NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Geospatial Question Answering on the YAGO2geo Knowledge Graph

Theodoros C. Stefou

Supervisors: **Manolis Koubarakis,** Professor
Dharmen Punjani, PhD Candidate

ATHENS

OCTOBER 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Γεωχωρικές Ερωτήσεις Φυσικής Γλώσσας στο Γράφο Γνώσης YAGO2geo

Θεόδωρος Χ. Στέφου

Επιβλέποντες: **Μανόλης Κουμπαράκης,** Καθηγητής
Dharmen Punjani, Υποψήφιος Διδάκτορας

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2019

BSc THESIS

Geospatial Question Answering on the YAGO2geo Knowledge Graph

Theodoros C. Stefou
S.N.: 1115201400193

SUPERVISORS: **Manolis Koubarakis,** Professor
Dharmen Punjani, PhD Candidate

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Γεωχωρικές Ερωτήσεις Φυσικής Γλώσσας στο Γράφο Γνώσης YAGO2geo

Θεόδωρος Χ. Στέφου
Α.Μ.: 1115201400193

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Μανόλης Κουμπαράκης,** Καθηγητής
Dharmen Punjani, Υποψήφιος Διδάκτορας

ABSTRACT

In the recent years there have been many attempts to develop systems that can process natural language questions and return meaningful answers in order to make information available to everyone and not only to people who can write queries for databases. Such systems can be designed to work for different types of questions varying from facts about historical figures all the way to questions about science problems.

In this thesis, we will be working with geospatial questions. We use an already existing geospatial natural language QA system (GeoQA system) that is currently using the DBpedia, GADM (Database of Global Administrative Areas) and OSM (Open Street Map) knowledge graphs and changing it to use the YAGO2geo knowledge graph which has been extended with Open Street Map, Ordnance Survey and GADM data.

The purpose of this change is to achieve more accurate results using the geospatial information that is in Open Street Map and Ordnance Survey and the huge amount of classes that are included in the YAGO2 knowledge graph.

The code that extends the system mentioned above can be found at the following link:

Github repository

SUBJECT AREA: Natural Language Geospatial Question Answering

KEYWORDS: geospatial, natural language, question answering, knowledge graph

ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια έχουν γίνει πολλές προσπάθειες για την ανάπτυξη συστημάτων που να μπορούν να επεξεργαστούν ερωτήσεις σε φυσική γλώσσα και να επιστρέψουν έυστοχες απαντήσεις ώστε να γίνει η πληροφορία διαθέσιμη σε όλους και όχι μόνο σε όσους μπορούν να γράψουν ερωτήματα σε βάσεις δεδομένων. Τέτοια συστήματα μπορούν να σχεδιαστούν έτσι ώστε να δουλεύουν για διάφορα είδη ερωτήσεων, από γεγονότα για ιστορικά πρόσωπα μέχρι επιστημονικά προβλήματα.

Σε αυτή την πτυχιακή εργασία θα δουλέψουμε με γεωχωρικές ερωτήσεις. Χρησιμοποιούμε ένα ήδη υπάρχον σύστημα γεωχωρικών ερωτήσεων-απαντήσεων φυσικής γλώσσας που μέχρι τώρα χρησιμοποιεί τους γράφους γνώσης Dbpedia, GADM (Database of Global Administrative Areas) και OSM (Open Street Map) και το αλλάζουμε ώστε να χρησιμοποιεί το γράφο γνώσης YAGO2geo ο οποίος έχει επεκταθεί με δεδομένα από το Open Street Map, το Ordnance Survey και το GADM.

Ο σκοπός της αλλαγής αυτής είναι η επίτευξη αποτελεσμάτων μεγαλύτερης ακρίβειας χρησιμοποιώντας τα γεωχωρικά δεδομένα του Open Street Map και του Ordnance και τον τεράστιο αριθμό κλάσεων που περιέχονται στο γράφο γνώσης YAGO2.

Ο κώδικας που επεκτείνει το σύστημα που αναφέρθηκε παραπάνω βρίσκεται στον ακόλουθο σύνδεσμο:

Github repository

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Γεωχωρικές Ερωτήσεις-Απαντήσεις Φυσικής Γλώσσας

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: γεωχωρικός, φυσική γλώσσα, ερωτήσεις-απαντήσεις, γράφος γνώσης

To my parents.

ACKNOWLEDGMENTS

I would like to thank my supervisor Professor Manolis Koubarakis for the opportunity to work on this subject and also his PhD student Dharmen Punjani who provided great guidance and support throughout the whole process.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

The work for this thesis was completed between March 2019 and October 2019 in Athens, Greece. The project was developed on a Linux machine using the Java programming language, Eclipse for IDE and the Stardog triple store. It was of crucial importance to understand the concepts of the semantic web, the SPARQL querying language and its extension GeoSPARQL as well as the different approaches of implementation for natural language processing systems.

