

# CubeQA—Question Answering on RDF Data Cubes

Konrad Höffner<sup>1(✉)</sup>, Jens Lehmann<sup>2,3</sup>, and Ricardo Usbeck<sup>1</sup>

<sup>1</sup> University of Leipzig, Institute of Computer Science, AKSW Group,  
Augustusplatz 10, 04109 Leipzig, Germany

{[hoeffner](mailto:hoeffner@informatik.uni-leipzig.de), [usbeck](mailto:usbeck@informatik.uni-leipzig.de)}@informatik.uni-leipzig.de

<sup>2</sup> Computer Science Institute, University of Bonn, Bonn, Germany

[jens.lehmann@cs.uni-bonn.de](mailto:jens.lehmann@cs.uni-bonn.de)

<sup>3</sup> Knowledge Discovery Department, Fraunhofer IAIS, Sankt Augustin, Germany

[jens.lehmann@iais.fraunhofer.de](mailto:jens.lehmann@iais.fraunhofer.de)

**Abstract.** Statistical data in the form of RDF Data Cubes is becoming increasingly valuable as it influences decisions in areas such as health care, policy and finance. While a growing amount is becoming freely available through the open data movement, this data is opaque to laypersons. Semantic Question Answering (SQA) technologies provide intuitive access via free-form natural language queries but general SQA systems cannot process RDF Data Cubes. On the intersection between RDF Data Cubes and SQA, we create a new subfield of SQA, called RDCQA. We create an RDQCA benchmark as task 3 of the QALD-6 evaluation challenge, to stimulate further research and enable quantitative comparison between RDCQA systems. We design and evaluate the domain independent CubeQA algorithm, which is the first RDCQA system and achieves a global  $F_1$  score of 0.43 on the QALD6T3-test benchmark, showing that RDCQA is feasible.

## 1 Introduction

Statistical data influences decisions in domains such as health care, policy, governmental decision making and finance. The general public is increasingly interested in accessing such open information [19]. This coincides with the open data movement and has led to an increased availability of statistical government data in the form of data cubes. Initiatives that publish those statistics include OpenSpending<sup>1</sup> and World Bank Open Data<sup>2</sup>. However, this type of data is multidimensional, numerical and often voluminous, and thus not easily approachable for laypersons.

While singular data points can be queried using a tabular and faceted browsing interfaces offered by those initiatives, common questions often require [12] the combination and processing of many different datapoints. This processing can be performed by specialized tools but they require knowledge of a specific

<sup>1</sup> <http://openspending.org/>.

<sup>2</sup> <http://data.worldbank.org/>.

vocabulary or a formal query language and are thus also hindering the access for laypersons. To provide a more intuitive interface, we present the CubeQA approach.

Our contributions are as follows: (1) To the best of our knowledge, we are the first to tackle the intersection between Data Cubes, Question Answering (QA) and RDF, creating a new subfield of QA, which we call RDCQA. (2) We stimulate further research with the creation of the QALD-6 Task 3 benchmark (QALD6T3): *Statistical Question Answering over RDF datacubes*. This enables quantitative comparison between RDCQA systems, see Sect. 4.2. Moreover, the introduction of this task has led to the development of another system, QA<sup>3</sup> (“QA cube”), and the extension of the SPARQL query builder Sparklis (see Sect. 5). (3) With the CubeQA algorithm, which achieves a global  $F_1$  score of 0.43 on QALD6T3-test (see Sect. 4), we show that RDCQA is feasible.

The rest of the paper is structured as follows: Section 2 introduces the preliminaries. Section 3 defines the CubeQA algorithm. Section 4 presents our benchmark and evaluates the CubeQA algorithm. Section 5 summarizes general SQA approaches and work in progress on RDCQA. We summarize our contributions in Sect. 6 and present challenges to be addressed by future work.

## 2 Preliminaries

Unlike common data representations such as tables or relational databases, the *data cube*<sup>3</sup> formalism adequately represents multidimensional, numerical data. A data cube is a multidimensional array of *cells*. Each cell is uniquely identified by its associated dimension values and contains one or more numeric *measurement* values. Data cubes are often *sparse*, i.e., for most combinations of dimension values there is no cell in the cube. Data cubes, such as in Fig. 2a, allow the following operations supported by CubeQA:

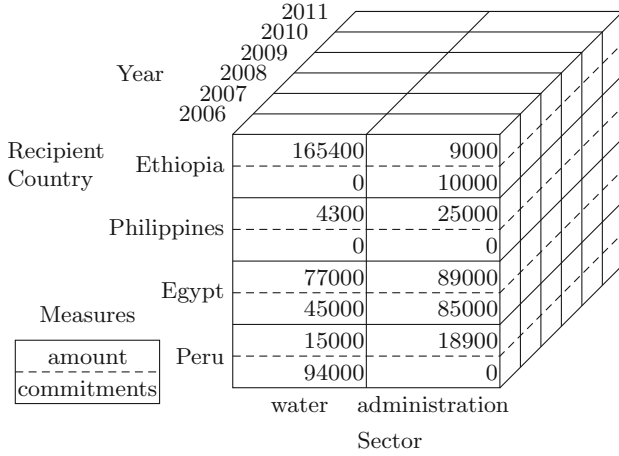
1. *Dicing* a data cube creates a subcube by constraining a dimension to a subset of its values, see Fig. 2b.
2. *Slicing* a data cube reduces its dimensionality by one by constraining a dimension to one specific value, see Fig. 2c.
3. *Rolling Up* a data cube means summarizing measure values along a dimension, such as a sum, count, or arithmetic mean. A roll-up of Fig. 2c answers Fig. 1.

