Semantic Web 0 (0) 1
IOS Press

Towards a Question Answering System over the Semantic Web

Dennis Diefenbach [a], Andreas Both [b], Kamal Singh [a] and Pierre Maret [a]
[a] *Université de Lyon, CNRS UMR 5516 Laboratoire Hubert Curien, France*
[b] *DATEV eG, Germany*

Abstract. With the development of the Semantic Web, a lot of new structured data has become available on the Web in the form of knowledge bases (KBs). Making this valuable data accessible and usable for end-users is one of the main goals of question answering (QA) over KBs. Most current QA systems query one KB, in one language (namely English). The existing approaches are not designed to be easily adaptable to new KBs and languages.
We first introduce a new approach for translating natural language questions to SPARQL queries. It is able to query several KBs simultaneously, in different languages, and can easily be ported to other KBs and languages. In our evaluation, the impact of our approach is proven using 5 different well-known and large KBs: Wikidata, DBpedia, MusicBrainz, DBLP and Freebase as well as 5 different languages namely English, German, French, Italian and Spanish. Second, we show how we integrated our approach, to make it easily accessible by the research community and by end-users.
To summarize, we provide a conceptional solution for multilingual, KB-agnostic question answering over the Semantic Web. The provided first approximation validates this concept.

Keywords: Question Answering, Multilinguality, Portability, QALD, SimpleQuestions

1. Introduction

Question answering (QA) is a research field in computer science that started in the sixties [30]. In the Semantic Web, a lot of new structured data has become available in the form of knowledge bases (KBs). Nowadays, there are KBs about media, publications, geography, life-science and more[1]. The core purpose of a QA system over KBs is to retrieve the desired information from one or many KBs, using natural language questions. This is generally addressed by translating a natural language question to a SPARQL query. Current research does not address the challenge of multilingual, KB-agnostic QA for both full and keyword questions (Table 1).

There are multiple reasons for that. Many QA approaches rely on language-specific tools (NLP tools), e.g., SemGraphQA [4], gAnswer [53] and Xser [48]. Therefore, it is difficult or impossible to port them to a language-agnostic system. Additionally, many approaches make particular assumptions on how the knowledge is modelled in a given KB (generally referred to as "structural gap" [12]). This is the case of AskNow [17] and DEANNA [49].
There are also approaches which are difficult to port to new languages or KBs because they need a lot of training data which is difficult and expensive to create. This is for example the case of Bordes et al. [6]. Finally there are approaches where it was not proven that they scale well. This is for example the case of SINA [39].
In this paper, we present an algorithm that addresses all of the above drawbacks and that can compete, in terms of F-measure, with many existing approaches.
This publication is organized as follows. In Section 2, we present related works. In Section 3 and 4, we describe the algorithm providing the foundations of our approach. In Section 5, we provide the results of our evaluation over different benchmarks. In Section 6, we show how we implemented our algorithm as a service so that it is easily accessible to the research community, and how we extended a series of existing services

[1] http://lod-cloud.net

QA system	Lang	KBs	Type
gAnswer [53] (QALD-3 Winner)	en	DBpedia	full
Xser [48] (QALD-4 & 5 Winner)	en	DBpedia	full
UTQA [36]	en, es, fs	DBpedia	full
Jain [27] (WebQuestions Winner)	en	Freebase	full
Lukovnikov [31] (SimpleQuestions Winner)	en	Freebase	full
Ask Platypus (https://askplatyp.us)	en	Wikidata	full
WDAqua-core1	en, fr, de, it, es	Wikidata, DBpedia, Freebase, DBLP, MusicBrainz	full & key

Table 1

Selection of QA systems evaluated over the most popular benchmarks. We indicated their capabilities with respect to multilingual questions, different KBs and different typologies of questions (full = "well-formulated natural language questions", key = "keyword questions").

so that our approach can be directly used by end-users. We conclude with Section 7.

2. Related work

In the context of QA, a large number of systems have been developed in the last years. For a complete overview, we refer to [12]. Most of them were evaluated on one of the following three popular benchmarks: WebQuestions [5], SimpleQuestions [6] and QALD[2].

WebQuestions contains 5810 questions that can be answered by one reified statement. SimpleQuestions contains 108,442 questions that can be answered using a single, binary-relation. The QALD challenge versions include more complex questions than the previous ones, and contain between 100 and 450 questions, and are therefore, compared to the other, small datasets.

The high number of questions of WebQuestions and SimpleQuestions led to many supervised-learning approaches for QA. Especially deep learning approaches became very popular in the recent years like Bordes et al. [6] and Zhang et al. [51]. The main drawback of these approaches is the training data itself. Creating a new training dataset for a new language or a new KB might be very expensive. For example, Berant et al. [5], report that they spent several thousands of dollars for the creation of WebQuestions using Amazon Mechanical Turk. The problem of adapting these approaches to new dataset and languages can also be seen by the fact that all these systems work only for English questions over Freebase.

A list of the QA systems that were evaluated with QALD-3, QALD-4, QALD-5, QALD-6 can be found in Table 3. According to [12] less than 10% of the approaches were applied to more than one language and 5% to more than one KB. The reason is the heavy use of NLP tools or NL features like in Xser [48], gAnswer [53] or QuerioDali [29].

The problem of QA in English over MusicBrainz[3] was proposed in QALD-1, in the year 2011. Two QA systems tackled this problem. Since then the MusicBrainz KB[4] completely changed. We are not aware of any QA system over DBLP[5].

In summary, most QA systems work only in English and over one KB. Multilinguality is poorly addressed while portability is not addressed at all. The few systems that address multilinguality rely on syntactic parsing techniques [1][24][36].

The fact that QA systems often reuse existing techniques and need several services to be exposed to the end-user, leads to the idea of developing QA systems in a modular way. At least four frameworks tried to achieve this goal: QALL-ME [19], openQA [32], the Open Knowledge Base and Question-Answering (OKBQA) challenge[6] and Qanary [7, 14, 40]. We integrated our system as a Qanary QA component called WDAqua-core1. We choose Qanary for two reasons. First, it offers a series of off-the-shelf services related to QA systems and second, it allows to freely configure a QA system based on existing QA components.

3. Approach for QA over Knowledge Bases

In this section, we present our multilingual, KB-agnostic approach for QA. It is based on the observation that many questions can be understood from

[2] http://www.sc.cit-ec.uni-bielefeld.de/qald/

[3] https://musicbrainz.org

[4] https://github.com/LinkedBrainz/MusicBrainz-R2RML

[5] http://dblp.uni-trier.de

[6] http://www.okbqa.org/

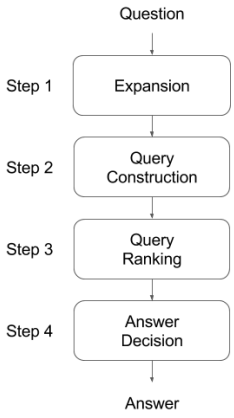


Fig. 1. Conceptual overview of the approach

the semantics of the words in the question while the syntax of the question has less importance. For example, consider the question "Give me actors born in Berlin". This question can be reformulated in many ways like "In Berlin were born which actors?" or as a keyword question "Berlin, actors, born in". In this case by knowing the semantics of the words "Berlin", "actors", "born", we are able to deduce the intention of the user. This holds for many questions, i.e. they can be correctly interpreted without considering the syntax as the semantics of the words is sufficient for them. Taking advantage of this observation is the main idea of our approach. The KB encodes the semantics of the words and it can tell what is the most probable interpretation of the question (w.r.t. the knowledge model described by the KB).
Our approach is decomposed in 4 steps: question expansion, query construction, query ranking and response decision. A conceptual overview is given in Figure 1. In the following, the processing steps are described. As a running example, we consider the question "Give me philosophers born in Saint-Etienne". For the sake of simplicity, we use DBpedia as KB to answer the question. However, it is important to recognize that no assumptions either about the language or the KB are made. Hence, even the processing of the running example is language- and KB-agnostic.

3.1. Expansion

Following a recent survey [12], we call a lexicalization, a name of an entity, a property or a class. For example, "first man on the moon" and "Neil Armstrong" are both lexicalizations of `dbr:Neil_Armstrong`. In this step, we want to identify all Internationalized Resource Identifiers (IRIs) of entities, properties and classes, which the question could refer to. To achieve this, we use the following rules:

- All IRIs are searched whose lexicalization (up to stemming) is a word n-gram (up to stemming) in the question.
- If an n-gram is a stop word (like "is", "are", "of", "give", ...), then we exclude the IRIs associated to it. This is due to the observation that the semantics are important to understand a question and the fact that stop words do not carry a lot of semantics. Moreover, by removing the stop words the time needed in the next step is decreased.