1. INTRODUCTION

The purpose of question answering systems is to make information available to people who do not have a programming background and therefore are not able to use a querying language to retrieve information from a database. The system is able to process a question which is provided in a natural language like English and returns a result or a list of results that answer the question. Not only is this of scientific interest from a programmer's perspective, but can also play a significant role in education, business, customer support automation etc.

There are many ways to implement a QA system; the one that we use here is a component based structure which works as a pipeline that isolates the different functions of the question parsing. Each component processes the question in some way and pushes the results in a triple store to make them available to the other component processes.

In this thesis, the purpose of our QA system is to be able to give answers to geospatial questions. An example of a geospatial question is: “Which river crosses London?” and the answer provided by the system should be “river Thames”. The information for the answer is stored in a semantic knowledge base, in our case YAGO2 extended with OSM and Ordnance Survey (YAGO2geo), in a triple store fashion and is retrieved once the question is processed by the system. GeoQA [4] was developed to work with DBpedia, GADM and OSM. DBpedia is a knowledge base based of Wikipedia facts. Although it might be sufficient for other types of question answering, it does not contain as useful and as accurate geospatial information as our combination of knowledge bases. YAGO2geo is extended with Ordnance Survey as well as GADM with administrative divisions of the United Kingdom and Ireland so we do not need to use the GADM dataset for GeoQA.

The extension of YAGO2 with OSM, GADM and Ordnance Survery administrative division polygons resulting in YAGO2geo [5] was developed by two members of professor Koubarakis' team, N. Karalis et el under professor Koubarakis' supervision, and the GeoQA system [4] that we will be using was implemented by D. Punjani et el, a PhD candidate at our university.

2. SEMANTIC WEB

The idea behind the semantic web is to make information readable by machines. This, apart from quantitative values which can be represented by numbers or characters, means that also qualitative information must be represented in some way (e.g. relationships between entities). For example, consider Elvis Presley; born in 1935, also known as the “King of Rock and Roll”, recorded 24 studio albums, had one daughter named Lisa Marie Presley who was born in 1968. All this information has to be stored in some way that does not expect other entries to also have the same characteristics, since we are storing a varying number of facts for each one of our entries. After all, it wouldn't make sense to have the number of studio albums recorded by the Eiffel Tower; remember we need to be able to accommodate any type of fact-information. The point of the semantic web is to accommodate such information. Knowledge is represented in triples known as subject, predicate and object. This is very similar to how natural languages work and in most cases can be represented by a graph (not when predicates are also subjects or objects).

Let's provide an example to make things clear. Consider the facts about Elvis Presley given above. Also, let's assume that our Uniform Resource Identifiers (URIs) in our semantic knowledge base start with http://domain.org in order to avoid name ambiguity and conflicts with other databases. We can represent this information in this way:

<http://domain.org/Elvis_Presley><http://domain.org/birthDate> 1935.

<http://domain.org/Elvis_Presley><http://domain.org/aka> “King of Rock and Roll”.

<http://domain.org/Elvis_Presley><http://domain.org/numberOfStudioAlbums> 24;

<http://domain.org/daughter><http://domain.org/Lisa_Marie_Presley>.

<http://domain.org/Lisa_Marie_Presley><http://domain.org/birthDate> 1968.

Figure 1: Example of simple knowledge base

Each fact consists of three parts, the subject, the predicate and the object and ends with a period or with a semicolon if the next fact uses the same subject to avoid repetition. This syntax is called “Turtle” for “Terse RDF Triple Language” and is not the only way to represent semantic web facts, but we will stick to this one because it is very similar to that of SPARQL, a querying language we will discuss later on. This information can be represented by a graph as shown below:

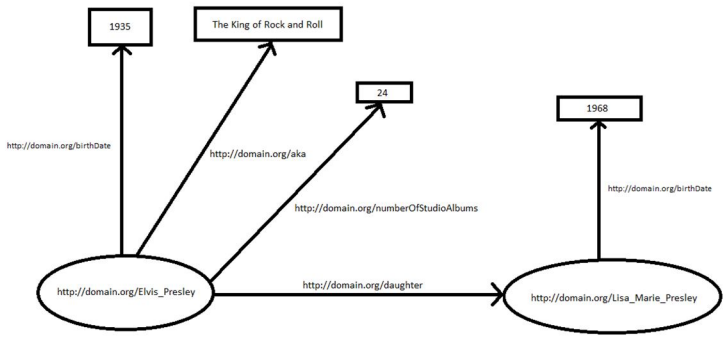

Figure 2: Example graph of semantic web data

The facts above are a small semantic knowledge base. The subject is some entity to which we want to attach some information. The predicate is the type of information we wish to attach, for example the date of birth. The object is the actual value of the information and it can be anything from the primitive types of integer, string etc to a reference to some other entity's URI.

