International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 4 Issue: 12 218 – 225

Intelligent Geographical Information Retrieval using Ontology

Mansi A. Radke
Visvesvaraya National Institute of Technology, Nagpur, India
mansiaradke@gmail.com

Omkar B. Nagare
Ittiam Systems
Bangalore, India
omkar.balkrishna@gmail.com

Umesh A. Deshpande
Visvesvaraya National Institute of Technology, Nagpur, India
uadeshpande@gmail.com

Abstract:- With Web 3.0 gaining popularity, efficiently retrieving geographical information from ever growing geospatial data is an important task. We address two issues in this work.Firstly, consider the query *"Find all restaurants towards the east of Singhania school within a distance of 50km"*. In current systems to get the required result, first all the objects of type restaurant are extracted, then those within a required distance range are filtered and finally the approximate direction is determined by comparing co-ordinates. This processing is done at run-time i.e. dynamically when the query is executed. In this paper, we suggest a technique to avoid this computational overhead by constructing triples after pre-processing data from the existing ontologies to make implicit information explicitly available.Secondly, to address queries like *"Find all schools in Mumbai"*, the current systems manually construct a polygon which encloses Mumbai and then the required schools are filtered out. The task of determining a polygon which encircles the required locality is laborious if done manually and inaccurate with APIs like Google Maps. We propose an accurate technique which automatically forms the enclosing polygon for a region under consideration.

Keywords:- *GeoSPARQL, Semantic Web, Ontology, Parliament.*

*****

I. INTRODUCTION

With the advancement in internet technologies, the amount of digital information is expanding rapidly. Lot ofthe data content on the internet includes geographic references in the form of place names or spatial relationship to geographical places. For example, consider the sentence, VJTI engineering college is towards the east of Matunga railway station in which *VJTI* and *Matunga* are place names bound with the spatial relationship "*east of"*. Retrieving information specific to a place of interest is a common activity observed in search engines. Significant part of the total queries fired on a search engine contain queries related to geographic information retrieval [23]. Geographic information retrieval (GIR) [2] is an augmentation of information retrieval with geographic metadata. GIR aims at solving keyword based queries that involve geographic content. Relying only on keyword based search causes problem for newcomers who may not be familiar with keywords used in the system. Also, using different terminology for the same concept in the system will have low recall. For example, street and road are semantically similar but spelt differently. On the other hand, using homonymous terms will result into low precision. For example, the word close may refer to something nearby or may refer to something which is shut depending on the scenario. Hence, the ambiguities present in natural language inherently restrict the state-of-the-art keyword based search approaches. On the other hand, sometimes overwhelming results might be brought by the keyword search. A lot of time will be spent by the user to browse through many unwanted query results. Therefore a semantic based approach is needed to overcome these challenges by providing a meaningful web to machines. OGC (Open Geospatial Consortium) [16] proposed GeoSPARQL [6] as an extension to existing SPARQL [14] (which is a query language and a protocol for RDF data) for representing and querying geospatial data. However, GeoSPARQL has its own limitations. The runtime cost of the query is dominated by spatial join operations because the RDF data model involves great attention to detail. It is also dominated by filter expressions involving spatial operators. Pre-computing the spatial indices does not ensure improved performance as the RDF queries are much more flexible, making it difficult to predict which object to index and how the indexing should be done. Another problem is the way of storing spatial attributes corresponding to geometrical objects in the RDF data sets. They are usually stored as string literals which adhere to specific formats such as GML [3] or WKT [17]. So, for spatial computation, the GeoSPARQL query engine has to parse the strings which is a large runtime overhead as such functionsare stateless. Hence, the already parsed literals cannot be cached. This shows that GeoSPARQL provides a way to utilize the geospatial data available in order to make geospatial applications more accurate and useful. However, representing such data can result into inefficient data access. This paper has two distinct contributions to facilitate solving specific kind of geographical queries. The first kind of query is the one in which an individual wants to find the places within a threshold distance and towards a particular direction of a known place (or the current place). An example is a query -