An example is given in Table 2. The stop words and the lexicalizations used for the different languages and KBs are described in Section 5.1. In this part, we used the well-known Apache Lucene[7] technology which allows fast retrieval, while providing a small disk and memory footprint.

3.2. Query construction

In this step, we construct a set of queries that represent possible interpretations of the given question within the given KB. Therefore, we heavily utilize the semantics encoded into the particular KB. We start with a set R of IRIs from the previous step. The goal is to construct all possible queries containing the IRIs in R which give a non-empty result-set. Let V be the set of variables. Based on the complexity of the questions in current benchmarks, we restrict our approach to queries satisfying 4 patterns:

```
SELECT / ASK var
 WHERE { s1 s2 s3 . }


SELECT / ASK var
WHERE { s1 s2 s3 .
        s4 s5 s6 . }
```

with

$$s1, \ldots, s6 \in R \cup V$$

and

$$var \in \{s1, \ldots, s6\} \cap V$$

[7] https://lucene.apache.org

n	start	end	n-gram	resource
1	2	3	philosophers	dbrc:Philosophes
2	2	3	philosophers	dbr:Philosophes
3	2	3	philosophers	dbo:Philosopher
4	2	3	philosophers	dbrc:Philosophers
5	2	3	philosophers	dbr:Philosopher
6	2	3	philosophers	dbr:Philosophy
7	2	3	philosophers	dbo:philosophicalSchool
8	3	4	born	dbr:Born,_Netherlands
9	3	4	born	dbr:Born_(crater)
10	3	4	born	dbr:Born_auf_dem_Dar?
11	3	4	born	dbr:Born,_Saxony-Anhalt
⋮				
42	3	4	born	dbp:bornAs
43	3	4	born	dbo:birthDate
44	3	4	born	dbo:birthName
45	3	4	born	dbp:bornDay
46	3	4	born	dbp:bornYear
47	3	4	born	dbp:bornDate
48	3	5	born in	dbp:bornIn
49	3	5	born in	dbo:birthPlace
50	3	5	born in	dbo:hometown

n	start	end	n-gram	resource
52	5	6	saint	dbr:SAINT_(software)
53	5	6	saint	dbr:Saint
54	5	6	saint	dbr:Boxers_and_Saints
55	5	6	saint	dbr:Utah_Saints
56	5	6	saint	dbr:Saints,_Luton
57	5	6	saint	dbr:Baba_Brooks
58	5	6	saint	dbr:Battle_of_the_Saintes
59	5	6	saint	dbr:New_York_Saints
⋮				
106	5	6	saint	dbp:saintPatron
107	5	6	saint	dbp:saintsDraft
108	5	6	saint	dbp:saintsSince
109	5	6	saint	dbo:patronSaint
110	5	6	saint	dbp:saintsCollege
111	5	6	saint	dbp:patronSaintOf
112	5	6	saint	dbp:patronSaint(s)
113	5	6	saint	dbp:patronSaint'sDay
114	5	7	saint etienne	dbr:Saint_Etienne_(band)
115	5	7	saint etienne	dbr:Saint_Etienne
116	5	7	saint etienne	dbr:Saint-Étienne
117	6	7	etienne	dbr:Étienne

Table 2

Expansion step for the question "Give me philosophers born in Saint Étienne". The first column enumerates the candidates that were found. Here, 117 possible entities, properties and classes were found from the question. The second, third and fourth columns indicate the position of the n-gram in the question and the n-gram itself. The last column is for the associated IRI. Note that many possible meanings are considered: line 9 says that "born" may refer to a crater, line 52 that "saint" may refer to a software and line 114 that the string "Saint Étienne" may refer to a band.

These correspond to all queries containing one or two triple patterns that can be created starting from the IRIs in R. Moreover, for entity linking, we add the following two patterns:

```
SELECT ?x
WHERE { VALUES  ?x {iri} . }
```

```
SELECT ?x
WHERE { VALUES  ?x {iri} .
  iri ?p iri1 . }
```

with iri, iri1 $\in$ R. These correspond to all queries returning directly one of the IRIs in R with possibly one additional triple.
Note that these last queries just give back directly an entity and should be generated for a question like: "What is Apple Company?" or "Who is Marie Curie?". An example of generated queries is given in Figure 2.
The main challenge is the efficient construction of these SPARQL queries. The main idea is to perform in the KB graph a breadth-first search of depth 2 starting from every IRI in R. While exploring the KB for all IRIs $r_j \in R$ (where $r_j \neq r_i$) the distance d_{r_i,r_j} between two resources is stored. These numbers are used when constructing the queries shown above. For a detailed algorithm of the query construction phase, please see Section 4. Concluding, in this section, we computed a set of possible SPARQL queries (candidates). They are driven by the lexicalizations computed in Section 3.1 and represent the possible intentions expressed by the question of the user.

3.3. Ranking

Now the computed candidates need to be ordered by their probability of answering the question correctly. Hence, we rank them based on the following features:

- Number of the words in the question which are covered by the query. For example, the first query in Figure 2 is covering two words ("Saint" and "born", where "born" is covered by the property dbo:hometown).
- The edit distance of the label of the resource and the word it is associated to. For example, the edit

```
– SELECT DISTINCT ?y WHERE {
    dbr:Saint_(song) ?p ?x .
    ?x  dbo:hometown  ?y . }


– SELECT ?x {
  VALUES ?x { dbr:Saint_Etienne_(band) } }


– SELECT DISTINCT ?y WHERE {
    ?x dbo:birthPlace dbr:Saint-Etienne .
    ?x dbo:birthDate  ?y . }


– SELECT DISTINCT ?y WHERE {
    ?x ?p dbr:Philosophy  .
    ?x dbo:birthDate  ?y . }
```

Fig. 2. Some of the 395 queries constructed for the question "Give me philosophers born in Saint Etienne.". Note that all queries could be semantically related to the question. The second one is returning "Saint-Etienne" as a band, the third one the birth date of people born in the city of "Saint-Etienne" and the forth one the birth date of persons related to philosophy.

distance between the label of `dbp:bornYear` (which is "born year") and the word "born" is 5.
- The sum of the relevance of the resources, (e.g. the number of inlinks and the number of outlinks of a resource). This is a knowledge base independent choice, but it is also possible to use a specific score for a KB (like page-rank).
- The number of variables in the query.
- The number of triples in the query.

If no training data is available, then we rank the queries using a linear combination of the above 5 features, where the weights are determined manually. Otherwise we assume a training dataset of questions together with the corresponding answers set, which can be used to calculate the F-measure for each of the SPARQL query candidates. As a ranking objective, we want to order the SPARQL query candidates in descending order with respect to the F-measure. In our exemplary implementation we rank the queries using RankLib[8] with Coordinate Ascent [33]. At test time the learned model is used to rank the queries, the top-ranked query is executed against a SPARQL endpoint, and the result is computed. An example is given in Figure 3. Note that, we do not use syntactic features. However, it is possible to use them to further improve the ranking.

[8]https://sourceforge.net/p/lemur/wiki/RankLib/

3.4. Answer Decision

The computations in the previous section lead to a list of ranked SPARQL queries candidates representing our possible interpretations of the user's intentions. We could directly give back the result of first ranked query from the previous step. This will (nearly) always generate an answer. However, there are situations where no suitable interpretation is generated for the question, or where the question is not answerable over the given KB. To determine if such a situation occurred, we add an additional step in which we decide if the result-set of the first query should be returned or if the system should not give back any result. This corresponds to a binary classification problem. We train a model based on logistic regression. We use a training set consisting of SPARQL queries and two labels equal to True or False. True indicates if the F-score of the SPARQL query is greater than a threshold θ_1 or false otherwise. Once the model is trained, it can compute a confidence score $p_Q \in [0,1]$ for a query Q. In our exemplary implementation we assume a correctly ordered list of SPARQL query candidates computed in Section 3.3. Hence, it only needs to be checked whether $p_{Q_1} \geqslant \theta_2$ is true for the first ranked query Q_1 of the SPARQL query candidates, or otherwise it is assumed that the whole candidate list does not reflect the user's intention. Hence, we refuse to answer the question. We answer the question if it is above a threshold θ_2 otherwise we do not answer it. Note that p_Q can be interpreted as the confidence that the QA system has in the generated SPARQL query Q, i.e. in the generated answer.

3.5. Multiple KBs

Note that the approach can also be extended, as it is, to multiple KBs. In the query expansion step, one has just to take in consideration the labels of all KBs. In the query construction step, one can consider multiple KBs as one graph having multiple unconnected components. The query ranking and answer decision step are literally the same.

