Offline Question Answering over Linked Data using Limited Resources

Paramjot Kaur[1], Vincent Blücher[3], Rricha Jalota[1], Diego Moussallem[1,3]
Axel-Cyrille Ngonga Ngomo[1,3], Ricardo Usbeck[1,2,3][0000−0002−0191−7211]

[1] Data Science Group, University of Paderborn, Germany
[2] Fraunhofer IAIS, Standort Dresden, Germany
[3] Leipzig University, Germany
first.lastname@upb.de

Abstract. Question Answering over Linked Data provides concise information to the user from a natural language request instead of flooding them with documents. However, the accessibility of Linked Data resources, e.g., SPARQL endpoints, is bound to an online connection. We present OQA, the first offline Question Answering system over Linked Data for mobile devices. We built OQA with the limited resources of an Android mobile device, such as battery power, computational power, or memory consumption in mind. Our OQA system has three main components: 1) question analysis and 2) query generation which identify the type of the question and reform it into a semantically meaningful data structure, i.e., a SPARQL query. Finally, the 3) query execution uses a novel mobile triple store, implemented with RDF4J. Our evaluation suggests that OQA is feasible for daily use in terms of battery consumption and able to answer domain-specific questions with up to 72% accuracy.

1 Introduction

The goal of our offline mobile Question Answering (QA) system is to answer a spoken or typed user question without a connection to a high-performance server using only the resources of a mobile device such as a smartphone. The OQA system was built to explore research challenges in QA, e.g., workers asking questions in steel factory buildings or tourists walking through buildings and inner cities with weak internet coverage. Most mobile devices today are limited in their resources w.r.t. CPU, memory, or storage. Thus the components of the mobile QA have to be efficient as well as effective.

In this demo, we present OQA, the first offline QA algorithm which uses 1) an own mobile Linked Data (LD) triple store, 2) a simple but yet effective algorithm for the transformation of natural language to SPARQL which does not require machine learning and 3) focuses on a low resource, i.e., battery and storage consumption. All software is publicly available on our GitHub repository[4] as well as a video of the App in the README. The latest release of OQA on Android is also available online.[5]

[4] https://github.com/dice-group/qamel/tree/master/app
[5] https://github.com/dice-group/qamel/releases

2 Related Work

Despite the plethora of QA over LD approaches, none focuses on offline capabilities or using limited resources. Offline Question Answering connects to QA over LD as well as the storage of LD resources in triple stores on mobile devices. Due to space limitations, we only present a brief overview of both areas. Note, we use LD because its underlying graph structure is more concise than a textual corpus and thus easier to ship and customize for particular use cases while being semantically unambiguous. There are several high-quality, but computationally expensive multilingual and rule-based QA approaches such as WDAqua [2]. WDAqua uses a combinatorial approach, which is computationally expensive, to formulate SPARQL queries by leveraging the semantics of a given underlying knowledge base. We refer the interested reader to an extensive survey of the field [3]. DeQA [1] is an on-device QA system to run locally on mobile devices with a set of latency and memory optimizations that can be applied without requiring any changes to the QA system. For storing LD resources on mobile devices, various open-source solutions exist, e.g., Mobile LD, Microalign, TriplePlace, and Androjena.[6]. All solutions are written in Java but lack an active community or were last updated before 2013. Also, some have only proprietary licenses and thereby cannot be considered as viable options for OQA.

3 The OQA system

OQA is based on 1) linguistic and semantic analysis of the input, and a subsequent 2) classification of the question type to assign a template. Finally, OQA 3) fills the template and executes it against the mobile triple store. Before describing our system, we introduce the creation of our **mobile triple store**, dubbed RDF4A. We based the development on RDF4J[7], which is an open-source Java framework for processing LD data which has an active community. We ported RDF4J to Android using version 1.0.3, which is a backport of the current RDF4J version for Java 1.7 supported on Android. To reduce the storage footprint, OQA creates subsets of LD datasets. OQA uses two synchronizers to transport LD to a mobile device. (1) **Server-side synchronizer:** OQA reads an RDF file as input and decides for each triple if it is relevant or not, e.g., based on the frequency of the contained entities. That is, a triple is considered to be relevant if all three entries of a triple have at least n occurrences. This mechanism can be modified in the future, e.g., to extract only user-relevant parts of an RDF graph or to contain continuous updates. The target triples are stored as an RDF4J SailRepository, which then is gzipped for distribution. Such offline data packages are identified by a hash code. (2) **Mobile-side synchronizer:** If the mobile device is connected to the internet, OQA downloads the offline data package and imports it into RDF4A.

