.w3.org/TR/vocab-data-cube/`.
[6] QA^3 website. `http://qa3.link/`.
[7] Maurizio Atzori and Carlo 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.
[8] Andrea Dessi and Maurizio Atzori. A machine-learning approach to ranking RDF properties. *Future Generation Computer Systems*, 54:366–377, 2016.
[9] Sébastien Ferré. Expressive and scalable query-based faceted search over SPARQL endpoints. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandecic, Paul T. Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble, editors, *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II*, volume 8797 of *Lecture Notes in Computer Science*, pages 438–453. Springer, 2014.
[10] Sébastien Ferré. SPARKLIS: a SPARQL endpoint explorer for expressive question answering. In Matthew Horridge, Marco Rospocher, and Jacco van Ossenbruggen, editors, *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.*, volume 1272 of *CEUR Workshop Proceedings*, pages 45–48. CEUR-WS.org, 2014.
[11] Sébastien Ferré. SQUALL: the expressiveness of SPARQL 1.1 made available as a controlled natural language. *Data Knowl. Eng.*, 94:163–188, 2014.
[12] Sébastien Ferré. Sparklis: An expressive query builder for sparql endpoints with guidance in natural language. *Semantic Web*, 7(1):95–104, 2016.
[13] Rasmus Hahn, Christian Bizer, Christopher Sahnwaldt, Christian Herta, Scott Robinson, Michaela Bürgle, Holger Düwiger, and Ulrich Scheel. Faceted wikipedia search. In *Business In-*

formation Systems, 13th International Conference, BIS 2010, Berlin, Germany, May 3-5, 2010. Proceedings, pages 1–11, 2010.

[14] Konrad Höffner and Jens Lehmann. Towards question answering on statistical linked data. In *Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014*, pages 61–64, 2014.

[15] Konrad Höffner, Jens Lehmann, and Ricardo Usbeck. Cubeqa - question answering on RDF data cubes. In Paul T. Groth, Elena Simperl, Alasdair J. G. Gray, Marta Sabou, Markus Krötzsch, Freddy Lécué, Fabian Flöck, and Yolanda Gil, editors, *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, volume 9981 of *Lecture Notes in Computer Science*, pages 325–340, 2016.

[16] Konrad Höffner, Michael Martin, and Jens Lehmann. Linkedspending: Openspending becomes linked open data. *Semantic Web*, 7(1):95–104, 2016.

[17] Konrad Höffner, Sebastian Walter, Edgard Marx, Ricardo Usbeck, Jens Lehmann, and Axel-Cyrille Ngonga Ngomo. Survey on challenges of Question Answering in the Semantic Web. *Semantic Web Journal*, 2016.

[18] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, 7-12 July 2003, Sapporo Convention Center, Sapporo, Japan.*, pages 423–430, 2003.

[19] Anca 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.

[20] Michael Martin, Konrad Abicht, Claus Stadler, Axel-Cyrille Ngonga Ngomo, Tommaso Soru, and Sören 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.

[21] Giuseppe M. Mazzeo and Carlo 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.

[22] David Milne and Ian H. Witten. An open-source toolkit for mining wikipedia. *Artif. Intell.*, 194:222–239, January 2013.

[23] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. Center for the Study of Language and Information/SRI, 2011.

[24] Kun Xu, Yansong Feng, Songfang Huang, and Dongyan 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.

[25] Carlo Zaniolo, Shi Gao, Maurizio Atzori, Muhao Chen, and Jiaqi Gu. User-friendly temporal queries on historical knowledge bases. *Information and Computation*, 54, (to appear).