**Definition 1.** We define Question Answering (QA) [11] as users (1) asking questions in natural language (NL) (2) using their own terminology to which they (3) receive a concise answer. In Semantic Question Answering (SQA), the

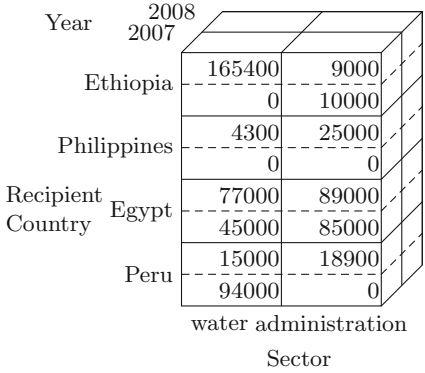
How much did the Philippines receive in the years of 2007 to 2008?

**Fig. 1.** The running example used throughout this paper

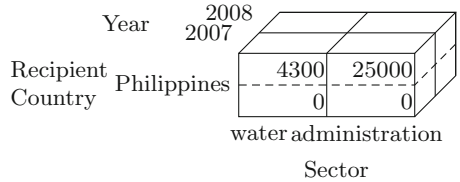
<sup>3</sup> also *OLAP cube* or *hypercube*.



(a) Simplified excerpt of the LinkedSpending RDC *Finland Aid Data* for Fig. 1. Measure units are provided by the *currency* attribute in each cell (omitted for brevity).



(b) A *dice* of Fig. 2a created by constraining the *year* dimension to 2007 and 2008.



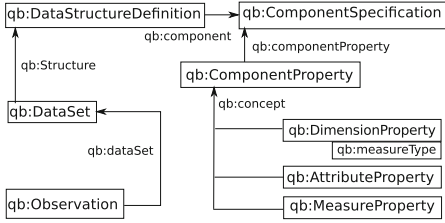
(c) A *slice* of Fig. 2b created by constraining the *Recipient Country* dimension to the Philippines.

**Fig. 2.** Example of a data cube and its operations.

natural language question is transformed into a formal query on an RDF knowledge base, commonly using SPARQL.

The *RDF Data Cube (RDC) Vocabulary* models both the schema and the observations of a data cube to RDF (see Fig. 3a). Each data cube, an instance of `qb:DataSet`, has an attached schema, the *data structure definition*, which specifies the *component properties*<sup>4</sup>. A component property is either a dimension, an attribute or a measure. *Measures*, of which there has to be at least one, represent the measured quantities, while the dimensions and the optional attributes provide context. Because the RDC vocabulary is focused on statistical data, its cells

<sup>4</sup> `qb:ComponentProperty` in Fig. 3a, not to be confused with `rdf:Property`.



(a) Simplified structure of the RDF Data Cube (RDC) vocabulary, which determines the triple patterns required for the SPARQL query. A more detailed explanation of the RDC vocabulary is presented in [12]. Figure originally published in [13].

```
SELECT sum(?amount)
FROM :finland-aid
{
  ?o a qb:Observation .
  ?o :amount ?amount .
  ?o :recipient-country :ph .
  ?o :refYear ?y .
  filter (year(?y)=2007
          OR year(?y)=2008)
}
```

(b) A SPARQL query answering the running example (Fig. 1).

**Fig. 3.** The RDC vocabulary and an exemplary SPARQL query for Fig. 1.

are called *observations*. Each observation contains a triple that specifies a value for each component property<sup>5</sup>.

RDCs allow data cubes to profit from the advantages of Linked Data [3], such as ontologies, reasoning and interlinking. For example, the dimension value for *recipient country* :ph can be linked to [dbpedia:Philippines](https://dbpedia.org/ontology/Philippines), whose properties can then be queried from DBpedia [16].

### 3 CubeQA Algorithm

The CubeQA algorithm converts a natural language question to a SPARQL query using a linear pipeline. Its first step, preprocessing, indexes the target datasets, extracts simple constraints and creates the parse tree used by the following steps. Next, the matching step recursively traverses the parse tree downwards until it identifies reference candidates at each branch. Starting at those candidates, the combination step merges those candidates upwards until it creates a final template in the root of the parse tree. Finally, in the execution step, the template is converted to a SPARQL query that is executed to generate the result set containing the answer.

#### 3.1 Preprocessing

*Number Normalization.* First, numbers are normalized, for example “5 thousand” to “5000”, as the other components do not recognize numbers in words.

*Keyphrase Detection.* In this step, phrases referring to data cube operations are detected. These operations are typically referenced by certain keyphrases and are thus detected using regular expressions during preprocessing, see Table 1.

<sup>5</sup> except attributes, which may also apply to the whole data cube.

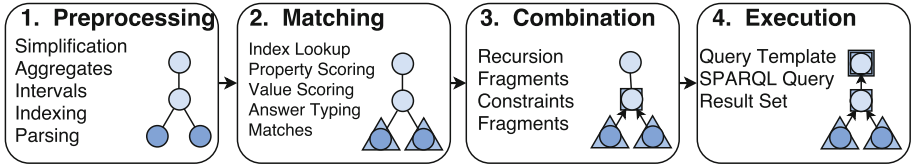


Fig. 4. The CubeQA pipeline

Those keyphrases, as well as the following entity recognition and disambiguation steps, are domain independent, so that CubeQA can be used with any set of RDCs. Nevertheless, the keyphrase to operation mapping can be extended for a specialized vocabulary to increase the recall on a particular domain.

