The Question Answering System GeoQA2

Dharmen Punjani[1], Sergios-Anestis Kefalidis[2], Konstantinos Plas[2], Eleni Tsalapati[2], Manolis Koubarakis[2], and Pierre Maret[1]

[1] Hubert Curien Laboratory, Universite Jean Monnet, St. Etienne, France
dharmen.punjani@gmail.com, pierre.maret@univ-st-etienne.fr
[2] National and Kapodistrian University of Athens, Greece
{s.kefalidis, etsalapati, koubarak}@di.uoa.gr

Abstract. We present the geospatial question answering engine GeoQA2 which is the most recent version of the engine GeoQA, originally proposed by Punjani et al. GeoQA2 has been designed to work over the geospatial knowledge graph YAGO2geo. GeoQA2 improves GeoQA by being able to answer more complex geospatial questions (e.g., questions involving aggregates and arithmetic comparisons), having more effective individual components (e.g., for named entity recognition and disambiguation), supporting more templates for generating GeoSPARQL queries, and executing these queries more efficiently by using a technique based on materializing topological relations.

Keywords: geospatial question answering engines · geospatial knowledge graphs · GeoQA2 · YAGO2geo

1 Introduction

Users are often interested in posing questions or information requests with a geospatial dimension to search engines, chatbots, and virtual personal assistants. Examples of such geospatial questions are: “Which rivers cross London?”, “Is there a Levi’s store in Hannover?”, “Which countries border Greece?”. Answering such questions or information requests requires structured data enriched with a geospatial dimension. There is plenty of such data available today in the form of *geospatial knowledge graphs (KGs)* (e.g., YAGO2 [4] YAGO2geo [6], WorldKG [1] and KnowWhereGraph [5]) or linked geospatial data [8].

The standard way to retrieve knowledge from geospatial KGs (or RDF stores storing linked geospatial data) is by using the query language SPARQL and its geospatial extensions GeoSPARQL and stSPARQL. However, to better serve the needs of non-technical end users, a system that will enable posing geospatial questions in natural language is needed. Although the developments in *question answering (QA)* over structured or unstructured data have been significant during the last few years, the geospatial dimension introduces additional challenges [9]: i) the QA system has to automatically identify the spatial representation (e.g., point, polygon) of the entities to choose, depending on the context of the question; ii) the interpretation of spatial operations and relations is subject to the map scale tied to the question (e.g., the word "near" is interpreted differently for the questions "Which countries are near Greece?" and "Which POIs are near Acropolis?"); iii) the problem of spatial relation recognition can be difficult due to the variability of the spatial language (north of Greece vs. northern Greece); iv) calculations among the geometries of the entities mentioned in the question may be required, a process that can be computationally very expensive.

In this paper we address the above challenges by developing the geospatial QA system GeoQA2 which answers questions over the KG YAGO2geo. GeoQA2 is based on GeoQA, the first geospatial QA engine proposed by [10] and in a revised version in [11]. GeoQA2 is available as open source at https://github.com/AI-team-UoA/GeoQA2. The differences between GeoQA2 and GeoQA can be summarized as follows. First, GeoQA was targeting DBpedia, and the parts of the Global Administrative Areas dataset (GADM, https://gadm.org/) and OpenStreetMap (OSM) for the United Kingdom and Ireland. The targeted KG of GeoQA2 is YAGO2geo [6], which is a large geospatial KG based on YAGO2, GADM, official geospatial data sources for some countries of Europe and the USA, and a subset of OSM. Targeting a single KG makes the step of query generation easier, and hence the QA process faster. Second, GeoQA2 can answer a greater variety of questions with respect to their complexity, including questions with *quantities and aggregates* (e.g., "What is the area of the city of London?" and "How many lakes are in Greece?"), *superlatives* (e.g., "Which municipality is the largest by population in Greece?") and *comparatives* (e.g., "Is Lake Baikal bigger than the Ioannina Lake?"). Finally, GeoQA2 is implemented utilizing optimized versions of the individual modules of the original GeoQA pipeline.

In recent work [7], we have also developed the dataset GeoQuestions1089 and used it to evaluate GeoQA2 and compare its accuracy with the geospatial question answering engine of Hamzei et al. [3]. GeoQuestions1089 contains simple but also semantically complex questions that require a sophisticated understanding of both natural language and GeoSPARQL to be answered. [7] argues that although GeoQA2 outperforms the engine of Hamzei et al., mainly because of its disambiguation component, neither engine is able to process complex questions caused by both a limited vocabulary of geospatial relations and a template-based approach to query generation. [7] also shows that the precomputation and materialization of entailed, but not stored explicitly, topological relations between entities in geospatial KGs, as done in the implementation of GeoQA2, can lead to substantial savings in geospatial query processing time.