[6] QAMEL report `https://tinyurl.com/QAMEL-Report`
[7] `http://rdf4j.org/`

We use the following question over the DBpedia 2016-04 as running example: *When was the Leipzig University founded?*, cf. Figure 1. Note, to highlight research challenges, OQA focuses on simple questions containing exactly one binary relation, which are the most used questions in voice-driven apps. Also, all processes are designed to be multilingual and use efficient. To this end, we rely on deterministic algorithms to keep the computational complexity low.

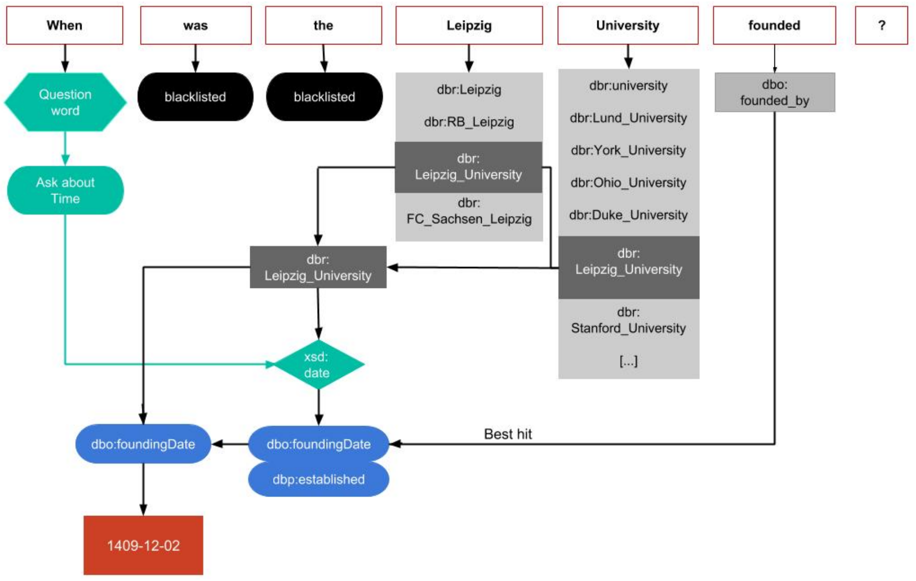

Fig. 1. Overview of the OQA algorithm w.r.t. our running example.

Preprocessing: OQA chunks the question into individual tokens separated by white-space and removes stop words and single-character tokens. For our running example, we are left with "When Leipzig University founded".

Question Type Determination: OQA determines the type of question and the associated type of answer based on the question word. For example, time questions are represented by "When" and the unknown people question type can be represent by "Who, What, Which". This helps OQA to reduce the number of candidates for possible slots significantly. For our running example, we figure out that we are looking for an answer of type *time* and remove *When* from the list of tokens. The resulting query is "Leipzig University founded".

Entity Candidates: We assume that either an entity, a literal or a property is missing in the binary relation. Thus, we perform a look-up in the mobile triple store. We try to assign each token to one or several entity candidates without using an additional dictionary, entity linking algorithm or other computational expensive processes. OQA exploits the `rdfs:label` property using the following query:

```
SELECT DISTINCT ?x ?z WHERE { ?x rdfs: label ?z .
    FILTER (regex (str(?x), ".* < TOKEN >.*") && lang (?z)='en ') }
```

This query returns all entities containing the token in their label. For the tokens left over in our running example "Leipzig University founded", the entity candidate finding would generate the results in Table 1. For "founded", our query returns only dbo:foundedBy but not dbo:foundingDate. Thus, we need to find better property candidates in the next step.

Table 1. Entity search excerpt for our running example.