Table 1. Data cube operations extracted in the preprocessing step

Element	SPARQL	Example phrase
Dice	Filter	2007 to 2008
Roll-up	Aggregate	In total
Slice	Filter	In 2008
Modifier	ORDER LIMIT	The 5 highest amounts

If a roll-up is not explicitly expressed in the question, a default aggregation is assumed for some answer types. A SPARQL aggregate rolls up all dimensions that are not bound by query variables. The roll-up aggregates *sum*, *arithmetic mean* and *count* are handled differently than *minimum* and *maximum*. The former aggregates return new values, so that they can be mapped to the SPARQL aggregation keywords SUM, AVG and COUNT. Minimum and maximum, however, choose a value among the existing ones, which allows identification of a cell and thus of a different component value. For example, in “Which company has the highest research spending?”, the user probably asks for the total, which can be achieved by a roll-up with addition followed by selecting the company with the highest sum.

*Dataset Detection.* CubeQA uses a dataset index that is initialized once with the set of available RDCs. It is implemented as a Lucene index with fields for the labels, comments and property labels of each RDC. The dataset name alone is not sufficient because questions (Definition 1) often do not refer to the dataset. For example, in Fig. 1, the dataset *finland-aid* is not mentioned but “the Philip-pines”, “2007” and “2008” will all be found by the index.

*Parsing.* At the end of the preprocessing step, a syntactic parse tree is generated for the modified question. This tree structure is traversed for matching nodes as described in Sect. 3.2.

### 3.2 Matching

Our query model starts with the whole RDC. The question is then split into phrases that are mapped to constraints, which exclude cells from the target datacube. In order to increase accuracy and resolve ambiguity, however, phrases of the question are first mapped to potential observation values in the matching step, based on the following definitions:

**Definition 2 (RDF Knowledge Base (KB)).** Let  $U$  be a set of URIs,  $U = I \cup P \cup C$ , where  $I$  is the instances,  $P$  the properties and  $C$  the classes. Let  $L \subset \Sigma^*$  be a set of literals, where  $\Sigma$  is the unicode alphabet. We define a KB  $k$  as a set of triples  $k \subseteq U \times P \times (U \cup L)$  (we disregard blanknodes). In the context of this work, the KB contains the triples of a set of RDCs.

**Definition 3 (Values).** The values of a component property  $p$  for an RDC  $c$  and a KB  $k$  are defined as:  $V_{p,c,k} = \{v | \exists o : \{(o, p, v), (o, qb:dataset, c), (o, rdf:type, qb:Observation)\} \subset k\}$ . The numerical values  $D_{p,c,k}$  are the values representing numbers, converted to double values. The temporal values  $T_{p,c,k}$  are the union of all values representing dates or years, converted to time intervals  $\tau(v) : T_{p,c,k} = \bigcup_{v \in V_{p,c,k}} \tau(v)$ .

Example: `:ph ∈ V{recipient-country,finland-aid,linkedspending}`.

**Definition 4 (Scorer).** A scorer for a component property  $p$  is represented formally by a partial function  $d_p : \Sigma^* \rightarrow (L \cup U) \times (0, 1]$ , Table 3 shows the three types of scorers, which are assigned to a property based on its range (see Table 2). Informally, the scorer of  $p$  returns the value with the closest match to a given phrase and its estimated probability.

**Table 2.** Component property scorer and answer type assignment. Integers include datatypes derived by constraint

Range	Scorer	Answer type
<code>xsd:integer</code>	Numeric	Countable
<code>xsd:float</code> , <code>xsd:double</code>	Numeric	Uncountable
<code>xsd:gYear</code> , <code>xsd:date</code>	Temporal	Temporal
No match	String	Entity

For the query template, the scorer results are converted to constraints. A naive approach is to create a value reference for the highest scored property of a phrase but this penalizes short phrases and suffers from ambiguity as it does not take the context of the phrase into account. Accordingly, CubeQA inserts an intermediate step: the *match*, which represents the possible references to component properties and their values.

**Table 3.** Definitions of the different types of scorers. The String Scorer uses both the Levenshtein distance, to quickly find candidates, and bigrams, for a more accurate scoring. All three scorers are partial functions whose result is undefined if no value is found. Only String Scorers return scores  $< 1$ , as they can correct for typographical errors in the input, while Numeric and Temporal Scorers are either undefined or return the input number, respectively time interval, with a score of 1. Type casting and conversion is omitted for brevity, e.g. in Fig. 1 the phrase “Philippines” is equated to the language tagged label “Philippines"@en and the phrase “2007” to the year `2007^^xsd:Year`.