2 The GeoQA2 pipeline

In this section we present the engine GeoQA2. Note that this is the first paper that presents this engine in detail; the recent paper [7] gives only a brief summary of GeoQA2's pipeline and capabilities.

GeoQA2 takes as input a question in natural language (English), the KG YAGO2geo and produces one or more answers. Question answering is performed by translating the input question into a set of SPARQL/GeoSPARQL queries,

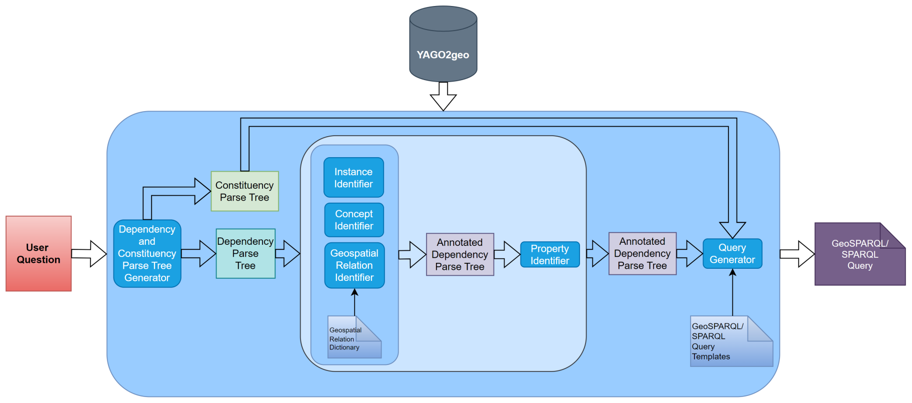
YAGO2geo
User Question
Dependency and Constituency Parse Tree Generator
Constituency Parse Tree
Dependency Parse Tree
Instance Identifier
Concept Identifier
Geospatial Relation Identifier
Geospatial Relation Dictionary
Annotated Dependency Parse Tree
Property Identifier
Annotated Dependency Parse Tree
Query Generator
GeoSPARQL/ SPARQL Query Templates
GeoSPARQL/ SPARQL Query

Fig. 1: The conceptual architecture of the GeoQA2 system

ranking these queries, and executing the top ranked query over a YAGO2geo endpoint. GeoQA2 has been implemented using Frankenstein platform which uses lightweight component-based QA methodology for efficient engineering of QA systems. In Figure 1, we illustrate the conceptual view of the GeoQA2 pipeline, which contains the six components described below.

Dependency and constituency parse tree generator. This component carries out part-of-speech tagging and generates a dependency and a constituency parse tree for the input question using the Stanford CoreNLP toolkit. The dependency parse tree is produced in CoNLL-U format and will be annotated by components coming next in the pipeline. The constituency parse tree is utilized in the query generator module.

Concept identifier. This component identifies the *types of features (concepts)* present in the input question and maps them to the corresponding classes of the YAGO2geo ontology. These concepts are identified by the elements of the question that are tagged as nouns (NN, NNS, NNP, NNPS) by the dependency parse tree generator. Then, these elements are mapped to the ontology classes using n-grams. As an example, let us consider the user question $\mathcal{Q}$: "*Which bays intersect with county councils that border with County Mayo*?" and use it to illustrate the details of various components of the pipeline. For question $\mathcal{Q}$, the concept identifier identifies the concepts "county councils" (as well as "county" and "councils") and "bays", as they are tagged as NN, NNS, NNP and maps them to the class yago2geoo:OSI_County_Council and yago2geoo:OSM_bay, respectively. To simplify the process, we have added labels to the YAGO2geo classes, for instance, "Bay" corresponds to the class yago2geoo:OSM_bay. Hence, the concept identifier iterates through this list of class labels from the ontology, it generates the n-grams (where n is the number of words present in each class label) for $\mathcal{Q}$, and compares the n-grams with the respective class labels. In the running example, the 2-gram "*county councils*", which has string similarity 0.998

with the class label "county council" (all letters of the class labels are converted to lowercase at the pre-processing phase), is mapped to the class yago2geoo:OSI_County_Council. If the n-grams contain more than one word then the Jaro-Winkler string similarity measure is used, otherwise we use the Levenshtein distance instead.

