Towards GeoSpatial Semantic Data Management: Strengths, Weaknesses, and Challenges Ahead

Kostas Patroumpas[†,§] Giorgos Giannopoulos[§] Spiros Athanasiou[§]

[§]Institute for the Management of Information Systems
"Athena" Research Center, Hellas

[†]School of Electrical and Computer Engineering
National Technical University of Athens, Hellas

kpatro@dblab.ece.ntua.gr, {giann, spathan}@imis.athena-innovation.gr

ABSTRACT

An immense wealth of data is already accessible through the Semantic Web and an increasing part of it also has geospatial context or relevance. Although existing technology is mature enough to integrate a variety of information from heterogeneous sources into interlinked features, it still falls behind when it comes to representation and reasoning on spatial characteristics. It is only lately that several RDF stores have begun to accommodate geospatial entities and to enable some kind of processing on them. To address interoperability, the OGC has recently adopted the GeoSPARQL standard, which defines a vocabulary for representing geometric types in RDF and an extension to the SPARQL language for formulating queries. In this paper, we provide a comprehensive review of the current state-of-the-art in geospatially-enabled semantic data management. Apart from an insightful analysis of the available architectures in industry and academia, we conduct an evaluation study on prominent RDF stores with geospatial support. We also compare their performance and attested capabilities to renowned DBMSs widely used in geospatial applications. We introduce a methodology suitable to assess RDF stores for robustness against large geospatial datasets, and also for expressiveness on a variety of queries involving both spatial and thematic criteria. As our findings demonstrate, the potential for query optimization, advanced indexing schemes, and spatio-semantic extensions is significant. Towards this goal, we point out several challenging issues for joint research by the GIS and Semantic Web communities.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Management, Performance

Keywords

evaluation, GeoSPARQL, geospatial linked data, RDF, triple store.

SIGSPATIAL'14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA

ACM 978-1-4503-3131-9/14/11...$15.00
http://dx.doi.org/10.1145/2666310.2666410.

1. INTRODUCTION

In recent years, Semantic Web technologies have strengthened their position in data and knowledge management. Standards and protocols for organizing and querying semantic information in the Web, such as RDF [47], RDFS [48], OWL [40], and SPARQL [50], are commonplace in related academic research. RDF stores[1] like [4, 9, 30, 36, 39, 49, 51] have become robust enough to support billions of triples, and can offer storage and querying functionalities similar to those in traditional relational database systems (DBMS) [19, 26, 28, 39, 44]. Corporate vendors have begun adopting semantic technologies to organize, expose, exchange and retrieve such data for various applications, such as online shopping facilities, personalized content delivery, social networking, etc.

Yet, it is remarkable how much geographic context exists in information searched on the Web, either explicitly or implicitly. For instance, a portal or a mobile application may provide to users not only clips, photographs or reviews for a film, but also its screening times in nearby cinemas with their locations pinpointed on a city map. A professional may wish to combine publicly available statistics (e.g., demographics, epidemics, etc.) or other analytics (e.g., product sales, road traffic) with open, crowdsourced, up-to-date geographic maps like OpenStreetMap (OSM) [38]. And if a visitor looks for a suitable hotel, apart from pricing, facilities, recent ratings or customer opinions on its quality of service, she may be also interested in its proximity to public transport or city attractions. But almost certainly, not all this information is available on a single site, so its pieces must be retrieved from multiple disparate sources and carefully interlinked to provide the answers, according to the Linked Data paradigm [5].

For over three decades, Geographic Information Systems (GIS) have been successfully used for editing large geographic datasets, map visualization, and spatial analysis in diverse domains (such as geology, hydrology, city planning, real estate, marketing, etc.). Typical GIS software for desktop computation or mapping services must cope with multiple, heterogeneous data sources (e.g., proprietary datasets, coming in various formats). Particular effort is also required to associate non-spatial data (e.g., census, multi-lingual content, etc.) to varying geographic layers (e.g., administrative areas, points of interest, transportation networks). Such operations usually require specialized skills and technical know-how that not all software developers have.

Core database technology offers consistency and integrity for scalable datasets, as well as powerful means for indexing and query optimization. But, even though DBMSs can nowadays host large geospatial datasets, they do not inherently support qualitative anal-

[1]A system for storing and managing RDF data is usually called *RDF store* or *triple store*, but terms such as *semantic repository*, *semantic store*, or *RDF database* are also used in the literature.

ysis, e.g., deduce directional relations between geometric shapes (such as that Italy is to the west of Greece, although there is no land border between them). Thanks to their ability for inferencing and linkage of data, triple stores are being increasingly attractive for advanced searching and reasoning [23]. Geospatial information often includes complex type hierarchies, which cannot be fully expressed or exploited in typical GIS platforms. For instance, a river can be a waterway, a transportation route, and an administrative boundary. Thanks to schema versatility, RDF stores are better equipped to cope with complex requests, such as multiple joins across entities, queries with variable properties, or ontological inference on datasets. Interlinking data sources on the Web is of major importance as well. For example, points of interest combined with hotel pricing, user recommendations and driving directions could offer much more refined and personalized travel planning services.

Therefore, a synergy of the mature geospatial database technology with the knowledge and reasoning capabilities of RDF stores could be extremely promising, both for the Semantic Web and GIS domains. In particular, we think that efficient integration of geospatial data representations and processing methodologies into the Semantic Web is a priority. In order to lower the burden for application development and facilitate creation of value added services, we deem that two main challenges should be addressed.

First, proper standards and vocabularies must be defined to describe geospatial information according to RDF(S) and SPARQL protocols, also conforming to the principles of established geographic standardization, such as OGC standards [33, 34], GML [31, 32], INSPIRE [8], etc. There have been several incomplete attempts from the Semantic Web community [2, 11, 13, 14, 15, 18, 42, 54] towards a geospatial RDF standard. Recently, *GeoSPARQL* [35] has been endorsed by the Open Geospatial Consortium (OGC), with the aim of standardizing representation, querying, and reasoning for geospatial RDF data. GeoSPARQL provides various conformance classes concerning its implementation of advanced reasoning capabilities (e.g. quantitative reasoning), as well as several sets of terminology for topological relationships between geometries. Since GeoSPARQL adheres to existing OGC standards for spatial data types, operators, and functions, it can facilitate exchange with geographic formats and swift migration to/from spatial databases. Although few triple stores are currently compliant with GeoSPARQL, this standardization is a major step towards interoperability among geospatial RDF data infrastructures.

In contrast, we observe less progress on another key challenge, namely development of technologies for efficient storage, robust indexing, and native processing of geospatial semantic data. Many proprietary or open-source RDF stores can efficiently handle large volumes of RDF data, but relatively few of them actually support triples with geospatial features. Even fewer can store all types of geometric objects (i.e., not only points, but also polygons, lines, etc.), or fully conform to the GeoSPARQL specification. A related open issue concerns the efficiency of currently available RDF stores with geospatial capabilities; regrettably, no such store is yet comparable to established spatial DBMSs in terms of performance.

In this paper, we present the actual state-of-the-art in geospatially-enabled semantic data management. We review several RDF stores with geospatial support, offering insight into their architecture, indexing capabilities, and deployment options. Our approach caters to the emerging need of accurately and objectively evaluating such platforms. We not only examine the degree of geospatial support they provide, but we also carry out qualitative and quantitative comparisons. In an attempt to capture real-world relevance and user requirements in geographic applications, we introduce a methodology that aims at inspecting functionality of RDF stores through rich and diverse geospatial semantics in the query workload. Farther beyond typical topological operations (e.g., objects within a given spatial range, intersecting entities, etc.), these queries cover advanced cases involving spatial aggregation, joins, nearest neighbor search, or negation.

To the best of our knowledge, this is the first attempt to conduct a comprehensive evaluation of all currently available, fully-fledged, geospatially-enabled RDF stores. We examine both *commercial platforms* (AllegroGraph [9], OWLIM [30], Virtuoso [36], and another proprietary triple store), as well as *research prototypes* (OpenSahara/uSeekM [37], Parliament [46], and Strabon [24]). We also compare them to established *spatial database systems*: the open source PostGIS [43] and another commercially available DBMS. Performance results reveal that RDF stores are considerably less efficient than DBMSs in data loading and query execution. Our findings highlight shortcomings in expressiveness for certain queries, and sometimes indicate insufficient support for geometries other than points. Most importantly, our evaluation draws attention to challenging issues for research in geospatial RDF data management, principally regarding storage, indexing, scalability, and query optimization, as well as potential specialized extensions.

The remainder of this paper proceeds as follows. Section 2 surveys basic concepts and approaches regarding geospatial RDF data. In Section 3, we examine features and spatial capabilities of prominent RDF stores. In Section 4, we develop a methodology for evaluating geospatially-enabled RDF stores. Section 5 reports comprehensive performance results against OSM datasets. In Section 6, we propose a tentative research agenda towards efficient management of geospatial RDF data. Section 7 concludes the paper.

2. GEOSPATIAL FEATURES IN RDF

The *Resource Description Framework* (RDF) [47] is a language for representing information about entities (i.e., resources) on the Web as *triples* with ⟨*Subject*, *Predicate*, *Object*⟩, e.g., ⟨Homer, wrote, Odyssey⟩. Resources are uniquely identified via Internationalized Resource Identifiers (IRIs), and a collection of triple statements forms a labeled, directed graph. Enhancements include the *RDF Schema* [48], which offers a *vocabulary* for managing resources and their relationships via a set of reserved words, as well as the *Web Ontology Language* (OWL) [40], which can be used to define and instantiate *Web ontologies*. Recommended by the W3C, the *SPARQL Protocol and RDF Query Language* resembles SQL with its main blocks `SELECT-WHERE`, but query evaluation is actually based on graph pattern matching. Its latest version SPARQL 1.1 [50] also supports aggregates, nested subqueries, negation, and data management operations (i.e., to insert or delete triples).

However, the particular demands for geospatial data storage and operations are not handled at all by these generic protocols. Besides full-fledged triple stores and vendor platforms (to be reviewed in Section 3), several attempts have been made towards encoding simple geometry data in RDF. The *W3C Basic Geo Vocabulary* [2] was one of the earliest works and enabled representation of point geometries with latitude/longitude coordinates in WGS84 reference system. *GeoRSS* [15] provided support for more geometric objects (such as lines, rectangles, or polygons) and GML application profiles. In another approach, *GeoOWL* [13] was developed as a more flexible model for geospatial concepts with an ontology that matches the existing GeoRSS vocabulary. The *NeoGeo Geometry Ontology* [29] was proposed for the topological modelling of various geometric shapes in RDF. According to this ontology, each coordinate must be retained as a resource, so polygons and lines are represented with an RDF collection of points. Hence, this model increases verbosity of the data without significant benefits in terms

of expressiveness and query processing with SPARQL. *GeoRDF* [14] was intended as an RDF-compatible profile for geographic points, lines, and polygons. Its two vocabularies (RDFGeom, RDFGeom2d) provide a framework that is extensible via subclassing to all kinds of geometric data in Cartesian coordinates, although the class hierarchy is currently only sparsely populated. *GeoJSON* [11] is a geospatial data interchange format based on JavaScript Object Notation (JSON) and can encode a variety of shapes (points, polygons, etc.). Overall, the aforementioned schemes mainly supported coordinates with Cartesian or WSGS84 georeferences (thus leading to gross errors in other reference systems), they were often limited to point geometries, and offered insufficient capabilities for spatial operations in real-world GIS applications.

Besides, querying geospatial RDF data also attracts a growing research interest. An extension to SPARQL, termed *SPARQL-ST* [42] suggested a modified syntax for specifying spatial queries against data modeled in a GeoRSS-like ontology, also including temporal and thematic (i.e., non-spatial) properties. But this dialect deviates from the standard SPARQL, hence the exposed data cannot be accessed from third-party systems. Adding topological predicates to SPARQL was briefly examined in [54] with an ontology that takes advantage of OGC Simple Features [33]. However, relations have to be specifically encoded in RDF, without any support for multiple Coordinate Reference Systems (CRS). The model in [18] nicely abstracts spatial knowledge from its underlying representation at several hierarchical levels, but mappings must be defined for each dataset at the instantiation level.

Regarding query optimization schemes, the RDF store prototype in [7] models spatial features as complex geometry literals and augments SPARQL with spatial range filter functions based on OGC Simple Features relations [33]. Geo-Store [52] focuses on range and k-nearest neighbor search and makes use of a Hilbert curve encoding of IRIs for accelerating spatial predicate evaluation. Also based on Hilbert encoding, but coupled with a hierarchical space decomposition, the scheme proposed in [25] handles complex geometries and offers significant cost savings when evaluating range and spatial join queries with thematic criteria. The SS-tree hybrid index [53] combines a bitmap encoding of entities in the graph along with the MBRs of geometries, so at query time it allows pruning of candidate answers with a top-down search algorithm.

The recent (2012) OGC standard on *GeoSPARQL* [35] suggests a concrete ontology for representing features and geometries in RDF, as well as an extension to SPARQL for querying such data. A core component defines top-level RDFS/OWL classes for representing spatial objects with georeferences at well-known CRS. Adhering to OGC standards [33, 34], GeoSPARQL offers two ways to represent geometry literals and their associated type hierarchies, namely WKT and GML. Properties `geo:asWKT` and `geo:asGML` link a feature (e.g., a bus stop, a road, or a parcel) to its geometry serializations encoded as `geo:wktLiteral` or `geo:gmlLiteral`. A vocabulary defines RDF properties for asserting and querying topological relations between spatial objects using SPARQL extension functions (`geof:sfIntersects`, `geof:sfContains`, `geof:sfWithin`, `geof:Crosses`, etc.) according to OGC Simple Features [34]. Spatial analysis is also possible using functions like `geof:union`, `geof:buffer`, `geof:distance`, etc.

Overall, GeoSPARQL is designed to accommodate systems for *qualitative* spatial reasoning (e.g., "is there a statue of Lord Byron inside Hyde Park?") and systems based on *quantitative* spatial computations (e.g., measuring distances). Using a common set of topological relations, GeoSPARQL allows conclusions from quantitative applications to be used by qualitative systems, and offers a single language for both types of reasoning. Thus, spatial ontologies may be exchanged, combined, indexed and queried along with other proprietary ontologies from data providers. With such standardization, vendors and users can achieve uniform, transparent, platform-independent access to geospatial RDF data with a rich collection of query operators. Example queries in Section 4 provide more intuition on how geometries and features are related, as well as how spatial functions and topological operators can be applied[2].

3. SPATIAL SUPPORT IN RDF STORES

Support for spatial data in RDF stores is still a work-in-progress. Datasets have been published using the W3C vocabularies (like [1]) and some RDF stores (e.g., [30]) support data represented by these ontologies in WGS84 datum. So, in order to be valid, data in any other CRS must be reprojected, which may incur some geometric distortion or inaccuracy in measurements. But most of these proposals have not exceeded an incubator state and the respective ontologies never became official W3C recommendations. Several vendors support spatial data, but not all vendors follow the same representation of data or share identical specifications for spatial queries. Some triple stores use W3C ontologies, some employ OGC specifications, while others have invented their own. Table 1 provides a concise comparison of advertised features for widely used triple stores and research prototypes with geospatial support. Next, we review these platforms in more detail.

3.1 RDF Stores with Geospatial Capabilities

AllegroGraph [9] is a commercial graph database offering a wide range of data types (including geospatial and temporal) as native data structures. It can perform reasoning efficiently, combined with its indexing and range query mechanisms. AllegroGraph stores *quints* of five slots in its *SPOGI* index: a *S*ubject, a *P*redicate, an *O*bject, a named *G*raph, and an internally assigned unique *I*dentifier. The *I* slot can be referred to by other quints directly; this drastically reduces storage size and query response time.

For geospatial features, AllegroGraph uses a custom data type and actually performs a mapping into a *"strip"* of space in the 2-dimensional index that contains all geometries. This novel indexing scheme divides the Y range into strips of a known width, chosen according to the expected size of searched regions, which is a subtle limitation. Apart from points, there is also support for simple polygonal regions. A modified SPARQL syntax offers a `GEO` operator, which can take as arguments a Cartesian or spherical point and a radius, a bounding box, or a polygon, and it is used to specify the spatial conditions of a query. Implementation of the geospatial component for SPARQL is still ongoing. Currently, all spatial operations are defined in terms of points, so there is no support for OGC types and operations [33], or for GeoSPARQL [35].

OWLIM is a family of semantic repositories, most recently (as of August 2014) rebranded as GraphDB [30]. Free-of-charge OWLIM-*Lite* is designed for medium data volumes, i.e., below 100 million triples. Commercial OWLIM-*SE* (*Standard Edition*) offers advanced features for managing huge volumes of data and multi-user query performance. OWLIM-*Enterprise* is a replication cluster infrastructure for resilience and parallel query answering.

OWLIM only supports 2-dimensional points that use the W3C Basic Geo Vocabulary [2] with coordinates in WGS84 only. A

[2]Examples make use of the following RDF namespace prefixes:

```
rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
geo:  <http://www.opengis.net/ont/geosparql#>
geof:  <http://www.opengis.net/def/function/geosparql/>
sf:  <http://www.opengis.net/ont/sf#>
uom:  <http://www.opengis.net/def/uom/OGC/1.0/>
```

Table 1: Qualitative comparison of RDF stores according to their support for geospatial features and processing.

Platform	*Supported geometry types*	*Geometry literals*	*Data storage*	*Spatial index*	*Query language*
AllegroGraph (free) ver. 4.10	points and simple polygons	Custom geolocation format	file repository for all statements and indices	Custom "strips"	A few custom topological operators as SPARQL filters; *not compliant to GeoSPARQL*
OWLIM-SE ver. 5.4	only points for storage; ad-hoc polygons in queries	W3C Basic WGS84 GeoVocabulary	file repository for all statements and indices	R-tree	A few custom topological operators as SPARQL filters; *not compliant to GeoSPARQL*
Parliament ver. 2.7.6	most OGC geometries incl. points, lines, polygons	WKT	file repository for all statements and indices	R-tree	GeoSPARQL
Strabon ver. 3.2.9	most OGC geometries incl. points, lines, polygons	WKT GML	hosted inside PostgreSQL + PostGIS	GiST	Close, but not fully compliant to GeoSPARQL (differing prefixes and spatial functions)
uSeekM ver. 1.2.0-a5	most OGC geometries incl. points, lines, polygons	WKT	hosted inside PostgreSQL + PostGIS	GiST	GeoSPARQL
Virtuoso Universal Server ver. 7.1 (ColumnStore)	most geometries for storage; limitations in processing	Custom namespace in WGS84	internal database, also accessible via SQL	R-tree	A few custom topological operators as SPARQL filters; *not compliant to GeoSPARQL*

standard R-tree [16] must be built before using any geospatial operations, so that related graph patterns are treated differently by the query engine. The spatial index allows SPARQL queries to quickly find points *within* ad-hoc rectangles, polygons and circles and to compute *distances* (in kilometers) between points. However, none of these spatial data types and query constructs is compliant with GeoSPARQL [35]. Currently, neither is there support for other geometric objects (e.g., lines, stored polygons, etc.), nor for any other topological or spatial functions.

Parliament [46] runs on a single machine and incorporates a number of open-source third-party packages. It is compatible with the RDF, RDFS, OWL, SPARQL, and GeoSPARQL standards. Its storage scheme interweaves the data with a unique index, and consists of a resource table, a statement table, and a dictionary for resource mappings [22]. Thanks to that, Parliament can answer queries efficiently by reordering execution so that the most selective parts of the query are executed first. This research prototype includes a rule engine, but not a query processor, so it is typically deployed with a third-party processor (e.g., Sesame [49]).

With the exception of query rewriting rules, the query engine supports GeoSPARQL and enables searching over indexed geometries using standard R-trees [16]. Queries are split into multiple parts during evaluation, allowing for an optimized query plan leveraging their spatial and thematic components, as exemplified in [3].

Strabon is an academic prototype [24] developed specifically for spatiotemporal RDF data. Strabon proposes *stSPARQL*, a spatially-enabled dialect of SPARQL against data represented in *stRDF*, an extension of RDF that can also capture changes over time. Strabon is built by extending the evaluator and optimizer of Sesame [49] for managing thematic, spatial and temporal data in a backend DBMS. Sesame's components were extended to support geometry literals expressed in stRDF. Thanks to its native spatial indexing with GiST [17], PostGIS [43] is used for storage of stRDF data in PostgreSQL [44] and evaluation of stSPARQL queries.

Developed independently from GeoSPARQL, the latest versions of stRDF and stSPARQL [24] follow the OGC standards for WKT and GML literals. stSPARQL has certain limitations, as it does not support binary topological relations to be used as RDF properties. However, stSPARQL extends SPARQL 1.1 [50] and overtakes GeoSPARQL by offering spatial aggregate functions and triple update commands. Particular optimization techniques allow spatial operations to take advantage of PostGIS functionality (instead of relying on external libraries), whereas spatial joins are also efficiently handled by the underlying PostgreSQL/PostGIS optimizer.

uSeekM [37] is an add-on library designed for triple stores that can be exposed with the Sesame Java interface [49]. Most of its functionality is provided through wrappers; one of them is called *IndexingSail* and extends an RDF database with indexing and querying capabilities, adding geospatial support, full-text search, etc. A *PostgisIndexer* builds a GiST spatial index [17], so it requires a PostgreSQL database with enabled PostGIS extension [43].

Thanks to these two indexing schemes, uSeekM supports all OGC geometry types specified in [33] and most operations in the GeoSPARQL standard [35]. WKT literals can be used to describe 2- or 3-dimensional geometries (points, lines, polygons, etc.), always georeferenced in WGS84; there are no plans to implement support for GML serializations. GeoSPARQL syntax is slightly relaxed, e.g., instead of `geo:asWKT`, any predicate name can be used to attach a given geometry to a resource or subject. Spatial relationships and methods are those inherently supported by PostGIS, suitably implemented as SPARQL filters and functions.

Virtuoso Universal Server [36] is a middleware and database engine. Concerning RDF data, it actually implements a *quad GSPO*, composed of *G*raph, *S*ubject, *P*redicate, and *O*bject. All quads are stored in one table, where *GSP* values are IRIs, whereas *O* is any SQL serializable object. SPARQL is embedded and translated into SQL for querying RDF data stored in the database. Lately, Virtuoso introduced a compressed column store representation for RDF.

Virtuoso can handle 2-dimensional points expressed with WGS84 coordinates as in [2]. Storage of other geometric shapes (e.g., lines, polygons) has been included recently, but their use in query processing is still under development. A geometry is defined via a special RDF typed literal with custom type `virtrdf:Geometry` that gets automatically indexed in a native R-tree [16]. Topological operations are limited to some built-in predicates (`ST_contains`, `ST_within`, `ST_intersects`), which can check whether two geometries are related. A handful of geometric functions are also available (e.g. `ST_distance`, `ST_x`, `ST_y`, `ST_AsText`). Overall, spatial types and functions are limited in Virtuoso and currently not compliant with GeoSPARQL [35].

Other commercial triple stores. Support for geospatial features varies in other commercial RDF platforms. Oracle Spatial and Graph [39] in its latest release (12c) provides integrated support for storing, loading and rich DML operations on RDF/OWL models along with most geometric objects and operators, although not fully compatible with GeoSPARQL. IBM offers the NoSQL Graph Store [20], which provides an optimized way to retain graph triples inside a DB2 database, but there is no indication about support for geospatial features. Spatial indexing is also missing from Bigdata store [4], so spatial queries cannot be executed. The most recent release of Stardog [51] does not include geospatial features, but GeoSPARQL support has been announced for a future release.

3.2 Qualitative Comparison

As listed in Table 1, not all geometric types are supported in

Table 2: Data contents of OSM layers (originally in ESRI shapefile format).

OSM Layer	*SPARQL prefix*	*Geometry type*	*Contents*	*File size (MBytes)*	*Number of features*
Points	`poi`	Point	Points of general interest (*bus stops, pubs, hotels, traffic signals, places of worship, etc.*)	74.3	590,390
Roads	`roads`	LineString (spaghetti lines)	Road network links (classified as *motorway, primary, secondary, tertiary, residential etc.*)	706	2,601,040
Natural	`zones`	Polygon	Surfaces like *parks, forests, waterbodies (i.e., lakes, riverbanks), etc.*	152	264,570

each RDF store. Some of the examined stores (OWLIM-SE, Virtuoso) are seriously limited, as they either allow storage of points only or offer a small collection of spatial operators. AllegroGraph may store simple polygons, but very few topological predicates and geometric functions can be applied on them. Admittedly, spatial analysis is severely constrained when no other vector data (such as lines or polygons) is allowed in the triple store. Other systems (Parliament, Strabon, uSeekM) allow most OGC geometries with a wealth of spatial operations and are very close to the functionality offered by acclaimed DBMSs [19, 26, 28, 39, 43].

Geometries are typically represented as WKT literals or similar variants. This proves that this OGC serialization is capable to capture most shapes (including georeferencing) in a uniform and interoperable manner. Varying representations in platforms such as AllegroGraph, OWLIM-SE, or Virtuoso, diverge from WKT and cannot be considered a viable solution for complex geometries apart from points. AllegroGraph, in particular, opts for a custom geometric format, which requires some preprocessing of original geometries and reconstruction of polygons from their constituent vertices. Overall, GeoSPARQL compliance [35] for many triple stores is still an objective. Interestingly, even those implementing GeoSPARQL (Parliament, uSeekM) have a few differences in namespaces for WKT literals and variations in spatial functions. Other systems do not follow GeoSPARQL specifications, although this would mostly require renaming functions and namespaces for supported geometries (e.g., in Virtuoso or Strabon). Hopefully, discrepancies will be remedied in forthcoming releases, but even such minimal deviations clearly demonstrate the necessity for standardization.

Regarding spatial indexing, it seems that R-trees [16] are preferred, since they are an established, provably efficient structure, and widely used in geospatial DBMSs as well. Two triple stores (uSeekM, Strabon) make use of PostGIS infrastructure [43], so they build a GiST index on geometries [17], basically equivalent to a more efficient 2-dimensional R-tree. The only exception to the R-tree family of indices amongst the examined platforms is AllegroGraph. For successful "strip"-based indexing, one must have a clue for the expected spatial range of user requests in order to determine a suitable "strip" size. This could add considerable overhead for data maintenance in AllegroGraph, but it also makes spatial predicates less intuitive and more difficult to express in queries.

4. EVALUATION METHODOLOGY

Several benchmarks (e.g., [6, 27]) are currently utilized to assess performance of RDF stores, but they are absolutely devoid of any kind of spatial processing. In contrast, spatial benchmarks like Jackpine [45] are mostly geared towards DBMSs and ignore the case of knowledge inferencing from multiple, schema-agnostic, interlinked RDF data sources. A geospatial RDF benchmark called *Geographica* was recently developed in [10], inspired by and closely adhering to Jackpine. Actually, it is a straightforward porting of Jackpine query workloads into a RDF context with minor adjustments on its original "micro" and "macro" options (concerning primitive spatial operations and real-world scenarios, respectively). Geographica has been applied against three selected RDF stores only (Strabon, uSeekM, Parliament) that provide support for at least a subset of GeoSPARQL. Further, it uses about 125,000 features in its real-world dataset and less than 300,000 synthetic features, a data volume so limited that may not reveal subtle issues in spatial query processing. In our tests, we employ an order of magnitude more data and geometric features, as well as a more representative workload with a broad variety of spatial query types.

In this section, we introduce a methodology for evaluating RDF stores with geospatial support. We start with a description of the datasets used in our experiments. Then, we present the types of spatial queries, which can be expressed either in GeoSPARQL (or SPARQL, depending on which idiom each triple store supports) or in SQL (against a geospatially-enabled DBMS).

4.1 Datasets

OpenStreetMap [38] data for Great Britain covering England, Scotland, and Wales were initially downloaded in ESRI shapefile format. Of the available geographic layers, only those concerning points of interest (*points*), the entire road network (*roads*), and natural parks and waterbodies (*natural*) were actually utilized, retaining all original OSM features as detailed in Table 2.

These three OSM layers were chosen as representatives for basic geometry types (points, polylines, polygons) and were deemed most meaningful for queries involving multiple geospatial layers. They also contained many more features compared to other similar layers (e.g., the number of railway links is about 3% of the road features). All data was used "as is", without editing the original WGS84 geometries (attribute *shape*). We made use of the open source ETL utility *TripleGeo* [41] to convert geometries into serialized literals recognisable by each platform. In addition, three thematic attributes were extracted: the unique OSM identifier *osm_id* served as a reference label for each feature, whereas not null values for *name* and *type* were turned into string literals.

4.2 Query Workload

As existing RDF benchmarks ignore spatial processing, we focused on creating rich and diverse *geospatial semantics* in our query workload, showcasing real world needs and emphasizing on checks for GeoSPARQL compliance. Despite its importance in knowledge systems, reasoning has not been examined in our evaluation, as the focus was predominantly on assessment of geospatial support. So, tested queries required no inferencing at all.

According to [21] there are four primary types of spatial queries that should be covered: location queries, range queries, spatial joins, and nearest neighbor search. In addition to these types, we also tested queries that make sense in geospatial analysis, i.e., computing derived geometries, employing both thematic and spatial criteria, calculating aggregates, or involving negation. As listed in Table 3, we have prescribed the following categories:

i. **Location queries** simply request the whereabouts of a given feature (i.e., like a map click). So, they check whether the examined system can provide spatial retrieval efficiently.

ii. **Range search** within an area of interest is specified with a list of coordinates, e.g., a rectangle, a circle, or a polygon.

Table 3: Operations and queries tested in the evaluation study.

Symbol	Type	Operation / Query semantics
BL	*Bulk Loading*	Time to insert all data into a RDF store or a DBMS.
SI	*Spatial Indexing*	Time to build a spatial index (usually an R-tree) on all stored geometries.
BL+SI	*Loading + Indexing*	Total cost for Bulk Loading and Spatial Indexing (when a platform performs them together).
L1	*Location query*	Find the geographic location of entity *"Westminster Abbey"*.
L2	*Location query*	Which entity is at given coordinates $(longitude, latitude)$?
R1	*Range search*	Retrieve all points within a given rectangle of area 1250 km^2 (about 0.5% of the total area).
R2	*Range search*	Retrieve all roads intersecting a given rectangle of area 31400 km^2 (about 13% of the total area).
SJ1	*Spatial join*	Find all road segments traversing any forest.
SJ2	*Spatial join*	Find all distinct pairs of points within a distance less than 50 meters from each other.
SJ2a	*Spatial join*	Variant of **SJ2**, with a `LIMIT 100` or `LIMIT 1000` modifier to report only partial results.
kNN1	*k-Nearest Neighbors*	Find the 3 closest points of interest to a given location $(longitude, latitude)$.
kNN1a	*k-Nearest Neighbors*	Variant of **kNN1**, with an additional distance filter to eliminate irrelevant candidates.
kNN2	*k-Nearest Neighbors*	Find the 3 points of interest closest to a given road segment.
kNN2a	*k-Nearest Neighbors*	Variant of **kNN2**, involving point geometries only.
G1	*Geoprocessing*	Return the portions of any motorway overlapping with a given zone (e.g., a flooded area).
G2	*Geoprocessing*	Take the geometric union of all segments of a given road (e.g., *"Barnaby Road"*).
G3	*Geoprocessing*	Compute a buffer around a given road (e.g., *"Barnaby Road"*).
TS1	*Thematic+Spatial Filter*	Find all pubs within a square of side 20 km (or a *radius* of 20 km, if polygons are not supported).
TS2	*Thematic+Spatial Filter*	Find all pubs within a square of side 200 km (*or an equivalent circle*) centered at *Westminster*.
A1	*Aggregation*	Count points of interest located within each forest area.
N1	*Negation query*	Locate pubs farther than 5 km from any bus stop (i.e., difficult to reach by public transport).

R2:
```
SELECT ?f ?fName ?fWKT
WHERE {?f geo:hasGeometry ?fGeom .
       ?fGeom geo:asWKT ?fWKT .
       ?f roads:name ?fName .
       FILTER (geof:sfIntersects(?fWKT,
       "POLYGON((-2.8164 51.6576, -0.277 51.6576,
       -0.277 54.1969, -2.8164 54.1969,
       -2.8164 51.6576))"^^geo:wktLiteral))};
```

TS1:
```
SELECT ?f ?fName ?fGeom
WHERE {?f geo:hasGeometry ?fGeom .
       ?fGeom geo:asWKT ?fWKT .
       ?f poi:name ?fName .
       ?f rdf:type poi:pub .
       FILTER (geof:sfWithin(?fWKT, "POLYGON((-1.5244 51.4924,
       -1.5244 51.9964, -1.0087 51.9964, -1.0087 51.4924,
       -1.5244 51.4924))"^^geo:wktLiteral))};
```

SJ1:
```
SELECT ?r ?rName ?z ?zName
 WHERE {?r rdf:type roads:roads .
        ?r geo:hasGeometry ?rGeom .
        ?rGeom geo:asWKT ?rWKT .
        ?r roads:name ?rName .
        ?z rdf:type zones:natural .
        ?z geo:hasGeometry ?zGeom .
        ?zGeom geo:asWKT ?zWKT .
        ?z zones:name ?zName .
        FILTER (geof:sfCrosses(?rWKT, ?zWKT))};
```

N1:
```
SELECT ?pName
WHERE {?p rdf:type poi:pub .
       ?p geo:Geometry ?pGeom .
       ?pGeom geo:asWKT ?pWKT .
       ?p poi:name ?pName .
       FILTER NOT EXISTS {
        ?f rdf:type poi:bus_stop .
        ?f geo:Geometry ?fGeom .
        ?fGeom geo:asWKT ?fWKT .
        FILTER (geof:distance(?pWKT, ?fWKT, uom:metre)<5000)}};
```

Figure 1: GeoSPARQL expressions for four indicative queries in Table 3. Important geospatial semantics are shown in bold.

Different areas can be tested (e.g., covering different percentages of the map extent), in order to assess advantages of spatial indexing over varying selectivities. This search would certainly benefit from a spatial index, as many locations may be found inside the given region. The area extent is important, as its size could incur significant execution cost.

iii. **Spatial join** is an expensive operation, as it requires interaction between two datasets. This is a tough task for RDF stores, since joins may be based on diverse geospatial operations (`intersects`, `crosses`, `distance`, etc.), the involved features can be numerous, and the query planner may need to handle thematic criteria as well.

iv. **Nearest-neighbor queries.** For a given location or spatial feature, a k-NN search returns its k closest geometries, typically under a Euclidean distance. If not natively supported, this can be simulated by a `LIMIT k` modifier to return k geometry triples having the smallest distances from the query point. But the cost of an indirect top-k search may be considerable compared to a native k-NN operator (as in [39]).

v. **Geoprocessing** returns new, *derived* geometries, as opposed to other queries that simply retrieve existing, *stored* entities. Hence, any intersections or unions over geometric shapes must be calculated on-the-fly and then returned as answers. This is specifically intended to verify support for spatial analysis according to OGC Simple Features [34].

vi. **Queries combining thematic and spatial criteria** include meaningful filters on thematic (i.e., non-spatial) attributes as well as spatial predicates. They are intended to examine whether the query planner can benefit from spatial and thematic selectivities. As argued in [21], if a spatial condition is selective enough, then it should be prioritized by the planner; otherwise, the spatial operation should be executed after any thematic matching. Two indicative queries were specified, each involving different sizes for spatial ranges and diverse distribution of features for the given thematic filter.

vii. **Aggregation queries** *with spatial predicates*. For each distinct geometry (e.g., region), functions like `avg` or `count` can be applied against related features (e.g., points therein), which usually involves a spatial join (e.g., on containment).

viii. **Negation queries** involve a `NOT EXISTS` clause. Typically, negation is a costly operation and here it is also coupled with a spatial predicate, which makes its optimization and evaluation even more difficult. Our example query involves different types of points, hence additional thematic

properties must be examined apart from the spatial criterion.

The aforementioned workload of SPARQL queries was tested in seven triple stores. For comparison, equivalent SQL statements were submitted against the same datasets in two DBMSs. Due to lack of space, in Figure 1 we provide GeoSPARQL statements for some indicative queries only. As each system follows its own conventions regarding geospatial management, variant query expressions were required. For instance, queries SJ1 and kNN2 cannot run in a store that supports points only. Extensive remarks on queries, their variants and execution details can be found in [12].

5. EVALUATION STUDY

5.1 Experimental Setup

Apart from the systems reviewed in Section 3 (their versions as in Table 1), we also tested a commercially available RDF store with native support for geospatial features, hereafter nicknamed *'Triple-Store X'*. For comparison, we conducted similar tests on PostGIS 2.1 [43] for PostgreSQL 9.3, as well as in a commercially available geospatial DBMS (hereafter called *'DBMS Y'*). These tests establish a baseline for assessing performance and robustness of current approaches to geospatial data management in RDF stores. Our evaluation study demonstrates their strengths and weaknesses regarding storage, indexing, scalability and expressiveness, not only to each other, but also to mature geospatial support in DBMSs.

For the experiments, we set up a XEN hypervisor on a server with Intel Core i7-3820 CPU at 3.60GHz and 10240KB cache, hosting a group of Virtual Machines (VM). Each platform (RDF store or DBMS) was installed on a separate VM with 8GB RAM, 2GB swap space, 4 CPU cores and 40GB disk space. During each experiment, only the system under evaluation was active[3].

Although most platforms provided a web interface and SPARQL endpoints, data loading and query executions were performed locally in order to avoid network delays in performance measurements. For some stores (AllegroGraph, OWLIM-SE, Parliament, uSeekM), bootstrap code was written in Java for direct interaction (bulk loading, querying) without resorting to the web interface. In all other systems, their native command-line interface was used. Query times do not include the cost of reporting triples or rows.

Measurements were performed in cold caches and include:

- *Amount of inserted triples*, including all supported geometries. Due to triplification, this differs substantially from the number of corresponding records in DBMSs.
- *Data insertion time using bulk loading (BL)*. The total time includes insertion of both geometry and non-spatial triples (i.e., those concerning name or type literals).
- *Spatial indexing (SI) cost*. In some triple stores (e.g., uSeekM, Virtuoso), the spatial index is adjusted upon every new insertion. In this latter case, only the combined time (*BL+SI*) can be reported, which covers data insertion and spatial indexing.
- *Response time* for each query involving spatial operations; no other requests were active at the same time. As discussed in Section 4.2, the query workload offered eight different categories with a few typical examples per category. Occasionally, due to platform limitations regarding geospatial support, variant query statements were expressed. Due to such variations, some queries are not directly comparable in semantics (and consequently in performance) among all platforms.

[3]For repeatability, images of preconfigured VMs, as well as data and queries are all publicly available at `http://bit.ly/1pPI5aI`.

5.2 Performance Results

Several aspects in the data or the user requests can heavily influence the expected performance of platforms. Concerning the data, these factors include shape complexity (e.g., number of vertices in polygons), as well as the amount of geometries that require indexing. With respect to queries, primary characteristics involve spatial semantics, selectivity, as well as the presence of expensive operators like join, geoprocessing, or negation. Performance results from our evaluation involving storage operations (*BL*: bulk loading, *SI*: spatial indexing) and the query workload are shown in Table 4.

First, note the differing number of triples (or records) imported in each system. Some platforms (OWLIM-SE, AllegroGraph) support points only, so they retained almost an order of magnitude less triples. Besides, OWLIM-SE employs two triples per point (for longitude and latitude); a single geometry triple is used in all other systems. Occasionally, a small number of geometries were discarded during loading due to inconsistencies w.r.t. supported data types (e.g., polygons with holes). Still, millions of triples were inserted in each store; about 20% of them involved geometry literals.

It must be pointed out that *spatial indexing* cost varies widely among platforms. Of course, OWLIM-SE builds an R-tree over 590,390 points and this is a relatively easy task, hence the resulting cost (less than a minute) is acceptable. AllegroGraph takes much longer to import points, as it incrementally builds its custom index of "strips". AllegroGraph also supports simple polygons, but indirectly: point literals must be created first, which are then used as vertices for the polygon. But it was not possible to import all polygons from the dataset, as many of them consist of hundreds of vertices and there was an upper limit of 5 million triples in the free edition of AllegroGraph we used.

Regarding management of complex geometries (not only points, but also lines and polygons), Parliament builds its R-tree index in less than 3 hours; the cost is considerable, but not prohibitive. In contrast, uSeekM seems problematic in that respect. Indeed, as all geometries are inserted into a PostGIS backend, the spatial index must be frequently updated and the combined operation takes unacceptably long time (about 33 hours to insert all layers and rearrange the spatial index). Strabon took slightly longer to import all data into its PostGIS backend, again due to repeated reorganization of the GiST index. This is orders of magnitude higher compared to the spatial indexing cost in DBMSs, and clearly sets a challenging task for improvement in triple stores. Thanks to its multi-thread import utility, Virtuoso was the absolute champion, with loading and indexing completed in just a couple of minutes.

Query semantics involving spatial expressions, functions and topological predicates is another concern. Quite disappointingly, no query expression was exactly the same in every RDF store. With the exception of those adopting GeoSPARQL (uSeekM, Parliament), all other stores have their own syntax, so it required much effort to identify all differences in spatial clauses. In AllegroGraph, OWLIM-SE, and Virtuoso only basic geospatial functionality is available, so typical queries such as range search (R1) or distance-based joins (SJ2) can be executed. But more advanced spatial analysis is rather limited, such as in aggregates over spatial features (A1), or completely absent, e.g., for derived geometries (G1, G2, G3).

In terms of *selectivity*, the examined queries vary widely. For instance, range query R1 involves a small area (around Oxford) with sparse distribution of points, whereas query R2 specifies a much wider area (the greater London) with the highest density of sought points. Ideally, a system should perform well both when the result set is small (R1) and potentially large (R2).

Regarding queries with *combined* thematic and spatial filters, note that the area of interest in TS2 is 100 times greater than the one

Table 4: Performance statistics for the geospatial platforms tested (RDF stores and DBMS).

	OWLIM-SE (*points only*)	AllegroGraph (*points only*)	TripleStore X (*all layers*)	Virtuoso (*all layers*)	uSeekM (*all layers*)	Strabon (*all layers*)	Parliament (*all layers*)	DBMS Y (*all layers*)	PostGIS (*all layers*)
Triples	3,265,602	2,675,212	25,796,666	25,796,652	25,796,782	25,796,782	25,796,652	N/A	N/A
Records	N/A	N/A	N/A	N/A	N/A	N/A	N/A	3,455,983	3,456,000
Geometries	590,390	590,390	3,455,975	3,455,714	3,456,000	3,456,000	3,455,961	3,455,983	3,456,000
BL	15,300 ms	N/A	7,771,304 ms	N/A	N/A	N/A	20,491,671 ms	1,464,000 ms	473,356 ms
SI	18,100 ms	N/A	4,980,000 ms	N/A	N/A	N/A	10,393,209 ms	291,313 ms	412,772 ms
BL+SI	N/A	3,163,122 ms	N/A	146,291 ms	118,451,387 ms	117,879,000 ms	N/A	N/A	N/A
L1	23 ms	110 ms	735 ms	1,960 ms	516 ms	104,281 ms	496 ms	71 ms	63 ms
L2	18 ms	45 ms	8,279 ms	4,078 ms	145 ms	108,057 ms	211,015 ms	23 ms	5 ms
R1	254 ms	766 ms	38,767 ms	3,961 ms	2,249 ms	100,711 ms	231,077 ms	24 ms	14 ms
R2	N/A	N/A	50,628 ms	42,359 ms	N/A	135,709 ms	402,740 ms	15,317 ms	12,964 ms
SJ1	N/A	N/A	*incomplete after 3.5 hours*	*incomplete after 5 hours*	N/A	175,190 ms	*incomplete after 4 hours*	1,023,189 ms	295,462 ms
SJ2	N/A	*crashed after* 3,573,521 ms	*incomplete after 3.5 hours*	*incomplete after 8 hours*	*incomplete after 8 hours*	*incomplete after 4 hours*	*incomplete after 4 hours*	*incomplete after 2 hours*	*incomplete after 2 hours*
SJ2a	*1000 results* in 49,034 ms	N/A	N/A	*100 results* in 38,457 ms	N/A	*100 results in* 16,804,349 ms	*100 results in* 2,108,014 ms	*100 rows* in 341 ms	*100 rows in* 140,896 ms
kNN1	11,743 ms	9,418 ms	3,951 ms	2,487 ms	*crashed after* 157,196 ms	133,260 ms	N/A	32 ms	N/A
kNN1a	N/A	N/A	N/A	N/A	17,290 ms	371,884 ms *but no results!*	143,540 ms	N/A	1,756 ms
kNN2	N/A	N/A	N/A	3,914 ms	*crashed after* 147,485 ms	444,836 ms	223,422 ms	223 ms	12,150 ms
kNN2a	7,385 ms	3,533 ms	N/A	N/A	N/A	N/A	N/A	N/A	N/A
G1	N/A	N/A	221,890 ms	N/A	2,255 ms	105,204 ms	17,952 ms	19,643 ms	1,133 ms
G2	N/A	N/A	3,605 ms	N/A	82 ms	101,472 ms	512 ms	225 ms	14,335 ms
G3	N/A	N/A	7,667 ms	N/A	100 ms	98,602 ms	926 ms	252 ms	385 ms
TS1	396 ms	212 ms	73,536 ms	2,468 ms	4,772 ms	101,803 ms	254,768 ms	53 ms	20 ms
TS2	2,426 ms	2,121 ms	1,403,180 ms	21,500 ms	21,489 ms	102,662 ms	328,862 ms	726 ms	136 ms
A1	N/A	N/A	*incomplete after 3.5 hours*	4,982,061 ms	*incomplete after 5 hours*	126,503 ms	*incomplete after 30 min*	727,360 ms	21,001 ms
N1	*incomplete after 90 min*	2,011 ms *, but incorrect result*	2,029 ms	2,704,046 ms	*incomplete after 8 hours*	321,162 ms	594 ms, *but no results!*	221,245 ms	*100 rows in* 248,759 ms

in TS1, but response times escalate sublinearly in many platforms. Of course, such requests take longer to execute than pure range search, but existence of mixed conditions in the `WHERE` clause seems to pose difficulties to the optimizer. Perhaps, this occurs because joins are not reordered or the spatial filtering does not make use of the underlying index (e.g., as currently done in Parliament [3]). Strabon incurs similar costs for range and combined search, basically due to good estimates of the respective selectivities.

Presence of *expensive operators* also varies from query to query. Some requests just ask for an entity and a few properties, so they are quite straightforward to execute (like L1, L2). But other operations, such as spatial joins or negations, incur huge overhead to the query processor and could push it to its limits.

Starting from *spatial joins*, SJ1 was answered by PostGIS and DBMS Y within minutes. Strabon managed even better (in less than 3 minutes), thanks to powerful query planning by PostgreSQL and PostGIS. All other triple stores either did not support the operation or failed to provide an answer. Similarly, execution of SJ2 remained incomplete after several hours in all systems, with partial results reported only for its relaxed variant SJ2a. As distance joins are quite natural in spatial analysis, this seems a major deficiency.

Indirect support for *k-NN search* with a `LIMIT k` modifier was possible in several triple stores. Especially Virtuoso reported answers rather fast, although execution times exceed by far those in the examined DBMS.

Insufficient support for *geoprocessing* is a serious flaw for many triple stores. While database systems were competent to provide derived geometries instantly, RDF stores offer rudimentary or no functionality at all. For example, GeoSPARQL implementations in uSeekM and Parliament can compute buffer zones (i.e., influence areas), but cannot unify adjacent geometries into a single one; this latter operation is not specified in [35], yet offered by DBMSs. Strabon offers complete answers, although at increased cost. In contrast, AllegroGraph cannot return dynamically generated polygons without explicitly storing their vertices as literals; Virtuoso and OWLIM-SE are ineligible as they can handle strictly points.

Quite interestingly, *negation* involving spatial predicates was supported in all systems, at least in query syntax. However, only TripleStore X, Virtuoso, Strabon, and DBMS Y returned a correct answer. In all other platforms, either the query was executing for many hours without results, or the reported answer was wrong (AllegroGraph), or returned instantly with no results (Parliament).

5.3 Other Observations

During query execution, Parliament was somewhat unstable, probably due to the considerable volume of data it had to manage. Curiously enough, datasets had to be imported four times, because the spatial index was causing serious troubles when resuming the service, and no processing could be started. uSeekM was also problematic is some cases, especially in k-NN search, when it crashed unexpectedly some time after submission; its performance on spatial joins also seems to require considerable improvement.

RDF stores that utilize a DBMS as their repository (i.e., Strabon, uSeekM) can really take advantage of its geospatial support for storing various geometries and answering most types of spatial queries. But they really suffer when it comes to bulk loading of millions of features, because the index (essentially, an R-tree) is predefined to host all geometries and requires reorganization upon new entries (or deletes/updates on geometries). Instead, platforms with a native support for storing triples in their own structures (e.g., Virtuoso) coped far better with massive insertions.

Performance of spatial operations in triple stores is rather poor compared to DBMSs; sometimes orders of magnitude worse. Many queries could not even be expressed (in grey in Table 4), and others failed or did not complete within a reasonable time (highlighted in yellow). Evaluation did not involve inferencing, a key advantage of triple stores but absolutely out of scope even for advanced DBMSs; otherwise, response times might get exacerbated even more.

6. RESEARCH PERSPECTIVES

We deem that our study offers much more than quantitative metrics on performance and a qualitative comparison on conformance with existing standards. Indeed, we were able to identify certain deficiencies of RDF stores concerning expressiveness, scalability, query optimization, and real-world relevance. Next, we point out some indicative, challenging topics that may be attractive for joint research by the Semantic Web and GIS communities.

6.1 Enhancing Core Geospatial Functionality

Most triple stores could not easily cope even with moderately large geodatasets, as those used in our tests. For larger areas (e.g., a continent, or the entire planet), performance would suffer both in terms of bulk loading and query processing. Therefore, *support for billions of geometry triples* is a major challenge for RDF stores. GeoSPARQL compliance would also remedy discrepancies in namespaces, geometry representations, georeferencing, etc.

Efficiency of *spatial indexing* over triples should be also improved, as it is evident when compared to similar functionality offered by DBMSs. Building or updating the spatial index (typically, an R-tree) should be performed faster. Apart from points, this structure should be able to host diverse geometric shapes (lines, polygons, etc.) and facilitate related computations. Taking advantage of that index to natively evaluate more types of spatial queries (e.g., k-NN, or geoprocessing) would also be worthwhile.

There is much room for *query optimization* techniques in presence of both spatial and thematic criteria. Currently, many optimizers split a query into blocks, and suggest that spatial predicates be evaluated first, followed by thematic filtering on the intermediate results. Yet, evaluation of such mixed filtering requests or spatial joins should rather take advantage of the spatial index, perhaps also assisted by estimated spatial selectivities. More advanced query plans might also make use of hybrid "spatio-semantic" access methods like [25] that interweave spatial and semantic properties in a common structure to enable aggressive pruning strategies.

In addition, APIs should be developed to seamlessly use RDF stores as backends in GIS applications (e.g., instead of PostGIS) and enable interoperability between different spatial formats, RDF repositories, or databases. Tools for integrating geospatial entities from diverse vocabularies (e.g., via mappings into GeoSPARQL ontology) would also be valuable, at least until GeoSPARQL becomes the de facto protocol for geospatial RDF data.

6.2 Advanced Geospatial Processing

As the amount of geospatial RDF content will inevitably increase on the Web, implementation of *query rewriting rules* could simplify query declaration (also recommended by GeoSPARQL [35]). For instance, a user request may include a triple pattern that tests a topological relation between two features; this could be transformed into an equivalent clause involving their geometries.

Support for *geometry-based inferencing* could strengthen topological consistency of the stored features, facilitate query expressiveness, and offer more intelligence to spatial reasoning. For example, built-in methods could be used to deduce, for every pair of adjacent polygons, their common borderline and identify which polygon is on either side (left/right) of each line vector. For such derived geometry features, several OWL assertions could be attached (e.g., a borderline could signify separation between counties, regions, or states), and the inference engine should apply consistency checks. Such assertions (along with other, non-spatial ones) could then be readily returned at query time. Turning the raw geometries into semantically enriched ones could also increase spatial accuracy of the original data, e.g., for easily detecting discrepancies in edge matching of lines or slight topology overlaps (a.k.a. "sliver polygons") due to digitization errors.

Another important research direction concerns *spatially-aware interlinking and fusion* of RDF datasets. More specifically, methods are needed to recognize potential matches between different geospatial datasets and connect resources that correspond to the same real world entity despite differing RDF representations. For instance, one dataset may represent a museum as a point and indicate its admission times and pricing, while another dataset may capture it as a polygon along with detailed information about its current exhibition. Such integration would improve the quality and augment the quantity of stored triples. If this process involved geometry matchings (e.g., point in polygon), it could produce richer, combined metadata for the matched entities (e.g., source, scale, last update). This could greatly leverage the value of linked data, benefiting a lot from geospatial information either implicitly (e.g., names, types of places) or explicitly (e.g., coordinates, distances).

6.3 Further Extensions

Many opportunities also arise for building extensions or add-ons to existing systems. Next, we mention three notable cases:

Open, crowdsourced geographic data like OpenStreetMap [38] undergo frequent changes from multiple contributors, so the system has to keep track of all successive updates. Thanks to global IRIs, *managing geospatial versions* of RDF data could be carried out in a more sophisticated way, while also keeping metadata for provenance. In case that an existing geometry (e.g., a road segment) gets modified by diverse users, this would require a more advanced mechanism for reconciliation of conflicting updates, as it might also affect other adjacent entities (e.g., nearby roads).

Support for map visualization would also be a precious tool for faceted spatial analysis and could attract GIS practitioners to start using triple stores. With the exception of AllegroGraph [9] and a KML exporting functionality from Strabon [24], there is no indication about mapping capabilities in triple stores. But, the growing interest on geospatial DBMS over the past decade indicates that interconnection of spatial data infrastructure with commercial or open-source GIS platforms provides a wider user community, an increased number of applications, and leads to further improvements in terms of processing efficiency and offered functionality.

Last, but not least, integration of *temporal and spatiotemporal semantics* [23] would greatly enrich expressiveness of the RDF schema. With a temporal component, the model can be used for representation and querying of linked geospatial data that changes over time (e.g., boundaries of land parcels, water levels in lakes etc.). With spatiotemporal support, movement of spatial entities can be linked with thematic data and provide a wealth of information about moving objects (e.g., vehicles, cargo containers, etc.).

7. CONCLUSIONS

In this paper, we presented the current state-of-the-art in geospatially-enabled semantic data management. Starting from an overview of the recently proposed GeoSPARQL standard, we examined whether and to what extent it is actually supported in several, widely used semantic repositories. As no purposeful benchmark is established yet, we designed a methodology for specifically evaluating geospatial capabilities of RDF stores. Moreover, we conducted a comprehensive empirical study on some well known RDF stores either available from commercial vendors or research prototypes. We utilized OpenStreetMap layers as a large data source, and a wide range of spatial queries as workload. We also compared efficiency of RDF stores against traditional geospatial databases in terms of bulk loading, spatial indexing and query response time.

We expect a mounting interest on handling geospatial data on the Semantic Web, both in terms of practical applications and innovative processing algorithms. Hopefully, this would bridge the gap between the GIS and Semantic Web communities. To assist this effort, we plan to extend our baseline approach with ready-to-use software, to include scaling datasets for several real-world scenarios, and to refine the query workload with reasoning semantics.

8. ACKNOWLEDGEMENTS

This work was partially supported by the European Commission under EU/FP7 grant #318159 for project *"GeoKnow: Making the Web an Exploratory Place for Geospatial Knowledge"*. We particularly wish to thank Michalis Alexakis for his valuable support in VM configuration and software installation. Zoi Kaoudi and Nikos Bikakis provided suggestions on an early draft of this paper.

9. REFERENCES

[1] S. Auer, J. Lehmann, and S. Hellmann. LinkedGeoData – Adding a Spatial Dimension to the Web of Data. In *ISWC*, pp. 731-746, 2009.

[2] Basic Geo (WGS84 lat/long) Vocabulary. URL: http://www.w3.org/2003/01/geo/

[3] R. Battle and D. Kolas. Enabling the Geospatial Semantic Web with Parliament and GeoSPARQL. *Semantic Web Journal*, 3(4):355-370, 2012.

[4] Bigdata Triple Store. URL: http://www.systap.com

[5] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *IJSWIS*, 5(3): 1-22, 2009.

[6] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *IJSWIS*, 5(2):1-24, 2009.

[7] A. Brodt, D. Nicklas, and B. Mitschang. Deep Integration of Spatial Query Processing into Native RDF Triple Stores. In *ACM GIS*, pp. 33-42, 2010.

[8] European Commission. *INSPIRE Directive:* Infrastructure for Spatial Information in the European Community. URL: http://inspire.jrc.ec.europa.eu/

[9] Franz Inc. AllegroGraph Triple Store. URL: http://www.franz.com/agraph/allegrograph/

[10] G. Garbis, K. Kyzirakos, and M. Koubarakis. Geographica: A Benchmark for Geospatial RDF Stores. In *ISWC*, 2013.

[11] GeoJSON 1.0. URL: http://geojson.org/

[12] GeoKnow EU/FP7 project. Deliverable 2.1.1: Market and Research Overview. URL: http://bit.ly/1pElP7L

[13] Geo OWL Ontology. URL: http://bit.ly/1q9Th3Q

[14] GeoRDF Profile. URL: http://www.w3.org/wiki/GeoRDF

[15] GeoRSS. URL: http://georss.org/

[16] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*, pp. 47-57, 1984.

[17] J. Hellerstein, J. Naughton, A. Pfeffer. Generalized Search Trees for Database Systems. In *VLDB*, pp. 562-573, 1995.

[18] H. Hu and X. Du. Linking Open Spatiotemporal Data in the Data Clouds. In *RSKT*, pp. 304-309, 2010.

[19] IBM DB2 Spatial Extender. URL: http://ibm.co/1ohWGfv

[20] IBM DB2 NoSQL Support. URL: http://ibm.co/1mdHnWF

[21] D. Kolas. A Benchmark for Spatial Semantic Web Systems. In *SSWS*, 2008.

[22] D. Kolas, I. Emmons, and M. Dean. Efficient Linked-List RDF Indexing in Parliament. In *SSWS*, pp. 17-32, 2009.

[23] W. Kuhn, T. Kauppinen, and K. Janowicz. Linked Data – A Paradigm Shift for Geographic Information Science. In *GIScience*, pp. 173-186, 2014.

[24] K. Kyzirakos, M. Karpathiotakis, M. Koubarakis. Strabon: A Semantic Geospatial DBMS. In *ISWC*, pp. 295-311, 2012.

[25] J. Liagouris, N. Mamoulis, P. Bouros, and M. Terrovitis. An Effective Encoding Scheme for Spatial RDF Data. *PVLDB*, 7(12): 1271-1282, 2014.

[26] Microsoft SQL Server. URL: http://bit.ly/1lJYj62

[27] M. Morsey, J. Lehmann, S. Auer, and A.C.N. Ngomo. DBpedia SPARQL Benchmark: Performance Assessment with Real Queries on Real Data. In *ISWC*, pp 454-469, 2011.

[28] MySQL Database. URL: http://www.mysql.com/

[29] NeoGeo Geometry Ontology. URL: http://geovocab.org/

[30] Ontotext AD. GraphDB (formerly OWLIM) Triple Store. URL: http://www.ontotext.com/ontotext-graphdb-owlim/

[31] OGC Geography Markup Language Encoding Standard, Version 3.2.1, 2007. URL: http://bit.ly/1x52Cw4

[32] OGC Geography Markup Language (GML) Simple Features Profile, Version 2.0, 2012. URL: http://bit.ly/1pPKD8I

[33] OGC Implementation Specification for Geographic Information - Simple Feature Access - Part 2: SQL Option, Version 1.2.1, 2010. URL: http://bit.ly/1mN6NJR

[34] OGC Implementation Standard for Geographic Information - Simple Feature Access - Part 1: Common Architecture, Version 1.2.1, 2011. URL: http://bit.ly/1qK0gje

[35] OGC GeoSPARQL Standard - A Geographic Query Language for RDF Data, 2012. URL: http://bit.ly/1vnWA76

[36] OpenLink Software. Virtuoso Universal Server. URL: http://virtuoso.openlinksw.com/

[37] OpenSahara uSeekM library. URL: http://bit.ly/1ATPPyC

[38] OpenStreetMap. URL: http://www.openstreetmap.org/

[39] Oracle Spatial and Graph. URL: http://bit.ly/1qQo07k

[40] OWL Web Ontology Language Overview. URL: http://www.w3.org/TR/owl-features/

[41] K. Patroumpas, M. Alexakis, G. Giannopoulos, and S. Athanasiou. TripleGeo: an ETL Tool for Transforming Geospatial Data into RDF Triples. In *LWDM*, pp. 275-278, 2014.

[42] M. Perry. A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data. *PhD thesis*, Wright State University, 2008.

[43] PostGIS for PostgreSQL. URL: http://postgis.net/

[44] PostgreSQL Database. URL: http://www.postgresql.org/

[45] S. Ray, B. Simion, and A. Demke Brown. Jackpine: a Benchmark to Evaluate Spatial Database Performance. In *ICDE*, pp. 1139-1150, 2011.

[46] Raytheon BBN Technologies Inc. Parliament Triple Store. URL: http://parliament.semwebcentral.org/

[47] Resource Description Framework Primer. URL: http://www.w3.org/TR/rdf-primer/

[48] RDF Schema. URL: http://www.w3.org/TR/rdf-schema/

[49] Sesame RDF Framework. URL: http://www.openrdf.org/

[50] SPARQL 1.1 Query Language for RDF. URL: http://www.w3.org/TR/sparql11-query/

[51] Stardog RDF database. URL: http://www.stardog.com/

[52] C.-J. Wang, W.-S. Ku, and H. Chen. Geo-Store: A Spatially-Augmented SPARQL Query Evaluation System. In *ACM SIGPATIAL GIS*, pp. 562-565, 2012.

[53] D. Wang, L. Zou, Y. Feng, X. Shen, J. Tian, and D. Zhao. S-store: An Engine for Large RDF Graph Integrating Spatial Information. In *DASFAA*, pp. 31-47, 2013.

[54] X. Zhai, L. Huang, and Z. Xiao. Geo-spatial Query based on Extended SPARQL. In *Geoinformatics*, pp. 1-4, 2010.