Template-based Question Answering over RDF Data

Christina Unger
Bielefeld University, CITEC
Universitätsstraße 21–23,
33615 Bielefeld
cunger@cit-ec.uni-bielefeld.de

Lorenz Bühmann
Universität Leipzig, IFI/AKSW
PO 100920, D-04009 Leipzig
buehmann@informatik.uni-leipzig.de

Jens Lehmann
Universität Leipzig, IFI/AKSW
PO 100920, D-04009 Leipzig
lehmann@informatik.uni-leipzig.de

Axel-Cyrille Ngonga Ngomo
Universität Leipzig, IFI/AKSW
PO 100920, D-04009 Leipzig
ngonga@informatik.uni-leipzig.de

Daniel Gerber
Universität Leipzig, IFI/AKSW
PO 100920, D-04009 Leipzig
dgerber@informatik.uni-leipzig.de

Philipp Cimiano
Bielefeld University, CITEC
Universitätsstraße 21–23,
33615 Bielefeld
cimiano@cit-ec.uni-bielefeld.de

ABSTRACT

As an increasing amount of RDF data is published as Linked Data, intuitive ways of accessing this data become more and more important. Question answering approaches have been proposed as a good compromise between intuitiveness and expressivity. Most question answering systems translate questions into triples which are matched against the RDF data to retrieve an answer, typically relying on some similarity metric. However, in many cases, triples do not represent a faithful representation of the semantic structure of the natural language question, with the result that more expressive queries can not be answered. To circumvent this problem, we present a novel approach that relies on a parse of the question to produce a SPARQL template that directly mirrors the internal structure of the question. This template is then instantiated using statistical entity identification and predicate detection. We show that this approach is competitive and discuss cases of questions that can be answered with our approach but not with competing approaches.

Categories and Subject Descriptors

H.5.2 [**Information systems**]: User Interfaces—*Natural language, Theory and methods*

General Terms

Algorithms, Experimentation, Theory

Keywords

Question Answering, Semantic Web, Natural Language Patterns, SPARQL

1. INTRODUCTION

As more and more RDF data is published as Linked Data, developing intuitive ways of accessing this data becomes increasingly important. One of the main challenges is the development of interfaces that exploit the expressiveness of the underlying data model and query language, while hiding their complexity. As a good compromise between intuitiveness and expressivity, question answering approaches allow users to express arbitrarily[1] complex information needs in natural language without requiring them to be aware of the underlying schema, vocabulary or query language. Several question answering systems for RDF data have been proposed in the past, for example, Aqualog [14, 23], PowerAqua [24], NLP-Reduce [6] and FREyA [1]. Many of these systems map a natural language question to a triple-based representation. For example, consider the simple question Who wrote The Neverending Story?. PowerAqua[2] would map this question to the triple representation

⟨[person,organization], wrote, Neverending Story⟩.

Then, by applying similarity metrics and search heuristics, it would retrieve matching subgraphs from the RDF repository. For the above query, the following triples would be retrieved from DBpedia, from which the answer "Michael Ende" can be derived:

⟨Writer, IS_A, Person⟩
⟨Writer, author, The_Neverending_Story⟩

While this approach works very well in cases where the meaning of the query can be captured easily, it has a number of drawbacks, as in many cases the original semantic structure of the question can not be faithfully captured using triples. For instance, consider the questions 1a and 2a below. PowerAqua would produce the triple representations in 1b and 2b, respectively. The goal, however, would be SPARQL queries[3] like 1c and 2c, respectively.

1. (a) Which cities have more than three universities?
   (b) ⟨[cities], more than, universities three⟩
   (c) `SELECT ?y WHERE {`

[1]At least as complex as can be represented in the query language.
[2]Accessed via the online demo at `http://poweraqua.open.ac.uk:8080/poweraqualinked/jsp/index.jsp`.
[3]Assuming a DBpedia namespace with `onto` as prefix `<http://dbpedia.org/ontology/>`.

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.

```
   ?x rdf:type onto:University .
   ?x onto:city ?y .
}
HAVING (COUNT(?x) > 3)
```

2. (a) Who produced the most films?
   (b) ⟨[person,organization], produced, most films⟩
   (c)
```
SELECT ?y WHERE {
   ?x rdf:type onto:Film .
   ?x onto:producer ?y .
}
ORDER BY DESC(COUNT(?x)) OFFSET 0 LIMIT 1
```

Such SPARQL queries are difficult to construct on the basis of the above mentioned triple representations, as aggregation and filter constructs arising from the use of specific quantifiers are not faithfully captured. What would be needed instead is a representation of the information need that is much closer to the semantic structure of the original question. Thus, we propose a novel approach to question answering over RDF data that relies on a parse of the question to produce a SPARQL template that directly mirrors the internal structure of the question and that, in a second step, is instantiated by mapping the occurring natural language expressions to the domain vocabulary. For example, a template produced for Question 2a would be:

3.
```
SELECT ?x WHERE {
   ?x ?p ?y .
   ?y rdf:type ?c .
}
ORDER BY DESC(COUNT(?y)) LIMIT 1 OFFSET 0
```

In this template, c stands proxy for the URI of a class matching the input keyword films and p stands proxy for a property matching the input keyword produced. In a next step, c has to be instantiated by a matching class, in the case of using DBpedia `onto:Film`, and p has to be instantiated with a matching property, in this case `onto:producer`. For instantiation, we exploit an index as well as a pattern library that links properties with natural language predicates.

We show that this approach is competitive and discuss specific cases of questions that can be precisely answered with our approach but not with competing approaches. Thus, the main contribution of this paper is a domain-independent question answering approach that first converts natural language questions into queries that faithfully capture the semantic structure of the question and then identifies domain-specific entities combining NLP methods and statistical information.

In the following section we present an overview of the system's architecture, and in Sections 3, 4 and 5 we explain the components of the system in more detail. In Section 6 we report on evaluation results and then point to an online interface to the prototype in Section 7. In Section 8 we compare our approach to existing question answering systems on RDF data, before concluding in Section 9.