Instance identifier. This component identifies the *features* (*instances*) present in the input question. These can be, for example, the Corfu island or County Mayo. The features are identified by the elements of the question that are tagged as (NN, NNS, NNP) by the dependency parse tree generator. Then, these elements are mapped to YAGO2geo resources using an entity recognition and disambiguation tool. In previous work [11] we tested a set of well-known tools for named entity recognition and/or disambiguation over GeoQuestions201 [10] and concluded in TagMeDisambiguate [2], which had the best performance. As TagMeDisambiguate links the identified instances with instances only from Wikipedia (hence, from YAGO2 as well [4]), we, also, query YAGO2geo to disambiguate the instances that are contained in YAGO2geo, but not in YAGO2 (e.g., instances in GADM), along with the total number of YAGO2geo triples that contain these instances. In the running example, once the term "County Mayo" is identified from TagMeDiambiguate, it is mapped to yago2geor:geoentity_Mayo_3302545, which is found by executing the SPARQL query `SELECT DINSTICT ?x WHERE{?x yago: hasName "County Mayo"@en}` over the YAGO2geo endpoint.

Geospatial relation identifier. Geospatial questions often include some qualitative geospatial relation, such as "borders", or some quantitative ones, such as "at most 2km". GeoQA2 supports various geospatial relations and their synonyms (not listed here due to space) including topological, distance and cardinal direction relations. Similarly to the previous modules, this module first identifies the geospatial relations in the input question, based on the POS tags {VB, IN, VP, VBP, VBZ}, generated by the dependency parse tree. Then, it maps them (or their synonyms) to the respective spatial function of the GeoSPARQL or stSPARQL vocabulary using the handcrafted Geospatial Relation Dictionary. In the running example question, the geospatial relations "intersect" and "border" are identified from their POS tag (VBP) in the dependency tree, and they are mapped to the spatial functions `geof:sfIntersects` and `geof:sfTouches` of the GeoSPARQL vocabulary. Strabon is used as the back-end grospatial RDF store [8] (GeoSPARQL does not support any cardinal direction functions or relations, therefore stSPARQL is used instead) similar to previous work [11].

Property Identifier. The property identifier module identifies *attributes of types of features* and *attributes of features* specified by the user in input questions and maps them to the corresponding properties in YAGO2geo. For instance, for the question "Which village in Rhodes has the biggest population?", the "population" attribute of the type of feature "village" is required.

This module first identifies the properties in the input question, based on the POS tags {NN, JJ, NNP, NP}, generated by the dependency parse tree. Additionally, from the concepts identified by the concept identifier it performs pattern matching between the filtered terms and the labels of the 1-hop connected relations of the YAGO2geo classes. For instance, for the question "Which village in Rhodes has the biggest population?", the terms $\mathcal{T} = \{village, Rhodes, population\}$ are filtered. Also, the 1-hop relations connected to the class yago2geoo:OSM_village, returned by the concept identifier are selected. Finally, the property identifier performs pattern matching between these relations and the terms in $\mathcal{T}$. This way, the YAGO2geo property yago2geop:hasGAG_Population is retrieved.

For two cases, though, this process is not straightforward: first, when the property is not explicitly mentioned in the question and, second, when the KG does not explicitly contain the implied property. Consider the question "Which is the largest lake in Greece?". Here, the qualification "by area" is not clearly stated but implied. For such cases, we have defined the rules (not listed here due to space limit) to identify the implied properties. It is to be noted that the list of rules can be extended further without having any impact on the process as required. In particular, the implied property is specified from the classes participating in the question (returned by the concept identifier), the JJS or NN POS tags of the edges of the dependency parse and the implied properties. Hence, in the previous example, to capture the property "area", after identifying with the concept identifier the class yago2geoo:OSM_lake, the property identifier checks if the POS tags of the edges of the dependency parse are annotated as JJS (adjective, superlative, e.g. "biggest") or NN (noun, singular, e.g. "lake") and if so, it then checks if any of the keywords {smallest, biggest, largest} appears in the question. If this is the case, according to the table of implied properties, the question is annotated with the "area" property. Supposing, now, that YAGO2geo does not contain any property related to the term "area" for the class yago2geoo:OSM_lake, then the areas of the lakes in Greece will be calculated by applying the stSPARQL function strdf:area() on their geometries.