3.6. Discussion

Overall, we follow a combinatorial approach with efficient pruning, that relies on the semantics encoded

1.
```
SELECT DISTINCT ?x WHERE {
  ?x dbp:birthPlace dbr:Saint-Etienne .
  ?x rdf:type dbo:Philosopher . }
```

2.
```
SELECT DISTINCT ?y WHERE {
  ?x  dbo:birthPlace dbr:Saint-Etienne .
  ?x  dbo:philosophicalSchool ?y . }
```

3.
```
SELECT DISTINCT ?x WHERE {
 ?x  dbp:birthPlace dbr:Saint-Etienne . }
```

4.
```
SELECT DISTINCT ?x WHERE {
 ?x  dbo:hometown dbr:Saint-Etienne . }
```

Fig. 3. The top 4 generated queries for the question "Give me philosophers born in Saint Étienne.". (1) is the query that best matches the question; (2) gives philosophical schools of people born in Saint-Étienne; (3)(4) give people born in Saint-Étienne or that live in Saint-Étienne. The order can be seen as a decreasing approximation to what was asked.

in the underlying KB.
In the following, we want to emphasize the advantages of this approach using some examples.

- **Joint disambiguation of entities and relations:** For example, for interpreting the question "How many inhabitants has Paris?" between the hundreds of different meanings of "Paris" and "inhabitants" the top ranked queries contain the resources called "Paris" which are cities, and the property indicating the population, because only these make sense semantically.
- **Portability to different KBs:** One problem in QA over KBs is the semantic gap, i.e. the difference between how we think that the knowledge is encoded in the KB and how it actually is. For example, in our approach, for the question "What is the capital of France?", we generate the query

  ```
  SELECT ?x WHERE {
    dbr:France dbp:capital ?x .
  }
  ```

  which probably most users would have expected, but also the query

  ```
  SELECT ?x {
    VALUES ?x {
      dbr:List_of_capitals_of_France
    }
  }
  ```

  which refers to an overview article in Wikipedia about the capitals of France and that most of the users would probably not expect. This important feature allows to port the approach to different KBs while it is independent of how the knowledge is encoded.
- **Ability to bridge over implicit relations:** We are able to bridge over implicit relations. For example, given "Give me German mathematicians" the following query is computed:

  ```
  SELECT DISTINCT ?x WHERE {
    ?x  ?p1 dbr:Mathematician .
    ?x  ?p2 dbr:Germany .
  }
  ```

  Here `?p1` is:

  - `dbo:field`
  - `dbo:occupation`,
  - `dbo:profession`

  and `?p2` is:

  - `dbo:nationality`,
  - `dbo:birthPlace`,
  - `dbo:deathPlace`,
  - `dbo:residence`.

  Note that all these properties could be intended for the given question, even if `dbo:deathPlace` could be seen as an over-generalization.
- **Easy to port to new languages:** The only parts where the language is relevant are the stop word removal and stemming. Since these are very easy to adapt to new languages, one can port the approach easily to other languages.
- **Permanent system refinement:** It is possible to improve the system over time. The system generates multiple queries. This fact can be used to easily create new training datasets as is shown in [13]. Using these datasets one can refine the ranker to perform better on the asked questions.
- **System robust to malformed questions and keyword questions:** We are not using part-of-speech tagging or dependency parsers in the approach which makes it very robust to malformed questions. For this reason, keyword questions are also supported.

A disadvantage of our exemplary implementation is that the identification of relations relies on a dictionary. Note that, non-dictionary based methods follow one of the following strategies. Either they try to learn ways to express the relation from big training corpora (like in [6]), s.t. the problem is shifted to create suitable training sets. Or text corpora are used to either extract lexicalizations for properties (like in [5]) or learn word-embeddings (like in [24]). Hence, possible improvements might be applied to this task in the future.

4. Fast candidate generation

In this section, we explain how the SPARQL queries described in Section 3.2 can be constructed efficiently.

Let R be a set of resources. We consider the KB as a directed labeled graph G:

Definition 1. *(Graph) A directed labeled graph is an ordered pair $G = (V, E, f)$, such that:*

- *V is a non-empty set, called the vertex set;*
- *E is a set, called edge set, such that $E \subset \{(v, w) : v, w \in V\}$, i.e. a subset of the pairs of V;*
- *For a set L called labeled set, f is a function $f : E \to L$, i.e. a function that assigns to each edge a label $p \in L$. We indicate an edge with label p as $e = (v, p, w)$.*

To compute the pairwise distance in G between every resource in R, we do a breadth-first search from every resource in R in an undirected way (i.e. we traverse the graph in both directions).

We define a distance function d as follows. Assume we start from a vertex r and find the following two edges $e_1 = (r, p_1, r_1)$, $e_2 = (r_1, p_2, r_2)$. We say that $d_{r,p_1} = 1, d_{r,r_1} = 2, d_{r,p_2} = 3$ and so on. When an edge is traversed in the opposite direction, we add a minus sign. For example, given the edges $e_1 = (r, p_1, r_1)$ and $e_2 = (r_2, p_2, r_1)$, we say $d_{r,p_2} = -3$. For a vertex or edge r, and a variable x we artificially set $d_{r,x}$ to be any possible integer number. Moreover, we set $d_{x,y} = d_{y,x}$ for any x, y. The algorithm to compute these numbers can be found in Algorithm 1.

The algorithm of our exemplary implementation simply traverses the graph starting from the nodes in R in a breadth-first search manner and keeps track of the distances as defined above. The breadth-first search is done by using HDT [18] as an indexing structure[9]. Note that HDT was originally developed as an exchange format for RDF files that is queryable. A rarely mentioned feature of HDT is that it is perfectly suitable for performing breadth-first search operations over RDF data. In HDT, the RDF graph is stored as an adjacency list which is an ideal data structure for breadth-first search operations. This is not the case for traditional triple-stores. The use of HDT at this point is key for two reasons, (1) the performance of the

[9] https://www.w3.org/Submission/2011/03/

Data: Graph $G = (V, E, f)$ and a set R of edges and labels
Result: The pairwise distance between elements in R

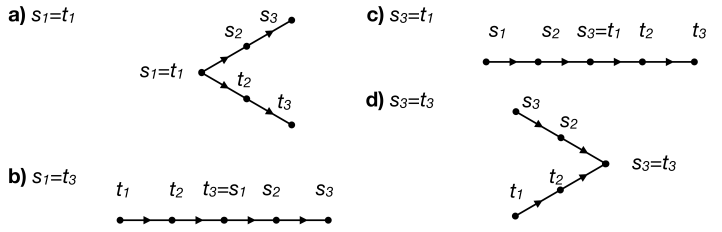
for $r \in R \cap V$ **do**
  for e_1=$(r,p_1,r_1) \in E$ **do**
    if $p_1 \in R$ **then** $d_{r,p_1} = 1$; **if** $r_1 \in R$ **then** $d_{r,l_1} = 2$
    for $(e_2 = (r_1, p_2, r_2) \in E)$ **do**
      if $p_2 \in R$ **then** $d_{r,p_2} = 3$; **if** $r_2 \in R$ **then** $d_{r,2_2} = 4$;
      if $p_1, p_2 \in R$ **then** $d_{p_1,p_2} = 2$: **if** $p_1, r_2 \in R$ **then** $d_{p_1,r_2} = 3$
    end
    for $(e_2 = (r_2, p_2, r_1) \in E)$ **do**
      if $p_2 \in R$ **then** $d_{r,p_2} = -3$; **if** $r_2 \in R$ **then** $d_{r,r_2} = -4$
      if $p_1, p_2 \in R$ **then** $d_{p_1,p_2} = -2$; **if** $p_1, p_2 \in R$ **then** $d_{p_1,2_2} = -3$
    end
  end
  for e_1=$(r_1,p_1,r) \in E$ **do**
    if $p_1 \in R$ **then** $d_{r,p_1} = -1$; **if** $r_1 \in R$ **then** $d_{r,l_1} = -2$
    for $(e_2 = (r_1, p_2, r_2) \in E)$ **do**
      if $p_2 \in R$ **then** $d_{r,p_2} = 3$; **if** $r_2 \in R$ **then** $d_{r,2_2} = 4$
      if $p_1, p_2 \in R$ **then** $d_{p_1,p_2} = 2$; **if** $p_1, r_2 \in R$ **then** $d_{p_1,r_2} = 3$
    end
    for $(e_2 = (r_2, p_2, r_1) \in E)$ **do**
      if $p_2 \in R$ **then** $d_{r,p_2} = 3$; **if** $r_2 \in R$ **then** $d_{r,2_2} = 4$
      if $p_1, p_2 \in R$ **then** $d_{p_1,p_2} = 2$; **if** $p_1, p_2 \in R$ **then** $d_{p_1,2_2} = 3$
    end
  end
end

Algorithm 1: Algorithm to compute the pairwise distance between every resource in a set R appearing in a KB.