2. OVERVIEW

Figure 1 gives an overview of our approach. The input question, formulated by the user in natural language, is first processed by a POS tagger. On the basis of the POS tags, lexical entries are created using a set of heuristics. These lexical entries, together with pre-defined domain-independent lexical entries, are used for parsing, which leads to a semantic representation of the natural language query, which is then converted into a SPARQL query template. This process is explained in Section 3. The query templates contain *slots*, which are missing elements of the query that have to be filled with URIs. In order to fill them, our approach first generates natural language expressions for possible slot fillers from the user question using WordNet expansion. In a next step, sophisticated entity identification approaches are used to obtain URIs for those natural language expressions. These approaches rely both on string similarity as well as on natural language patterns which are compiled from existing structured data in the Linked Data cloud and text documents. A detailed description is given in Section 4. This yields a range of different query candidates as potential translations of the input question. It is therefore important to rank those query candidates. To do this, we combine string similarity values, prominence values and schema conformance checks into a score value. Details of this mechanism are covered in Section 5. The highest ranked queries are then tested against the underlying triple store and the best answer is returned to the user.

3. TEMPLATE GENERATION

The main assumption of template generation is that the overall structure of the target SPARQL query is (at least partly) determined by the syntactic structure of the natural language question and by the occurring domain-independent expressions. Consequently, our goal is to generate a SPARQL query template such as the one in Example 3 in the introduction, which captures the semantic structure of the user's information need and leaves open only specific slots for resources, classes and properties, that need to be determined with respect to the underlying dataset.

3.1 SPARQL templates

SPARQL templates specify the query's select or ask clause, its filter and aggregation functions, as well as the number and form of its triples. Subject, predicate and object of a triple are variables, some of which stand proxy for appropriate URIs. These proxy variables, called *slots*, are defined as triples of a variable, the type of the intended URI (resource, class or property), and the natural language expression that was used in the user question, e.g. ⟨?x, class, films⟩.

For example, for the question in 4 (taken from the QALD-1 benchmark, cf. Section 6) the two SPARQL templates given in 4a and 4b are built.

4. How many films did Leonardo DiCaprio star in?
   (a)
```
SELECT COUNT(?y) WHERE {
   ?y rdf:type ?c .
   ?x ?p ?y .
}
```
   Slots:
   - ⟨?x, resource, Leonardo DiCaprio⟩
   - ⟨?c, class, films⟩
   - ⟨?p, property, star⟩

   (b)
```
SELECT COUNT(?y) WHERE {
   ?x ?p ?y .
}
```
   Slots:

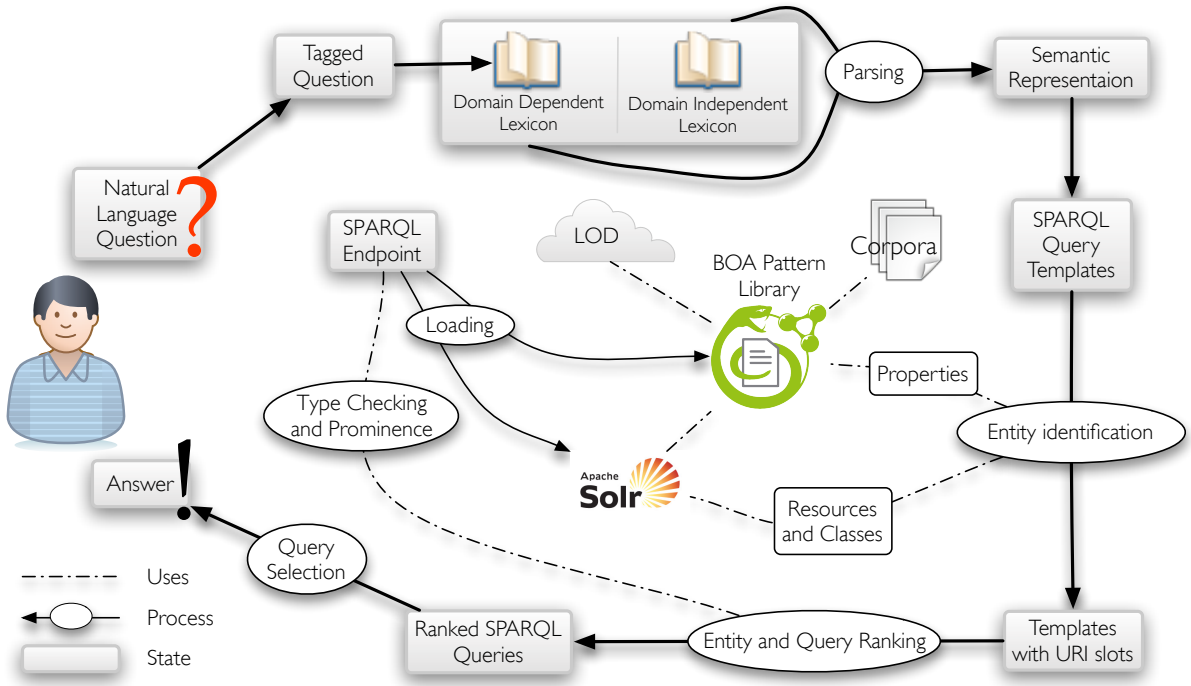


Figure 1: Overview of the template based SPARQL query generator.

- ⟨?x, resource, Leonardo DiCaprio⟩
- ⟨?p, property, films⟩

The reason for constructing two templates is that the noun films could either correspond to a class (as in 4a) or to a property (as in 4b). Since we want the template generation mechanism to use only domain-independent, linguistic information, such that it does not depend on how a particular dataset is modelled, we start by generating all possible templates and later select the one that captures the structure of the considered data.

3.2 Constructing SPARQL templates

In order to get from a natural language question to a SPARQL template, we adopt the parsing and meaning construction mechanism of the question answering system Pythia [22]. The main reason for doing so is ease of adaption to our purposes, but another parser, e.g. the Stanford parser, together with some semantic interpretation process could do as well.