TOKEN	Label	URI
Leipzig	Leipzig	http://dbpedia.org/resource/Leipzig
founded	founded by	http://dbpedia.org/ontology/foundedBy

Candidate Ranking - For each result from the query, a tuple $t = (W, E, L)$ consisting of the input token ($w \in W$), and the resulting entity ($e \in KB$) as well as label ($l = label(e)$) is stored. OQA first ranks by the number of token $w_i \in W$ that have found the same entity e_j. , formally: $rank(e_j) = |\{ t|t = (w_i, \_, \_)\}|$. For example, dbr:Leipzig_University has a higher priority in the question "When was the Leipzig University founded?" as dbr:Leipzig since it is found by both "Leipzig" and "University". In case of a tie exists, we sort by the Levenshtein distance between the label of an entity and the words it is covering. If a tie still exists, we use the frequency of the entity in the knowledge base, i.e., its popularity as a third comparison criterion. After this step, there is a list of entities sorted by priority. Note,OQA also assigns properties a priority.

Property Candidates and Ranking: The literal and entity values for a subject-property or property-object pair have specific data types. Using the question type mapping, we can significantly reduce the number of possible answers. Table 2 shows the assignments between question types and data types based on the `http://dbpedia.org` ontology. This list can be extended to cover different LD knowledge bases and their ontologies from various domains. Starting with the highest ranked entity candidate, all properties for this candidate are retrieved from the mobile triple store. The labels of these properties are compared via the Levenshtein distance with each word of the question. If a property is found which was already found in the entity candidate search phase, its reliability value is increased proportionally. Finally, the list of entity-property pairs

Table 2. Question type to data type mapping.

Question Type	time	place	discrete	numerical number	person
Data Type	xsd:date xsd:Year	xsd:place dbo:Place	xsd:integer xsd:nonNegative Integer	xsd:float xsd:double xsd:decimal	foaf:person dbo:Person

is sorted according to their reliability value. By executing the SPARQL query with the missing slot as a variable against the mobile triple store we retrieve the final result.

4 Evaluation

The evaluation is twofold. First, a use case driven dataset about Cologne is used. OQA was able to answer 32 out of 44 questions on the mobile device (72% accuracy).[8] Second, we include preliminary results for the battery consumption. To test the battery consumption, we used Battery Historian[9]. OQA is only consuming 5.28% battery in offline mode for answering 200 questions using the QALD-9 [4] dataset on a reduced version of roughly 120 MB DBpedia data.

5 Summary

We presented OQA, an offline question answering system over Linked Data on mobile devices. OQA is lightweight and able to work on questions with incorrect grammar. The OQA system is easily extensible to other languages as it relies on simple look-ups only. In the future, we plan to implement caching to reduce battery consumption further and to evaluate the quality of OQA against well-known benchmarks and analyze further the resource consumption choke points. We also plan to add a feature of auto-correction of incorrect grammar. By presenting this demo at SEMANTiCS, we will collect feedback from users and discuss future research directions to bring Linked Data-based application to the masses.

Acknowledgments This work was supported by the EuroStars project QAMEL (no. 01QE1549C) and by the German Federal Ministry of Transport and Digital Infrastructure (BMVI) through the project LIMBO (no. 19F2029I).

References

1. Q. Cao, N. Weber, N. Balasubramanian, and A. Balasubramanian. Deqa: On-device question answering. 2019.
2. D. Diefenbach, K. D. Singh, and P. Maret. WDAqua-core0: A Question Answering Component for the Research Community. In *Semantic Web Challenges*, pages 84–89, 2017.
3. K. Höffner, S. Walter, E. Marx, R. Usbeck, J. Lehmann, and A. N. Ngomo. Survey on challenges of question answering in the semantic web. *Semantic Web*, pages 895–920, 2017.
4. R. Usbeck, R. H. Gusmita, A. N. Ngomo, and M. Saleem. 9th Challenge on Question Answering over Linked Data (QALD-9). In *Joint proceedings of SemDeep-4 and NLIWOD-4*, pages 58–64, 2018.

[8] `https://tinyurl.com/QAMEL-Accuracy-Report`

[9] `https://developer.android.com/studio/profile/battery-historian`