Data: Graph $G = (V, E, f)$ and a set R of vertices and edges, and their pairwise distance d
Result: All connected triple patterns in G from a set R of vertices and edges with maximal K triple patterns

$L = \emptyset$ #list of triple patterns
$V_{s,o} = \emptyset$ #set of variables in subject, object position
$V_p = \emptyset$ #set of variables in predicate position
k=0
Function *generate (L,k)*
  for $s_1 \in (R \cap V) \cup V_{s,o} \cup \{x_{k,1}\}$ **do**
    for $s_2 \in (R \cap P) \cup V_p \cup \{x_{k,2}\}$ **do**
      for $s_3 \in (R \cap V) \cup V_{s,o} \cup \{x_{k,3}\}$ **do**
        if $k = 0 \wedge d_{s_2,s_3} = -1 \wedge d_{s_1,s_2} = 1 \wedge d_{s_1,s_3} = 2$ **then** $L \leftarrow L \cup \{(s_1, s_2, s_3)\}$
        for $T \in L^{(k)}$ **do**
          $b_1 = true; b_2 = true; b_3 = true; b_4 = true;$
          for $(t_1, t_2, t_3) \in T$ **do**
            if not
              $(s_1 = t_1 \wedge d_{t_1,s_2} = 1 \wedge d_{t_1,s_3} = 2 \wedge d_{t_2,s_2} = -2 \wedge d_{t_2,s_3} = -3 \wedge d_{t_3,s_2} = -3 \wedge d_{t_3,s_3} = -4)$
              then $b_1 = false$
            if not $(s_1 = t_3 \wedge d_{t_1,s_2} = 3 \wedge d_{t_1,s_3} = 4 \wedge d_{t_2,s_2} = 2 \wedge d_{t_2,s_3} = 3 \wedge d_{t_3,s_2} = 1 \wedge d_{t_3,s_3} = 2)$
              then $b_2 = false$
            if not $(s_3 = t_1 \wedge d_{t_1,s_2} = -1 \wedge d_{t_1,s_1} = -2 \wedge d_{t_2,s_2} = -2 \wedge d_{t_2,s_3} = -1 \wedge d_{t_3,s_2} = -3 \wedge d_{t_3,s_3} = -2)$ **then** $b_3 = false$
            if not $(s_3 = t_3 \wedge d_{t_1,s_2} = -3 \wedge d_{t_1,s_3} = 2 \wedge d_{t_2,s_2} = -2 \wedge d_{t_2,s_3} = -3 \wedge d_{t_3,s_2} = -1)$
              then $b_4 = false$
          end
          if $b_1 = true \vee b_2 = true \vee b_3 = true \vee b_4 = true$ **then**
            $L \leftarrow L \cup \{T \cup (s_1, s_2, s_3)\}$;
            $V_{s,o} \leftarrow V_{s,o} \cup \{s_1, s_3\}$;
            $V_p \leftarrow V \cup \{s_2\})$
          end
          if *(k!=K)* **then**
            return *generate(L,k+1)*
          end
        end
      end
    end
  end

Algorithm 2: Recursive algorithm to create all connected triple patterns from a set R of resources with maximal K triple patterns. L contains the triple patterns created recursively and $L^{(k)}$ indicates the triple patterns with exactly k triples. Moreover $x_{k,1}, x_{k,2}, x_{k,3}$ are new variables that are added in step k. Note that the "if not" conditions correspond to the four possibilities a),b),c),d) of joining two triples which are depicted below. Note that they are very often not fulfilled. This guarantees the speed of the process.



breadth-first search operations, and (2) the low footprint of the index in terms of disk and memory space. Roughly, a 100 GB RDF dump can be compressed to a HDT file of a size of approx. 10 GB [18].
Based on the numbers above, we now want to construct all triple patterns with K triples and one projection variable recursively. Given a triple pattern T, we only want to build connected triple-pattern while adding triples to T. This can be done recursively using the algorithm described in Algorithm 2. Note that thanks to the numbers collected during the breadth-first search operations, this can be performed very fast. Once the triple patterns are constructed, one can choose any of the variables, which are in subject or object position, as a projection variable.
The decision to generate a SELECT or/and ASK query, is made depending on some regex expressions over the beginning of the question.

5. Evaluation

To validate the approach w.r.t. multilinguality, portability and robustness, we evaluated our approach using multiple benchmarks for QA that appeared in the last years. The different benchmarks are not comparable and they focus on different aspects of QA. For example SimpleQuestions focuses on questions that can be solved by one simple triple-pattern, while LC-QuAD focuses on more complex questions. Moreover, the QALD questions address different challenges including multilinguality and the use of keyword questions. Unlike previous works, we do not focus on one benchmark, but we analyze the behaviour of our approach under different scenarios. This is important, because it shows that our approach is not adapted to one particular benchmark, as it is often done by existing QA systems, and proves its portability.
We tested our approach on 5 different datasets namely Wikidata[10], DBpedia[11], MusicBrainz[12], DBLP[13] and Freebase[14]. Moreover, we evaluated our approach on five different languages namely: English, German, French, Italian and Spanish. First, we describe how we selected stop words and collected lexicalizations for the different languages and KBs, then we describe and discuss our results.

[10] https://www.wikidata.org/
[11] http://dbpedia.org
[12] https://musicbrainz.org
[13] http://dblp.uni-trier.de
[14] https://developers.google.com/freebase/

5.1. Stop Words and lexicalizations

As stop words, we use the lists, for the different languages, provided by Lucene, together with some words which are very frequent in questions like “what”, “which”, “give”.
Depending on the KB, we followed different strategies to collect lexicalizations. Since Wikidata has a rich number of lexicalizations, we simply took all lexicalizations associated to a resource through `rdfs:label`[15], `skos:prefLabel`[16] and `skos:altLabel`. For DBpedia, we only used the English DBpedia, where first all lexicalizations associated to a resource through the `rdfs:label` property were collected. Secondly, we followed the disambiguation and redirect links to get additional ones and took also into account available demonyms `dbo:demonym` (i.e. to `dbr:Europe` we associate also the lexicalization “European”). Thirdly, by following the interlanguage links, we associated the labels from the other languages to the resources. DBpedia properties are poorly covered with lexicalizations, especially when compared to Wikidata. For example, the property `dbo:birthPlace` has only one lexicalization namely “birth place”, while the corresponding property over Wikidata `P19` has 10 English lexicalizations like “birthplace”, “born in”, “location born”, “birth city”. In our exemplary implementation two strategies were implemented. First, while aiming at a QA system for the Semantic Web we also can take into account interlinkings between properties of distinguished KBs, s.t. lexicalizations are merged from all KBs currently considered. There, the `owl:sameAs` links from DBpedia relations to Wikidata are used and every lexicalization present in Wikidata is associated to the corresponding DBpedia relation. Secondly, the DBpedia abstracts are used to find more lexicalizations for the relations. To find new lexicalizations of a property p we follow the strategy proposed by [20]. We extracted from the KB the subject-object pairs *(x,y)* that are connected by p. Then the abstracts are scanned and all sentences are retrieved which contain both $label(x)$ and $label(y)$. At the end, the segments of text between $label(x)$ and $label(y)$, or $label(y)$ and $label(x)$ are extracted. We rank the extracted text segments and we choose the most frequent ones. This was done only for English.
For MusicBrainz we used the lexicalizations attached

[15] `rdfs: http://www.w3.org/2000/01/rdf-schema#`
[16] `skos: http://www.w3.org/2004/02/skos/core#`

to purl:title[17], foaf:name[18], skos:altLabel and rdfs:label. For DBLP only the one attached to rdfs:label. Note, MusicBrainz and DBLP contain only few properties. We aligned them manually with Wikidata and moved the lexicalizations from one KB to the other. The mappings can be found under http://goo.gl/ujbwFW and http://goo.gl/ftzegZ respectively. This took in total 1 hour of manual work.
For Freebase, we considered the lexicalizations attached to rdfs:label. We also followed the few available links to Wikidata. Finally, we took the 20 most prominent properties in the training set of the SimpleQuestions benchmark and looked at the lexicalizations of them in the first 100 questions of SimpleQuestions. We extracted manually the lexicalizations for them. This took 1 hour of manual work. We did not use the other (75,810 training and 10,845 validation) questions, i.e., unlike previous works we only took a small fraction of the available training data.
We want to briefly discuss the strategies we used here. We do not see any option other than manually indicating the lexicalizations for instances. For example, in MusicBrainz the property purl:title must be selected otherwise one cannot find any existing album. On the other hand there could be a property expressing the cover description of an album. We do not see any method to determine automatically why the property purl:title should be used as a lexicalization, while not the one about the cover description. We therefore think that the only available solution is to make the standard more clear on how to express such an information. Regarding the lexicalization of relations, the situation is different. The literature contains a number of approaches that can be used to generate them. For an overview, we refer to Section 7 in [12]. All works suppose one of the three following situations: there is some free text that also contains such knowledge, or they are expressed in some external databases like WordNet, or a training repository to learn them from question and answer pairs is available. On knowledge bases like Musicbrainz any of the 3 alternatives is not available, so also in this case we believe that the manual work cannot be avoided.