Pythia's parsing process relies on a lexicon, which specifies for each expression a syntactic and a semantic representation. The former are trees from Lexicalized Tree Adjoining Grammar [16] and the latter are representations similar to Underspecified Discourse Representation Theory (see [22] for more details). Such a lexicon consists of two parts. One part comprises domain-independent expressions, which were specified manually and can be re-used across all domains. This part contains 107 entries, mainly light verbs (to be, to have, and imperatives like give me), question words (what, which, how many, when, where) and other determiners (some, all, no, at least, more/less than, the most/least), together with negation words, coordination and the like. The other part of the lexicon comprises domain-dependent expressions. But since in our approach it is not known beforehand which URIs these expressions should be mapped to, their lexical entries cannot be fully specified. So instead, they contain slots and are built on-the-fly while parsing, based on part-of-speech information provided by the Stanford POS tagger [18], and a set of simple heuristics that specify which POS tag corresponds to which syntactic and semantic properties, such as the following:

- Named entities are noun phrases and are usually modelled as resources, thus a lexical entry is built comprising a syntactic noun phrase representation together with a corresponding semantic representation containing a resource slot.
- Nouns are often referring to classes, while sometimes to properties, thus two lexical entries are built – one containing a semantic representation with a class slot and one containing a semantic representation with a property slot.
- Verbs most often refer to properties, thus a lexical entry with a property slot is built. However, in some cases, the verb does not contribute anything to the query structure (like have in Which cities have more than 2 million inhabitants?), thus an additional entry is built, that does not contain a property slot corresponding to the verb but assumes that the property slot is contributed by a noun (inhabitants in this case).

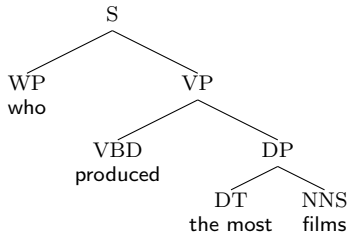
The workflow of template generation thus is the following: The input natural language question is first tagged with part-of-speech information. Then all lexical entries for domain-independent expressions are looked up in the predefined lexicon, and for each expression not covered in this

lexicon are built based on its POS tag and a set of heuristics. For example, processing the question 2a (Who produced the most films?) starts with the tagged input in 5a. The expressions who and the most are domain-independent expressions found in the lexicon, while for produced and films entries need to be build on-the-fly.

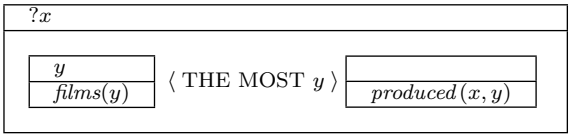
5. (a) `who/WP produced/VBD the/DT most/JJS films/NNS`
   (b) Covered tokens: `who, the most, the, most`
   (c) Building entries for: `produced/VBD, films/NNS`

Now, all lexical entries are input to the parser, which constructs a set of syntactic and corresponding semantic representations of the whole question. An example is given in 6 and 7, respectively.

6.



7.



The semantic representations are finally translated into SPARQL templates, in the case of Who produced the most films? yielding the following two templates (one where the property is contributed by the verb, corresponding to 7, and one where the verb is assumed to be empty and the property is contributed by the noun):

8. (a)
```
SELECT ?x WHERE {
    ?x ?p ?y .
    ?y rdf:type ?c .
}
ORDER BY DESC(COUNT(?y)) LIMIT 1 OFFSET 0
```
Slots:
- ⟨?c, class, films⟩
- ⟨?p, property, produced⟩

   (b)
```
SELECT ?x WHERE {
    ?x ?p ?y .
}
ORDER BY DESC(COUNT(?y)) LIMIT 1 OFFSET 0
```
Slots:
- ⟨?p, property, films⟩

In order to arrive at fully specified SPARQL queries, all slots need to be replaced by appropriate URIs. The mechanism achieving this is explained in detail in the following section.

4. ENTITY IDENTIFICATION

The entity identification problem can be formalized as follows: Given a string s and a knowledge base K, retrieve and assign a score to entities (i.e., classes, instances or properties) that are similar to the input string s. This problem is particularly complex when retrieving properties, as the semantics of a property can be expressed by using a wide variation of natural language expressions. Hence, we use the following entity detection approach: We run our generic approach to entity detection on all labels. In addition, if s stands for a property label, we also compare s with the natural language expressions stored in the BOA pattern library. These two strategies return the highest ranking entities which are then used to fill the query slots. In the following we describe both approaches.

4.1 Generic approach

We begin by identifying the most common synonyms of s. This is carried out by retrieving the union $\mathcal{S}(s)$ of all synsets of s from WordNet. Given this synset, our goal is now to retrieve all entities e with label $label(e)$ from K that abide by the restrictions of the slot which the entity is supposed to fill. We then retrieve the set $\mathcal{E}(s)$ of entities e that are such that their label is highly similar to the elements of the synset $\mathcal{S}(s)$. Formally,

$$\mathcal{E}(s) = \arg\max_{s' \in \mathcal{S}(s)} \sigma(s', label(e)), \quad (1)$$

where the string similarity function σ is the average of the trigram, Levenshtein and substring similarities.

4.2 Property Detection

While the detection of resources and classes can be reduced to a retrieval task, the detection of predicates from natural language is a difficult task. This is mostly due to the large number of expressions that can be used to denote the same predicate. For example, the expressions `X, the creator of Y` and `Y is a book by X` are difficult to match by using synset expansion but they both imply that `X` is the author of `Y`. To address this problem, we make use of the pattern library extracted by the BOA framework[4] [9] in addition to string matching to detect properties. The basic idea behind BOA is to use the considerable amount of instance knowledge available in the Linked Data cloud to compute natural language expressions that stand for predicates from the knowledge base K. By these means, expressions used in natural language questions can be mapped to predicates automatically.

