Geo-Store: A Framework for Supporting Semantics Enabled Location-Based Services with RDF Triple Stores

Wei-Shinn Ku† Haiquan Chen‡ Chih-Jye Wang† Chuan-Ming Liu♯

†Dept. of Computer Science and Software Engineering
Auburn University, Auburn, AL 36849, USA
{weishinn,wangchj}@auburn.edu
‡Dept. of Mathematics and Computer Science
Valdosta State University, Valdosta, GA 31698, USA
hachen@valdosta.edu
♯Dept. of Computer Science and Information Engineering
National Taipei University of Technology, Taipei 106, Taiwan
cmliu@csie.ntut.edu.tw

Abstract

Today, location-based services (LBS) are very popular information services for mobile users who access the services with location-aware mobile devices through wireless networks. The data sources of advanced LBSs are on the Web, which consists of a huge volume of data in human-readable format. The main goal of the Semantic Web is to augment the Web with information so that computers can understand and exchange Web data. Therefore, it will be a trend for LBS providers to employ Semantic Web data sources to develop semantics-enabled location-based services. However, there are currently limited solutions to process spatial queries, the building blocks of most LBSs, on Semantic Web data for supporting location-based services. This article describes our novel spatial query techniques, which are able to efficiently evaluate spatial queries on RDF triple stores (Semantic Web data management systems) for providing semantics-enabled location-based services.
1 Introduction

With the popularity of wireless networks and mobile devices, Location-Based Services (LBS) have become indispensable applications to mobile users. The global GPS navigation and LBS market size has been predicted to grow significantly from $1.6 billion in 2009 to $13.4 billion in 2014 according to IEMR’s report “Global GPS Navigation and Location Based Services Forecast”. In addition, the Web consists of a huge volume of data which require the use of human intelligence. The main goal of the Semantic Web is to augment the Web with information so that computers can understand and exchange Web data. Consequently, it will be a trend for LBS providers to employ Semantic Web data sources to develop semantics-enabled location-based services.

The Resource Description Framework (RDF) data model is designed for interchanging schema-relaxable (or schema-less) data on the Semantic Web [16]. RDF models the linking structure of the Web as triples of the form \( \langle s, p, o \rangle \), where \( s \) is a subject, \( p \) is a predicate, and \( o \) is an object. Each triple represents the relationship between the subject and the object. A collection of triples forms a directed graph, where the edges represent predicates between subjects and objects, which are represented by the graph nodes. With the increasing amount of RDF data on the Web, researchers developed specialized architectures for RDF data management named triple stores [4, 5, 1, 18, 14]. Generally, these solutions employ various indexing, compression, and query optimization techniques for scalable and efficient management of RDF data.

The Semantic Web is an ideal data source for supporting the state-of-the-art location-based services that employ dynamic or near real-time information. For example, a mobile user may utilize LBS to search for nearby restaurants based on recent reviews on the Web. However, to the best of our knowledge, there are limited solutions to process spatial queries, the building blocks of most LBSs, on triple stores for supporting advanced location-based services. Therefore, the goal of the Geo-Store project is to develop novel spatial query techniques that are able to efficiently evaluate spatial queries on RDF triple stores for providing semantics-enabled location-based services. The main features of our Geo-Store system are as follows.
The Geo-Store system employs a novel representation to model spatial features and utilizes a spatial mapping mechanism to preserve spatial locality.

The Geo-Store system is able to effectively process both range and \( k \) Nearest Neighbor (\( k \text{NN} \)) queries, the building blocks of many LBS applications, on RDF triple stores.

Users are able to integrate and operate the Geo-Store system on existing triple stores (e.g., RDF-3X) with limited changes.

2 Related Work

Location-based services are any service that takes into account the geographic location of an entity and are accessible with mobile devices through wireless networks [8]. With the prevalence of GPS-enabled mobile devices and the introduction of 4G mobile telecommunications services, various commercial LBSs, such as location-based dating, location-targeted advertisement, and child safety services, appear in our lives [2]. In addition, novel LBS applications are able to exploit online Semantic Web sources (e.g., LinkedGeoData\(^1\)) about nearby physical entities of a user to provide personalized services [19]. Today, location-based services are applied in different fields, such as emergency response, navigation, product tracking, social networks, etc.

The Semantic Web is a group of techniques for machines to understand information and exchange knowledge on the World Wide Web. The cornerstone of the Semantic Web is a logical data model named RDF which employs triples to represent the relationships between subjects and objects. In order to efficiently manage RDF data, there are numerous systems invented for storing and querying triple collections [4, 5, 1, 18, 14]. For improving performance and scalability, Abadi et al. [1] introduced a solution by vertically partitioning the RDF data. Their solution’s performance can be further improved by utilizing a column-oriented DBMS, which is a database designed specially for the vertically partitioned case. Weiss et al. [18] proposed a sextuple-indexing scheme, named Hexastore, which allows for quick and scalable general purpose query evaluation for RDF data management. Hexastore

\(^1\)http://linkedgeodata.org
achieves significant advantages in performance compared with previous solutions for managing RDF triples. The RDF-3X engine [14] is an implementation of the SPARQL query language [17] for RDF by pursuing a simplified architecture with streamlined indexing and query processing. The design of RDF-3X completely eliminates the need for index tuning by exhaustive indexes for all permutations of subject-predicate-object triples and their binary and unary projections. However, all the aforementioned triple stores do not consider the unique features of spatial data when they encode and store RDF triples.

Recently, Perry et al. [15] presented an extension to SPARQL, named SPARQL-ST, for complex spatiotemporal queries. They implemented SPARQL-ST by extending a relational database, which is not an effective way of managing RDF triples. The Strabon system is an implementation of the data model stRDF and the query language stSPARQL [11], which are extensions of RDF and SPARQL for managing spatial and temporal data. Nevertheless, Strabon is built on top of the RDF store Sesame2 which is not as efficient as the aforesaid new generation RDF triple stores [12]. Brodt et al. [3] proposed a solution to integrate spatial query processing into RDF triple stores. However, their design cannot efficiently evaluate queries on large-scale spatial data sets, which are essential for the state-of-the-art LBSs. Furthermore, Parliament [9, 10] is a triple store and employs a similar approach to [3] for spatial queries and storage of the data. The GeoSPARQL draft standard [6] is currently being implemented in Parliament.

3 The Geo-Store System

Although many RDF triple stores have been proposed during the past few years, most of them were designed and optimized mainly for non-spatial Semantic Web data. In order to enable spatial query processing on RDF triple stores, the state-of-the-art method [3] is to treat all the Universal Resource Identifiers (URIs) and literals, non-spatial or spatial, equally and replace each URI and literal with an integer ID by dictionary encoding. As a result, each spatial literal (e.g., the latitude and longitude coordinates or the address) is mapped to a randomly generated ID. Afterwards, an R-tree (or its variant) [7] is created to index all the IDs referred to spatial literals.

2http://www.openrdf.org/
However, this dictionary encoding-based method may incur extreme inefficiency in evaluating spatial queries on RDF triple stores, especially when coping with large-scale spatial data sets [3]. In this article, we employ a novel representation, GeoHilbert – an RDF vocabulary as an extension of the Geography Markup Language (GML), to model spatial features based on their Hilbert curve [13] transformation information and to utilize Spatially Aware Mapping (SAM), instead of dictionary mapping, to encode URIs and literals for preserving spatial locality.

3.1 Data Representation

Because many huge data repositories published online contain geographic location or spatial relationship information, we are witnessing a new research trend of modeling geographic entities ontologically and querying their spatial relations in the Semantic Web community. As one of the Semantic Web technologies, the Resource Description Framework treats relationships as first-class citizens and, consequently, can work as an ideal tool for modeling and querying complex and large amounts of relations between spatial entities. The XML formatted semantic data can be converted to the corresponding RDF representation with minor modifications. For example, Listing 1 shows an XML data excerpt from the OpenStreetMap project. This snippet describes the referred information (semantic and spatial) about a restaurant in Pasadena, California, USA. Its corresponding RDF representation is demonstrated in Listing 2. Specifically, gml represents the namespace of the GML standard, and gstore corresponds to the namespace of our Geo-Store system.

Listing 1. An XML data excerpt from the OpenStreetMap project.

```xml
- <node id="738330640" lat="34.1135498" lon="-118.1235345" user="AM909" uid="82317" visible="true" version="1" changeset="4737707" timestamp="2010-05-18T11:26:40Z">
  <tag k="amenity" v="restaurant" />
  <tag k="cuisine" v="american" />
  <tag k="name" v="Colonial Kitchen" />
  <tag k="source" v="usgs_imagery;survey;image" />
  <tag k="source_ref" v="AM909_DSCU3253" />
</node>
```

3http://www.openstreetmap.org/
4http://www.opengis.net/gml
5http://example.org/gstore
3.2 System Architecture

Figure 1 shows the system framework of the Geo-Store system, which consists of four main components: query parser and planner module, spatially aware mapping module, internal processing module, and dictionary decoding module.

3.2.1 Spatially Aware Mapping Module

In most of the existing triple stores, each URI or literal is replaced with a unique integer ID by dictionary mapping because the triples may contain very long URIs and string literals. In our system, in order to support efficient spatial query evaluation on triples, we extend the standard dictionary encoding and design a Spatially Aware Mapping (SAM) approach to encode all the URIs and literals, as well as preserve spatial locality. SAM includes two major
components, *GeoHilbert Transformation* and *Discriminative Encoding*. With GeoHilbert Transformation, original RDF triples are transformed to the corresponding GeoHilbert representation. In addition, SAM assigns an ID to each URI and literal in a discriminative way in order to maintain spatial locality by executing the discriminative encoding component.

**GeoHilbert Transformation**  We utilize a novel RDF representation, GeoHilbert, to model spatial data in GeoStore. GeoHilbert is designed for incorporating the Hilbert curve-based spatial transformation information into original RDF data. GeoHilbert consists of a subject, *HilbertMapping* and four predicates: *StartPoint*, *Order*, *Orientation*, and *Pos*. Specifically, the values of *StartPoint*, *Order*, *Orientation*, and *Pos* are the location of the Hilbert curve starting point, the curve order $O$, the curve orientation $\theta$, and a Hilbert value translated from a given spatial location, respectively.

Listing 3 demonstrates the corresponding RDF data in the GeoHilbert representation for the restaurant described in Listing 1. As shown in Listings 2 and 3, the GeoHilbert representation appends to the original RDF representation an additional RDF statement, \{*gstore:Point738330640*, *gstore:Pos*, “208083”\}, which stores the relative position of the referred point along the Hilbert space-filling curve.

```
Listing 3. The GeoHilbert representation of the restaurant described in Listing 1.

```...```
gstore:HilbertMapping gstore:StartPoint "32.3372200 -114.1369000"
gstore:HilbertMapping gstore:Order "10"
gstore:HilbertMapping gstore:Orientation "Up Left"
...
```...```
gstore:Point738330640 gml:amenity "restaurant"
gstore:Point738330640 gml:cuisine "american"
gstore:Point738330640 gml:name "Colonial Kitchen"
gstore:Point738330640 gml:pos "34.1135498 -118.1235345"
gstore:Point738330640 gstore:Pos "208083"
...
```

**Discriminative Encoding**  The Discriminative Encoding (DE) component takes the GeoHilbert representation based data as the input. For each literal following the predicate *Pos*, DE assigns it a positive ID which is exactly
the same as its Hilbert value. For example, in Listing 3, the literal “208083” in the triple \{\text{gstore:Point738330640}, \text{gstore:Pos}, “208083”\} will be assigned with ID 208083. For all the other literals or URIs, DE acquires a random integer by calling a dictionary encoding function, and then returns the opposite of that integer as the ID (e.g., a randomly generated integer “1000” will be returned as “−1000”). All the assigned IDs are maintained in a B+-tree to speed up the dictionary lookup.

3.2.2 Internal Processing Module

Generally, the evaluation of SPARQL queries is based on pattern matching. In this system, we maintain in memory all six possible permutations of subject (S), predicate (P), and object (O) in six separate indices in order to guarantee that every possible query pattern in a SPARQL query with variables in any position of a triple can be answered by only performing a single index scan. Specifically, these permutations are named as SPO, SOP, OSP, OPS, PSO, and POS indices, respectively. Notice that instead of the original URIs or literals, all the six indices here consist of integer IDs, which are assigned by the discriminative encoding component. This ID-based indexing scheme can both save memory space and accelerate join processing.

3.2.3 Query Parser and Planner Module

When Geo-Store receives a SPARQL query \(q\), it parses \(q\), identifies the IDs that have been assigned to the literals in \(q\) according to SAM, and employs the retrieved IDs to replace the corresponding literals in \(q\). By executing these processes, the subsequent evaluation of \(q\) purely relies on the comparisons of IDs instead of the original literals. A query evaluation plan can then be generated accordingly.

3.2.4 Dictionary Decoding Module

After the SPARQL query is evaluated by the internal processing module, the dictionary decoding module transforms the resulting IDs back to their original literals as the query results. In Geo-Store, we employ a B+-tree structure to implement this ID-to-literal mapping.
4 Semantics Enabled Location-based Services with Spatial Encoding

The SPARQL query language is standardized by the World Wide Web Consortium (W3C) for querying RDF data. Evaluation of SPARQL queries is based on pattern matching on the target RDF graph. A pattern may contain variables that are bound to a URI or a literal. In addition, the SPARQL query language provides a number of built-in FILTER functions, which can be easily extended to support spatial operations (or constraints). Because range and \( k \) nearest neighbor queries are the building blocks of location-based services, we elaborate on how to evaluate the two important query types by utilizing our system in this section.

4.1 Range Queries

Listing 4 shows a sample range query which is launched to retrieve the latitude and longitude values of all the qualified gas stations with the two selection conditions: (1) the location is less than 10 miles away from the query point (the mobile user’s current location), and (2) the gas price is cheaper than $3.00 per gallon.

```
SELECT ?coordinates
WHERE { ?point a "Gas Station" .
  FILTER (gstore:range(?point, CURRENT_LOCATION, 10, "mile"))
  FILTER (?price < 3.00)
}
```

In Geo-Store, each spatial object is annotated with its Hilbert value information based on the GeoHilbert representation. Figure 2(a) illustrates an example of mapping spatial objects in a two dimensional space into their Hilbert values. In Figure 2(a), the entire space is divided by the Hilbert curve into 64 grids with their unique Hilbert values, and we can acquire the Hilbert values of the spatial objects \( A \), \( B \), and \( C \), as 29, 32, and 7, respectively. Depending on the desired resolution, more fine-grained curves can be recursively generated based on the Hilbert curve generation algorithm. Figure 2(b) demonstrates how a range query can be processed in our system by taking advantage of the Hilbert values of spatial objects. As depicted in Figure 2(b), the query point is \( q \) and the
query window of the range query $Q_R$, highlighted in red, covers the three Hilbert curve segments [10-12], [17-18], and [28-31]. After obtaining the above three curve segments, our system retrieves all the spatial objects whose Hilbert values are embraced by the three curve sections and treats the retrieved spatial object set $R'$ as the inclusive query result. Subsequently, our system examines all the spatial objects in the grids that partially overlap with the query window (i.e., grids [10-12], [28] and [31]) to check if their exact locations (i.e., latitude and longitude values) are within the query window by dictionary lookup. Finally, the exact query result $R$ is returned to the user after filtering out those objects whose locations are outside the query window in the partially overlapping grids.

4.2 $k$ Nearest Neighbor Queries

In this subsection, we extend our range query solution to evaluate $k$ nearest neighbor queries efficiently in Geo-Store. Listing 5 demonstrates a sample $k$ nearest neighbor query which is issued to retrieve the latitude and longitude values of the gas stations that are among the $k$ closest gas stations to the query point with a listed gas price cheaper than $3.00 per gallon.
Listing 5. The sample $k$ nearest neighbor query in Geo-Store (use case 2).

```sql
SELECT ?coordinates
WHERE { ?point a "Gas Station" .
  FILTER (gstore:NN(?point, CURRENT_LOCATION, k))
  FILTER (?price < 3.00)
}
```

Given a query point $q$, Geo-Store searches the spatial objects in both the ascending and descending directions of Hilbert values until $k$ spatial objects are found, and then records the result set as $S$. Supposing the object $o$ has the longest distance to $q$ in $S$, $Distance(q, o)$ (the distance between $q$ and $o$) is set as the search upper bound for the subsequent range query. Afterwards, Geo-Store launches a range query $Q_R$ with $Distance(q, o)$ to decide the query window size and then acquires the query result $R'$. Next, Geo-Store identifies the top $k$ objects in $R'$ based on their respective distances to $q$ in order to derive the final query result $R$.

Figure 3 demonstrates the evaluation of a $k$ nearest neighbor query with Geo-Store. As shown in Figure 3(a), based on the query point $q$, Geo-Store first searches for spatial objects with Hilbert values $\geq 30$ and $< 30$ in parallel until $k$ objects are discovered. Next, assume that the spatial object that has the longest distance to $q$ among the above $k$ objects is object $B$. Then, a range query $Q_R$ with the distance between $q$ and $B$ as the search upper bound is issued; $Q_R$ returns the result set $R'$, as depicted in Figure 3(b). In this example, set $R'$ encompasses objects which fall on the five Hilbert curve segments, [8-20], [23-24], [27-32], [35-36], and [53-54]. Finally, Geo-Store computes the top $k$ objects in $R'$ with the shortest distance to $q$ as the exact query result.

5 Experimental Validation

For performance evaluation, we compared our Geo-Store system with Brodt’s system [3] and the Strabon system [12] by measuring their response time on processing two popular location-based spatial query types, range and $k$NN. We acquired the locations of points of interest (POI) in California from the U.S. Geological Survey\(^6\). This real-world data set contains one million POIs distributed over the state of California with 63 different POI

\(^6\)http://cumulus.cr.usgs.gov/
categories, including airport, hospital, school, populated place, road junction, etc. Each category exhibits a distinct density and distribution. For each result in this section, we ran 500 corresponding queries with distinct query points whose locations were randomly generated within the state of California. All the experiments were conducted on the same Windows machine.

5.1 Efficiency Comparison

In this subsection, we focus on comparing the efficiency of Geo-Store, Brodt’s, and Strabon in terms of query response time.

5.1.1 Range query

We first report our experimental results of range queries. We gradually increased the area of the query window to investigate its impact on query response time. Here we specify the area of a query window by using the percentage of the region of California. For example, a query window with the percentage of 0.01% represents an area of around 16 square miles (given the region of California is around 160,000 square miles). As shown in Figure 4 (a), with the enlargement of the query window, the response time kept increasing for all systems. However, our Geo-Store system always outperformed Brodt’s and Strabon. For instance, with the query window of 0.01%, Geo-Store only required 0.422 seconds on average to execute the range query, while Brodt’s and Strabon needed 1.916 seconds.
and 2.792 seconds, respectively. The advantage of Geo-Store over the other two systems, in terms of efficiency in evaluating range queries, can be explained as follows.

By utilizing the Hilbert curve based transformation, Geo-Store manages to maintain a roughly one-to-one mapping between Hilbert values and POIs. Therefore, in most cases, the spatial relation between two entities can be determined by comparing their respective Hilbert values. As a result, if a POI is identified as beyond the query window by the examination of its Hilbert value, it can be filtered out at an earlier stage, and there will be no need to check its exact latitude/longitude information on disk. On the contrary, in Brodt’s and Strabon (both employ the R-tree index), each Minimum Bounding Rectangle (MBR) contains numerous POIs (i.e., usually more than half of the fan-out value), resulting in a much larger amount of I/Os to retrieve the latitude/longitude values on disk in order to decide if a particular POI satisfies a spatial selection operation or not.

5.1.2 $k$ nearest neighbor query

Next we study the efficiency of all three systems in evaluating $k$ nearest neighbor queries. We varied the $k$ value from 1 to 10. Figure 4 (b) shows that when we elevated the $k$ value, all the systems required a longer response time to identify the $k$ closest neighbors. Nevertheless, as demonstrated in Figure 4 (b), the response time needed by Geo-Store was significantly reduced, compared to the other two systems. For example, with $k$ equal to 3, Geo-Store only required 0.325 seconds on average, while Brodt’s and Strabon needed 0.913 seconds and 1.642 seconds, respectively. Geo-Store demonstrates a much higher efficiency than the other two systems in processing $k$ nearest neighbor queries because Hilbert values employed in Geo-Store can provide a more precise location estimation of each POI than MBRs used in R-trees, which improves query evaluation performance.

5.2 Integration with Existing Triple Stores

As discussed in Section 3, one of the clear advantages that Geo-Store provides over Brodt’s solution is that no complicated spatial index, such as an R-tree, is needed to be maintained in Geo-Store. The indexing mechanism in Geo-Store is completely consistent to the six separate indices design [18], which is employed in most existing
triple stores [4, 1, 18, 14] for RDF query evaluation. On the other hand, in [3], a separate R-tree is responsible for indexing all the spatial IDs while all the IDs, spatial or non-spatial, are indexed by a $B^+$-tree. Therefore, by using our design, current RDF triple stores can be easily extended to support location-based services with limited cost on integration.

6 Conclusions

The increasing amount of RDF data containing location-based information calls for the development of systems which support effective evaluation of location-based services on RDF triple stores. In our Geo-Store project, we implement a system which is capable of querying heterogenous data sources and providing semantics-enabled location-based services with high efficiency. The Geo-Store system confers the following advantages. First, Geo-Store utilizes Spatially Aware Mapping to preserve spatial locality during encoding. Second, as our experiments demonstrate, Geo-Store allows for effective processing of range and $k$NN queries. Third, existing triple stores can be easily integrated with Geo-Store with limited integration cost. In the future, we plan to extend Geo-Store to support other novel RDF query languages specifically designed for spatial data management, such as GeoSPARQL and stSPARQL, by expanding the query parser module and related components. In addition, we will support more spatial query types, such as spatial join, in the next phase of this project.
Acknowledgements

This research has been funded in part by the National Science Foundation (NSF) Grants CNS-0831502 (CT) and CNS-0855251 (CRI).

References


Author Biography

Wei-Shinn Ku received his Ph.D. degree in Computer Science from the University of Southern California (USC) in 2007. He also obtained both the M.S. degree in Computer Science and the M.S. degree in Electrical Engineering from USC in 2003 and 2006, respectively. He is a graduate of National Taiwan Normal University. He is currently an assistant professor with the Department of Computer Science and Software Engineering at Auburn University. His research interests include spatial and temporal data management, mobile data management, geographic information systems, and location-based services. He has published more than 50 research papers in refereed international journals and conference proceedings. He is a member of the ACM and the IEEE. Dr. Ku can be reached at weishinn@auburn.edu.

Haiquan Chen is an assistant professor at Valdosta State University. He obtained his Ph.D. degree in Computer Science from Auburn University. His research interests lie in the field of database and information systems, including data mining and cleaning, social network analysis, and the semantic web. He is a member of the ACM and the ACM SIGMOD. Contact him at hachen@valdosta.edu.

Chih-Jye Wang received his B.S. in Computer Science in 2008 from Auburn University. He has 3 years of industrial experience as software engineer implementing enterprise and interoperable systems. He is currently a Ph.D. candidate in the Department of Computer Science and Software Engineering at Auburn University. His research interests include wireless peer-to-peer systems, network privacy, and semantic web data management.

Chuan-Ming Liu is an associate professor in the Department of Computer Science and Information Engineering, National Taipei University of Technology (NTUT), Taiwan. He received his Ph.D. in Computer Sciences from Purdue University in 2002 and B.S. and M.S. degrees both in Applied Mathematics from National Chung-Hsing University, Taiwan, in 1992 and 1994, respectively. Dr. Liu’s research interests include data management and data dissemination in various emerging computing environments, query processing in moving objects, location-based services, ad-hoc and sensor networks, parallel and distributed computation, and analysis and design of algorithms.