5.2. Experiments

To show the performance of the approach on different scenarios, we benchmarked it using the following benchmarks.

5.2.1. Benchmarks

QALD: We evaluated our approach using the QALD benchmarks. These benchmarks are good to see the performance on multiple languages and over both full-natural language questions and keyword questions. We followed the metrics of the original benchmarks. Note that the metrics changed in QALD-7. The results are given in Table 3 together with state-of-the-art systems. To find these, we used Google Scholar to select all publications about QA systems that cited one of the QALD challenge publications. Note that, in the past, QA systems were evaluated only on one or two of the QALD benchmarks. We provide, for the first time, an estimation of the differences between the benchmark series. Over English, we outperformed 90% of the proposed approaches. We do not beat Xser [48] and UTQA [36]. Note that these systems required additional training data than the one provided in the benchmark, which required a significant cost in terms of manual effort. Moreover, the robustness of these systems over keyword questions is probably not guaranteed. We cannot prove this claim because for these systems neither the source code nor a web-service is available.
Due to the manual effort required to do an error analysis for all benchmarks and the limited space, we restricted to the QALD-6 benchmark. The error sources over the 100 questions are the following:

- 40% (26 errors) are due to lexical gap (e.g. for "Who played Gus Fring in Breaking Bad?" the property dbo:portrayer is expected)
- 28% (18 errors) come from wrong ranking
- 12% (8 errors) are due to the missing support of superlatives and comparatives in our implementation (e.g. "Which Indian company has the most employees?")
- 9% (4 errors) from the need of complex queries with unions or filters (e.g. the question "Give me a list of all critically endangered birds." requires a filter on dbo:conservationStatus equal "CR")
- 6% (4 errors) come from out of scope questions (i.e. question that should not be answered)
- 2% (1 error) from too ambiguous questions (e.g. "Who developed Slack?" is expected to refer to a "cloud-based team collaboration tool" while we interpret it as "linux distribution").

[17] purl: http://purl.org/dc/elements/1.1/
[18] foaf: http://xmlns.com/foaf/

One can see that keyword queries always perform worse as compared to full natural language queries. The reason is that the formulation of the keyword queries does not allow us to decide if the query is an ASK query or if a COUNT is needed (e.g. "Did Elvis Presley have children?" is formulated as "Elvis Presley, children"). This means that we automatically get these questions wrong.

To show the performance over Wikidata, we consider the QALD-7 task 4 training dataset. This originally provided only English questions. The QALD-7 task 4 training dataset reuses questions over DBpedia from previous challenges where translations in other languages were available. We moved these translations to the dataset. The results can be seen in Table 4. Except for English, keyword questions are easier than full natural language questions. The reason is the formulation of the questions. For keyword questions the lexical gap is smaller. For example, the keyword question corresponding to the question "Qui écrivit Harry Potter?" is "écrivain, Harry Potter". Stemming does not suffice to map "écrivit" to "écrivain", lemmatization would be needed. This problem is much smaller for English, where the effect described over DBpedia dominates. We can see that the best performing language is English, while the worst performing language is Italian. This is mostly related to the poorer number of lexicalizations for Italian. Note that the performance of the QA approach over Wikidata correlates with the number of lexicalizations for resources and properties for the different languages as described in [28]. This indicates that the quality of the data, in different languages, directly affects the performance of the QA system. Hence, we can derive that our results will probably improve while the data quality is increased. Finally we outperform the presented QA system over this benchmark.

SimpleQuestions: SimpleQuestions contains 108,442 questions that can be solved using one triple pattern. We trained our system using the first 100 questions in the training set. The results of our system, together with the state-of-the-art systems are presented in Table 5. For this evaluation, we restricted the generated queries with one triple-pattern. The system performance is 14% below the state-of-the-art. Note that we achieve this result by considering only 100 of the 75,810 questions in the training set, and investing 1 hour of manual work for creating lexicalizations for properties manually. Concretely, instead of generating a training dataset with 80,000 questions, which can

QA system	Lang	Type	Total	P	R	F	Runtime	Ref
QALD-3								
WDAqua-core1	**en**	**full**	**100**	**0.64**	**0.42**	**0.51**	**1.01**	-
WDAqua-core1	**en**	**key**	**100**	**0.71**	**0.37**	**0.48**	**0.79**	-
WDAqua-core1	**de**	**key**	**100**	**0.79**	**0.31**	**0.45**	**0.22**	-
WDAqua-core1	**de**	**full**	**100**	**0.79**	**0.28**	**0.42**	**0.30**	-
WDAqua-core1	**fr**	**key**	**100**	**0.83**	**0.27**	**0.41**	**0.26**	-
gAnswer [53]∗	en	full	100	0.40	0.40	0.40	≈ 1 s	[53]
WDAqua-core1	**fr**	**full**	**100**	**0.70**	**0.26**	**0.38**	**0.37**	-
WDAqua-core1	**es**	**full**	**100**	**0.77**	**0.24**	**0.37**	**0.27**	-
WDAqua-core1	**it**	**full**	**100**	**0.79**	**0.23**	**0.36**	**0.30**	-
WDAqua-core1	**it**	**key**	**100**	**0.84**	**0.23**	**0.36**	**0.24**	-
WDAqua-core1	**es**	**key**	**100**	**0.80**	**0.23**	**0.36**	**0.23**	-
RTV [21]	en	full	99	0.32	0.34	0.33	-	[8]
Intui2 [15]	en	full	99	0.32	0.32	0.32	-	[8]
SINA [39]∗	en	full	100	0.32	0.32	0.32	≈ 10-20s	[39]
DEANNA [49]∗	en	full	100	0.21	0.21	0.21	≈ 1-50 s	[53]
SWIP [37]	en	full	99	0.16	0.17	0.17	-	[8]
Zhu et al. [52]∗	en	full	99	0.38	0.42	0.38	-	[52]
QALD-4								
Xser [48]	en	full	50	0.72	0.71	0.72	-	[44]
WDAqua-core1	**en**	**key**	**50**	**0.76**	**0.40**	**0.52**	**0.32s**	-
WDAqua-core1	**en**	**full**	**50**	**0.56**	**0.30**	**0.39**	**0.46s**	-
gAnswer [53]	en	full	50	0.37	0.37	0.37	0.973 s	[44]
CASIA [26]	en	full	50	0.32	0.40	0.36	-	[44]
WDAqua-core1	**de**	**key**	**50**	**0.92**	**0.20**	**0.33**	**0.04s**	-
WDAqua-core1	**fr**	**key**	**50**	**0.92**	**0.20**	**0.33**	**0.06s**	-
WDAqua-core1	**it**	**key**	**50**	**0.92**	**0.20**	**0.33**	**0.04s**	-
WDAqua-core1	**es**	**key**	**50**	**0.92**	**0.20**	**0.33**	**0.05s**	-
WDAqua-core1	**de**	**full**	**50**	**0.90**	**0.20**	**0.32**	**0.06s**	-
WDAqua-core1	**it**	**full**	**50**	**0.92**	**0.20**	**0.32**	**0.16s**	-
WDAqua-core1	**es**	**full**	**50**	**0.90**	**0.20**	**0.32**	**0.06s**	-
WDAqua-core1	**fr**	**full**	**50**	**0.86**	**0.18**	**0.29**	**0.09s**	-
Intui3 [16]	en	full	50	0.23	0.25	0.24	-	[44]
ISOFT [35]	en	full	50	0.21	0.26	0.23	-	[44]
Hakimov [25]∗	en	full	50	0.52	0.13	0.21	-	[25]
QALD-5								
Xser [48]	en	full	50	0.74	0.72	0.73	-	[45]
UTQA [36]	en	full	50	-	-	0.65	-	[36]
UTQA [36]	es	full	50	0.55	0.53	0.54	-	[36]
UTQA [36]	fs	full	50	0.53	0.51	0.52	-	[36]
WDAqua-core1	**en**	**full**	**50**	**0.56**	**0.41**	**0.47**	**0.62s**	-
WDAqua-core1	**en**	**key**	**50**	**0.60**	**0.27**	**0.37**	**0.50s**	-
AskNow[17]	en	full	50	0.32	0.34	0.33		[17]
QAnswer[38]	en	full	50	0.34	0.26	0.29	-	[45]
WDAqua-core1	**de**	**full**	**50**	**0.92**	**0.16**	**0.28**	**0.20s**	-
WDAqua-core1	**de**	**key**	**50**	**0.90**	**0.16**	**0.28**	**0.19s**	-
WDAqua-core1	**fr**	**full**	**50**	**0.90**	**0.16**	**0.28**	**0.19s**	-
WDAqua-core1	**fr**	**key**	**50**	**0.90**	**0.16**	**0.28**	**0.18s**	-
WDAqua-core1	**it**	**full**	**50**	**0.88**	**0.18**	**0.30**	**0.20s**	-
WDAqua-core1	**it**	**key**	**50**	**0.90**	**0.16**	**0.28**	**0.18s**	-
WDAqua-core1	**es**	**full**	**50**	**0.88**	**0.14**	**0.25**	**0.20s**	-
WDAqua-core1	**es**	**key**	**50**	**0.90**	**0.14**	**0.25**	**0.20s**	-
SemGraphQA[4]	en	full	50	0.19	0.20	0.20	-	[45]
YodaQA[3]	en	full	50	0.18	0.17	0.18	-	[45]
QuerioDali[29]	en	full	50	?	?	?	?	[29]
QALD-6								
WDAqua-core1	**en**	**full**	**100**	**0.55**	**0.34**	**0.42**	**1.28s**	-
WDAqua-core1	**de**	**full**	**100**	**0.73**	**0.29**	**0.41**	**0.41s**	-
WDAqua-core1	**de**	**key**	**100**	**0.85**	**0.27**	**0.41**	**0.30s**	-
WDAqua-core1	**en**	**key**	**100**	**0.51**	**0.30**	**0.37**	**1.00s**	-
SemGraphQA [4]	en	full	100	0.70	0.25	0.37	-	[46]
WDAqua-core1	**fr**	**key**	**100**	**0.78**	**0.23**	**0.36**	**0.34s**	-
WDAqua-core1	**fr**	**full**	**100**	**0.57**	**0.22**	**0.32**	**0.46s**	-
WDAqua-core1	**es**	**full**	**100**	**0.69**	**0.19**	**0.30**	**0.45s**	-
WDAqua-core1	**es**	**key**	**100**	**0.83**	**0.18**	**0.30**	**0.35s**	-
WDAqua-core1	**it**	**key**	**100**	**0.75**	**0.17**	**0.28**	**0.34s**	-
AMUSE [24]	en	full	100	-	-	0.26	-	[24]
WDAqua-core1	**it**	**full**	**100**	**0.62**	**0.15**	**0.24**	**0.43s**	-
AMUSE [24]	es	full	100	-	-	0.20	-	[24]
AMUSE [24]	de	full	100	-	-	0.16	-	[24]