Formally, BOA assumes a set P of predicates `p` for which equivalent natural language expressions are to be detected from an arbitrary input corpus (e.g., Wikipedia, or the Web). For each `p`, BOA begins by computing the set of pairs $\mathcal{I}(\mathtt{p}) = \{(x, y) : (x\ \mathtt{p}\ y) \in \mathcal{K}\}$. BOA searches through the input corpus and retrieves all sentences that contains pairs ($label(\mathtt{x})$, $label(\mathtt{y})$) with $(\mathtt{x}, \mathtt{y}) \in \mathcal{I}(\mathtt{p})$, where $label(r)$ denotes the label of any resource r. From these sentences, it extracts the substrings that match the regular expressions "$label(x)$ * $label(y)$" or "$label(y)$ * $label(x)$". From these substrings, BOA finally generates natural language expressions (NLE) θ of the form `?D? representation ?R?` or `?R? representation ?D?`, where `?D?` resp. `?R?` are placeholders for the labels of x resp. y, i.e., of the entities which matched the domain resp. range of `p`. For example, the NLE `?D?, the creator of ?R?` and `?R? is a book by ?D?` both express the authorship relation.

[4]BOA stands for BOotstrapping linked datA, see `http://boa.aksw.org`

The result of the NLE extraction process is a large number of pairs (p,θ), which we call *BOA patterns*. Distinguishing the patterns that are specific to a given property p is carried out by computing a score based on the following assumptions:

1. A good NLE θ for p is used across several elements of $\mathcal{I}(\mathtt{p})$. This characteristic is modeled by computing the *support* of the pattern.
2. A good NLE θ for p allows to retrieve text segments such that the placeholders `?D?` resp. `?R?` can be matched to labels of entities whose `rdf:type` correspond with `rdfs:domain` resp. `rdfs:range` of p. We call this characteristic *typicity*.
3. A good NLE θ is used exclusively to express p, i.e, it occurs in a small number of pattern mappings. We call this last characteristic *specificity*.

To be able to compute these characteristics of good patterns numerically, BOA collects the following supplementary information during the NLE extraction process:

- the number of sentences that led to θ and that contained $label(x)$ and $label(y)$ with $(x, y) \in \mathcal{I}(\mathtt{p})$, which we denote $l(x, y, \theta, \mathtt{p})$, and
- $\mathcal{I}(\mathtt{p}, \theta)$, the subset of $\mathcal{I}(\mathtt{p})$ which contains only pairs (s, o) that led to θ.

4.2.1 Support

We calculate the support $sup(\theta, \mathtt{p})$ of the pattern θ for the predicate p as the product of the number of subject-object pairs the pattern has been learned from and the maximum value for a single subject-object pair:

$$sup(\theta, \mathtt{p}) = \log\left(\max_{(s,o)\in\mathcal{I}(\mathtt{p})} l(s, o, \theta, \mathtt{p})\right) \log(|\mathcal{I}(\mathtt{p}, \theta)|). \quad (2)$$

Since both components of the support follow a long-tail distribution, we use the logarithm to reduce the boosting of very popular patterns.

4.2.2 Typicity

A pattern θ is considered to display a high typicity with respect to a predicate p if its placeholders ?D? and ?R? match only labels of entities whose `rdf:type` matches the range and domain restrictions of p in the reference corpus. Let d resp. r be functions that map each p to its `rdfs:domain` resp. `rdfs:range`. Furthermore, let $d(\theta, s)$ resp. $r(\theta, s)$ be functions which map the class of the named entity used to substitute ?D? resp. ?R in the pattern θ for the given sentence s. Finally, let the function $\delta(x, y)$ be Kronecker's delta function, which returns 1 if $x = y$ and 0 in all other cases. We define the typicity of θ as

$$typ(\theta, \mathtt{p}) = \sum_{s\in S}\left(\frac{\delta(d(\mathtt{p}), d(\theta, s)) + \delta(r(\mathtt{p}), r(\theta, s))}{2|S|}\right) \log(|S|+1), \quad (3)$$

where S is the set of sentences used to evaluate the typicity of θ. Note that the first term of the typicity is simply the precision of the pattern. We multiply this factor with the logarithm of (|S|+1) to prevent overly promoting patterns which have a low recall, i.e., patterns that return only a small number of sentences.

4.2.3 Specificity

A NLE θ is considered to be specific if it is used to expressed a small number of predicates p. We adapted the idea of inverse document frequency (*idf*) as known from Information Retrieval to capture this characteristic. The specificity $spec(\theta)$ is thus given by the following expression:

$$spec(\theta) = \log\left(\frac{|P|}{|M(\theta)|}\right), \quad (4)$$

where $M(\theta)$ is the set of predicates of which θ is a NLE.

All three equations can now be combined to the global confidence score $c(\theta, \mathtt{p})$ used by BOA as shown in Equation 5:

$$c(\theta, \mathtt{p}) = sup(\theta, \mathtt{p}) \cdot typ(\theta, \mathtt{p}) \cdot spec(\theta). \quad (5)$$

5. QUERY RANKING AND SELECTION

After identifying entities that could fill the slots of a template, we arrive at a range of possible SPARQL queries. The task now is to rank these queries and to pick one, which is then used to retrieve the answer to the input question.

The goal of the query ranking step is to provide a function for deciding on the order of execution of queries that possibly match a question. Given a slot that is to be filled, we compute two scores for each possible entity e that can be used to fill a slot: a similarity score and a prominence score. The similarity score $\sigma(e)$ is the string similarity used during the entity detection phase. The prominence score $\varphi(e)$ is given by

$$\varphi(e) = \begin{cases} \log_2|\{(\mathtt{x}, \mathtt{y}) : \mathtt{x\ e\ y}\}| & \text{if e is a property} \\ \log_2|\{(\mathtt{x}, \mathtt{y}) : \mathtt{x\ y\ e}\}| & \text{else,} \end{cases} \quad (6)$$

where `x e y` holds when this triple can be found in the reference knowledge base K. The final score $score(e)$ of each entity is then definded as

$$score(e) = \alpha \max_{s'\in\mathcal{S}(s)} \sigma(s', label(e)) + (1 - \alpha)\varphi(e), \quad (7)$$

where $\alpha \in [0, 1]$ decides on the impact of similarity and prominence on the final score of each entity.