find all the restaurants towards the south of Mumbai within 50 km. In the current systems, to get the required result, first all the objects of the type restaurant are extracted and then those lying within a particular distance range are filtered. Finally the relative direction is determined by comparing latitude and longitude information of the places under consideration. In the current systems, this processing is done at runtime and requires a lot of computational overhead. Hence an approach for efficient retrieval of the required information is proposed in this paper with the help of a modified ontology. Secondly, we consider a use case where a person wants to retrieve all the entities within a particular geometrical region. Typical query will be, find all the historical places in New York city. In order to obtain such information, the current systems follow a two-step process in which, first an approximate polygon enclosing New York is constructed. Then all the historical places which lie inside that polygon are retrieved. This paper deals with the first step of the process which is construction of the required polygon. The task of determining a polygon which encloses the locality under consideration is to be done either manually or using APIs like Google Maps. The manual process is laborious, whereas Google maps API provides only specific shapes like a rectangle or a circle to represent a particular region. This is quite inaccurate as the geographical regions are irregular in shape. Hence the determination of a polygon for a geographical region is time-consuming, inconsistent and error-prone. Hence an approach to automate the polygon construction to facilitate solving these typical kind of queries is proposed. The paper is structured as follows. The next section reviews the currently existing systems. This is followed by section 3 in which we explain the proposed approach in detail. The experimental results are presented in section 4. Section 5 concludes the paper and points out some directions for future work.

II. LITERATURE REVIEW

Many approaches have been proposed to solve the problem of efficient geographic information retrieval with the help of ontologies till date. Mei Kun et al. [20] proposed an architecture in 2007 which is purely based on an ontology to support semantic interoperability of Geographic Information Systems (GIS). This helps the users to refer to data as well as semantics behind it to retrieve the required result. Their system looks for the semantics related to individual entities in the data under consideration and fetches the results considering interaction among the different ontologies.

In 2009, Du Ping et al. [22] built a place name ontology (PNO) and implemented the PNO knowledge base (PNOKB) to substitute the state-of-the-art gazetteers. These gazetteers consist of a huge list of places and their geographical information. But due to the poor semanticsand simple data structure of gazetteers, GIR systems which rely on them have many limitations. Such systems have low recall value due to synonymous concepts, while they have low precision due to homonymous keywords present. Hence the systems suffer mainly because they do not consider the semantics behind the data present in the gazetteers. The authors have shown that, efficient retrieval can be achieved by using an ontology in GIR systems. In this paper, we have used a similar kind of ontology which is built by modifying PNO.

In 2010, Liu et al. [21] used their own ontology to consider the spatial and non-spatial attributes related to geographical information which was designed to achieve a higher level of retrieval, such that the results are retrieved not only based on a keyword but also using semantics which are related to direction, topology and distance. Earlier proposed researches focused primarily on the concept of topology and distance, but they did not consider spatial information associated with the data. The authors have given more importance to the spatial relations among the various geographical entities. By analyzing the current data, they have created additional data based on a reference ontology which describes the spatial relations. This was not considered before. In this paper, we have used a similar idea of adding additional data to make implicit information available explicitly.

In 2012, Bhattacharjee et al. [19] proposed an ontology based approach to manage geospatial information using semantics. Due to the semantic heterogeneity, a keyword based search in spatial catalogue containing a large amount of information becomes inaccurate. They created a standard using an ontology to handle the heterogeneity and incompatibility in order to effectively retrieve suitable data from the catalogues. They have used the Jena API [1] for building reasoner to resolve semantic ambiguity. Apache Jena (or Jena in short) is a free and open source Java framework for building semantic web and Linked Data applications. The framework is composed of different APIs interacting together to process the RDF data.

In the same year, Battle et al. [18] who are also authors of Parliament [11] and contributors to OGC, published an article focusing on GeoSPARQL, which discusses issues related to geospatial data access and indexing. Parliament is a triple store which complies with the RDF [12], RDFS [13], OWL [10], SPARQL, and GeoSPARQL standards. A triple consists of a subject and an object joined together by a predicate in the form subject-predicate-object. For example, Size is 45 or George knows Sally. A triple store or RDF store is a database made specifically for the retrieval and

storage of triples using semantic queries [15]. Many of the triple stores support partial or complete implementation of GeoSPARQL which shows that the geospatial applications are important. Efficient geospatial queries are required to fully utilize the geospatial RDF content present over the semantic web. So, for the users to truly realize geospatial semantic web, tools like Parliament and the technologies such as GeoSPARQL are necessary. In this paper, we have used Parliament triplestore for experimental purposes.