Type	Scoring function $d_p(a)$
String	$(\arg \max_{b \in \beta_p(a)}(\text{ngram}(a, b)), \max_{b \in \beta_p(a)}(\text{ngram}(a, b)))$ , where $\beta_p(a) = \{b \in V_{p,c,k} \mid \text{lev}(a, b) \leq 2\}$ , using $n$ -gram similarity [15] ( $\text{ngram}, n=2$ ) and a Levenshtein-Automaton [21] ( $\text{lev}$ )
Numeric	$(a, 1)$ , if $a \in [\min(D_{p,c,k}), \max(D_{p,c,k})]$ , otherwise undefined
Temporal	$(a, 1)$ , if $T_{p,c,k} \cap \tau(a) \neq \emptyset$ , otherwise undefined,

**Definition 5 (Match).** A match  $m$  is represented formally by a pair  $(\rho, \gamma)$ , where  $\rho$  is the partial property scoring function,  $\rho : P \rightarrow (0, 1]$  and  $\gamma$  is the partial value scoring function,  $\gamma : P \rightarrow (L \cup U) \times (0, 1]$ .

### 3.3 Combining Matches to Constraints

The recursive combination process is used because (1) it favours longer phrases over shorter ones, giving increased coverage of the question and (2) it favours combination of phrases that are nearby in the question parse tree.

**Definition 6 (Constraint).** A constraint  $c$  is represented by a triple  $(G, \omega, \lambda)$ , where:

- $G$  is a set of SPARQL triple patterns and filters as defined in [20]
- $\omega$  is an optional order by modifier,  $\omega \in (\{\text{ASC}, \text{DESC}\} \times P) \cup \{\text{null}\}$
- $\lambda$  is an optional limit modifier,  $\lambda \in \mathbb{N}^+ \cup \{\text{null}\}$

Constraints are based on three different criteria:

1. A **Value Constraint** can be applied to any component property to confine it to an exact value, which can be a string, a number or a URI. It consists of a single SPARQL triple pattern:  $c_v = (\{(?o, p, v), (?o, \text{qb:DataSet}, d), (?o, a, \text{qb:Observation})\}, \text{null}, \text{null})$ , with  $p \in P$  and  $v \in L \cup U$ .
2. An **Interval Constraint** confines a value to a numeric or temporal interval. Accordingly, it can only apply to a component property whose range is an XSD numeric or temporal data type. It consists of a SPARQL triple pattern and a filter:  $c_i = (\{?o p ?x, \text{filter}(?x > x_1) \text{ AND } (?x < x_2)\}, \text{null}, \text{null})$ , with  $p \in P$ , the lower limit  $x_1$  and an upper limit  $x_2$ . Example:  $(\{?o : \text{refYear } ?y, \text{filter}(\text{year}(?y) \geq 2007 \text{ AND } \text{year}(?y) \leq 2008)\}, \text{null}, \text{null})$ . Closed or half-bounded intervals are defined analogously.

3. **Top/Bottom n Constraints** place an upper limit on the number of selected cells. They consist of three parts: The order (ascending or descending), the limit and the numeric component property whose values imply the order. Formally,  $c_t = (\emptyset, (\text{DESC}, p), n)$ ,  $c_b = (\emptyset, (\text{ASC}, p), n)$

To identify Value Constraints, each component property has a scorer (Definition 4), which tries to find a value similar to an input phrase. For example, “How much total aid was given to the regional FLEG programme in Mekong?”, could refer to a dimension “programme” with a value of “FLEG” and a dimension “region” with a value of “Mekong”. Equally possible would be a dataset description of “aid to Mekong” and a dimension “target” with a value of “FLEG programme”. The other types of constraints are matched in the preprocessing step because they are identified by certain keyphrases, such as “the 5 highest  $X$ ”.

In the example question, “How much did the Philippines receive in the year of 2007?”, there are multiple candidates for the number “2007”. The candidates can be disambiguated using the property scoring function of the “year” node by upward combination. As a *match* only holds the information collected from a single node in the question parse tree, there is additional information needed to represent a whole subtree. This extended representation is called a *fragment* and holds: (1) multiple matches collected in the recursive merge and (2) constraints extracted from fitting matches.

**Definition 7 (Fragment).** *Formally, a fragment  $f$  is a pair  $(M, R)$ , where  $M$  is a set of matches (see Definition 5) and  $R$  is a set of constraints.*

Algorithm 1 describes the process that combines the fragments of a list of child nodes into the fragment for their parent node.

### 3.4 Execution

Algorithm 1 combines the fragments of child nodes to create a fragment for the parent node. When this recursive process reaches the root node, Algorithm 2 transforms the fragment that results from the successive combination up to that point into a template (see Definition 8). All leftover value references whose property has not been referenced yet over a certain score threshold are transformed into additional Value Constraints. Other name and value references are discarded. All constraints, as well as the aggregate, if available, are then used to construct a SPARQL *select* query.

**Definition 8 (Template).** *A template  $t$  is a tuple  $(R, a, \alpha)$ , where  $R$  is as defined in Definition 7,  $a \in P$  is the answer property and  $\alpha$  is an optional aggregate function,  $\alpha(X) \in \{\min(X), \max(X), \sum_{x \in X} x, |X|, \sum_{x \in X} \frac{x}{|X|}, \text{null}\}$ .*

Next, the values of the answer properties are requested. If the set of answer properties is empty, the default measure of the dataset is used as an answer property to determine the properties. Executing the SPARQL query on the target knowledge base results in the set of answers requested by the user. The algorithm