The predicates are defined in the schema of a knowledge base along with their domains (legal subjects) and ranges (legal objects) and can be anything we might need for our application. However, the Resource Description Framework Schema RDF(S) provides some basic elements for the description of ontologies in order to define a standard across all triple store databases. Some very common are:

- rdf:type to state that a resource is an instance of a class
- rdf:subClassOf to allow the declaration of a hierarchy between classes
- rdfs:label to provide a human readable version of a resource

The prefixes rdf and rdfs are just links provided to uniquely identify each property. “rdf” is short for “http://www.ww3.org/1999/02/22-rdf-syntax-ns#” and “rdfs” is short for “http://www.w3.org/2000/01/rdf-schema#“. These prefixes are so common that they are usually implicitly defined by the endpoints of databases without specific code.

This way, it is clear to new users how to get basic information about the data of a knowledge base. For example, if we wanted to add the fact that Elvis is an instance of the class “Artist”, then we would add this fact to our knowledge base:

```
<http://domain.org/Elvis_Presley> rdf:type <http://domain.org/Artist>.
```

Figure 3: Adding instances of classes to a knowledge base

and someone that knew the URI of Elvis and nothing else about our database, would be able to retrieve Elvis's class by querying using the rdf:type predicate. More on that in the next chapter.

It is considered good practice to organize the resources based on the schema of the knowledge base, but we skipped that above for simplicity. A better way to define the subject and the object above would be <http://domain.org/resource/Elvis_Presley> and <http://domain.org/ontology/Artist>, because Elvis is a resource-instance of a class and Artist is a class, therefore part of our knowledge base ontology. With the term “ontology” we refer to the hierarchy of classes that model our data.

3. QUERYING SEMANTIC DATABASES

3.1 SPARQL

A database is useful only if it provides a way to query its data. In the semantic web, the most common way of querying databases is the SPARQL querying language. SPARQL is a recursive acronym and stands for “SPARQL Protocol and RDF Query Language”. Its syntax is very close to what we described in the previous chapter. Like SQL, querying consists of two basic parts: a) Selecting which fields we wish to be returned, b) The conditions under which the triples will be returned. Instead of tables, a database consists of a number of graphs, one of them being the default. However, it is closer to logic programming languages like Prolog and Haskell than to SQL, because it tries to match the variables of the given triples to facts in the knowledge base.

The most interesting part of the language is the WHERE block, in which we can describe the conditions that will filter our data. The language tries to satisfy the triples that we pass in this block by replacing variables with instances and returns the results that passed all the tests. Variables start with the “?” character. Prefixes are a way to keep the code clean and repetition-free. The “distinct” keyword is used when we are not interested in duplicates in our results. Let's look at a few examples to make this a bit more clear:

```
SELECT ?subject ?predicate ?object
WHERE {
   ?subject ?predicate ?object.
}
```

Figure 4:Selecting every single triple in the database

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT distinct ?class
WHERE {
   ?subj rdf:type ?class.
}
```

Figure 5: Selecting every single class

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT distinct ?class
WHERE {
   ?subj rdf:type ?class.
   ?class rdfs:label ?label.
   FILTER(CONTAINS(?label, “art”)).
}
```

Figure 6: Selecting classes whose labels contain the string “art”

The FILTER function receives boolean values and returns a boolean value as well. Only if it returns true are the triples that matched the variables returned to be displayed.

Since we are using YAGO2geo, let's also take a look at a query that returns the URIs and the English labels of all entities in YAGO2 that are of type “restaurant”. The URI for the class “restaurant” in YAGO2 is:

http://yago- knowledge.org/resource/wordnet_restaurant_104081281

The query is the following:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX yago: <http://yago-knowledge.org/resource/>
SELECT distinct ?subj ?label
WHERE {
   ?subj rdf:type yago:wordnet_restaurant_104081281.
   ?subj rdfs:label ?label.
   FILTER(langMatches(lang(?label), "eng")).
}
```

Figure 7: Selecting all restaurants in YAGO2 along with their english labels

Here are the first few results out of almost 3000 that were returned when we ran this query on the 3rd of October 2019 at the YAGO2 endpoint provided by the Max-Planck institute:

subj	label
http://yago-knowledge.org/resource/B.T.'s_Smokehouse	"B.T.'s Smokehouse"@eng
http://yago-knowledge.org/resource/Workshop_Kitchen_+_Bar	"Workshop Kitchen + Bar"@eng
http://yago-knowledge.org/resource/Caru'_cu_Bere	"Caru' cu Bere"@eng
http://yago-knowledge.org/resource/Pershing_Square_(restaurant)	"Pershing Square"@eng
http://yago-knowledge.org/resource/Pershing_Square_(restaurant)	"Pershing Square (restaurant)"@eng
http://yago-knowledge.org/resource/ReConnect_Café	"ReConnect Café"@eng
http://yago-knowledge.org/resource/ReConnect_Café	"ReConnect Cafe"@eng
http://yago-knowledge.org/resource/Vinkeles	"Vinkeles"@eng
http://yago-knowledge.org/resource/Restaurants_in_Washington,_D.C.	"Restaurants in Washington, D.C."@eng
http://yago-knowledge.org/resource/Restaurants_in_Washington,_D.C.	"Restaurants in Washington"@eng
http://yago-knowledge.org/resource/Eiffel_Tower	"Eiffel Tower"@eng
http://yago-knowledge.org/resource/Bawabet_Dimashq	"Bawabet Dimashq"@eng
http://yago-knowledge.org/resource/Khan_Murjan	"Khan Murjan"@eng
http://yago-knowledge.org/resource/ar/مطعم_الساعة	"مطعم الساعة"@eng
http://yago-knowledge.org/resource/Piz_Gloria	"Piz Gloria"@eng
http://yago-knowledge.org/resource/Conveyor_belt_sushi	"Conveyor belt sushi"@eng
http://yago-knowledge.org/resource/Hofbräukeller	"Hofbräukeller"@eng
http://yago-knowledge.org/resource/Hofbräukeller	"Hofbraeukeller"@eng
http://yago-knowledge.org/resource/Sunsphere	"Sunsphere"@eng
http://yago-knowledge.org/resource/Hangar-7	"Hangar-7"@eng
http://yago-knowledge.org/resource/Osteria	"Osteria"@eng
http://yago-knowledge.org/resource/de/Haus_Goldschmieding	"Haus Goldschmieding"@eng
http://yago-knowledge.org/resource/Savoy_Hotel	"Savoy Hotel"@eng
http://yago-knowledge.org/resource/Windows_on_the_World	"Windows on the World"@eng
http://yago-knowledge.org/resource/Sibylla_(fast_food)	"Sibylla"@eng
http://yago-knowledge.org/resource/Sibylla_(fast_food)	"Sibylla (fast food)"@eng

Figure 8: Results of SPARQL query example in browser

3.2 GeoSPARQL

GeoSPARQL, defined by OGC (Open Geospatial Consortium), is an extension of SPARQL that provides several geospatial functions which operate on points and polygons and are useful to extract spatial relations from the database. It is our main tool in finding out geospatial relations between points of interest to answer the questions that are fed into our system.

The functions and classes that are provided by GeoSPARQL help us query our database and retrieve geospatial information like whether a point is inside a polygon, the distance between two points, whether two polygons cross etc.

Here is an example of a query that answers the question: “Which river crosses Limerick?”

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX osmo: <http://www.app-lab.eu/osm/ontology#>
SELECT ?x
WHERE {
   ?x rdf:type osmo:River;
      geo:hasGeometry ?xGeom.
   ?xGeom geo:asWKT ?xWKT.
   gadmr:Limerick geo:hasGeometry ?iGeom.
   ?iGeom geo:asWKT ?iWKT.
   FILTER(geof:sfCrosses(?xWKT, ?iWKT)).
}
```

Figure 9: Query for the question: “Which river crosses Limerick?”

As you can see from this query, the first two triples returns all the rivers from the database and the URIs of their geometries. Then for those geometries we retrieve their “Well known text representation” known as WKT, which is a markup language for representing vector geometry objects. We do the same for Limerick and then filter the results with the sfCrosses GeoSPARQL function that returns “true” if the polygon of a river that's been assigned to the variable ?x crosses that of Limerick. The polygons in this query are provided by the OSM semantic geospatial database.

4. GEOQA

The GeoQA system [4] that we are extending in this paper is based on a pipeline oriented architecture called Frankenstein [2]. The several routines of editing and parsing of the question are divided into autonomous components that receive data from previous components, complete their part of the work load and proceed to pass the results to the next components. This enables us to have more control over the functionality of the system, better scalability and maintainability.

The intermediate stage of the data transferring within this pipeline is achieved using the Stardog triple store with an academic license.

The pipeline itself is implemented using Qanary, which is specifically created to assist in the creation of component based QA systems. It uses Spring Boot to deploy a server that recognizes which components are running and provides a simple browser interface for the user to decide which components they wish to be executed and in what order.

The system consists of 5 basic components:

1. Dependency Parse Tree Generator
2. Instance Identifier
3. Geospatial Relation Identifier
4. Concept Identifier
5. Query Generator

Here is a visual representation of the system taken from the paper that it was published in:

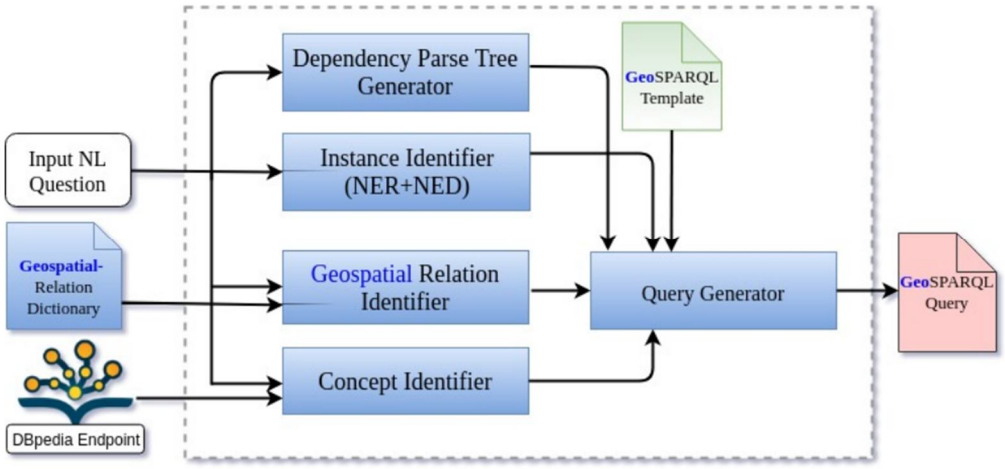

Figure 10: A visual representation of the GeoQA system

4.1 Changes in the GeoQA system

For the purposes of this thesis, we made changes to the instance identifier, concept identifier and query generator components in order to rub GeoQA on YAGO2, YAGO2geo and OSM. In the next pages we are going to give a brief description for the rest of the components and provide a detailed explanation of the changes we did on the mentioned components.

4.2 Dependency Parse Tree Generator

This is the first step in which we generate a tree that represents the grammatical structure of the question. The code uses the NLP library provided by the Stanford University. No changes were done to this component.

4.3 Instance identifier

An instance identifier consists of two sub-components:

1. The named entity recognition (NER)
2. The named entity disambiguation (NED).

4.3.1 Named entity recognizer (NER)

This is the process of looking at the chunks of unstructured text and annotating them with predefined categories. Let's look at an example:

"Elvis Presley was born on January 8, 1935, in Tupelo, Mississippi"

If we feed this text into a good NER system, it should return something like this:

"(Elvis Presley)$_{Person}$ was born on (January 8, 1935)$_{Date}$, in (Tupelo, Mississippi)$_{Location}$"

4.3.2 Named entity disambiguator (NED)

After NER has done its part of the processing, it is the NED system's responsibility to find an appropriate URI for each annotation provided by the NER that refers to an instance of a class. For the example given above for NER, NED would have to annotate with URIs the texts "Elvis Presley" and "Tupelo, Mississippi". A good NED system would return something like this (with disambiguation to the YAGO2 knowledge graph):

Elvis Presley → http://yago-knowledge.org/resource/Elvis_Presley

Tupelo, Mississippi → http://yago-knowledge.org/resource/Tupelo,_Mississippi

4.3.3 Choosing an instance identifier that disambiguates to YAGO

We looked at as many different instance identifiers as we could, some available for free, some with limitations regarding the maximum number of API calls per day and some paid ones. We found these systems from the paper describing Frankenstein [3]. Unfortunately, we were not able find any that disambiguate directly to the YAGO knowledge graph.

We tested them against part of our golden standard benchmark. Our mini-benchmark and the answers can be found at the github repository of this thesis. The first column contains the questions with their entities highlighted in bold and the second contains the correct URIs for the entities respectively.

The following table contains the instance identifier systems we tested and their percentage of correctly annotated questions from our benchmark.

Table 1: Percentage of correctly annotated questions by the instance identifiers

AIDA[i]	55.81%
Agdistis [10]	81.40%
DBpedia Spotlight [11]	69.77%
TagMe [12]	88.37%
Meaning Cloud[ii]	60.47%
Text Razor[iii]	70.93%
Babelfly [13]	36.05%
Entity fishing[iv]	83.72%

i. https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/ambiverse-nlu/aida/

ii. https://www.meaningcloud.com/developer

iii. https://www.textrazor.com/docs/rest

iv. https://nerd.readthedocs.io/en/latest/

The best results were returned by TagMe, so this is the instance identifier API that we used in our code. To be fair, most of these systems are designed to annotate general purpose questions, hence the terrible results for some.

None of these systems, including TagMe, disambiguate to YAGO2. Therefore we have to use owl:sameAs links in our SPARQL queries which are provided by the Max-Planck institute and their purpose is to link databases.

4.4 Geospatial relation identifier

This component is responsible for identifying the geospatial relations that might be in the questions. The system supports 14 different geospatial relations divided into 3 categories. No changes were done to this component.

Table 2: Categories of supported geospatial relations

Category	Geospatial relation
Topological relations	within, crosses, borders
Distance relations	near, at most x units, at least x units

Cardinal direction relations	north of, south of, east of, west of, northwest of , northeast of, southwest of, southeast of

4.5 Concept identifier

This component annotates the questions with the URIs of identified classes. Let's look at an example:

Which cities are in Berkshire?

The concept identifier should return the URI for the class “city” in YAGO and annotate the text “cities” of the question with it.

Which (cities)http://yago-knowledge.org/resource/wordnet_city_108524735 are in Berkshire?

We have created a mapping of class labels to YAGO2 URIs. You can see part of this mapping in the following table. The full table can be found at the github repository of this thesis. Note that for some classes, YAGO has multiple URIs, because classes are in the form wordnet_XXX_YYY where XXX is the name of the class and YYY its synset id in Wordnet 3.0. Everything in the second column is implicitly prefixed by “http://yago-knowledge.org/resource/ “.

Table 3: Mapping of class labels to YAGO URIs

LABEL	URI(s)
Airport	wordnet_airport_102692232
Beach	wordnet_beach_109217230
Church	wordnet_church_103028079
Forest	wordnet_forest_108438533, wordnet_forest_109284015
Hospital	wordnet_hospital_103540595
Hotel	wordnet_hotel_103542333
Island	wordnet_island_109316454
Library	wordnet_library_103660664, wordnet_library_103660909, wordnet_library_107977870

Restaurant	wordnet_restaurant_104081281
River	wordnet_river_109411430
Town	wordnet_town_108226514, wordnet_town_108665504

We got the class labels above from the paper by D. Punjani et el [4] and from the questions of the benchmark. To retrieve the class URIs for each of those labels we ran the query shown in the 5th figure of this thesis, downloaded an XML file of the results and then manually searched for class names that were close to what we had in the label.

4.6 Query generator

The query generator is the component that creates the queries that will be executed to retrieve the results. In this component we changed a few things in the structure of these queries to make them work with our new knowledge base. In the paper by D. Punjani et el [4] there is a table that describes the several patterns of questions that have been concluded from the benchmark. These patterns consist of 3 elements: “C” for concept, “R” for relation and “I” for instance. Each pattern corresponds to one parameterized query template that produces the results of the question. The patterns with examples are the following:

1. CRI → Is there a church in London?
2. CRIRI → Which restaurants are north of river Thames in London?
3. CRC → Are there libraries close to Big Ben?
4. CRCRI → Which hotels are close to colleges in Limerick?
5. IRI → Is Somerset west of Westmorland?

If no pattern is identified no query will be executed.

4.6.1 Using schema information to improve accuracy

Although YAGO2 has a proven accuracy of over 95%, it is important to make sure that the results of our GeoQA system [4] do not contain unwanted values. For this reason, for each YAGO2 class that was mapped to a label earlier, we have also defined a range of classes that make sense for this class to be located in. YAGO provides the geospatial predicate yago:isLocatedIn which links two geospatial entities if the first is located in the first one. For example:

```
yago:Hanover_Town yago:isLocatedIn yago:Virginia.
```

The mapping can be found at the github repository of this thesis.

The challenging part of this was to make sure that the ranges make sense but are not too strict; at a first glance something might not seem reasonable but it exists. For example, museums in lakes!

The query that we used for this was the following:

```
SELECT DISTINCT ?parent
WHERE {
  ?s a yago:_CLASS_.
  ?s <http://yago-knowledge.org/resource/isLocatedIn> ?o.
  ?o a ?type.
  ?type rdfs:subClassOf ?parent.
}
order by ?parent
```

Figure 11:Getting the domain and the range of the predicate yago:isLocatedIn

For each one of our classes we replaced the _CLASS_ segment in the query above with their URI in YAGO and downloaded a JSON file with the results. Then, we manually decided which classes were allowed to be “located in” other classes.

We got the superclasses of the actual classes of the instances because YAGO2 has 2 layers of classes: The first is derived from wordnet and is a more general ontology like “restaurant” and the second is derived from wikipedia facts and it's more specific like “restaurants in London”.

4.6.2 Mapping instances of OSM to YAGO2 using owl:sameAs links

Even though YAGO2 has already been extended with the geospatial knowledge of OSM resulting in the YAGO2geo graph we have been talking about, it only contains polygons for administrative divisions and not for points of interest. In order to link YAGO2 points of interest to OSM polygons, such as hotels, tourist attractions, monuments etc, we downloaded all the owl:sameAs links provided by YAGO2 which link YAGO2 entities to DBpedia. We also have OSM owl:sameAs links at our disposal that link OSM to DBpedia as well. [4]

We ran a simple python3 script that using dictionaries created a file that links OSM points of interest to YAGO2. The file is in the form

```
<osm_entity> owl:sameAs <yago2_entity>.
```

Out of 23016 OSM entities that we had available we managed to link 21317 to YAGO2 entities. Unfortunately, for the rest 1699 YAGO2 did not have DBpedia owl:sameAs links.

The script can be found at the github repository of this thesis.

5. RESULTS

5.1 Questions that GeoQA is now able to answer

After implementing what we have mentioned in the chapters above, we ran our benchmark against our system and saved the query that we produced for each question. The queries that we produced can be found at the github repository of this thesis.

In the 7th table of the paper of the GeoQA system by D. Punjani et el [4] that we used can be seen some questions that were not able to be answered. After our changes to the system, it now produces correct queries for the following questions from that table:

Question 17: Which hotels are near Big Ben?

```
select ?x
where {
   ?x rdf:type <http://www.app-lab.eu/osm/ontology#Hotel>;
      geo:hasGeometry ?cGeom.
   ?cGeom geo:asWKT ?cWKT.
   ?instance owl:sameAs <http://yago-knowledge.org/resource/Big_Ben>;
      geo:hasGeometry ?geom.
   ?geom geo:asWKT ?iWKT.
   FILTER(geof:distance(?cWKT,?iWKT,uom:metre) <= 1000)
}
```

Figure 12: Query produced for: "Which hotels are near Big Ben?"

Question 98: Which restaurants are at most 1km away from Big Ben?

```
select ?x
where {
   ?x rdf:type <http://www.app-lab.eu/osm/ontology#Restaurant>;
       geo:hasGeometry ?cGeom.
   ?cGeom geo:asWKT ?cWKT.
   ?instance owl:sameAs <http://yago-knowledge.org/resource/Big_Ben>;
       geo:hasGeometry ?geom.
   ?geom geo:asWKT ?iWKT.
   FILTER(geof:distance(?cWKT,?iWKT,uom:metre) <= 1000)
 }
```

Figure 13: Query produced for: "Which restaurants are at most 1km away from Big Ben?"

Question 182: Is there any airfield in Wellesbourne village in England?

```
select ?x
where {
   ?x rdf:type <http://www.app-lab.eu/osm/ontology#Airfield>;
       geo:hasGeometry ?geom.
   ?geom geo:asWKT ?cWKT.
   <http://yago-knowledge.org/resource/England> geo:hasGeometry ?iGeometry1.
   ?iGeometry1 geo:asWKT ?iWKT1.
   ?y2 owl:sameAs <http://yago-knowledge.org/resource/Wellesbourne>;
        geo:hasGeometry ?iGeometry2. ?iGeometry2 geo:asWKT ?iWKT2.
```

```
    FILTER(geof:sfWithin(?cWKT, ?iWKT1) && geof:sfWithin(?cWKT, ?iWKT2) )
}
```

Figure 14: Query produced for: "Is there any airfield in Wellesbourne village in England?"

Question 235: Which county is east of county Dorset?

```
select ?x
where {
    ?x rdf:type <http://kr.di.uoa.gr/yago2geo/ontology/OS_County>;
        geo:hasGeometry ?cGeom.
    ?cGeom geo:asWKT ?cWKT.
    <http://yago-knowledge.org/resource/Dorset> geo:hasGeometry ?geom.
    ?geom geo:asWKT ?iWKT.
    FILTER(strdf:right(?cWKT, ?iWKT))
}
```

Figure 15: Query produced for: "Which county is east of county Dorset?"

Question 269: Which restaurants are near Stonehenge?

```
select ?x
where {
    ?x rdf:type <http://www.app-lab.eu/osm/ontology#Restaurant>;
        geo:hasGeometry ?cGeom.
    ?cGeom geo:asWKT ?cWKT.
    ?instance owl:sameAs <http://yago-knowledge.org/resource/Stonehenge>;
        geo:hasGeometry ?geom.
    ?geom geo:asWKT ?iWKT.
    FILTER(geof:distance(?cWKT,?iWKT,uom:metre) <= 500)
}
```

Figure 16: Query produced for: "Which restaurants are near Stonehenge?"

Question 312: Which restaurants in London are near Kensington Palace?

```
select ?x
where {
    ?x rdf:type <http://www.app-lab.eu/osm/ontology#Restaurant>;
        geo:hasGeometry ?geom.
    ?geom geo:asWKT ?cWKT.
    <http://www.app-lab.eu/osm/england/places/id/107775>
geo:hasGeometry ?iGeometry1.
    ?iGeometry1 geo:asWKT ?iWKT1.
    ?y2 owl:sameAs <http://yago-knowledge.org/resource/Kensington_Palace>;
          geo:hasGeometry ?iGeometry2.
    ?iGeometry2 geo:asWKT ?iWKT2.
    FILTER(geof:sfWithin(?cWKT, ?iWKT1) && (geof:distance(?cWKT, ?iWKT2,
    uom:metre) <= 1000)) }
```

Figure 17: Query produced for: "Which restaurants in London are near Kensington Palace?"

For the questions numbered 17, 98, 269, 312, the component that was responsible for the improvement was the new NED system that we added. For the questions numbered 182, 235, the component that was responsible was the query generator.

5.2 Statistics for the rest of the questions

For the purposes of this evaluation, we have divided the questions into two categories; simple YAGO2 queries and geospatial queries on YAGO2geo. Simple YAGO2 queries are the queries where all the spatial relations were “within” and also both concepts and instances were all from YAGO2. We ran those queries at the YAGO2 SPARQL endpoint using the yago:isLocatedIn predicate. YAGO2geo queries were the rest of the queries which included filtering with a geospatial function on the polygons of the concepts and entities.

The following table shows these statistics. The total number of questions is 86:

Table 4: Statistics for the two categories of queries

Query category	Number of queries produced
Simple YAGO2 query	11
YAGO2geo geospatial query	41

We were able to generate queries which target YAGO2 and YAGO2geo for all the questions that GeoQA was able to generate before for DBpedia. In addition to that, changing the instance identifier resulted in the generation of queries that we were not able to generate before. The queries that we are producing at the moment can be found at the github repository of this thesis.

6. CONCLUSION

In this BSc thesis we have described fundamentals regarding the semantic web and the dominant querying language for semantic knowledge bases called SPARQL. We studied a component-based geospatial question answering system GeoQA [4] and successfully modified it to use a different set of underlying knowledge graphs from DBpedia and GADM to YAGO2 and YAGO2Geo. We also changed its NED component, something that led to the generation of more queries than the previous version of the system. This change allowed the system to pose better queries which now have a geospatial dimension using the polygons that have been added to the knowledge base in combination with the huge amount of classes that YAGO2 has derived from wordnet and wikipedia.

ABBREVIATIONS - ACRONYMS

YAGO	Yet Another Great Ontology
OSM	Open Street Map
IDE	Integrated Development Environment
SPARQL	SPARQL Protocol and RDF Query Language
QA	Question Answering
URI	Uniform Resource Identifier
RDF	Resource Description Framework
SQL	Structured Query Language
HTML	HyperText Markup Language
WKT	Well-Known Text
NLP	Natural Language Processing
NER	Named Entity Recognition
NED	Named Entity Disambiguation
API	Application Programming Interface
OGC	Open Geospatial Consortium

REFERENCES

[1] Singh, Kuldeep & Lytra, Ioanna & Vidal, Maria-Esther & Punjani, Dharmen & Lange, Christoph & Auer, Sören & Thakkar, Harsh. (2017). QAestro – Semantic-Based Composition of Question Answering Pipelines. 10.1007/978-3-319-64468-4_2.

[2] Singh, Kuldeep & Both, Andreas & Sethupat, Arun & Shekarpour, Saeedeh. (2018). Frankenstein: A Platform Enabling Reuse of Question Answering Components.

[3] Singh, Kuldeep & Sethupat, Arun & Both, Andreas & Shekarpour, Saeedeh & Lytra, Ioanna & Usbeck, Ricardo & Vyas, Akhilesh & Khikmatullaev, Akmal & Punjani, Dharmen & Lange, Christoph & Vidal, Maria-Esther & Lehmann, Jens & Auer, Sören. (2018). Why Reinvent the Wheel-Let's Build Question Answering Systems Together. 10.1145/3178876.3186023.

[4] Punjani, Dharmen & Singh, Kuldeep & Both, Andreas & Koubarakis, Manolis & Angelidis, I & Bereta, Konstantina & Beris, Themis & Bilidas, Dimitris & Ioannidis, T & Karalis, N & Lange, Christoph & Pantazi, D & Papaloukas, C & Stamoulis, G. (2018). Template-Based Question Answering over Linked Geospatial Data. 10.1145/3281354.3281362.

[5] Karalis N., Mandilaras G., Koubarakis M. (2019) Extending the YAGO2 Knowledge Graph with Precise Geospatial Knowledge. In: Ghidini C. et al. (eds) The Semantic Web – ISWC 2019. ISWC 2019. Lecture Notes in Computer Science, vol 11779. Springer, Cham

[6] V. Teller, "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition Daniel Jurafsky and James H. Martin (University of Colorado, Boulder) Upper Saddle River, NJ: Prentice Hall (Prentice Hall series in artificial intelligence, edited by Stuart Russell and Peter Norvig), 2000, xxvi+934 pp; hardbound, ISBN 0-13-095069-6, $64.00,"*Computational Linguistics,* vol. 26, *(4),* pp. 638-641, 2000.

[7] S. Hakimov, S. Jebbara and P. Cimiano, "AMUSE: Multilingual Semantic Parsing for Question Answering over Linked Data," 2018.

[8] D. Diefenbach *et al*, "Towards a Question Answering System over the Semantic Web," 2018.

[9] H. Thakkar *et al*, "Towards an Integrated Graph Algebra for Graph Pattern Matching with Gremlin (Extended Version)," 2019.

[10] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Michael Röder, Daniel Gerber, Sandro Athaide Coelho, Sören Auer, and Andreas Both. 2014. AGDISTIS – Graph- Based Disambiguation ofNamed Entities Using Linked Data. In The SemanticWeb - ISWC 2014. Springer, 457–471. https://doi.org/10.1007/978-3-319-11964-9_29

[11] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. 2011. DBpedia spotlight: shedding light on the web ofdocuments. In Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011. ACM, 1–8. https://doi.org/10.1145/2063518.2063519

[12] Paolo Ferragina and Ugo Scaiella. 2010. TAGME: on-the-fly annotation of short text fragments (by wikipedia entities). In Proceedings ofthe 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010. 1625–1628. https://doi.org/10.1145/1871437.1871689

[13] Andrea Moro, Alessandro Raganato, and Roberto Navigli. 2014. Entity Linking meets Word Sense Disambiguation: a Unified Approach. TACL 2 (2014), 231–244. https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/291