These approaches show the importance of using ontologies to solve the problems related to geographical information retrieval. In this paper, we propose and use a modified ontology approach based on PNO to achieve interoperability between two different kinds of data sets and their ontologies. This modified ontology is then used to generate additional information using Jena and GeoSPARQL to reduce the retrieval time of specific geospatial queries.

III. PROPOSEDAPPROACH

In this section, we present our two distinct contributions.

3.1 Reducing the Information Retrieval Time usinganOntology

Consider the query, "find all the places towards the east of Badlapur town within 50 km". Listing 1 shows a conventional query for getting the required information in the existing systems.

Listing 1: Conventional Query Example

```
1 #Find all the places within 50 km towards the east of Badlapur town
2 SELECT DISTINCT
3 ?object
4 WHERE {
5 ?object rdf:type lgdo:Place;
6 w3cGeo:long ?longitude_object;
7 w3cGeo:lat ?latitude_object;
8 geovocab:geometry
9 [geo:asWKT ?geometry_object].
10 lgeodata:node969958566
11 w3cGeo:long ?longitude_badlapur ;
12 w3cGeo:lat ?latitude_badlapur;
13 geovocab:geometry
14 [geo:asWKT ?geometry_badlapur].
15 FILTER ((geof:distance(?geometry_badlapur
,?geometry_object , uom:metre)/1000) <= 50)
16 FILTER  (?longitude_object>? longitude_badlapur)
17 }
```

In the given query, lgeodata:node9699585661 represents a geometrical object corresponding to Badlapur town. The execution plan for the query is explained in the following steps.

1. Firstly, a triple pattern, *?object rdf:type lgdo:Place* is matched against the triples from the data set (linkedgeodata) by an exhaustive search.
2. All the matched triples are then stored in the document *Result1*.
3. In a similar way, the matched triples for the triple patterns, *?object w3cGeo:long ?longitude object* and *?object w3cGeo:lat ?latitude* object are stored in the documents *Result2* and *Result3* respectively.
4. The documents *Result1* and *Result2* are then joined by *?object* to get the document *Join1*. Similarly, the documents *Result2* and *Result3* are joined in the same way to obtain the document *Join2*.
5. Line 8 and 9 from listing 1 is split into two triple patterns, *?object geovocab:geometry_ :blank_node* and *_:blank_node_geo:asWKT ?geometry_object*, where *_:blank_node* represents a blank node. 1This data is taken from linkedgeodata [9], which is a large RDF knowledge base containing geospatial RDF data. Linkedgeodata is in conformance with GeoSPARQL standards.
6. The matched triples from the above two triple patterns are joined together using the blank node to get the document *Join3*.
7. The documents *Join1* and *Join2* are joined again using *?object* to form the document *Join4* which is further joined with the document *Join3* to form the document *Object*.
8. The document *Object* contains various solutions to the variables *?object*, *?longitude_object*, *?latitude_object* and *?geometry_ object* which satisfy all the triple patterns mentioned between line 5 to line 9 from the Listing 1.
9. A similar process is followed to obtain the solutions for variables corresponding to *Badlapur town*. The difference here is that, the join operation will not be costly. This is because each document will contain only one triple specific to *lgeodata:node969958566*.
10. The filter expression, shown in line 15, is used to extract only those solutions for which value of the expression, *((geof:distance(?geometry badlapur, ?geometry object,uom:metre)/1000) <= 50)* is true.
11. For the expression to be evaluated, the solutions corresponding to the variable *?geometry_object* are parsed to recover their spatial coordinates.
12. The distance between the two objects is evaluated with the help of coordinates by using a spatial join.

13. All such filtered objects are again filtered by comparing their longitudes which also requires a normal join operation.
14. The solutions obtained after all these steps, represent the places towards the east of Badlapur town within 50 km range.
15. The query engine has to follow the above mentioned steps, at runtime to fetch the required result.

The above query takes 9 normal join operations and 1 spatial join operation to process the result. The join operations are costly. One of the filter expression requires a spatial join as well as parsing of string literals. Both the operations require heavy computation, thus increasing the cost of the query. The places which are towards the east of Badlapur town can be obtained by comparing latitude-longitude information available. In this case, if longitude of the object under consideration is greater than that of Badlapur town then we can say that the object is towards the east of Badlapur town approximately as shown in Figure 1. Similar comparisons can be made to figure out the other directions. All these mentioned tasks are executed at runtime, resulting into a high response time. In order to retrieve implicit information, complex queries are to be written which requires heavy join operations, filter expressions and literal parsing. In order to avoid such computations, additional triples can be generated by pre-processing the existing data to make the implicit information explicit. This pre-processing is an one time activity which is done statically.