**Algorithm 1.** Fragment Combination

---

**Input:** A list of fragments  $F = \{(M_1, R_1), \dots, (M_n, R_n)\}$ , with  $M_i = (\rho_i, \gamma_i)$   
**Output:** The combined fragment  $f = (M, R)$   
 $R \leftarrow \cup_{i=1}^n R_i$ ;  
 $P' \leftarrow (P \setminus \delta(R)) \cap \cup_{i=1}^n (\text{dom}(\rho_i) \cup \cup_{x \in \text{dom}(\gamma_i)} \pi_1(x))$ ;  
 $M \leftarrow \{M_1, \dots, M_n\}$ ;  
**foreach**  $p' \in P'$  **do**  
     $m_{\text{property}} \leftarrow \arg \max_{(\rho, \gamma) \in M'} \rho(p')$ ;  
     $m_{\text{value}} \leftarrow \arg \max_{(\rho, \gamma) \in M' \setminus \{m_{\text{property}}\}} \gamma(p')$ ;  
     $g \leftarrow ?o \ p \ \pi_2(m_{\text{value}})(p) \cdot ;$   
     $R \leftarrow R \cup \{\{g\}, \text{null}, \text{null}\}$ ;  
     $M \leftarrow M \setminus \{m_{\text{property}}, m_{\text{value}}\}$ ;  
**return**  $(M, R)$

---

$\pi_i(t)$  is the projection on the  $i$ -th element of the tuple  $t$ . The domain  $\text{dom}(f)$  is the set of all elements for which the (partial) function  $f$  is defined.  $\delta(R)$  is the set of all component properties that occur in at least one triple pattern in  $R$ .

implementation is publicly available under an open license at (link temporarily removed for anonymity). The algorithm implementation is publicly available under an open license at <https://github.com/AKSW/cubeqa>.

## 4 Evaluation

### 4.1 Research Questions

The goal of the evaluation was to obtain answers to the following research questions: Q1: Is CubeQA powerful enough to be practically useful on challenging statistical questions? Q2: Is there a tendency towards either high precision or recall? Q3: How do other RDCQA systems perform? Q4: What types of errors occur? How frequently are they? What are the reasons?

### 4.2 Experimental Setup and Benchmark

As there was no existing benchmark for RDCQA, we created a benchmark based on a statistical question corpus [12] and included it in the QALD-6<sup>6</sup> evaluation challenge. We used the existing corpus and significantly extended it to 100 questions, forming the training set QALD6T3-train. While keeping a similar structure, we adapted it to 50 of the, at this time, 983 financial datasets of LinkedSpending [13]. Chosen are the first 50 datasets that are manually confirmed as English from a list of all datasets. The list was sorted in descending order by their proportion of English labels (having at least 100 labels) as determined by automatic language detection. The datasets contain in total 158 dimensions, 81 measures, 176 attributes, 950149 observations and 16359532 triples (Table 4).

<sup>6</sup> <http://www.sc.cit-ec.uni-bielefeld.de/qald/>.

**Algorithm 2.** Fragment to Template Conversion.

---

**Input:** A fragment  $f = \{(M, R)\}$ , an optional aggregate function  $\alpha$  identified in preprocessing. The set of expected answer types  $E$  is defined in Table 4.  $\text{answerType}(p)$  is defined by Table 2.

**Output:** A template  $t = (R', a, \alpha')$

$R' = R;$

$P' \leftarrow (P \setminus A \setminus \delta(R));$

**foreach**  $(\rho, \gamma) \in M'$  **do**

$p_{\max} = \arg \max_{p \in (\text{dom}(\gamma) \cap P')} (\pi_2(\gamma(p)));$

**if**  $(p_{\max} \neq \text{null}) \wedge (\pi_2(\gamma(p_{\max})) \leq \theta)$  **then**

$R' \leftarrow R' \cup (\{?o p_{\max} \pi_1(\gamma(p_{\max}))\}, \text{null}, \text{null});$

$A \leftarrow \bigcup_{(\rho, \gamma) \in M} \text{dom}(\rho);$

$A' \leftarrow \{p \in A \mid \text{answerType}(p) \in E\};$

**if**  $A' = \emptyset$  **then**

$a \leftarrow \text{DEFAULT\_MEASURE};$

**else**

$a \leftarrow \arg \max_{(\rho, \gamma) \in M, p \in A'} \rho(p);$

**return**  $(R', a, \alpha)$

---

Using the same 50 datasets, the test set QALD6T3-test was created in the same way, but with slightly less complex questions. The questions, correct SPARQL queries, correct answers and our evaluation results are available online.<sup>7</sup>

QALD6T3 provides several challenges that are supported by the CubeQA algorithm. These are implied aggregations, intervals, implied or differently referenced measures and numerical values that are contained in several component properties. It also includes questions that require features not provided by CubeQA, such as SPARQL subqueries. The performance of CubeQA on the benchmark is measured as follows: Given  $C$  the correct set of resources and  $O$  the output of the algorithm, we define precision  $p = \frac{|C \cap O|}{|O|}$ , recall  $r = \frac{|C \cap O|}{|C|}$  and the  $F_1$ -score  $F_1 = 2 \frac{pr}{p+r}$ . The average global  $F_1$  score calculates  $p = 0$  for empty answers.