QA system	Lang	Type	Total	P	R	F	Runtime	Ref
QALD-7								
WDAqua-core1	**en**	**full**	**100**	**0.25**	**0.28**	**0.25**	**1.24s**	-
WDAqua-core1	**fr**	**key**	**100**	**0.18**	**0.16**	**0.16**	**0.32s**	-
WDAqua-core1	**en**	**key**	**100**	**0.14**	**0.16**	**0.14**	**0.88s**	-
WDAqua-core1	**de**	**full**	**100**	**0.10**	**0.10**	**0.10**	**0.34s**	-
WDAqua-core1	**de**	**key**	**100**	**0.12**	**0.10**	**0.10**	**0.28s**	-
WDAqua-core1	**fr**	**full**	**100**	**0.12**	**0.10**	**0.10**	**0.42s**	-
WDAqua-core1	**it**	**key**	**100**	**0.06**	**0.06**	**0.06**	**0.28s**	-
WDAqua-core1	**it**	**full**	**100**	**0.04**	**0.04**	**0.04**	**0.34s**	-

Table 3

This table (left column and upper right column) summarizes the results obtained by the QA systems evaluated with QALD-3 (over DBpedia 3.8), QALD-4 (over DBpedia 3.9), QALD-5 (over DBpedia 2014), QALD-6 (over DBpedia 2015-10), QALD-7 (2016-04). We indicated with "$*$" the systems that did not participate directly in the challenges, but were evaluated on the same benchmark afterwards. We indicate the average running times of a query for the systems where we found them. Even if the runtime evaluations were executed on different hardware, it still helps to give an idea about the scalability.

QA System	Lang	Type	Total	P	R	F	Runtime	Ref
QALD-7 task 4, training dataset								
WDAqua-core1	**en**	**full**	**100**	**0.37**	**0.39**	**0.37**	**1.68s**	-
WDAqua-core1	**en**	**key**	**100**	**0.35**	**0.38**	**0.35**	**0.80s**	-
WDAqua-core1	**es**	**key**	**100**	**0.31**	**0.32**	**0.31**	**0.45s**	-
Sorokin et al. [41]	en	full	100	-	-	0.29	-	[41]
WDAqua-core1	**de**	**key**	**100**	**0.27**	**0.28**	**0.27**	**1.13s**	-
WDAqua-core1	**fr**	**key**	**100**	**0.27**	**0.30**	**0.27**	**1.14s**	-
WDAqua-core1	**fr**	**full**	**100**	**0.27**	**0.31**	**0.27**	**1.05s**	-
WDAqua-core1	**es**	**full**	**100**	**0.24**	**0.26**	**0.24**	**0.65s**	-
WDAqua-core1	**de**	**full**	**100**	**0.18**	**0.20**	**0.18**	**0.82s**	-
WDAqua-core1	**it**	**full**	**100**	**0.19**	**0.20**	**0.18**	**1.00s**	-
WDAqua-core1	**it**	**key**	**100**	**0.17**	**0.18**	**0.16**	**0.44s**	-

Table 4

The table shows the results of WDAqua-core1 over the QALD-7 task 4 training dataset. We used Wikidata (dated 2016-11-28).

QA System	Lang	Type	Total	Accuracy	Runtime	Ref
Lukovnikov et al.	en	full	21687	0.712	-	[31]
Golub and He	en	full	21687	0.709	-	[23]
Yin et al.	en	full	21687	0.683	-	[50]
Bordes et al.	en	full	21687	0.627	-	[6]
Dai et al.*	en	full	21687	0.626	-	[9]
WDAqua-core1*	**en**	**full**	**21687**	**0.571**	**2.1 s**	**-**

Table 5

This table summarizes the QA systems evaluated over SimpleQuestions. Every system was evaluated over FB2M except the ones marked with $(*)$ which were evaluated over FB5M.

Benchmark	Lang	Type	Total	P	R	F	Runtime
LC-QuAD	en	full	5000	0.59	0.38	0.46	1.5 s
WDAquaCore0Questions	mixed	mixed	689	0.79	0.46	0.59	1.3 s

Table 6

This table summarizes the results of WDAqua-core1 over some newly appeared benchmarks.

Dataset	Lang	Type	Total	P	R	F	Runtime
DBpedia	en	full	100	0.55	0.34	0.42	1.37 s
All KBs supported	en	full	100	0.49	0.39	0.43	11.94s

Table 7

Comparison on QALD-6 when querying only DBpedia and multiple KBs at the same time.

cost several thousands of euros, we invested 1 hour of manual work with the result of loosing (only) 14% in accuracy!

Note that the SimpleQuestions dataset is highly skewed towards certain properties (it contains 1629 properties, the 20 most frequent properties cover nearly 50% of the questions). Therefore, it is not clear how the other QA systems behave with respect to properties not appearing in the training dataset and with respect to keyword questions. Moreover, it is not clear how to port the existing approaches to new languages and it is not possible to adapt them to more difficult questions. These points are solved using our approach. Hence, we provided here, for the first time, a quantitative analysis of the impact of big training data corpora on the quality of a QA system.