Figure 1: Filtering Entities towards the East of Bad
-lapur within 50 km

For that, first all the geometrical objects are retrieved from linkedgeodata. Then for each such object, the distance from every other object is calculated using the function *geof:distance*. After all such distances have been computed, we classify each pair of the geometrical objects into 4 different categories depending upon the distance. If the distance between them is less than or equal to 10 km then the pair is added to category-10. Similar process is done for categorizing the object pairs into category-50, category-100 or category-500. Consider a scenario where a user is walking or running. In such cases, a user might be interested only in those entities which are within 10 km distance. However, if a user is travelling by a car, he or she might travel up to 100 km or 500 km. This categorization, depending upon the distance, is needed to add more flexibility to the system, as we do not know the mode of transport the user is using. All the pairs with the distance of separation greater than 500 km are discarded. The relative direction between the objects is then determined, after every such pair is categorized. The determination of direction is done by calculating angular separation between the objects using their latitude and longitude information [7]. For example if the angular separation is between -22.5^0 to 22.5^0 then the relative direction is east. Figure 2 shows the determination of the direction for pair of objects from category-50 where one of the object from the pair is Badlapur town. These computations are carried out using Jena API which is a Java framework designed for creation, storage and retrieval of semantic data from the triple stores.

Figure 2: Determination of the Direction using An
-gular Separation between Places

With the above information in hand, the triples can be generated by using an appropriate ontology to enable intelligent information retrieval. Table 1 represents the summarized structure of the proposed ontology. It consists of four columns namely, *Type*, *Properties*, *SameAs* and *ReverseOf*. Column *Type* is used to specify the type of object properties mentioned in the ontology. *Properties* column contains all the different type of predicates created in the ontology. Column *SameAs* is used to state the equivalence relation among object properties while Column *ReverseOf* represents an inverse-functional relation among them.

The properties listed can be divided into three major types namely directional, relative and proximity. The directional type properties can be further subdivided into 4 categories depending upon threshold limit used. They are threshold-10, threshold-50, threshold-100 and threshold-500, each containing all the eight directions. For example, *10-northof* represents a predicate of type threshold-10 where the relative direction between the objects is north and they are

 ISSN: 2321-8169
218 – 225

within 10 km distance. Relative position between the two objects is represented using the properties *leftof* and *rightof*. Also for specifying proximity, the predicates *nearof* and *farof* are used. The objects are said to be near if they are within 50 km distance of each other, otherwise far. For each categorized pair, a triple is generated using an appropriate directional property.

Table 1: Proposed Ontology

Type	Properties	SameAs	ReverseOf
Directional	10-southof	-	10-northof
	10-southwestof	-	10-northeastof
	10-westof	leftOf	10-eastof
	10-northwestof	-	10-southeastof
	10-northof	-	10-southof
	10-northeastof	-	10-southwestof
	10-eastof	rightOf	10-westof
	10-southeastof	-	10-northwestof
	50-southof	-	50-northof
	50-southwestof	-	50-northeastof
	50-westof	leftOf	50-eastof
	50-northwestof	-	50-southeastof
	50-northof	-	50-southof
	50-northeastof	-	50-southwestof
	50-eastof	rightOf	50-westof
	50-southeastof	-	50-northwestof
	100-southof	-	100-northof
	100-southwestof	-	100-northeastof
	100-westof	leftOf	100-eastof
	100-northwestof	-	100-southeastof
	100-northof	-	100-southof
	100-northeastof	-	100-southwestof
	100-eastof	rightOf	100-westof
	100-southeastof	-	100-northwestof
	500-southof	-	500-northof
	500-southwestof	-	500-northeastof
	500-westof	leftOf	500-eastof
	500-northwestof	-	500-southeastof
	500-northof	-	500-southof
	500-northeastof	-	500-southwestof
	500-eastof	rightOf	500-westof
	500-southeastof	-	500-northwestof
Relative	leftof	westof	Rightof
	rightof	eastof	Leftof
Proximity	nearof	-	Farof
	farof	-	Nearof

Threshold value for property is determined from the category to which the pair belongs. For each pair from the category-10 and the category-50, a triple is generated with the proximity type property *nearof*. In addition, while creating the triples with directional properties, the triples representing relative directions are also created only for those pairs where the determined direction is either west or east. Considering the proposed ontology, the query for the same problem can be represented as listed in Listing 2.