*Results.* Of the 100 questions, 82 resulted in a nonempty answer, with an average precision of 0.401, a recall of 0.324 and an  $F_1$  score of 0.392. Expected Answer Typing positively impacts the performance, as its removal results in a significant decrease in all three scores. Due to the cube index, many questions can be answered even if they do not specify their target dataset. With all the 50 datasets as candidates, the performance drops even more than without using answer typing, but the index chooses the dataset correctly for the majority of the questions (74 of 100). Answering the 100 questions on a PC with an Intel Core i5-3230M CPU, hosting both the SPARQL endpoint and the system implementation, took

<sup>7</sup> <https://github.com/AKSW/cubeqa/blob/master/benchmark/>.

**Table 4.** Mapping  $m$  of a question word to a set of expected answer types  $E$ , along with the frequency of each question word in the benchmark. When the question word is unknown or not found, or the unspecific “what” is used, all 6 answer types are possible.

Question word	Expected answer type	$f$
What	Uncountable, countable, count, temporal, location, entity	35
How much	Uncountable	33
Which	Temporal, location, entity	19
How many	Countable, count	6
When	Temporal	4
None, other	Uncountable, countable, count, temporal, location, entity	3
Total		100

87.45 s, 63.29 s and 63.33 s on three consecutive runs with preexisting index structures.<sup>8</sup> Table 5 shows the runtime distributions for the core tasks. Without preexisting index structures, the runs took 228.11 s, 228.77 s and 224.73 s, respectively.

### 4.3 Research Question Summary

A brief summary of the initial research questions is as follows: Q1: CubeQA is sufficiently powerful to be applied on challenging questions over statistical data and we believe it will be a strong baseline for future research. Q2: Precision is higher than recall, similar to general SQA systems, on QALD6T3-train but similar on QALD6T3-test. Q3: CubeQA achieves a global  $F_1$  score of 44 %, surpassed by QA<sup>3</sup> with 53 %, using a template-base algorithm. Q4: The most common cause for problems is ambiguity, followed by the lexical gap and query structure.

### 4.4 Discussion

*Comparison.* We believe that CubeQA will be a strong baseline in this new research subfield. As QALD6T3 was launched prior to submitting this publication to attract further research, two additional systems emerged: the yet unpublished QA<sup>3</sup> RDCQA system and the Sparklis [8] query builder (see Table 6). A query builder lets the user construct queries visually by selecting and combining SPARQL features and knowledge base resources. It enables users to create SPARQL queries and, if they build those queries correctly, achieves high accuracies. As such it occupies a middle ground, both in accuracy and usability, between RDCQA and manually creating SPARQL queries. QA<sup>3</sup> achieves a 9 %

<sup>8</sup> The higher initial time is assumed to be caused by cache warmup both in the system and the SPARQL endpoint.

**Table 5.** Runtimes and error causes.

task	$t(ms)$	error cause	$n$
SPARQL	25489	ambiguity	30
scoring	23326	lexical gap	18
index lookup	16070	query structure	17
parsing	5066	unknown	1
detectors	466	no error	34
answer typing	13	total errors	66
total	61673		

(a) Runtimes of the core tasks on QALD6T3- the different benchmark questions train with preexisting cache structures. SPARQL (at most one error per question), querying, scoring and index lookup are intersect-including the categories automatically and not all tasks are measured, so that the ically excluded before the evaluation. times do not add up to the total.

(b) Categorization of errors from

higher f-score than CubeQA but due to its purely template-base approach, it is unclear how it performs on open domain questions.

*Limitations.* CubeQA does not support query structures that require SPARQL subqueries, express negations of facts or unions of concepts. Ambiguities and lexical gaps are hard challenges that are not solved yet [14]. Nevertheless, they occur in almost every question and must be addressed by every SQA system to avoid massive penalties to precision and recall. Table 5 categorizes the different errors that prevented CubeQA from returning a correct result to a question.

The most common cause is ambiguity, which mainly results from a high number of similar resources or equal numbers in the observation values. In benchmark question 86, “How much was budgeted for general services for the Office of the President of Sierra Leone in 2013?”, two different properties contain the literal “Office of the President”. Because only the property value and not the property name is referenced, the algorithm cannot determine which property is correct. SQA systems like TBSL [26] resolve ambiguity by template scoring, so that the user chooses among the top  $n$ , where candidate combinations are ranked highest that maximize textual and semantic relatedness between the candidates [22]. But this approach is not applicable to RDCQA because of the RDC meta model, where component properties are not directly connected.

Instead, CubeQA relies on references consisting of a name reference as well as a value reference, as in “the year 2008”, where the name-value pair with the maximal score product of the name reference and the value reference is chosen. In case such a two-part reference does not occur, it is alleviated by giving temporal dimensions priority to others. For example, “2008” gets mapped to the year, if it exists, rather than the more improbable measurement value.

**Table 6.** QALD-6T3 performance of, indicated by average precision (over defined values), recall, and global  $F_1$ -score, rounded to 2 decimal places. The training set is used for evaluation, as it contains 100 harder questions compared to 50 in the test set. The correct target RDC was predefined for the training set, as the cube index is evaluated separately, with 74 of 100 correct choices.