LC-QuAD & WDAquaCore0Questions: Recently, a series of new benchmarks have been published. LC-QuAD [43] is a benchmark containing 5000 English questions and it concentrates on complex questions. WDAquaCore0Questions [13] is a benchmark containing 689 questions over multiple languages and addressing mainly Wikidata, generated from the logs of a live running QA system. The questions are a mixture of real-world keyword and malformed questions. In Table 6, we present the first baselines for these benchmarks.

Multiple KBs: The only available benchmark that tackles multiple KBs was presented in QALD-4 task 2. The KBs are rather small and perfectly interlinked. This is not the case over the considered KBs. We therefore evaluated the ability to query multiple KBs differently. We run the questions of the QALD-6 benchmark, which was designed for DBpedia, both over DBpedia (only) and over DBpedia, Wikidata, MusicBrainz, DBLP and Freebase. Note that, while the original questions have a solution over DBpedia, a good answer could also be found over the other datasets. We therefore manually checked whether the answers that were found in other KBs are right (independently from which KB was chosen by the QA system to answer it). The results are presented in Table 7. WDAqua-core1 choose 53 times to answer a question over DBpedia, 39 over Wikidata and the other 8 times over a different KB. Note that we get better results when querying multiple KBs. Globally we get better recall and lower precision which is expected. While scalability is an issue, we are able to pick the right KB

to find the answer!

Note: We did not tackle the WebQuestions benchmark for the following reasons. While it has been shown that WebQuestions can be addressed using non-reified versions of Freebase, this was not the original goal of the benchmark. More then 60% of the QA systems benchmarked over WebQuestions are tailored towards its reefication model. There are two important points here. First, most KBs in the Semantic Web use binary statements. Secondly, in the Semantic Web community, many different reefication models have been developed as described in [22].

5.2.2. Setting

All experiments were performed on a virtual machine with 4 cores of Intel Xeon E5-2667 v3 3.2GH, 16 GB of RAM and 500 GB of SSD disk. Note that the whole infrastructure was running on this machine, i.e. all indexes and the triple-stores needed to compute the answers (no external service was used). The original data dumps sum up to 336 GB. Note that across all benchmarks we can answer a question in less then 2 seconds except when all KBs are queried at the same time which shows that the algorithm should be parallelized for further optimization.

6. Provided Services for Multilingual and Multi-KB QA

We have presented an algorithm that can be easily ported to new KBs and that can query multiple KBs at the same time. In the evaluation section, we have shown that our approach is competitive while offering the advantage of being multilingual and robust to keyword questions. Moreover, we have shown that we can achieve acceptable run-times on a modern laptop. In this section, we describe how we integrated the approach to an actual service and how we combine it to existing services so that it can be directly used by end-users.

First, we integrated WDAqua-core1 into Qanary [7, 14], a framework to integrate QA components. This way WDAqua-core1 can be accessed via RESTful interfaces for example to benchmark it via Gerbil for QA[47]. It also allows to combine it with services that are already integrated into Qanary like a speech recognition component based on Kaldi[19] and a language detection component based on [34]. Moreover, the integration into Qanary allows to reuse Trill [10], a reusable front-end for QA systems. A screenshot of Trill using in the back-end WDAqua-core1 can be found in Figure 4.

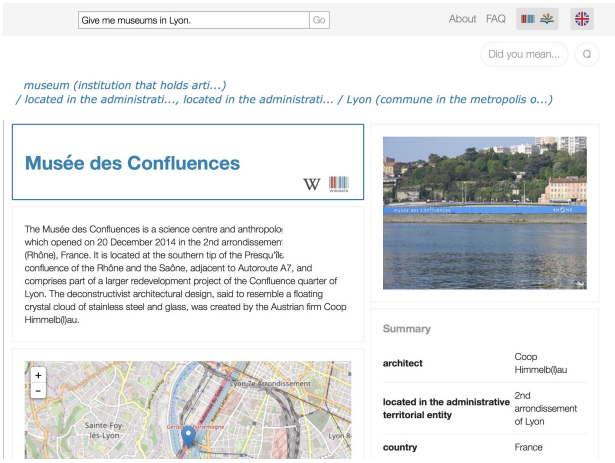


Fig. 4. Screenshot of Trill, using in the back-end WDAqua-core1, for the question "Give me museums in Lyon.".

Secondly, we reused and extended Trill to make it easily portable to new KBs. While Trill originally supported only DBpedia and Wikidata, now it also supports also MusicBrainz, DBLP and Freebase. We designed the extension so that it can be easily ported to new KBs. Enabling the support for a new KB is mainly reduced to writing an adapted SPARQL query for the new KB. Additionally, the extension allows to select multiple KBs at the same time.

Thirdly, we adapted some services that are used in Trill to be easily portable to new KBs. These include SPARQLToUser [11], a tool that generates a human readable version of a SPARQL query and LinkSUM [42] a service for entity summarization. All these tools now support the 5 mentioned KBs and the 5 mentioned languages.

A public online demo is available under:

www.wdaqua.eu/qa

Moreover, there is an open API available that is described at

http://wdaqua-frontend.univ-st-etienne.fr/faq

This is for example implemented by Gerbil for QA [47] and the DBpediaChat [2][20].

[19] http://kaldi-asr.org

[20] http://chat.dbpedia.org/

7. Conclusion and Future Work

In this paper, we introduced a novel concept for QA aimed at multilingual and KB-agnostic QA. Due to the described characteristics of our approach portability is ensured which is a significant advantage in comparison to previous approaches. We have shown the power of our approach in an extensive evaluation over multiple benchmarks. Hence, we clearly have shown our contributions w.r.t. qualitative (language, KBs) and quantitative improvements (outperforming many existing systems and querying multiple KBs) as well as the capability of our approach to scale for very large KBs like DBpedia.

We have applied our algorithm and adapted a set of existing services so that end-users can query, using multiple languages, multiple KBs at the same time, using a unified interface. Hence, we provided here a major step towards QA over the Semantic Web following our larger research agenda of providing QA over the LOD cloud.

In the future, we want to tackle the following points. First, we want to parallelize our approach, s.t. when querying multiple KBs acceptable response times will be achieved. Secondly, we want to query more and more KBs (hints to interesting KBs are welcome). Thirdly, from different lessons learned from querying multiple KBs, we want to give a set of recommendations for RDF datasets, s.t. they are fit for QA. And fourth, we want to extend our approach to also query reified data. Fifth, we would like to extend the approach to be able to answer questions including aggregates and functions. We believe that our work can further boost the expansion of the Semantic Web since we presented a solution that easily allows to consume RDF data directly by end-users requiring low hardware investments.

Note: There is a Patent Pending for the presented approach. It was submitted the 18 January 2018 at the EPO and has the number EP18305035.0.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 642795.

References

[1] 2018. Demoing Platypus - a Multilingual Question Answering Platform for Wikidata. In *ESWC P&D*.

[2] Ram G Athreya, Axel-Cyrille Ngonga Ngomo, and Ricardo Usbeck. 2018. Enhancing Community Interactions with Data-Driven Chatbots–The DBpedia Chatbot. In *Companion of the The Web Conference 2018 on The Web Conference 2018*. International World Wide Web Conferences Steering Committee, 143–146.

[3] Petr Baudiš and Jan Šedivỳ. 2015. QALD Challenge and the YodaQA System: Prototype Notes. (2015).

[4] Romain Beaumont, Brigitte Grau, and Anne-Laure Ligozat. 2015. SemGraphQA@QALD-5: LIMSI participation at QALD-5@CLEF. CLEF.

[5] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs.. In *EMNLP*.

[6] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale Simple Question Answering with Memory Networks. *CoRR* abs/1506.02075 (2015). arXiv:1506.02075 http://arxiv.org/abs/1506.02075

[7] A. Both, D. Diefenbach, K. Singh, S. Shekarpour, D. Cherix, and C. Lange. 2016. Qanary – a methodology for vocabulary-driven open question answering systems. In *ESWC 2016*.

[8] Philipp Cimiano, Vanessa Lopez, Christina Unger, Elena Cabrio, Axel-Cyrille Ngonga Ngomo, and Sebastian Walter. 2013. Multilingual question answering over linked data (qald-3): Lab overview. Springer.

[9] Zihang Dai, Lei Li, and Wei Xu. 2016. Cfo: Conditional focused neural question answering with large-scale knowledge bases. *arXiv preprint arXiv:1606.01994* (2016).

[10] D. Diefenbach, S. Amjad, A. Both, K. Singh, and P. Maret. 2017. Trill: A reusable Front-End for QA systems. In *ESWC P&D*.