Listing 2: Query Example using Proposed Ontology

```
1 #Find all the places within 50 km towards the east of
Badlapur town
2 SELECT DISTINCT
3  ?object
4  WHERE {
5              ?object
6              rdf:type lgdo:Place;
7             :50-eastof lgeodata:node969958566.
8 }
.
```

The query requires following steps to execute.

1. Firstly, All the triples from the data set are matched against the triple pattern, *?objectrdf:type lgdo:Place* and are then stored in the document *Result1*.
2. A similar process is done to find the matching triples for the pattern, *?object :50-eastof lgeodata:node969958566* which are stored in the document *Result2*.
3. In order to obtain the solutions for the variable *?object*, the documents *Result1* and *Result2* are joined by using *?object*.
4. The solutions obtained, represent the places towards the east of Badlapur town within 50 km range.

Only one normal join operation is required as compared to 9 in the conventional case. This query is free from filter expressions. There is no spatial information needed to execute the query, hence no spatial joins are required which reduces the time required. As literals are not represented as strings, parsing is not required. Hence the cost of the query is very low compared to that of the conventional query. The pre-processing step has a large overhead of memory. However, we assume large amount of memory is available for storage. Additionally if one wants to determine the places which are located within an arbitrary distance which is different from the standard categories, then using the proposed ontology, one can filter out categories which contain the potential candidates for retrieval. For example,

consider the query "Find all the restaurants within 5 km from Dadar railway station". Here, only the objects from the category threshold-10 are considered while the rest are discarded. Then among these objects, the objects within 5 km distance from Dadar railway station are found using a normal GeoSPARQL query. Thus by using the proposed ontology and GeoSPARQL, required geographic information can be efficiently retrieved with low query execution time.

3.2 Polygon Determination for a Particular Region

Consider the query, "find all the restaurants in Texel island". Listing 3 represents a GeoSPARQL query to retrieve all the restaurants in Texel island. The required results can be fetched by using the POLYGON which approximately represents the geographical boundary of the Texel island. The query extracts all the objects which are of the type *lgeodata:Restaurant* first by triple matching. The solutions which match with the triple patterns stated between line 5 to 8 are bound to the respective variables by join operations. The solutions which are bound to the variable *?geometry_texel*, are filtered (line 9) further by using *geof:sfWithin* function. This function needs a spatial join and literal parsing to determine the objects which lie within the polygon mentioned in the query.

Listing 3: Query using GeoSPARQL Polygon

```
1 #Find all the restaurants in Texel island.
2 SELECT DISTINCT
3 ?thing ?thingLabel
4 WHERE {
5 ?thing
6 a lgeodata:Restaurant;
7 rdfs:label ?thingLabel;
8 :geometry [geo:asWKT ? geometry_texel].
9FILTER (geof:sfWithin(?geometry_texel ,
"POLYGON((53.1 4.8,53.1 4.8,53.1 4.8,53 4.8,53 4.7,52.9
4.7,53 4.7,53.1 4.7,53.1 4.8))"^^geo: wktLiteral))
10 }
```

Conventionally to get the result of such queries, current systems approximately construct a square or a polygon around the region under consideration. The latitude and longitude of the vertices of polygon are found either manually or using the Google maps API. With these set of points in hand, a polygon of type *geo:wktLitera*l can be formed which is then used to retrieve the objects inside it using GeoSPARQL function - *geof:sfWithin*. The Google maps API can give the polygon in the form of a predefined shape like rectangle, square etc. which is not accurate most of the times because of the irregularities in the geographical shape of different places. Manual approach will be quite accurate but time-consuming, inconsistent and laborious. The task of determining a polygon which encloses the region under consideration is challenging. It might give erroneous results due to wrong polygon approximation such as getting places which are actually outside the region. This paper proposes an algorithm to automate the process of determining the vertices of the polygon which encloses the required region. The steps below represent a schematic of the algorithm for constructing a polygon. The detailed process is explained in following paragraphs.

