UNIVERSITY of CALIFORNIA
Santa Barbara

Computational Time and Space Tradeoffs in Geo Knowledge Graphs

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Doctor of Philosophy

in

Geography

by

Blake D. Regalia

Committee in charge:

Dr. Krzysztof Janowicz, Chair

Dr. Keith C. Clarke

Dr. Werner Kuhn

June 2020

The dissertation of Blake D. Regalia is approved.

Dr. Keith C. Clarke

Dr. Werner Kuhn

Dr. Krzysztof Janowicz, Chair

March 2020

Acknowledgements

Reflecting on the events that led me to pursue a PhD, I am reminded of the people who granted me with incredible opportunities to advance my education and propel my academic career. There are those who inspired me to reach further, those who encouraged me to think critically, and those who enabled me to work creatively; for all of whom I have great respect. Specifically, I want thank Dr. Jatila van der Veen for her long commitment to scientific education outreach and her unwavering enthusiasm to engage young students in research. I am grateful for the years of support and approval that Dr. Phil Lubin showed me throughout my time at UCSB both as an undergraduate and as a graduate student. I thank Dr. Keith Clarke for capturing my initial interest in Geography and Geographic Information Science, for giving me the opportunities to engage in research as an undergraduate, and for guiding me to ultimately pursue graduate school. I give special thanks to Dr. Grant McKenzie, who has been nothing but kind and supportive towards me both as a colleague and as a friend. Thank you to my colleagues: Gengchen, Rui, Lisa, Bo, Yingjie, Song, Yiting, Li, Ben, Jay, Ruben, and Ling. Thank you to my committee members: Dr. Werner Kuhn, Dr. Keith Clarke, and Dr. Krzysztof Janowicz. I am profoundly grateful for my advisor and mentor, Jano, who not only convinced me to pursue a PhD instead of leaving with a Master's degree, but has also demonstrated time and time again his commitment to his students and protecting their best interests. Jano has a strong sense of morals that have inspired me to care about issues I was previously ignorant about. Above all else, Jano supports freedom of thought and shows tremendous respect for his students' opinions. Thank you to my parents, to my sisters, and to my friends, for their unconditional love and support.

Curriculum Vitæ

Blake D. Regalia

Education

2020 Ph.D., Geography, University of California, Santa Barbara

2013 B.S., Computer Science, University of California, Santa Barbara

Publications

2019 **Regalia, B.**, Janowicz, K., and McKenzie, G. – Computing and Querying Strict, Approximate, and Metrically-Refined Topological Relations in Linked Geographic Data. *Transactions in GIS. (2019)*

2018 Janowicz, K., **Regalia, B.**, Hitzler, P., Mai, G., Delbecque, S., Frohlich. M., Martinent, P., and Lazarus, T. – On the Prospects of Blockchain and Distributed Ledger Technologies for Open Science and Academic Publishing. *In: Semantic Web Journal, Volume 9, Number 5 / 2018*

2018 **Regalia, B.**, Janowicz, K., Mai, G., Varanka D., and Usery E. L. – GNIS-LD: Serving and Visualizing the Geographic Names Information System As Linked Data. *At: ESWC 2018*

2018 Janowicz, K., Yan, B., **Regalia, B.**, Zhu, R., and Mai, G. – Debiasing Knowledge Graphs: Why Female Presidents are not like Female Popes. *At: ISWC 2018*

2018 Mai, G., Janowicz, K., Hu, Y., Gao, S., Zhu, R., Yan, B., McKenzie, G., Uppal, A., and **Regalia, B.** – Collections of Points of Interest: How to Name Them and Why it Matters. Workshop on Spatial Big Data and Machine Learning in GIScience. *Workshop on Spatial Big Data and Machine Learning@ GIScience 2018*

2017 **Regalia, B.** – Decentralizing the Persistence and Querying of RDF Datasets Through Browser-Based Technologies. *In: DeSemWeb@ ISWC 2017*

2017 **Regalia, B.**, Janowicz, K., and Mai, G. – Phuzzy.link: A SPARQL-Powered Client-Sided Extensible Semantic Web Browser. *In: VOILA@ ISWC 2017*

2017 **Regalia, B.**, Janowicz, K., and McKenzie, G. – Revisiting the Representation of and Need for Raw Geometries on the Linked Data Web. *In: LDOW@ WWW (2017)*