Query generator. This module generates the formal query using handcrafted query patterns, templates, and the outputs of the previous modules. In particular, the query generator reformulates the annotated (by the previous components of the pipeline) dependency parse tree and parses it with inorder traversal. From this process, it identifies the pattern of the question and, then, the respective template. Finally, the GeoSPARQL or SPARQL queries are generated from the templates and the resources identified from the previous modules of the pipeline. If the user question does not match any of the patterns, a message is passed to the query executor that no query has been generated.

We utilized the question patterns defined by [11], which we extended with one more question pattern (PCRCRI). These patterns contain different elements where "C" stands for "concept", "I" for "instance", "R" for "geospatial relation", "P" for "property" and "N" for "Count of", following the terminology introduced above. Notice that the identification of the intent of the question is implicitly executed through the identification of the proper query template. The query templates contain slots (strings starting with an underscore), which are replaced by the query generator with the outputs of the previous modules.

For instance, the question pattern of the running example question is CRCRI. The dependency parse tree is annotated with the concepts (C) yago2geoo:

OSM_bay and yago2geoo:OSI_County_Council, the geospatial relations (R) geof:sfIntersects and geof:sfTouches and the instance (I) yago2:County_Mayo. The question pattern is extracted by traversing the tree in the in-order traversal.

To capture more complex questions, containing superlatives, comparatives, or counts, the query generator uses, also, the constituency parse tree of the input question generated by the constituency parse tree generator module. Consider the question "Which civil parishes in Ireland have more than 10 townlands?". The identified templates are reformulated based on its dependency and constituency parse tree, and on the handcrafted rules. This way, for instance, the query generator will detect that the question contains the quantifier phrase (QP) "more than 10". Hence, it will automatically replace the "`SELECT ?x`" with "`SELECT ?x (COUNT(?y) AS ?total)`" and it will add `GROUP BY(?x) HAVING (?total > 10)` at the end of the query generated by the template-based approach.

For superlative questions (e.g., "Which county of England has the most parks?"), the query generator checks if there is an edge in the dependency parse tree which contains the set of POStags {RBS, JJS/DT} and does not contain QP from the constituency parse tree. In this case, the "`GROUP BY(?x) ORDER BY DESC(?total) LIMIT 1`" is added at the end of the query generated by the template-based approach. As the query contains "`GROUP BY(?x)`", the "`SELECT DISTINCT ?x`" appearing in the query is appended with "`(COUNT(?y) AS ?total)`". The reformulation of the queries for the rest of the aggregates is based on the keyword in question.

Lastly, the query generator ranks the generated queries. The ranking system employed is based on the estimated selectivity of the generated queries. It computes the selectivity of a SPARQL or GeoSPARQL query taking into account only the triple patterns present in the query. The generated query with the lowest selectivity is selected to be executed; in this way, it expects to generate more results to the user question. It is to be noted that preference to the generated GeoSPARQL query is given over the generated SPARQL query for the question.

Materialization of topological relations. We have found through experiments that some of the queries generated by the query generator of GeoQA2 have a significant execution overhead due to expensive computations of topological functions over complicated geometries. To alleviate this situation we precompute some topological relations ("within", "crosses", "intersects", "touches", "overlaps", "covers" and "equals") between any pair of geometries and store the result as a triple in the KG. Although this consumes space, it is very effective during query processing time. Detailed results are given in [7].

3 Evaluation

In [7] we present the benchmark GeoQuestions1089, which contains 1089 triples of geospatial questions, their answers, and the respective SPARQL or GeoSPARQL queries. GeoQuestions1089 is currently the largest geospatial QA benchmark and it is publicly available. GeoQuestions1089 follows the same file structure and question categorization as GeoQuestions201 introduced in [10], but is a much larger and more varied dataset. GeoQuestions1089 includes numerous complex questions that require both solid natural language understanding and advanced SPARQL features (nested queries, not-exists filters, arithmetic calculations) to be answered. For our evaluation we compare the results returned by each engine to the gold-result included in GeoQuestions1089. To accept an answer as correct, it must match the gold result exactly. We do not consider partially correct answers (e.g., when computed answers are a proper subset of the ones in the gold set) as correct. Likewise, we do not consider a superset of the answers in the gold set as correct.

Table 1: Evaluation of GeoQA2 and the engine of Hamzei et al. [3] over GeoQuestions1089.