Algorithm	Benchmark	$\varnothing p$	$\varnothing r$	$\varnothing F_1$
CubeQA	Train	0.40	0.32	0.32
QA <sup>3</sup>	Test	0.59	0.62	0.53
CubeQA	Test	0.49	0.41	0.44
Sparklis	Test	0.96	0.94	0.95

The second most common cause the lexical gap, where a reference could not be mapped to an entity due to the differences in surface forms. It is caused, among others, by different capitalization, typing errors, word transpositions (“extended amount”, “amount extended”) and different word forms (“committed”, “commitments”). Another issue with the lexical gap is that a measurement can be referenced using a quantity reference (“amount”), a unit (“How many dollars are given”), or the type (“aid”), of which only the first one guarantees a match. Thus, CubeQA matches the range of a property as well as a its label. The RDC vocabulary provides `sdmx-attribute:unitMeasure` to specify units of measurement, but it does not support multiple measures so that the fallback has the same effect. In case of future vocabulary specification updates, we plan to integrate measurement units into our approach.

All of those, except typing errors, occur in the benchmark. As these mentioned causes occur in document retrieval and Web search as well, full text indexes have been developed that robustly handle those problems. The employed Lucene index cannot overcome the lexical gap in some cases, which are not recognized by the stemmer and where the edit distance is too large for the fuzzy index as well. Sometimes a concept is implicitly required but there is no explicit reference at all. Implicit references are part of future work and include aggregates.

## 5 Related Work

SQA in general is an active and established area of research with too many systems to cite individually but *surveys* [2, 6, 9, 14, 17] give a qualitative overview of the field. Also, *evaluation campaigns* present quantitative comparisons with benchmarks on either general tasks like QALD [5] or specialized tasks like BioASQ [25]. RDCQA has not existed until recently, but non-semantic QA is implemented by Wolfram—Alpha, which queries several structured sources using the computational platform *Mathematica* [27], but the source code and algorithm are not published. We inspired the RDCQA sub-field by discussing RDCs in relation to SQA and by categorizing of a statistical question corpus [12]. Next, we

developed CubeQA and QALD6-T3 to stimulate further research, which led to the development of QA<sup>3</sup> and Sparklis (see Sect. 4.4).

CubeQA uses time intervals for handling dates, similar to the system in [24] that uses the *Clinical Narrative Temporal Relation Ontology (CNTRO)* to incorporate the time dimension in answering clinical questions. The ontology is based on Allen’s Interval Based Temporal Logic [1] but it represents time points as well. The framework includes a reasoner for time inference, for example based on the transitivity of the *before* and *after* relations. The time dimension is used there to identify the direction of possible causality between different events.

Furthermore, CubeQA generates query templates recursively, which is similarly employed by Intui2 [7], which uses DBpedia and is based on *synfragments*, minimal parse subtrees of a question, that are combined based on syntactic and semantic characteristics to create the final query.

The motivation to develop RDCQA algorithms and their benefit rises with the quantity, quality and significance of available RDCs. On the flipside, we expect that the emergence and improvement of RDCQA algorithms increases the value of RDCs. Because of this interdependence, we summarize efforts to improve the quality of, create and publish RDF in general and RDCs in particular: RDCs are usually created by transforming databases or other structured data sources using either custom software or mapping languages like R2RML<sup>9</sup> and SML [23]. Eurostat—Linked Data<sup>10</sup> transforms tabular data of Eurostat<sup>11</sup>, providing statistics for comparing the European countries and regions. Linked-Spending [13] uses the OpenSpending JSON API to provide finance data from countries around the world. The most widely used statistical data format is SDMX (Statistical Data and Metadata eXchange), which can be transformed to RDCs using SDMX-ML [4]. A systematic review of Linked Data quality [28] provides a qualitative analysis over established approaches, tools and metrics.

## 6 Conclusions and Future Work

We introduce RDCQA and design the CubeQA algorithm, provide a benchmark based on real data, and evaluate the results. In future work, we plan to continue contributing to the yearly QALD evaluation campaign by providing progressively more challenging benchmarks. The next iteration of CubeQA will answer questions that require the consolidation of several RDCs. We will also investigate how to integrate RDCQA techniques with SQA frameworks, such as OpenQA [18], so that all-purpose systems can also answer questions on RDCs. On the flipside, we also plan to integrate general SQA into RDCQA, to answer questions on RDCs that require world knowledge. We also identified the following improvements:

- Implement selection filters as logical formula of constraints instead of flat sets, including negations and unions.

<sup>9</sup> <https://www.w3.org/TR/r2rml>.

<sup>10</sup> <http://eurostat.linked-statistics.org/>.

<sup>11</sup> <http://ec.europa.eu/eurostat>.

- Support SPARQL subqueries to handle nested information dependencies.
- Support languages other than English using language detection components as well as fitting parsers, indexes and preprocessing templates.
- Incorporate measurement units if the RDC vocabulary adds support for them for multiple measures. For elaborate phrase patterns, like “How many people live in” for “population”, there are pattern libraries like BOA [10] which need to be adapted to statistical data by retraining on a comprehensive statistical question corpus.

Overall, we believe to have opened a novel research subfield within SQA, which will increase in importance due to the rise of both the volume of statistical data and the usage of QA approaches in everyday life.

**Acknowledgment.** This work was supported by a grant from the EU H2020 Framework Programme provided for the project HOBbit (GA no. 688227).