2016 Janowicz, K., Hu, Y., McKenzie, G., Gao, S., **Regalia, B.**, Mai, G., Zhu, R., Adams, B., and Taylor, K. – Moon Landing or Safari? A Study of Systematic Errors and their Causes in Geographic Linked Data. *In: R. Sieber, S. Bell (eds.) Ninth International Conference on Geographic Information Science (2016)*

2016 **Regalia, B.**, McKenzie, G., Gao, S., and Janowicz, K. – Crowdsensing Smart Ambient Environments and Services. *In: J. P. Wilson, Q. Zhou & A. Singleton (eds.) Transactions in GIS. (2016)*

2016 **Regalia, B.** and Janowicz, K. – Computing Linked Data On-demand Using the VOLT Proxy. *In: M. D'Aquin, E. Blomqvist (eds.) ESWC 2016. Springer LNCS (2016)*

2016 **Regalia, B.**, Janowicz, K., and Gao, S. – VOLT: A Provenance-Producing, Transparent SPARQL Proxy for the On-Demand Computation of Linked Data and its Application to Spatiotemporally Dependent Data. *In: M. D'Aquin, E. Blomqvist (eds.) ESWC 2016. Springer LNCS (2016)*

Research Experience

2018-2020 Graduate Student Intern at Jet Propulsion Laboratory, *California Institute of Technology*. Worked with CAE SSWE Team leveraging Semantic Web technology for the Model Management System (MMS) as part of the collaborative Open Model Based Engineering Environment.

2015 Graduate Student Intern at Jet Propulsion Laboratory, *California Institute of Technology*. Developed an experimental engineering tool intended for systems design using RDF (graph databases). Included work on a framework for the publication and querying of Commercial Off-The-Shelf (COTS) component meta-data for Cubesat developers.

2014-2020 Graduate Student Researcher at STKO Lab, *University of California, Santa Barbara*. Employed through a series of research grant funds (including by both industry and government) to develop projects rooted in the Semantic Web or Geospatial Linked Data. Employed as a Graduate Student Researcher in the Geography Department at University of California, Santa Barbara and as a member of the STKO Lab.

2011-2014 Researcher at Deepspace Cosmology Group, *University of California, Santa Barbara*. Employed by director of Deepspace group, Dr. Phil Lubin, to develop research experiment software including: an N-body particle simulator, CMB panorama viewer, and a kiosk interface for presenting research papers and animations.

2012-2013 Research Software Developer at Geography Department, *University of California, Santa Barbara*. Designed and developed the web application for an online Geography course pilot at UCSB that is still in use today (now offered to all UC campuses). Project was funded by research grants under Professor of Geography, Dr. Keith Clarke.

2011-2013 Research Software Developer at Geography Department, *University of California, Santa Barbara*. Worked as a volunteer on an experimental Interactive Campus Map.

2012 Research Software Developer at Geography Department, *University of California, Santa Barbara*. Employed as an undergraduate researcher to develop an Android application for surveying travel data with the GeoTrans Lab.

Professional Experience

2012-2014 Software Developer at Novacoast, *Santa Barbara*. Contracted by several companies to develop applications for internal or public-facing use. My duties as a consultant included meeting prospective clients, bidding on RFPs, developing full-stack applications, and providing IT support. Most notable projects included a Content Management System for an air quality management district, a secure cross-organizational file-sharing system for an international tech corporation, and a production QA tracking software for a prominent thermal imaging infrared company.

Awards and Fellowships

2019 The ESIP Community Fellowship Program - Semantic Technologies Committee, Earth Science Information Partners **(ESIP)**.

2016 The Jack & Laura Dangermond Graduate Fellowship - Excellence in Research Award, Department of Geography at **University of California**, Santa Barbara.

2016-2019 The Jack & Laura Dangermond Travel Grant, Department of Geography at **University of California**, Santa Barbara.

Services

Program committee member for conferences and workshops:

- International Semantic Web Conference (ISWC), 2020
- Extended Semantic Web Conference (ESWC), 2020
- International Semantic Web Conference (ISWC), 2019
- Extended Semantic Web Conference (ESWC), 2019
- Linked Data on the Web (LDOW), 2019
- The 9th International Conference on Knowledge Capture, 2017
- Linked Data on the Web (LDOW), 2018

Organization roles:

- **Metadata Chair**. Extended Semantic Web Conference (ESWC), 2019
- **Chair**. Distributed Ledger Technologies for Geographic Information (DLT-Geo), 2018

Abstract

Computational Time and Space Tradeoffs in Geo Knowledge Graphs

by

Blake D. Regalia

Over the past several years, the Web of Linked Data has continued to grow in size, both in terms of the breadth of domains covered as well as the depth and precision of knowledge. As a consequence to this growth, the community has been led to confront challenges that arise from incorporating large-scale geographic information into knowledge graphs. These challenges include data quality, data storage, data transmission, and the scaling of geospatial query processing. A crucial concern in real-time computing is about striking a balance between the *time* complexity of algorithms and memory consumption or data storage (i.e., *space*). Given a computational problem and the domain of its inputs, there are several decisions that researchers, engineers, and practitioners must make based on the constraints of available computational resources, as well as the desired program's 'reaction' time for the sake of human-computer interaction. Understanding how to strike such a balance requires a thorough understanding of the data structures and algorithms used to solve a problem. Geospatial data and geospatial queries in particular require innovators to possess deep background knowledge in order to research and develop viable solutions. As a geographic information scientist work-

ing with Linked Data, I attempt to improve the quality, accessibility, reliability, and query performance of geographic data in knowledge graphs. In this dissertation, I study three specific trade-offs: (i) whether certain geographic properties and relations should be computed on-demand or materialized beforehand; (ii) whether carefully precomputing topological relations is more useful than providing users with geometries to compute topological relations on-demand; and finally, (iii) whether the challenges of hosting public geographic knowledge graph services on the Web can be mitigated, and at what cost, by a peer-to-peer architecture in which the clients possess more intelligence.

Contents

Chapter 1

Introduction

This introductory chapter begins with a subsection on the background and motivation for the research presented in the dissertation. The background includes an overview of the Semantic Web and the potential it has to improve aspects of collaborative geographic information on the Web. The chapter then provides an outline of three research questions that will explore the computational time and space trade-offs involved when storing, transmitting, and querying geographic knowledge graphs. More broadly, the dissertation is a study of the computational costs and benefits associated with the organization and retrieval and geographic information in knowledge graphs. Following the chapters, I review the experiments that were conducted and the research contributions therein. I discuss how the results answer the three proposed questions and conclude on the lessons learned.

1.1 Background and Motivation

In 2001, Berners-Lee et al. [9] and contributing members of the World Wide Web Consortium (W3C) envisioned a framework that would facilitate the sharing and reuse of machine-readable data on the World Wide Web. Berners-Lee stated that "Like the Internet, the Semantic Web will be as decentralized as possible" [9]. Initially promoted as an extension to the 'Document Web' and bolstered by the same intent on openness that ushered the Web to mass adoption, the *decentralize everything* philosophy has largely guided the development and evolution of the Semantic Web and its technologies. Allemang and Hendler later described one of the principles of this framework as allowing "Anyone to say Anything about Any topic"[4]. This slogan neatly conveys both political and logical aspects of the Semantic Web. Politically, it expresses how the inherent decentralization of knowledge on the Web empowers freedom of thought and freedom of speech. Beyond local government censorship barriers in certain parts of the world, there is no global arbiter to the aggregate wealth of human knowledge shared across the World Wide Web. Logically, the slogan expresses how the Semantic Web is built to operate under the open-world assumption which asserts that if a statement is not explicitly stated by one party, they cannot reason that the given statement would be false, e.g., the statement might not yet exist or may exist already but persists outside their knowledge base. Another key component to the Semantic Web framework implicitly captured by the AAA slogan is that it allows for conflicting statements that stem from different perspectives and ontological interpretations of the world to coexist without

breaking the interconnected system of knowledge. This flexibility is especially useful for understanding the geographic world [71, 103, 97], where the categorization of an event or feature is done to serve a certain purpose such as for navigation or ecology, or even to function within a certain context such as social or political [57]. A simple example of this phenomenon can be seen in statements that locate people, places, or events within disputed territories. Russia asserts that it annexed Crimea in 2014, however this statement is contradicted by the United Nations who maintains that Crimea is part of Ukraine. While most knowledge bases require a consistent axiomatic system and thus forbids such contradictions, the Semantic Web avoids the need for global consensus as each statement is intrinsically framed with the additional context of *who* made the assertion. However, the framing actually goes beyond the *who* as any statement is also contextualized based on *where* and *when*. The validity and interpretation of statements can vary depending on temporal and spatial context, or as Janowicz puts it, "Anyone can say Anything about Any topic at Any time and Anywhere"[57].

As a complement to the existing document-oriented Web, the Semantic Web set forth a mission to integrate structured data from a variety of heterogeneous sources in an open and global web of interlinked machine-readable documents. Today, the cumulation of machine-readable knowledge on the Semantic Web is simply referred to as Linked Data, and the collection of those sources contributing open datasets make up the Linked Open Data cloud (LOD). Linked Open GeoData refers to the heterogeneous, decentralized cluster of machine-readable gazetteers, geographic ontologies, and spatially referenced

data on the Semantic Web. In addition to improving geographic information retrieval, data discovery and multimedia geo-enrichment, the linking of geographic concepts and identifiers plays a central role in connecting non-spatial datasets across domains. In fact, several of the most interlinked hubs on the Linked Open Data cloud are gazetteers, geographic thesauri, and collections of volunteered geographic information including the locations and descriptions of places and events.

10 years after Vice President Al Gore's 1998 vision of a Digital Earth [53], a joint reflection of geographers and environmental scientists concluded that the next generation Digital Earth would need to consist of "multiple connected globes/infrastructures addressing the needs of different audiences: citizens, communities, policymakers, scientists, educationalists"[19], which encouraged different 'views' of geographic observations. Indeed, one of the largest obstacles in the way of creating a unified digital representation of the world geologically, socially, ecologically, environmentally, and above all else geographically was how to reconcile the challenges of integrating knowledge across disciplines, i.e., the challenges on semantic interoperability [34, 10, 38, 44, 49, 62, 96]. These issues were even more pronounced in the age of big data and volunteered geographic information [45]. In 2002, Egenhofer predicted that the advent of the Semantic Web would enhance geographic information retrieval and provide a better means for data integration across multiple sources of geographic information on the Web [25]. Egenhofer invoked the call for a *Semantic Geospatial Web* which sought the curation of "multiple spatial and terminological ontologies, each with a formal semantics; the representation of those

semantics such that they are available both to machines for processing and to people for understanding" [25].

To demonstrate the need for semantic interoperability on the Geospatial Web, let us examine a hypothetical scenario using OpenStreetMap (OSM), a free and open, community-driven project that collaboratively produces a constantly updated vector-based street map of the world. While the OSM project openly welcomes edits from any contributor, those edits go through a rigorous quality control process before they are accepted into the map product. The scope, context, and type of information the project seeks to collect are for enhancing a thematic road map of the world. OSM therefore operates with a certain, albeit somewhat implicit, ontological perspective of *place* and geography more broadly. This is typically true for any centralized repository of crowd-sourced geographic information due to the inherent lack of ontological uniformity between observers and thus the need for a rigorous, structured data collection schema. Naturally, research communities are incentivized to take advantage of large, open geospatial data repositories by augmenting, reusing, or linking to their resources, but do so without the need nor intent to reintegrate their derivate work back into the original source. In many circumstances, data reintegration is even impossible if the modifier operates under a different ontological perspective which would violate the constraints of the original conceptual model. However, branching datasets can easily retain *semantic interoperability*, for example by using Semantic Web technologies. At least two such works, LinkedGeoData [6] and OSM Semantic Network [78, 7], set out to realize this very scenario by

creating Linked Data versions of OSM. The datasets have since garnered links from more than 20 major linked datasets spanning across government, historical, publishing, media, and social networking domains[1]. The lesson here is that the ability to store and reason on interlinked knowledge from different disciplines, each with a unique ontological perspective of the world, is perhaps one of the strongest cases for why the Semantic Web, and Linked Open Data more concretely, may become enrooted as the predominant collaborative method for sharing, reusing, and searching open geographic knowledge.

The Semantic Web Layer Cake is a conceptual architecture for the technologies that create the Semantic Web. It is represented as a vertical stack of technologies where the lower layers provide functionality for the layers above them. Towards the bottom of the stack, supporting all the technologies above it, the lowest layer specifically designed and standardized for the Semantic Web is called the Resource Description Framework (RDF). The RDF data model[2], describes how machine-readable information should be structured when publishing data to the Semantic Web. It describes itself as a "graph-based" data model, not to be confused with GIS network models. The structure of an RDF dataset is a set of *triples* by which two resources (a *subject* and *object*) are connected by a labeled, directed relationship (a *predicate*). RDF also allows for arbitrarily datatyped values to appear in the object position of the triple. So-called *literals* have two components, a string of data that encodes some value, and a datatype identifier. From a query perspective, the subject and object of a triple (and in certain cases, the datatype identifier of a