The score of a query is computed as the average of the scores of the entities used to fill its slots. In addition to this, we perform type checks on queries: We first extract all triple patterns of the form `?x rdf:type c` in the query, where `?x` stands for a variable and `c` for a class. We compute types$(?x, q) = \{c \mid (?x, \mathtt{rdf:type}, c) \in TP(q)\}$ where TP stands for the set of triple patterns in the considered query q. For each such variable, we search for triple patterns `?x p e` and `e p ?x` in the query. In the former case, we check whether the domain of the property p is disjoint with an element of types$(?x, q)$. In the latter case, we perform the same check with the range of p. If any of these type checks fails, the query is rejected. We perform this to avoid queries, which do not follow the schema of the knowledge base, but could still return results because of modelling errors in the data.

Once a ranked list of SPARQL queries is available, we need to decide which of those queries should be returned as answer. If only the highest ranking query would be returned, the problem arises that most of those queries actually do not return a result. The reason for this is that the query ranking method can only take limited information into account for reasons of efficiency. It uses string similarity, prominence

of entities and the schema of the knowledge base to score a query. However, this does not guarantee that the combination of triple patterns in a query is meaningful and leads to a non-empty result. Therefore it is necessary to execute and test queries before returning a result to the user. Our system returns the highest scored query with a non-empty result. A special case are COUNT queries: In most of those queries, a return value of 0 is also discarded in our method, since this usually means that the WHERE clause of the corresponding SPARQL query does not yield a match in the considered RDF graph.

6. EVALUATION AND DISCUSSION

The evaluation is based on the QALD[5] benchmark on DBpedia[6] [10]. It comprises two sets of 50 questions over DBpedia, annotated with SPARQL queries and answers. Each question is evaluated w.r.t. precision and recall defined as follows:

$$\text{Recall} = \frac{\text{number of correct resources returned by system}}{\text{number of resources in gold standard answer}}$$

$$\text{Precision} = \frac{\text{number of correct resources returned by system}}{\text{number of resources returned by system}}$$

Before we turn to the evaluation results, one important preliminary remark: The reported results are results based on natural language questions tagged with ideal part-of-speech information. The reason is that questions often lead to POS tagging errors. For example, in Which films did Leonardo di Caprio star in, the infinitive verb form star is tagged as a noun by the Stanford POS tagger as well as the Apache OpenNPL[7] POS tagger, which leads to a parse failure. The same holds for a range of infinitives such as play, border, die, cross and start. In order to separate such external errors from errors internal to our approach, we manually corrected erroneous POS tags in seven questions, that otherwise would not have been parsed. But this is only a temporal solution, of course; the next step is to train a POS tagger model on a corpus containing a sufficient amount of questions.

6.1 Evaluation results

Of the 50 training questions provided by the QALD benchmark, 11 questions rely on namespaces which we did not incorporate for predicate detection: FOAF[8] and YAGO[9]. Especially the latter poses a challenge, as YAGO categories tend to be very specific and complex (e.g., `FemaleHeadsOfGovernment` and `HostCitiesOfTheSummerOlympicGames`). We did not consider these questions, thus only 39 questions are processed by our approach. Of these 39 questions, 5 questions cannot be parsed due to unknown syntactic constructions or uncovered domain-independent expressions. This mainly concerns the noun phrase conjunction as well as and ordinals (the 5th, the first). These constructions will be added in the future; the only reason they were not implemented yet is that they require significant additional effort when specifying their compositional semantics.

[5] `http://www.sc.cit-ec.uni-bielefeld.de/qald`
[6] `http://dbpedia.org`
[7] `http://incubator.apache.org/opennlp/`
[8] `http://www.foaf-project.org/`
[9] `http://www.mpi-inf.mpg.de/yago-naga/yago/`

Of the remaining 34 questions, 19 are answered exactly as required by the benchmark (i.e. with precision and recall 1.0) and another two are answered almost correctly (with precision and recall > 0.8). Figure 3 at the very end of the paper lists the results of each of the 39 processed questions.

The mean of all precision scores is therefore 0.61 and the mean of all recall scores is 0.63, leading to an F-measure[10] of 0.62. These results are comparable with those of systems such as FREyA and PowerAqua. The key advantage of our system is that the semantic structure of the natural language input is faithfully captured, thus complex questions containing quantifiers, comparatives and superlatives pose no problem, unlike in PowerAqua. Moreover, our system does not need any user feedback, as FREyA does.

6.2 Discussion

In the following, we identify the main sources of errors and discuss how they can be addressed in future work.

In the examples given in this section, we will use the following abbreviations for relevant DBpedia namespaces:

- `res` for `<http://dbpedia.org/resource/>`
- `onto` for `<http://dbpedia.org/ontology/>`
- `prop` for `<http://dbpedia.org/property/>`

Incorrect templates

It only very rarely happens that a parse is found but no sensible template is constructed. However, it does happen that none of the constructed templates captures the structure of the data. One example is question 36 (Is there a video game called Battle Chess?), where the generated template assumes a property slot *title* or *name* corresponding to the participle called; however, none such property exists in DBpedia. The appropriate property `rdfs:label`, on the other hand, is not part of the index and thus is not found by the predicate detection algorithm.

Incorrect templates are most eminent when the semantic structure of the natural language question does not coincide with the triple structure of the target query. For example, the phrase join the EU would lead to a template containing a property slot *join* related to the resource *EU*; the appropriate property in DBpedia, however, is `prop:accessioneudate`. The same structural mismatch would arise with complex YAGO caterogies. Cases like these suggest that the fixed structure of the templates is sometimes too rigid. We are currently working on two solutions to this problem, see Section 9 below.

Another reason for incorrect templates is the sporadic failure of named entity recognition. E.g., if a phrase like Battle of Gettysburg is not recognized as a named entity, no resource slot is built – instead the template would contain a slot for a class *battle* related to an entity *Gettysburg*, which does not lead to a meaningful result.

Entity identification

Errors due to entity identification occur when a resource, class or property cannot be found on the basis of the slot. These are the most frequent errors in our approach.

A particularly hard case for entity identification is when a property in the intended target query does not have a correspondent in the natural language question. This is the

[10] $(2 \times \text{precision} \times \text{recall})/(\text{precision} + \text{recall})$

case in questions 11 (Give me all soccer clubs in the Premier League) and 29 (Give me all movies with Tom Cruise). The templates constructed for these questions contain a property slot that arises from the prepositions in and with; the correct properties `onto:league` (for 11) and `onto:starring` (for 29), however, could be found only by inferences on the basis of Premier League and films. This type of inferences is not part of our approach at the moment.

Examples for entities which do have a correspondent in the natural language input but are nevertheless hard to match are the following:

- inhabitants, the correct property being `prop:population` or `prop:populationTotal` (question 9)
- owns, the property specified in the Gold query being `onto:keyPerson` (question 10)
- higher, the target property being `prop:elevationM` (question 33)

These cases would require the incorporation of additional semantic similarity measures, such as *Explicit Semantic Analysis* [8].

Query selection

Sometimes the correct entity is among the entity candidates, but still a query with the wrong entity instantiating the slot is picked. An example of this is question 32 (Who wrote The pillars of the Earth?). The expression wrote is matched with the property `onto:writer`, as this is higher ranked than the property `onto:author`. Using the former, the name The pillars of the Earth is incorrectly matched with `res:The_Pillars_of_the_Earth_(TV_Miniseries)` because it gives a non-empty result in combination with `onto:writer`.

Another case in which the wrong entity is picked is when the slot contains too little information in order to decide among candidates. E.g., there are three questions (24, 41, 44) containing the participle founded. There are several candidates of properties that founded could correspond to, e.g. `prop:foundation`, `prop:foundingYear`, `prop:foundingDate`, `onto:foundationPerson`, `onto:foundationPlace`. Without a hint about the intended range of the property, the decision for one of these properties has to be quite arbitrary. In order to capture these cases, slots would need to comprise more information, e.g. also specify the property's range, in order to distinguish constructions like founded in 1950, founded in California and founded by Goofy. A first step towards this goal is already implemented: In case the argument is a numeral or the question contains a wh-word like when or where, the slot contains the information that a date or place is intended (thus question 41 works fine and for question 24 a sensible template is built, although it fails due to query ranking and selection).

Other reasons

In some cases our approach is doing the right thing, however not, or only partially, matching the Gold query. One example is question 13 (What languages are spoken in Estonia?). The target query specified in the Gold standard contains a union of countries related to Estonia via the property `onto:language` and countries related to Estonia via the property `onto:spokenIn`. Our approach finds the former property and stops, thus misses the latter and thereby achieves 1.0 precision but a lower recall. The solution would be to perform an exhaustive search, i.e. not stopping after one successful query is found.

Another example is question 38, which asks for the country with the most official languages. Our approach choses the property `onto:officialLanguage`, while the Gold query uses the more general (and arguably less appropriate) property `onto:language`.

In general, question answering over DBpedia has to face the challenge of two schemas – a manually created ontology modelling mostly neat and consistent data in the `ontology` namespace, and an automatically created one modelling a large amount of quite noisy data in the `property` namespace. The namespaces partly overlap and chosing one over the other often leads to different results of different quality.

7. PROTOTYPE

A prototype for the described algorithm was implemented and deployed, see Figure 2. It is a freely accessible web application, which allows a user to enter natural language questions. The answers are shown in a tabular view if appropriate. The view allows the user to enrich the generated answers by displaying further appropriate property values for the returned resources. Interesting queries can be saved and reused by other users.

For the prototype, we used DBpedia as underlying knowledge base. To be able to use the mentioned techniques, some components were created offline: Separate Lucene indices were created for resources, properties and classes by querying for the labels of those elements in the used DBpedia triple store. Additionally, a BOA index was created for properties, since it vastly improves the mapping of properties in natural language queries compared to using a text index. The same approach can be applied to other knowledge bases and we plan to evaluate this in future work.

8. RELATED WORK

Several approaches have been developed for the purpose of question answering.

PowerAqua is a question answering system over Linked Data that is not tailored towards a particular ontology; especially it does not make any assumptions about the vocabulary or structure of datasets. The main focus of the system is to combine and merge data from different sources, focusing on scalability, and using iterative algorithms, filtering and ranking heuristics to limit the search space. PowerAqua is therefore very strong on large, heterogeneous datasets, although it does struggle on complex mappings such as the aforementioned YAGO categories. For a detailed explanation of the system's architecture and an evaluation see, e.g., [15, 13]. The major shortcoming of PowerAqua is its limited linguistic coverage. In particular, PowerAqua fails on questions containing the most (such as question 31), and more than (such as question 12), which pose no problem for a system with a deeper linguistic analysis of the input question.

Pythia [22] is such a system. It relies on a deep linguistic analysis (on which the approach based in this paper is based) and can therefore handle linguistically complex questions, in particular questions containing determiners such as the most and more than. Pythia's major drawback is that it requires a lexicon, which up to this moment has to be created manually. It therefore fails to scale to very large datasets.