1. Firstly, all the objects which are part of the region under consideration are obtained along with their latitude-longitude information which is stored in the form of a point.
2. A central point is then obtained by taking the average of all the obtained points.
3. All the points are divided into eight categories according to their direction with respect to the central point.
4. For each region, one point is obtained which is having the largest distance from the central point.
5. Each farthest point represents one of the vertex of the required polygon.

In order to obtain information required in step 1, one of the way is to use the geonames API [5] which is a collection of various RESTful webservices built on the geonames data set [4]. The geonames API provides a webservice, http://api.geonames.org/contains?geonameId= <id>&username=<uname>, which returns list of objects in XML or JSON format which are part of the region. The query string contains two variables, *geonamesId* and *username*. The *geonamesId* is an identifier assigned to the region under consideration in the geonames data set, while *username* is the name of the user who is availing the webservice. Another way to get the latitude-longitude information is by using the *gn:parentFeature* property present in the geonames ontology. The predicate connects two entities where one object is part of the other geographically. For example, if objects which are part of Texel island are to be considered then the required query can be realized as shown in Listing 4.

Simple average can not be employed in step 2 because the surface of the earth follows elliptical geometry thus making latitude-longitude points non-cartesian. At the antimeridian, the longitude values wrap around, while the latitude values wrap around at the poles thus giving an incorrect simple average. In order to obtain the central point, all the latitude-longitude points are first converted to 3-D cartesian points and then their average is taken.

Listing 4: Places inside Texel island

```
1 #find all the places which are part of Texel island 2
SELECT DISTINCT
3 ?thing ?long ?lat
4 WHERE {
5?object_texel gn:name "Texel".
6?thing
7gn:parentFeature ?object_texel;
8 w3cGeo:long ?long ;
9 w3cGeo:lat ?lat.
10 }
```

This average value is then projected back to the surface to get the central latitude-longitude point.

In Step 3, points are distributed among the different categories, by calculating their angular separation with respect to the central point. For example, if the angular separation is between 67.5^0 to 112.5^0, then the point is added to the north category. The points can be added to the categories of the other directions in a similar way. Same process is repeated for each point.

Step 4 determines a latitude-longitude point which is farthest from the central point. For calculating the distances, we use haversine formula [8]. A polygon enclosing the region under consideration is constructed in step 5. Figure 3 shows the polygon obtained for Texel island using the proposed approach where the points are divided into eight categories corresponding to the eight standard directions.

Figure 3: Constructing Polygon for Texel island using 8 categories

The polygon formed for the region is not as accurate as the actual boundary of the corresponding geographical region because the physical boundaries of geographical regions are highly irregular in shape. However, the accuracy of the polygon will depend upon the accuracy with which places inside the region are obtained using API or geonames data set. Considering more number of categories will also improve the accuracy of the polygon which is evident from Figure 4. It shows the polygon obtained for Texel island when the points are divided into 16 categories. This will add to the computational overhead in the case where the number of objects lying inside some of the regions could be very large. Thus proposed approach automates the process of determination of a polygon which approximately encloses a particular region under consideration.

Figure 4: Constructing Polygon for Texel island us -ing 16 categories

IV. EXPERIMENTATION

We have computed the time required to fetch the required information using the proposed ontology and compared it with that of the conventional systems.We have then calculated the speed up achieved and averaged it over 1000 queries. All the quantities are in milliseconds. The experiments were performed on Intel core i3-2130 CPU (@3.40 GHz), 4 GB RAM, 64-bit Windows 8.1 OS. The average speed up with our proposed ontology was 23.8 times as compared to the conventional method

Table 2: Time Required for GIR

Direction	Current	Proposed	Speedup
southof	15715	822	19.12
southwestof	-	750	-
westof	20116	900	22.5
northwestof	-	650	-
northof	12632	420	30.8
northeastof	-	807	-
eastof	17798	453	23.64
southeastof	-	685	-
Average speedup			23.8

Polygon constructed for Texel island matches very closely to the *Texel island* which is evident from Figure 3. Polygon construction for the given example requires 1450 milliseconds at runtime which is quite less compared to the manual process.

V. CONCLUSION

Geographic information retrieval and storage is the crux of sharing spatial information across the web. Geographic semantic web helps in retrieving suitable information compared to the systems which rely only on keyword based search. However, querying geospatial information using GeoSPARQL has inherent limitations like, difficulties in

finding appropriate spatial indices, loosely connected RDF data, need of spatial joins etc. Ontologies can be employed for the task of efficient retrieval of geospatial information. We have successfully created additional triples and stored them with the help of the proposed ontology by preprocessing the data from the existing ontologies. This additional information removes the need of the spatial joins or spatial indices. We have observed average speed up of 23.8 times compared to the conventional way of querying using GeoSPARQL. We have also proposed a technique for automatic construction of a polygon for a region under consideration. This is an important problem since the boundary for a region might be highly irregular. The automated polygon construction can be done in much less time as compared to the manual process. It is more accurate than using techniques such as using Google maps APIs since specifying a polygon for the irregular boundary might be problematic.

This work focuses on two types of queries as mentioned in section 3. Our future work will consider more geospatial queries for which implicit information can be made explicitly available in order to reduce the retrieval time.

Listing 5: RDF Prefixes

```
1 rdf:<http://www.w3.org/1999/02/22-rdfsyntax -ns#>
2 rdfs:<http://www.w3.org/2000/01/rdfschema#>
3 geo:<http://www.opengis.net/ont/ geosparql#>
4 geof:<http://www.opengis.net/def/ function/geosparql/>
5 uom:<http://www.opengis.net/def/uom/OGC /1.0/>
6 geovocab:<http://geovocab.org/geometry #>
7 w3cGeo:<http://www.w3.org/2003/01/geo/ wgs84_pos#>
8 owl:<http://www.w3.org/2002/07/owl#>
9 lgeodata:<http://linkedgeodata.org/ triplify/>
10 lgdo:<http://linkedgeodata.org/ontology />
11 gn:<http://www.geonames.org/ontology#>
```

6. REFERENCES

[1] Apache jena - home. https://jena.apache.org/.

[2] Geographic information retrieval - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/ Geographic_information_retrieval.

[3] Geography markup language | ogc. http://www.opengeospatial.org/standards/gml.

[4] Geonames. http://www.geonames.org/.

[5] Geonames web service documentation. http://www.geonames.org/export/web-services.html.

[6] Geosparql - a geographic query language for RDF data | OGC. http://www.opengeospatial.org/standards/geosparql.

[7] Get direction (compass) with two longitude/latitude points - stack overflow. http://stackoverflow.com/questions/8502795/ get-direction-compass-with-two-longitude-latitude-points.

[8] Haversine formula - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Haversine_formula.

[9] linkedgeodata.org : About. http://linkedgeodata.org/About.

[10] Owl - semantic web standards. https://www.w3.org/2001/sw/wiki/OWL.

[11] Parliament high-performance triple store. http://parliament.semwebcentral.org/.

[12] RDF - semantic web standards. https://www.w3.org/RDF/.

[13] RDFS - semantic web standards. https://www.w3.org/2001/sw/wiki/RDFS.

[14] SPARQL query language for RDF. https://www.w3.org/TR/rdf-sparql-query/.

[15] Triplestore - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Triplestore.

[16] Welcome to the OGC | OGC. http://www.opengeospatial.org/.

[17] Well-known text representation of coordinate reference systems | OGC. http://www.opengeospatial.org/standards/wkt-crs.

[18] R. Battle and D. Kolas. Enabling the geospatial semantic web with parliament and geosparql. Semantic Web, 3(4):355–370, 2012.

[19] S. Bhattacharjee, R. R. Prasad, A. Dwivedi, A. Dasgupta, and S. K. Ghosh. Ontology based framework for semantic resolution of geospatial query. In Intelligent Systems Design and Applications (ISDA), 2012 12th International Conference on, pages 437–442. IEEE, 2012.

[20] M. Kun and B. Fuling. An ontology-based approach for geographic information retrieval on the web. In Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on, pages 5959–5962. IEEE, 2007.

[21] W. Liu, H. Gu, C. Peng, and D. Cheng. Ontology-based retrieval of geographic information. InGeoinformatics, 2010 18th International Conference on, pages 1–6. IEEE, 2010.

[22] D. Ping and L. Yong. Building place name ontology to assist in geographic information retrieval. In *Computer Science-Technology and Applications, 2009. IFCSTA'09. International Forum on, volume 1*, pages 306–309. IEEE, 2009.

[23] M. Sanderson and J. Kohler. Analyzing geographic queries. In *SIGIR Workshop on Geographic Information Retrieval, volume 2*, pages 8–10, 2004.