Category	Hamzei et al.			GeoQA2		
	Generated Queries	Correct Answers	Correct Answers*	Generated Queries	Correct Answers	Correct Answers*
Type-A	89.71%	10.85%	12.10%	84%	47.42%	56.45%
Type-B	95.68%	53.23%	55.63%	76.25%	58.99%	77.35%
Type-C	97.75%	30.33%	31.03%	79.21%	44.38%	56.02%
Type-D	100%	12%	12.00%	56%	12%	21.42%
Type-E	99.25%	7.40%	7.46%	80%	31.85%	39.81%
Type-F	79.16%	4.10%	5%	66.66%	16.66%	25%
Type-G	98.27%	11.49%	11.69%	74.13%	32.18%	43.41%
Type-H	97.18%	7.74%	7.97%	71.12%	26.05%	36.63%
Type-I	92%	0%	0.00%	84%	20%	23.80%
Total	**95.77%**	18.97%	19.81%	76.99%	**38.54%**	**50.06%**

Engine comparison. The results of our evaluation show that GeoQA2 significantly outperforms the QA engine of [3] (the only other geospatial QA engine working on YAGO2geo) by generating twice the amount of correct queries. The main factor of this performance gap is the existence of a dedicated named entity disambiguation step in GeoQA2 (instance identifier). Other than this main difference, the two engines are similar in a number of ways. Both utilize dependency and constituency parsing to understand the structure of the input question and the relations that exist among its tokens. Likewise, both engines have a rule-based query generator that uses a set of predefined templates that are filled in with instances and concepts to generate the final GeoSPARQL queries, although the engine of [3] uses a more of a dynamic approach of combining smaller templates which allows it to generate queries for a significantly larger portion of the dataset. Considering these similarities, it follows that the engines must share some weaknesses. That is the case, with the inability of either engine to reliably answer complex questions being their most important weakness.

4 Conclusions and Future Work

We have addressed the challenge of providing access to the YAGO2geo KG for non-expert users using the question answering engine GeoQA2. The effectiveness of our previous engine GeoQA has been greatly improved and now GeoQA2 can also answer much more complex questions containing e.g., aggregations, superlatives and comparatives. In related work [7], we showed that GeoQA2 outperforms the QA engine of [3] over the dataset GeoQuestions1089. However, there is still plenty of room for improvement for both engines; this is the topic on which we now concentrate our attention using ideas from deep learning and, especially, large language models.

Acknowledgements

This work was supported by the first call for H.F.R.I. Research Projects to support faculty members and researchers and the procurement of high-cost research equipment grant (HFRI-FM17-2351). It was also partially supported by the ESA project DA4DTE (subcontract 202320239), the Horizon 2020 project AI4Copernicus (GA No. 101016798), the Horizon Europe project STELAR (GA No. 101070122) and the Marie Skłodowska-Curie project QuAre (GA No. 101032307).

References

[1] Dsouza, A., Tempelmeier, N., Yu, R., Gottschalk, S., Demidova, E.: WorldKG: A world-scale geographic knowledge graph. In: CIKM (2021)
[2] Ferragina, P., Scaiella, U.: TAGME: on-the-fly annotation of short text fragments (by wikipedia entities). In: CIKM (2010)
[3] Hamzei, E., Tomko, M., Winter, S.: Translating place-related questions to GeoSPARQL queries. In: The Web Conference (2022)
[4] Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: Yago2: A spatially and temporally enhanced knowledge base from wikipedia. Artif. Intell. **194** (2013)
[5] Janowicz, K. et al.: Know, Know Where, Knowwheregraph: A densely connected, cross-domain knowledge graph and geo-enrichment service stack for applications in environmental intelligence. AI Mag. **43**(1), 30–39 (2022)
[6] Karalis, N., Mandilaras, G.M., Koubarakis, M.: Extending the YAGO2 knowledge graph with precise geospatial knowledge. In: ISWC (2019)
[7] Kefalidis, S.-A. et al.: Benchmarking geospatial question answering engines using the dataset GeoQuestions1089. In: ISWC (2023)
[8] M. Koubarakis (ed.): Geospatial data science: a hands-on approach based on geospatial technologies. ACM Books (2023)
[9] Mai, G. et al.: Geographic question answering: Challenges, uniqueness, classification, and future directions. CoRR **abs/2105.09392** (2021)
[10] Punjani, D. et al.: Template-based question answering over linked geospatial data. In: Workshop on Geographic Information Retrieval (2018)
[11] Punjani, D. et al.: Template-based question answering over linked geospatial data. CoRR **abs/2007.07060** (2020), https://arxiv.org/abs/2007.07060