The approach proposed in this paper tries to combine both

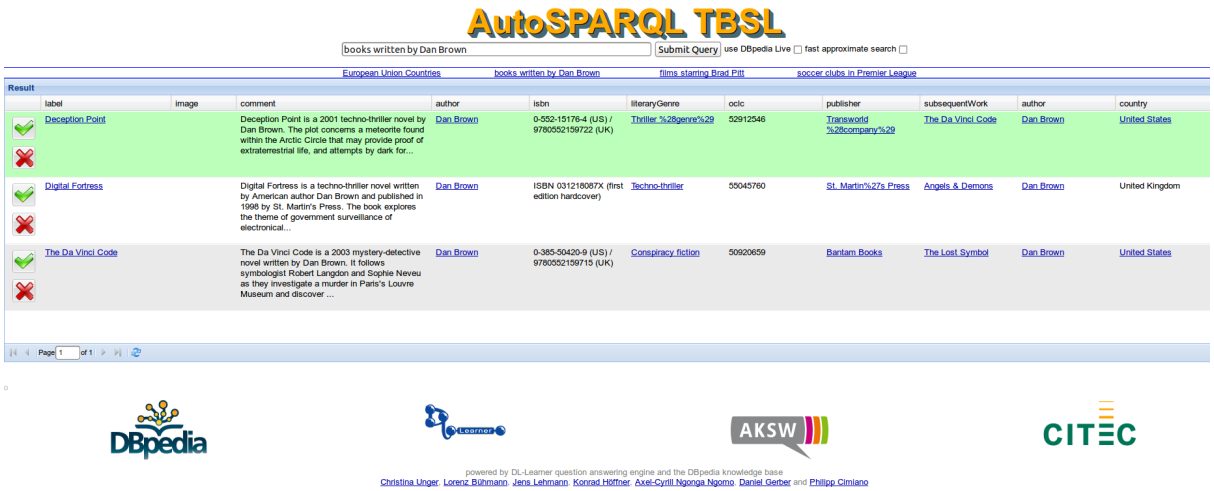


Figure 2: Screenshot of prototype available at `http://autosparql-tbsl.dl-learner.org`.

a deep linguistic analysis with the flexibility of approaches focusing on matching natural language questions to RDF triples. The triple structure is derived from the semantic structure of the question.

Another possibility to determine the triple structure is by exploration of the dataset, as in the question answering system *FREyA* [2, 3]. However, FREyA partly relies on the user's help in selecting the entity that is most appropriate as match for some natural language expression. The drawback of such an approach is that the naive end-user is often not informed about the modeling and vocabulary of the data and thus is not able to help.

Further approaches related to question answering over Linked Data include, e.g., *Treo* [7], which combines entity search, semantic relatedness and spreading activation for exploring RDF data, and *Ontolook* [12], which focuses on relation-based search. In addition to question answering, keyword-based approaches have been gaining momentum over the past years. This led to semantic search engines, such as Swoogle [5], Watson [4], Sigma [20] and Sindice [21], which aim to index RDF across the Web and make it available for entity search. The approaches described in [17] and [19] extend upon the paradigm of simple entity search and try to generate interpretations of keyword queries which exploit the semantics available on the Linked Data Web. Especially, [19] implements a graph exploration approach to detect subgraphs of the input knowledge base that can be used to compute an answer to the user's query. On the other hand, [17] uses schema knowledge to infer SPARQL queries that represent possible interpretations of the user-given keywords.

9. CONCLUSION AND FUTURE WORK

We presented a novel approach to question answering over Linked Data that relies on a deep linguistic analysis yielding a SPARQL template with slots that need to be filled with URIs. In order to fill those slots, possible entities were identified using string similarity as well as natural language patterns extracted from structured data and text documents. The remaining query candidates were then ranked and, on the basis of scores attached to the entities, one of them was selected as final result.

One of the strengths of this approach is that the generated SPARQL templates capture the semantic structure of the natural language input. Therefore questions containing quantifiers like the most and more than, comparatives like higher than and superlatives like the highest do not pose a problem – in contrast to most other question answering systems that map natural language input to purely triple-based representations.

However, in some cases the semantic structure of the question and the triple structure of the query do not coincide, thus faithfully capturing the semantic structure of the input question sometimes leads to too rigid templates. We are currently exploring two approaches to solve this problem. The first one concentrates on more flexible processing. On the one hand side, we are considering a preprocessing step that can detect complex (especially YAGO) categories before parsing the natural language question. On the other hand side, we are investigating the relaxation of templates, such that the triple structure is not completely fixed but is discovered through exploration of the RDF data.

The second approach concerns incorporating a more flexible fallback strategy in case no successful SPARQL query is found. In particular, we are working on combining our approach with active learning methods as described in [11]. Active learning allows the user to give feedback on the presented query results, i.e. the user can say whether particular query results are incorrect and/or whether further results should be returned. This will allow two enhancements over the presented question answering system: First, if the returned answers are incorrect or incomplete, then the user can indirectly modify the query via his feedback. And second, if our approach cannot generate a query at all, then the system can still recover by allowing the user to specify one or more query results. This procedure can be assisted with standard search and disambiguation methods.

Once these enhancements are in place, i.e. once the shortcomings mentioned in Section 6.2 are addressed, we will evaluate our approach on a larger scale, for example using the data provided by the second instalment of the QALD open challenge, which comprises 100 training and 100 test ques-

tions on DBpedia, and a similar amount of questions on MusicBrainz. In particular, we will test how well our approach carries over to different types of domains. Additionally, we plan to conduct a small usability study.

Ultimately, our goal is to provide robust question answering for large scale heterogeneous knowledge bases. Our vision is that this robustness can help to make the usage of question answering systems a standard task in everyday life in a similar but more powerful way as web search.

10. REFERENCES

[1] H. Cunningham D. Damljanovic, M. Agatonovic. Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010), Heraklion, Greece, May 31-June 3, 2010*. Springer, 2010.

[2] D. Damljanovic, M. Agatonovic, and H. Cunningham. Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. In *ESWC 2010*, volume 6088 of *LNCS*, pages 106–120. Springer, 2010.

[3] D. Damljanovic, M. Agatonovic, and H. Cunningham. FREyA: An interactive way of querying Linked Data using natural language. In *Proceedings of the 1st Workshop on Question Answering over Linked Data (QALD-1), ESWC 2011*, 2011.

[4] M. d'Aquin, E. Motta, M. Sabou, S. Angeletou, L. Gridinoc, V. Lopez, and D. Guidi. Toward a new generation of Semantic Web applications. *Intelligent Systems, IEEE*, 23(3):20–28, 2008.

[5] L. Ding, T.W. Finin, A. Joshi, R. Pan, R. Scott Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the Semantic Web. In David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *CIKM*, pages 652–659. ACM, 2004.

[6] L. Fischer E. Kaufmann, A. Bernstein. NLP-Reduce: A "naive" but domain-independent natural language interface for querying ontologies. In *Proceedings of the 4th European Semantic Web Conference (ESWC 2007), Innsbruck, Austria*, 2007.

[7] A. Freitas, J.G. de Oliveira, S. O'Riain, E. Curry, and J.C. Pereira da Silva. Querying Linked Data using semantic relatedness: A vocabulary independent approach. In *Proceedings of the 16th International Conference on Applications of Natural Language to Information Systems (NLDB)*, 2011.

[8] E. Gabrilovich and S. Markovitch. Computing semantic relatedness using Wikipedia-based Explicit Semantic Analysis. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI), Hyperabad, India*, 2007.

[9] D. Gerber and A.-C. Ngonga Ngomo. Bootstrapping the Linked Data Web. In *WekEx@ISWC*, 2011.

[10] J. Lehmann, C. Bizer, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia – A crystallization point for the Web of Data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[11] J. Lehmann and L. Bühmann. AutoSPARQL: Let users query your knowledge base. In *Proceedings of ESWC 2011*, volume 6643 of *Lecture Notes in Computer Science*, pages 63–79, 2011.

[12] Y. Li, Y. Wang, and X. Huang. A relation-based search engine in Semantic Web. *IEEE Trans. Knowl. Data Eng.*, 19(2):273–282, 2007.

[13] V. Lopez, M. Fernandez, E. Motta, and N. Stieler. PowerAqua: Supporting users in querying and exploring the Semantic Web. *Semantic Web Journal*, In Press (2011).

[14] V. Lopez and E. Motta. Ontology driven question answering in AquaLog. In *Proceedings of the 9th International Conference on Applications of Natural Language to Information Systems (NLDB 2004), Manchester, England*, 2004.

[15] V. Lopez, A. Nikolov, M. Sabou, V. Uren, and E. Motta. Scaling up question-answering to Linked Data. In *Proceedings of Knowledge Engineering and Knowledge Management by the Masses (EKAW-2010), Lisboa, Portugal*, 2010.

[16] Y. Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, University of Pennsylvania, 1990.

[17] S. Shekarpour, S. Auer, A.-C. Ngonga Ngomo, D. Gerber, S. Hellmann, and C. Stadler. Keyword-driven SPARQL query generation leveraging background knowledge. In *International Conference on Web Intelligence*, 2011.

[18] K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 2003*, pages 252–259, 2003.

[19] Thanh Tran, Tobias Mathäß, and Peter Haase. Usability of keyword-driven schema-agnostic search. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *ESWC (2)*, volume 6089 of *Lecture Notes in Computer Science*, pages 349–364. Springer, 2010.

[20] G. Tummarello, R. Cyganiak, M. Catasta, S. Danielczyk, R. Delbru, and S. Decker. Sig.ma: Live views on the Web of Data. *Journal of Web Semantics*, 8(4):355–364, 2010.

[21] G. Tummarello, R. Delbru, and E. Oren. Sindice.com: Weaving the Open Linked Data. pages 552–565, 2007.

[22] C. Unger and P. Cimiano. Pythia: Compositional meaning construction for ontology-based question answering on the Semantic Web. In *Proceedings of the 16th International Conference on Applications of Natural Language to Information Systems (NLDB 2011)*, 2011.

[23] E. Motta V. Lopez, V. Uren and M. Pasin. AquaLog: An ontology-driven question answering system for organizational semantic intranets. *Journal of Web Semantics*, 5(2):72–105, 2007.

[24] V. Uren V. Lopez, M. Sabou and E. Motta. Cross-ontology question answering on the Semantic Web – an initial evaluation. In *Proceedings of the Knowledge Capture Conference, 2009, California*, 2009.

id	question	precision	recall
2	Who has been the 5th president of the United States of America		
4	Who was Tom Hanks married to	1.0	1.0
5	Which people were born in Heraklion	0.91	1.0
7	Which companies work in the aerospace industry as well as on nuclear reactor technology		
8	Which people have as their given name Jimmy		
9	Who developed the video game World of Warcraft	1.0	1.0
10	Who was the wife of president Lincoln	1.0	1.0
12	Which caves have more than 3 entrances	1.0	1.0
13	Which cities have more than 2000000 inhabitants	0.04	0.26
14	Who owns Aldi		
16	Give me all soccer clubs in the Premier League	0.5	0.86
17	In which programming language is GIMP written	1.0	1.0
18	What languages are spoken in Estonia	1.0	0.14
20	Which country does the Airedale Terrier come from	1.0	1.0
21	What is the highest mountain	1.0	1.0
24	Which organizations were founded in 1950	0.0	0.0
25	Which genre does DBpedia belong to	1.0	1.0
26	When was DBpedia released	1.0	1.0
27	Who created English Wikipedia	1.0	1.0
28	Which companies are located in California USA	0.8	0.76
30	How many films did Leonardo DiCaprio star in	1.0	1.0
31	Who produced the most films	1.0	1.0
32	Is Christian Bale starring in Batman Begins	1.0	1.0
33	Which music albums contain the song Last Christmas		
34	Give me all films produced by Hal Roach	1.0	1.0
35	Give me all actors starring in Batman Begins	1.0	0.86
36	Give me all movies with Tom Cruise	0.08	0.75
37	List all episodes of the first season of the HBO television series The Sopranos		
38	Which books were written by Danielle Steel	1.0	1.0
39	Who wrote the book The pillars of the Earth	0.5	1.0
40	Which mountains are higher than the Nanga Parbat	0.0	0.0
41	When was Capcom founded	1.0	1.0
42	Which software has been published by Mean Hamster Software	1.0	1.0
43	Is there a video game called Battle Chess	0.0	0.0
44	Which software has been developed by organizations founded in California		
45	Which country has the most official languages	0.0	0.0
47	Is Natalie Portman an actress	1.0	1.0
48	Who produced films starring Natalie Portman	1.0	1.0
49	In which films did Julia Roberts as well as Richard Gere play		

Figure 3: This table shows precision and recall values for each processed question (i.e. all questions that do not require the YAGO or FOAF namespace). For questions with no precision and recall specified, no query was constructed. Questions printed in cells with red background were not parsed, questions in white cells succeeded and for questions in lightgray cells queries with quality equal or close to the Gold query were built, while questions in yellow cells fail due to a query selection problem and questions in orange cells fail due to some entity identification problem.