[11] D. Diefenbach, Y. Dridi, , K. Singh, and P. Maret. 2017. SPARQLtoUser: Did the question answering system understand me?. In *ISWC NLIWoD3 Workshop*.

[12] D. Diefenbach, V. Lopez, K. Singh, and P. Maret. 2017. Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information systems* (2017), 1–41.

[13] D. Diefenbach, T. Pellissier, K. Singh, and P. Maret. 2017. Question Answering Benchmarks for Wikidata. In *ISWC P&D*.

[14] D. Diefenbach, K. Singh, A. Both, D. Cherix, C. Lange, and S. Auer. 2017. The Qanary Ecosystem: getting new insights by composing Question Answering pipelines. In *ICWE*.

[15] Corina Dima. 2013. Intui2: A prototype system for question answering over linked data. *Proceedings of the Question Answering over Linked Data lab (QALD-3) at CLEF* (2013).

[16] Corina Dima. 2014. Answering natural language questions with Intui3. In *Conference and Labs of the Evaluation Forum (CLEF)*.

[17] Mohnish Dubey, Sourish Dasgupta, Ankit Sharma, Konrad Höffner, and Jens Lehmann. 2016. AskNow: A Framework for Natural Language Query Formalization in SPARQL. In *International Semantic Web Conference*. Springer.

[18] Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. 2013. Binary RDF representation for publication and exchange (HDT). *Web Semantics: Science, Services and Agents on the World Wide Web* 19 (2013), 22–41.

[19] Ó. Ferrández, Ch. Spurk, M. Kouylekov, and al. 2011. The QALL-ME Framework: A specifiable-domain multilingual Question Answering architecture. *J. Web Sem.* (2011).

[20] Daniel Gerber and A-C Ngonga Ngomo. 2011. Bootstrapping the linked data web. In *1st Workshop on Web Scale Knowledge Extraction@ ISWC*, Vol. 2011.

[21] Cristina Giannone, Valentina Bellomaria, and Roberto Basili. 2013. A HMM-based approach to question answering against linked data. *Proceedings of the Question Answering over Linked Data lab (QALD-3) at CLEF* (2013).

[22] José M Giménez-García, Antoine Zimmermann, and Pierre Maret. 2017. NdFluents: An Ontology for Annotated Statements with Inference Preservation. In *European Semantic Web Conference*. Springer, 638–654.

[23] David Golub and Xiaodong He. 2016. Character-level question answering with attention. *arXiv preprint arXiv:1604.00727* (2016).

[24] Sherzod Hakimov, Soufian Jebbara, and Philipp Cimiano. 2017. AMUSE: Multilingual Semantic Parsing for Question Answering over Linked Data. In *International Semantic Web Conference*. Springer, 329–346.

[25] Sherzod Hakimov, Christina Unger, Sebastian Walter, and Philipp Cimiano. 2015. Applying semantic parsing to question answering over linked data: Addressing the lexical gap. In *Natural Language Processing and Information Systems*. Springer.

[26] Shizhu He, Yuanzhe Zhang, Kang Liu, and Jun Zhao. 2014. CASIA@ V2: A MLN-based Question Answering System over Linked Data. *Proc. of QALD-4* (2014).

[27] Sarthak Jain. 2016. Question Answering over Knowledge Base using Factual Memory Networks. In *Proceedings of NAACL-HLT*.

[28] Lucie-Aimée Kaffee, Alessandro Piscopo, Pavlos Vougiouklis, Elena Simperl, Leslie Carr, and Lydia Pintscher. 2017. A Glimpse into Babel: An Analysis of Multilinguality in Wikidata. In *Proceedings of the 13th International Symposium on Open Collaboration*. ACM, 14.

[29] V. Lopez, P. Tommasi, S. Kotoulas, and J. Wu. 2016. QuerioDALI: Question Answering Over Dynamic and Linked Knowledge Graphs. In *International Semantic Web Conference*. Springer, 363–382.

[30] Vanessa Lopez, Victoria Uren, Marta Sabou, and Enrico Motta. 2011. Is question answering fit for the semantic web? a survey. *Semantic Web* 2, 2 (2011).

[31] Denis Lukovnikov, Asja Fischer, Jens Lehmann, and Sören Auer. 2017. Neural Network-based Question Answering over Knowledge Graphs on Word and Character Level. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1211–1220.

[32] E. Marx, R. Usbeck, A. Ngonga Ngomo, K. Höffner, J. Lehmann, and S. Auer. 2014. Towards an Open Question Answering Architecture. In *SEMANTiCS*.

[33] Donald Metzler and W Bruce Croft. 2007. Linear feature-based models for information retrieval. *Information Retrieval* 10, 3 (2007), 257–274.

[34] Shuyo Nakatani. 2010. Language Detection Library for Java. (2010). https://github.com/shuyo/language-detection.

[35] Seonyeong Park, Hyosup Shim, and Gary Geunbae Lee. 2014. ISOFT at QALD-4: Semantic similarity-based question answering system over linked data. In *CLEF*.

[36] Amir Pouran-ebn veyseh. 2016. Cross-Lingual Question Answering Using Common Semantic Space. In *NAACL-HLT 2016*.

[37] Camille Pradel, Ollivier Haemmerlé, and Nathalie Hernandez. 2012. A semantic web interface using patterns: the SWIP system. In *Graph Structures for Knowledge Representation and Reasoning*. Springer.

[38] Stefan Ruseti, Alexandru Mirea, Traian Rebedea, and Stefan Trausan-Matu. 2015. QAnswer-Enhanced Entity Matching for Question Answering over Linked Data. CLEF.

[39] Saeedeh Shekarpour, Edgard Marx, Axel-Cyrille Ngonga Ngomo, and Sören Auer. 2015. Sina: Semantic interpretation of user queries for question answering on interlinked data. *Web Semantics: Science, Services and Agents on the World Wide Web* 30 (2015).

[40] K. Singh, A. Both, D. Diefenbach, and S. Shekarpour. 2016. Towards a Message-Driven Vocabulary for Promoting the Interoperability of Question Answering Systems. In *ICSC 2016*.

[41] Daniil Sorokin and Iryna Gurevych. 2017. End-to-end Representation Learning for Question Answering with Weak Supervision. In *ESWC 2017 Semantic Web Challenges*.

[42] Andreas Thalhammer, Nelia Lasierra, and Achim Rettinger. 2016. LinkSUM: Using Link Analysis to Summarize Entity Data. In *Web Engineering: 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*. Lecture Notes in Computer Science, Vol. 9671. Springer International Publishing, Cham, 244–261. https://doi.org/10.1007/978-3-319-38791-8_14

[43] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. LC-QuAD: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference*. Springer, 210–218.

[44] Christina Unger, Corina Forascu, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. 2014. Question answering over linked data (QALD-4). In *Working Notes for CLEF 2014 Conference*.

[45] Christina Unger, Corina Forascu, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. 2015. Answering over Linked Data (QALD-5). In *Working Notes for CLEF 2015 Conference*.

[46] Christina Unger, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, and Cimiano. 2016. 6th Open Challenge on Question Answering over Linked Data (QALD-6). In *The Semantic Web: ESWC 2016 Challenges.*

[47] Ricardo Usbeck, Michael Röder, Michael Hoffmann, Felix Conrads, Jonathan Huthmann, Axel-Cyrille Ngonga-Ngomo, Christian Demmler, and Christina Unger. 2016. Benchmarking Question Answering Systems. *Semantic Web Journal* (2016).

[48] Kun Xu, Yansong Feng, and Dongyan Zhao. 2014. Xser@ QALD-4: Answering Natural Language Questions via Phrasal Semantic Parsing. (2014).

[49] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. 2012. Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural*

Language Processing and Computational Natural Language Learning. Association for Computational Linguistics.

[50] Wenpeng Yin, Mo Yu, Bing Xiang, Bowen Zhou, and Hinrich Schütze. 2016. Simple question answering by attentive convolutional neural network. *arXiv preprint arXiv:1606.03391* (2016).

[51] Yuanzhe Zhang, Kang Liu, Shizhu He, Guoliang Ji, Zhanyi Liu, Hua Wu, and Jun Zhao. 2016. Question Answering over Knowledge Base with Neural Attention Combining Global Knowledge Information. *arXiv preprint arXiv:1606.00979* (2016).

[52] Chenhao Zhu, Kan Ren, Xuan Liu, Haofen Wang, Yiding Tian, and Yong Yu. 2015. A Graph Traversal Based Approach to Answer Non-Aggregation Questions Over DBpedia. *arXiv preprint arXiv:1510.04780* (2015).

[53] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffer Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural language question answering over RDF: a graph data driven approach. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM.