Revisiting the Representation of and Need for Raw Geometries on the Linked Data Web

Blake Regalia
blake@geog.ucsb.edu
STKO Lab, University of California, Santa Barbara, USA

Krzysztof Janowicz
jano@geog.ucsb.edu
STKO Lab, University of California, Santa Barbara, USA

Grant McKenzie
gmck@umd.edu
Department of Geographical Sciences, University of Maryland, USA

ABSTRACT

Geospatial data on the Semantic Web historically stems from using point geometries to represent the geographic locations of places. As the practice evolved in the Semantic Web community, a demand for more complex geometries and geospatial query capabilities came about as a consequence of integrating traditional GIS and geo-data into the Linked Data cloud. However, recent projects have revealed that, in practice, these established techniques have major shortcomings that limit their storage, transmission and query potential. In this position paper, we examine these shortcomings, propose to treat geometries similar to how other binary data are stored and referenced on the Semantic Web, namely by representing them as resources via URIs instead of RDF literals, and demonstrate the utility of precomputing topological relations rather than computing them on-demand by arguing that end users are most often interested in topology and not raw geometries.

ACM Reference format:
Blake Regalia, Krzysztof Janowicz, and Grant McKenzie. 2016. Revisiting the Representation of and Need for Raw Geometries on the Linked Data Web. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Looking back to the origins of publishing geospatial data on the Semantic Web, the W3C Semantic Web Interest Group (SWIG) introduced the Basic Geo Vocabulary[1] circa 2003 in order to "explore the possibilities of representing mapping/location data in RDF." Initially, this vocabulary was considered good enough for annotating web documents and XML resources with basic location metadata (e.g., `<pos:lat>51.46</pos:lat> <pos:long>-0.45</pos:long>`[2]). It even brought about an immediate linking of resources via services such as GeoURL[3], which allowed users to find URLs by their proximity to a given location such as "your neighbor's blog" or "restaurants near you". The establishment and subsequent widespread usage of such a W3C vocabulary made it an obvious choice for early contributors of geospatial data on the Semantic Web. Among the first major contributors were gazetteers, such as GeoNames, who housed spatial databases comprised entirely of latitude/longitude coordinate pairs of geocoded places. To this extent, the Basic Geo Vocabulary was still sufficient. However, it soon became clear to the growing Semantic Web community that a more comprehensive geospatial vocabulary was needed in order to deal with geometries beyond single points, as well as a general support for coordinate reference systems, and most importantly the distinction between the entity on the surface of the Earth and the many possible geometries one can use to represent it given various contexts, scales, use cases, and so forth. In fact, the authors of the Basic Geo Vocabulary explicitly acknowledged that it "does not attempt to address many of the issues covered in the professional GIS world".[1]

Between 2006 and 2011, prior to the standardization of OGC's GeoSPARQL[10], several groups set out to establish a successor geospatial vocabulary that would support the serialization of various geometry types as RDF along with the means to reason and query on those geometries. A community known as NeoGeo drafted a vocabulary[4] that was guided by the idea to convert entire data structures (down to primitive datatypes) into RDF. A serialization of a polygon using NeoGeo is shown in Listing 1.

```
:polygon rdf:type ngeo:Polygon ;
    ngeo:exterior [ rdf:type ngeo:LinearRing ;
        ngeo:posList (
            [ geo:lat -29; geo:long 16 ]
            [ geo:lat -28; geo:long 33 ] ... )] ;
    ngeo:interior [ ... ]
```

Listing 1 Using the NeoGeo Vocabulary to serialize the geometry of a polygon.

While such a serialization certainly follows the Linked Data paradigm's call for raw data, it also drew criticisms for its excessive creation of blank nodes[1] and the burdens of storing and querying complex geometries with such a high degree of geometric decomposition. Furthermore, separating the latitude and longitude values could introduce ambiguity as to which two values belonged to a coordinate pair and more importantly, had no apparent query use case besides searching for points within a given bounding box. NeoGeo, and with it many other approaches proposed for different kinds of (non-geographic) data, raised the interesting question of what to triplify and what to consider a leaf node. On the one hand, only a direct RDF representation allows for reasoning, direct linkage, reuse of raw data, and so on. On the other hand, the same triplified data is often difficult or impossible to process by domain applications such as GIS, it is not efficient in terms of storage nor processing, and is often not easily read and understood by humans.

One solution that addressed these criticisms for geographic data was to store the entire geometry in a single RDF literal, thus

[1]https://www.w3.org/2003/01/geo/
[2]http://swig.planetrdf.com/2003/01/10/2003-01-10.html#1042200521.031970
[3]https://goo.gl/5ho0Pv
[4]http://geovocab.org/doc/neogeo.html

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

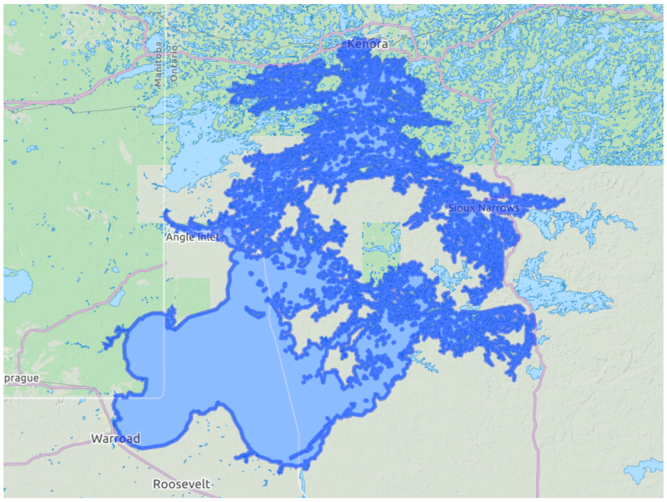

Figure 1 Multipart polygon representing the Lake of the Woods.

eliminating any issues brought on by embedding complex structures as RDF. Serialized geographic formats such as Geographic Markup Language (GML) and Well-Known Text (WKT) offered accessible means to encode geometries in a human-readable form. GeoSPARQL [10] adopted these two formats in its first iteration of the standard which was approved in 2012. GeoSPARQL was a break-through for storing serialized geometry data within RDF triples, supporting coordinate reference systems, maintaining the distinction between entities and their geometric representation, and enabling geospatial queries on linked geographic data. However, feedback from implementors and data publishers has revealed latent problems with scaling, e.g., challenges associated with the storage and transmission of large WKT strings, and timely execution of SPARQL queries that make use of geospatial functions. Some projects sidestepped these issues by storing multiple versions of a feature's geometry at different levels of simplification, sacrificing storage space for speed while not compromising on data quality.

Serializing geometries as WKT literals may be suitable in some cases but prohibitive in others. Consider, for example, the notable *Lake of the Woods* body of water (Fig. 1) and its nearly 15,000 islands. A geometric representation of the lake consists of 4,484 rings formed by 487,505 nodes. The resulting WKT literal[5] is 11 MB large, not human-readable, and is too cumbersome to be used directly as input for reasoning, e.g., to compute topological relations. These problems become apparent when dealing with a multitude of complex geometries[9], such as the USGS Digital Line Graph data[6]. While the *Lake of the Woods* might seem like an extreme example, the reality is that even WKT strings of 11 KB (the average size of OpenStreetMap geometries for the United States) far exceed the capacity for human-readability. By comparison, while it might make sense to store a single color value as a hexadecimal color code in an RDF literal, it does not follow that pixel data for an entire image should also be encoded as a literal.

In light of these issues and the responses they have precipitated, we reconsider the techniques set forth by GeoSPARQL in favor of

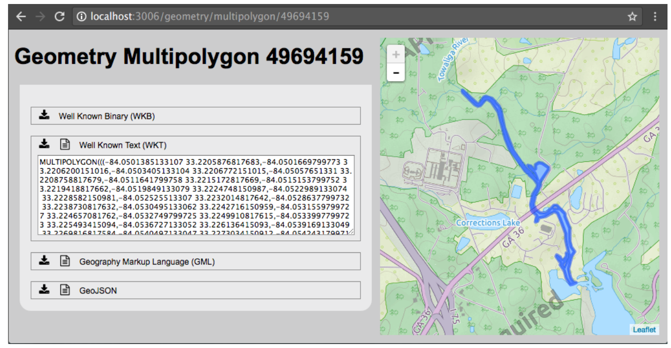

Figure 2 Screenshot of our web interface dereferencing a geometry's URI.

two proposed alternatives that are driven by practicality. Specifically, we argue that (1) serialized geometry data beyond points and bounding boxes do not need to be expressed in RDF and that (2) geospatial queries on Linked Data will benefit from storing precomputed topological relations instead of, or in addition to, raw geometries. We refer to our approach here by the nickname 'AGO'.

2 REPRESENT GEOMETRY WITH URIS

In our approach, rather than storing geometry as RDF literals in the triples of blank nodes (as is common with GeoSPARQL), we opt to use Uniform Resource Identifiers (URIs) to represent the geometry as a resource. This approach is compatible with existing GeoSPARQL datasets and implementations because it still *allows* for statements about a geometry (e.g., `geosparql:asWKT`), however it relieves the triplestore from *having* to bear responsibility for the entire geographic dataset. Instead, geometry data can persist elsewhere, such as within a local geodatabase or on a remote server.

The broader line of reasoning we emphasize here is that given the complexity of high-resolution geometries used by production-level Geographic Information Systems and many of the most prominent science datasets, e.g., provided by USGS National Map, raw geometries hold little value in a human-readable medium when compared to their more efficient binary formats. Without an application layer to render them on a map or perform some geospatial analysis, the only discernible information that complex geometries (e.g., WKT strings) can convey to humans are the types of features they contain (e.g., point, linestring, or polygon). Therefore, we believe geometry data should be treated similarly to how other binary data is stored and referenced on the Semantic Web, namely via URIs. Just as with other binary resources on the web, we envision the client having the option to download geometry data by dereferencing their URIs. Coupled with content negotiation, such an approach allows clients to fetch geometry in a format that suits their needs (in addition to being able to negotiate for its RDF). To give an example, we implemented an HTTP server that supports various 'Accept' header media (MIME) types for geometries as shown in Table 1.

When dereferencing a geometry's URI in a web browser (MIME type `text/html`), we designed a simple interface to display the geometry on a map and provide access to the actual data by triggering one of the other four supported content-types as shown in Fig. 2.

We mint each URI with human-readable metadata about the geometry it represents; this includes the geometry type, its unique ID,

[5] Available at: http://stko-testing.geog.ucsb.edu/blake/resource/lake-of-the-woods.txt
[6] https://lta.cr.usgs.gov/DLGs

```
curl "http://ex.co/geometry/polygon/12345" -H "Accept: $MIME_TYPE"
```

MIME Type	Description	Returns
text/html	Web interface	<!DOCTYPE html><html lang="en">...
text/plain	Well-Known Text	POLYGON((113.1016 -38.062 ...))
application/gml+xml	GML	<gml:Polygon><gml:Exterior>...
application/json	GeoJSON	{"type":"Polygon","coordinates":...}
application/octet-stream	Well-Known Binary	01 06 00 00 20 E6 10 00 00 01...

Table 1 Five MIME types and their associated return values that our experimental server software supports when dereferencing a geometry's URI.

and the bounding box coordinates of the geometry (in the WGS84 coordinate reference system). For point geometries, this bounding box metadata is reduced to just the single coordinate pair, effectively encoding the entire geometry within the text of the URI. While such practice is not (yet) standardized, it enables clients to filter geometries by type and to begin building a spatial index if their application supports it. In our experiment, we encoded URIs for 2.2 million geometries from the USGS Geographic Names Information System (GNIS) using this scheme (Listing 2).

```
           base uri         geometry ID
http://ex.co/geometry/polygon/12345#113.05281,-38.11945/153.30671,-11.15957
                     geometry type       WGS84 bounding box coordinates
```

Listing 2 Encoding scheme used to mint URIs representing a geometry.

When the GeoSPARQL specification succeeded NeoGeo's vocabulary and related approaches, a few resourceful concepts were unfortunately sacrificed in the process that were among the strengths of NeoGeo. Most notably, reusing existing geometries to create so-called 'Composite Geometries' was a promising way to derive new features while tracking the provenance of its constituents – something that is not practical to achieve with RDF literals and that we would therefore consider a shortcoming of the current GeoSPARQL specification. This idea is expanded even further when considering geometric operations such as creating a union, intersection, difference, buffer, convex hull, and so forth. With the use of URIs, however, the option to reuse geometries is feasible once again. For example, one could describe the geometry of a university's spatial extent by the union of the geometries for its constituent features such as its campus, sports stadium, and off-campus housing areas.

To summarize, we compare the strengths and weaknesses of three different approaches to storing and querying geospatial information as Linked Data for GeoSPARQL, NeoGeo, and our AGO proposal in Table 2.

While the last row in Table 2 indicates a weakness of using URIs to represent geometries, this is only from the perspective of an inactive client. If one considers the broader perspective of an application, this weakness is replaced by a strength. For instance, rather than downloading the geometry in a prescribed format, the client application can strategically schedule a bulk download of just the geometries it wants (e.g., of a certain type or within a given bounding box) in either a compact binary format (e.g., Well-Known Binary) or a ready-on-arrival format (e.g., GeoJSON for JavaScript applications). The ability to download multiple geometries in a single HTTP request certainly has its advantages, and we concede that coming up with an HTTP-friendly solution for doing this with geometry resource URIs needs discussion.

3 WHAT ABOUT TOPOLOGY?

With the advent of GeoSPARQL and other means to perform spatio(temporal) queries [7] over Linked Data, storing complex geometries as RDF is becoming more popular. The LinkedGeoData project [11], for example, provides different geometry types, such as polygons, extracted from OpenStreetMap. These geometries can be utilized for two types of queries, those that involve or infer topological relations and those that are non-topological such as distances, buffers, and convex hulls.

Replacing the simple geometries that dominate knowledge graphs and search engines today with more complex geometries will be of limited use (beyond applications such as routing and visualization). Instead, we believe that knowledge graphs and Linked Data more concretely will see a greater benefit from storing topological relations. One could argue that such topological relations can be computed using geometries but not the other way around. While this is true in an abstract mathematical sense, it does not hold for actual data. In fact, topological relations between places cannot be easily computed based on geometry alone. While there are many reasons for this [6, 12], our argumentation will focus on the role of domain knowledge, vagueness, and uncertainty [2] and not on computational issues.

3.1 Challenges

To understand how topology is handled in GIS, it is important to note that data collection, modeling, and pre-processing take about 80% of the time budget of a typical GIS project. When data are loaded into a GIS, the analyst uses a sequence of toolboxes to first correct common errors such as so-called *sliver polygons* and then applies domain-specific topological consistency rules.[7] Neither the pre-processing steps nor the domain-specific topological rules are available when computing topological relations on-demand using GeoSPARQL over Linked Data. In addition, the datasets used for any given GIS task that involves topological relations are orders of magnitude smaller than querying such relations over Linked Data hubs such as DBpedia, i.e., they involve dozens of hundreds of polygons or linestrings but not hundreds of thousands. Queries such as finding cities along the Mississippi River or counties along state borders cannot be effectively answered over Linked Data today.

Consider the following illustrative example. Given that Lynchburg, Tennessee is a consolidated city-county whose boundaries coincide with Moore County, Region Connection Calculus 8 (RCC8) dictates that the true topological relation between the city and the county must be *equal* (EQ). Computing the relation using the GeoSPARQL-enabled Apache Marmotta triplestore, however, will

[7] See, for example, the following overview of geodatabase topology rules by ArcGIS http://resources.arcgis.com/en/help/main/10.2/01mm/pdf/topology_rules_poster.pdf.

Trait	GeoSPARQL	NeoGeo	AGO
Uniform RDF structure	✓		✓
Efficient geometry storage			✓
Content-negotiation for geometry format		✓	✓
Composite geometries		✓	✓
Geometry can persist externally [1]			✓
Determine geometry type [2]	✓	✓	✓
Access bounding box [2]			✓
Access raw geometry [2]	✓	✓	

1 = Geometry can persist in a local geodatabase or even on a remote system and *without copies*.
2 = From the triples' RDF data alone (e.g., without using SPARQL).

Table 2 A comparison of strengths and weaknesses of the three different approaches to storing geometry data: GeoSPARQL, NeoGeo and AGO.

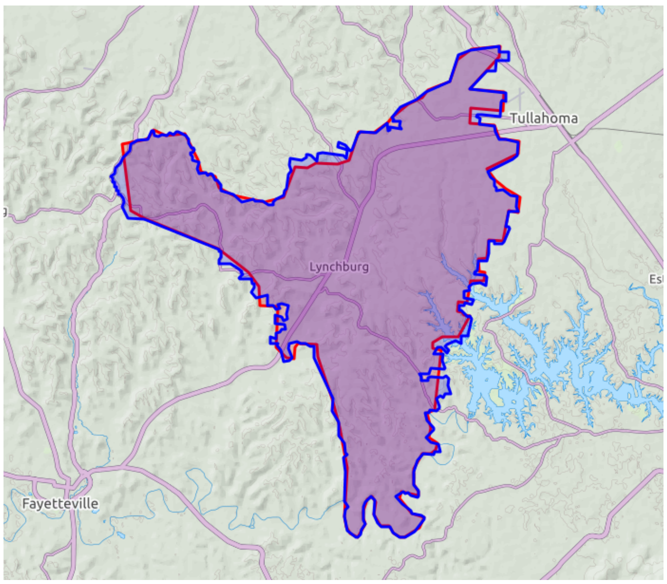

Figure 3 Lynchburg, Tennessee is a consolidated city-county whose boundaries coincide with Moore County. While proper topological RCC8 relation should be *equal* (EQ), computing the relation based on geometries alone will return *partial overlap* (PO).

return a *partial overlap*; see Fig. 3. The reason for this is due in large part to digitization errors. More concretely, the so-called *double-digitized boundaries problem* in which the blue boundary has been digitized to a greater degree of detail compared to the red boundary. While such differences in granularity are common sources of error, difficulties arising from uncertainty and vagueness are even more troublesome. Whereas uncertainty stems from a lack of precise knowledge, vagueness is caused by intrinsically underdetermined concepts that do not have clear borders [2]. For example, the true shape of a city can be determined in theory however, measurement accuracy, timeliness (the city may grow or shrink), and so forth, impact the results. In contrast, the shape of a mountain or forest cannot be exactly determined in practice nor theory as the transition zone between a mountain and a valley, as well as between a forest and isolated trees, is *conceptually* vague [8]. In fact, the Lake of the Woods example or the number of lakes in Minnesota more generally are famous examples for this challenge as the number of lakes depends on the size of many small lakes (under 10 acres) which in turn depend on the seasonal water level and so forth.

3.2 Strict Topological Relations

To experiment with the effect of precomputed relations on a linked dataset, we took a geospatial dataset consisting of counties, cities, parks, streams, and so on from the United States and computed several topological relations among polylines and polygons. Out of 18.6k polygons and 7.7k polylines, we extracted a total of 68.6k distinct relations to be materialized before querying takes place. We only materialize a relation in one direction and use reasoning to handle symmetric, transitive and disjoint properties. These strict relations represent the topology of polygons after being cleaned of digitization errors. We show the counts and statistics for this *precomputed* set in Table 3.

To give an example of the impact this practice can have on querying, we compared a topological GeoSPARQL query to its equivalent topological AGO (precomputed) query. The two SPARQL queries are shown in Listing 3. A simple trial shows that where a GeoSPARQL query takes 1318ms to complete due to the need for on-demand computation, an equivalent precomputed topological query takes 112ms, approximately 11× faster for our dataset. The other difference to our topological query is that it also returns more results since it has *cleaned* the digitization errors, an important step to GIS analysis that GeoSPARQL does not currently support.

```
# a) GeoSPARQL
select ?place where {
   <http://dbpedia.org/resource/Tulsa,_Oklahoma>
      geosparql:hasGeometry [geosparql:asWKT ?wktA].
   ?place geosparql:hasGeometry [geosparql:asWKT ?wktB].
   filter(geof:sfTouches(?wktA, ?wktB)) }

# b) Our AGO approach
prefix agt: <http://awesemantic-geo.link/topology/> .
select ?place where {
   <http://dbpedia.org/resource/Tulsa,_Oklahoma>
      agt:touches ?place. }
```

Listing 3 Comparison of a topological SPARQL query for places that *touch* the city of Tulsa, Oklahoma using (a) GeoSPARQL and (b) AGO.

3.3 Uncertainty and Vagueness

As we discussed above, strict/crisp topological relations (i.e., those derived from an intersection-matrix of source geometries) alone do not account for the vagueness and uncertainty principles that exist for geographic data. Therefore, we propose a multi-layered topological relations framework to encompass these principles in

Strict Topological Relations

			avg. area/length of...	
# instances	**relation**	**code**	**left geometry**	**right geometry**
19,134	polygon **touches** polygon	EC	$960km^2$	$2,049km^2$
1,272	polygon **overlaps** polygon	PO	$321km^2$	$2,974km^2$
1,287	polygon **tpp** polygon	TPP	$57km^2$	$2,653km^2$
2,577	polygon **ntpp** polygon	NTPP	$16km^2$	$3,052km^2$
3	polygon **equals** polygon	EQ	$830m^2$	$836m^2$
19,543	polyline **touches** polygon	TCH	$652km$	$701km^2$
7,871	polyline **crosses** polygon	PTH	$290km$	$2,733km^2$
5,733	polyline **within** polygon	INC	$6.5km$	$6,863km^2$
11,227	polyline **crosses** polyline		$395km$	$688km$

Table 3 Some statistics about the distinct, strict topological relations computed between combinations of polyline and polygon using RCC8/DE-9IM[4] or 16-Intersection-Matrix[5]. See references for codes. For cases comparing two geometries of the same type, the 'left' is the shorter/smaller of the two.

attempt to bring clarity to the fuzzy nature surrounding spatial relations for regions. In other words, we supplement the set of strict topological relations by computing additional topology for features that may have *broad boundaries* [3] as well as for features that may exhibit *cognitive relations*, e.g., Brazil is *mostlyInside* the Southern Hemisphere.

The challenges of computing topological relations for features with broad boundaries are not limited to designing an ontology that determines *which features* should be considered to have a broad boundary and *what types* of relations may ensue, but also deciding on a mathematical framework to use for calculating the boundaries [3]. A good place to start with an ontology might be by excluding cases that are forbidden by their definition. For example, two counties may qualify for the `agt:broadlyTouches` relation if they are located relatively nearby, however no two counties should ever be considered for the `agt:broadlyOverlaps` relation as any area in the U.S. can legally only be under the jurisdiction of one county.

Our method for calculating broad boundaries is to use the isoperimetric quotient of a polygon, given by $Q = \frac{4\pi Area}{Perimeter^2}$. After computing a polygon-to-polygon distance matrix for each combination of feature types (e.g., city-to-city, city-to-park, etc.) we sort the distances to create a cumulative distribution function and select the 0.05 percentile value as p. Then, a polygon's broad boundary radius R (i.e., buffer radius) is given by $R = Q \cdot p$. This model follows the rationale that simpler polygons deserve broader boundary radii than polygons having more complex structure because a finer resolution might generally imply a more precise digitization.

4 SUMMARY AND CONCLUSION

In this work we have revisited two pressing issues, how to store geometries and what role do they play in querying Linked Data. We showed that the established practices of storing complex geometries (beyond points and bounding boxes) as RDF literals, while suitable in some cases, should be reconsidered for other GIS applications and the many domain datasets that make use of complex geometries. We proposed an alternative method for storing geometry data by representing them via URIs in RDF and allowing the client to obtain data in the desired format by dereferencing the URI via content negotiation. We then argued that many GIS and geographic information retrieval queries do not utilize geometries directly but rely on topological relations, and that this may hold for Linked Data usage as well, i.e., users often want to know whether two places are adjacent rather than what their exact geometries are. Computing such topological relations on-demand using GeoSPARQL is possible but leads to very common data quality errors. Furthermore, we suggest that even with today's geospatial query tools that supplement Linked Data, supporting strict topological relations alone is not enough to satisfy the spectrum of user-driven topology queries about relations between regions due to uncertainty and vagueness.

Acknowledgments: We acknowledge support from USGS from the *Linked Data for the National Map* award.

REFERENCES

[1] Ghislain Auguste Atemezing and Raphaël Troncy. 2012. Comparing vocabularies for representing geographical features and their geometry. In *Terra Cognita 2012 Workshop*, Vol. 3.

[2] Brandon Bennett. 2001. What is a forest? On the vagueness of certain geographic concepts. *Topoi* 20, 2 (2001), 189–201.

[3] Eliseo Clementini and Paolino Di Felice. 1997. Approximate topological relations. *International journal of approximate reasoning* 16, 2 (1997), 173–204.

[4] Max J Egenhofer, Jayant Sharma, and David M Mark. 1993. A critical comparison of the 4-intersection and 9-intersection models for spatial relations: formal analysis. In *Autocarto Conference*. ASPRS American Society for Photogrammetry and Remote Sensing ASPRS.

[5] Anna Formica, Mauro Mazzei, Elaheh Pourabbas, and Maurizio Rafanelli. 2012. A 16-intersection matrix for the polygon-polyline topological relation for geographic pictorial query languages. In *International Conference on Availability, Reliability, and Security*. Springer, 302–316.

[6] Wm Randolph Franklin. 1984. Cartographic errors symptomatic of underlying algebra problems. In *International Symposium on Spatial Data Handling, Zurich, Switzerland*, Vol. 286.

[7] Manolis Koubarakis and Kostis Kyzirakos. 2010. Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL. In *Extended Semantic Web Conference*. Springer, 425–439.

[8] Daniel R Montello. 2003. Regions in geography: Process and content. *Foundations of geographic information science* (2003), 173–189.

[9] Kostas Patroumpas, Giorgos Giannopoulos, and Spiros Athanasiou. 2014. Towards GeoSpatial semantic data management: strengths, weaknesses, and challenges ahead. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 301–310.

[10] Matthew Perry and John Herring. 2012. OGC GeoSPARQL - A geographic query language for RDF data. *OGC Implementation Standard. Sept* (2012).

[11] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. 2012. LinkedGeoData: A core for a web of spatial open data. *Semantic Web* 3, 4 (2012), 333–354.

[12] Thierry Ubeda and Max J Egenhofer. 1997. Topological error correcting in GIS. In *International Symposium on Spatial Databases*. Springer, 281–297.