## References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. *Commun. ACM* **26**(11), 832–843 (1983)
2. Athenikos, S., Han, H.: Biomedical question answering: a survey. *Comput. Meth. Programs Biomed.* **99**(1), 1–24 (2010)
3. Berners-Lee, T.: Linked Data-Design issues, W3C design issue (2009)
4. Capadisli, S., Auer, S., Ngonga Ngomo, A.C.: Linked SDMX data. *Semant. Web J.* **6**(2), 105–112 (2015)
5. Cimiano, P., Lopez, V., Unger, C., Cabrio, E., Ngonga Ngomo, A.-C., Walter, S.: Multilingual question answering over linked data (QALD-3): lab overview. In: Forner, P., Müller, H., Paredes, R., Rosso, P., Stein, B. (eds.) *CLEF 2013. LNCS*, vol. 8138, pp. 321–332. Springer, Heidelberg (2013)
6. Cimiano, P., Minock, M.: Natural language interfaces: what is the problem? – A data-driven quantitative analysis. In: Horacek, H., Métais, E., Muñoz, R., Wolska, M. (eds.) *NLDB 2009. LNCS*, vol. 5723, pp. 192–206. Springer, Heidelberg (2010)
7. Dima, C.: Intui2: A prototype system for question answering over linked data. In: Forner, P., Navigli, R., Tufis, D. (eds.) *Question Answering over Linked Data (QALD-3), CLEF 2013 Evaluation Labs and Workshop, Online Working Notes* (2013)
8. Ferré, S.: Sparklis: an expressive query builder for SPARQL endpoints with guidance in natural language. *Semant. Web J.* (2015)
9. Freitas, A., Curry, E., Oliveira, J., O’Riain, S.: Querying heterogeneous datasets on the Linked Data Web: challenges, approaches, and trends. *IEEE Internet Comput.* **16**(1), 24–33 (2012)
10. Gerber, D., Ngomo, A.-C.N.: Extracting multilingual natural-language patterns for RDF predicates. In: ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d’Acquin, M., Nikolov, A., Aussenac-Gilles, N., Hernandez, N. (eds.) *EKAW 2012. LNCS*, vol. 7603, pp. 87–96. Springer, Heidelberg (2012)
11. Hirschman, L., Gaizauskas, R.: Natural language question answering: the view from here. *Nat. Lang. Eng.* **7**(4), 275–300 (2001)

12. Höffner, K., Lehmann, J.: Towards question answering on statistical linked data. In: Proceedings of the 10th International Conference on Semantic Systems, SEM 2014, pp. 61–64. ACM (2014)
13. Höffner, K., Martin, M., Lehmann, J.: LinkedSpending: OpenSpending becomes Linked Open Data. *Semant. Web J.* **7**, 95–104 (2015)
14. Höffner, K., Walter, S., Marx, E., Usbeck, R., Lehmann, J., Ngonga Ngomo, A.C.: Survey on challenges of Question Answering in the Semantic Web. *Semant. Web J.* (2016, submitted)
15. Kondrak, G.: *N*-Gram similarity and distance. In: Consens, M.P., Navarro, G. (eds.) SPIRE 2005. LNCS, vol. 3772, pp. 115–126. Springer, Heidelberg (2005)
16. Lehmann, J., Bizer, C., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - a crystallization point for the web of data. *J. Web Semant.* **7**(3), 154–165 (2009)
17. Lopez, V., Uren, V., Sabou, M., Motta, E.: Is question answering fit for the semantic web? A survey. *Semant. Web J.* **2**(2), 125–155 (2011)
18. Marx, E., Usbeck, R., Ngonga Ngomo, A.C., Höffner, K., Lehmann, J., Auer, S.: Towards an open Question Answering architecture. In: SEMANTiCS 2014 (2014)
19. Piotrowski, S.J., Van Ryzin, G.G.: Citizen attitudes toward transparency in local government. *Am. Rev. public Adm.* **37**(3), 306–323 (2007)
20. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation (2008)
21. Schulz, K.U., Mihov, S.: Fast string correction with Levenshtein automata. *Int. J. Doc. Anal. Recogn.* **5**(1), 67–85 (2002)
22. Shekarpour, S., Ngonga Ngomo, A.C., Auer, S.: Query segmentation and resource disambiguation leveraging background knowledge. In: Proceedings of WoLE Workshop (2012)
23. Stadler, C., Unbehauen, J., Westphal, P., Sherif, M.A., Lehmann, J.: Simplified RDB2RDF mapping. In: Proceedings of the 8th Workshop on Linked Data on the Web (LDOW2015), Florence, Italy (2015)
24. Tao, C., Solbrig, H.R., Sharma, D.K., Wei, W.-Q., Savova, G.K., Chute, C.G.: Time-oriented question answering from clinical narratives using semantic-web techniques. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part II. LNCS, vol. 6497, pp. 241–256. Springer, Heidelberg (2010)
25. Tsatsaronis, G., Schroeder, M., Paliouras, G., Almirantis, Y., Androutsopoulos, I., Gaussier, E., Gallinari, P., Artieres, T., Alvers, M.R., Zschunke, M., et al.: BioASQ: a challenge on large-scale biomedical semantic indexing and Question Answering. In: 2012 AAAI Fall Symposium Series (2012)
26. Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.C., Gerber, D., Cimiano, P.: Template-based Question Answering over RDF data. In: Proceedings of the 21st International Conference on World Wide Web, pp. 639–648 (2012)
27. Wolfram, S.: *The Mathematica Book*, vol. 100, pp. 61820–67237. Cambridge University Press and Wolfram Research Inc., New York (2000)
28. Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S.: Quality assessment for Linked Data: a survey. *Semant. Web J.* **7**, 63–93 (2015)