[1] https://www.lod-cloud.net/#diagram
[2] https://www.w3.org/TR/rdf11-concepts/

literal) behave as nodes, sometimes coinciding with other nodes from different triples in the dataset to form a traversable collection of links that can be queried jointly as a directed, edge-labeled multigraph. Another graph-based data model that has seen tremendous growth in recent years is the Property Graph (PG), which is incongruent with the RDF data model [48]. However, these graph-shaped data structures offer similar query capabilities and advantages over the relational model used by many traditional database systems. Not only do graph databases fall within the highly flexible 'schema-less' subcategory of 'NoSQL', i.e., non-relational, database systems popularized by the influx of big data, but they are also particularly well suited for arbitrary path queries with many joins, link traversal, inferencing, reasoning and knowledge organization. While the modern interpretation and usage of the term may vary depending on the community, *knowledge graphs* typically refer to datasets using either RDF or PGs.

Real-time computing is the expectation that applications will react to user input with 'real-time' constraints. These constraints are generally understood as being on the order of milliseconds, i.e., reacting to user input events and then responding with feedback or partial results within one second. The relationship between the time it costs to carry out some computational task and the amount of memory or storage required to sustain its data is referred to here as a *computational time and space trade-off*. Knowledge graph technology has notoriously struggled to keep up with providing real-time results in part due to the unpredictable query complexity afforded by query languages and the difficulty surrounding query optimization. Oftentimes, knowledge graph databases will target a

certain usage pattern, maintain database statistics, or sacrifice certain functionality in favor of better performance. These decisions are examples of computational time and space trade-offs. Finding the ideal balance between time and space however depends on many factors that include the shape of data and the types of applications that will eventually query them.

In the wild, the Web of Linked Data is currently facing challenges associated with sustaining and utilizing geographic linked data at Web scale, most notably: geographic data quality issues and geospatial query processing performance [58, 60, 109, 88]. It is primarily due to the demands of geographic linked data applications, e.g., real-time interactive maps, that these issues have received widespread attention. In order to best meet the demands of real-time systems, one must explore and understand the computational time and space trade-offs available within the geographic knowledge graphs that power these applications. The sorts of trade-off decisions that arise from studying geographic linked data require experts who have a deep understanding of spatial data science as well as the underlying technologies.

The broader impact of this research is about supporting observational science by improving the retrieval of geographic information so that researchers will be able to make better use of geographic knowledge. The goal is to broaden accessibility to open data, strengthen the reliability of question-answering systems, and improve data quality for geographic information and its services on the Web. I believe that geographers will benefit from Linked Data in pursuit of complex scientific problems by querying structured

cross-domain knowledge. In addition, I believe that Linked Data will benefit from geographers who possess the background knowledge needed to develop methods for modeling, generating, storing, transmitting and querying geospatial and temporal data.

1.2 Research Questions

1.2.1 Computing Geographic Properties and Relations On-Demand

A massive amount of knowledge that makes up today's LOD cloud is a result of semi-automatic extraction frameworks that semantically lift data collected by machines that crawl the contents of Web documents. Take DBpedia [12] for instance, the LOD cloud's most central hub for interconnecting resources. DBpedia extracts knowledge from the unstructured or semi-structured texts of articles from Wikipedia, the world's largest encyclopedia. The resulting knowledge graph is the essential core of geographic linked data as it describes more than 4.5 million things including more than 735,000 places in over 120 languages[3]. In fact, a recent study found that DBpedia contains more than 920,000 features with direct spatial footprints with nearly a third of its 1.4 million persons, and nearly 15% of events, having direct links to spatial footprints [58].

Within these derived knowledge graphs however, we often find variety in the robustness of statements. For example, a statement about the population density of a city may

[3] https://wiki.dbpedia.org/about/facts-figures

in fact be derived from two adjacent statements: (1) the city's population count, and (2) the area of the city's spatial extent. Consequently, the statement about population density can be said to be *dependent* since it relies on the presence of other statements. However, this dependency relation is not modeled or expressed in the ontology nor the data itself. Aside from lacking provenance information, dependent statements also raise concerns over reliability and data quality, as well as what should be considered redundant when publishing generated datasets.

The previously mentioned study [58] also identified a set of the most common systematic errors facing geographic linked data including those caused by triplification and extraction. Known artifacts to the extraction process for geographic information include data sparsity and poor data quality. For example, articles about populated places often include references to other 'nearby' places by using cardinal direction relations such as, north, northeast, east, southeast, and so on. However, the presence of such statements is entirely arbitrary as it is primarily driven by the topic of the article and the social or political relevance of relating one place to another. Furthermore, the statements are not always accurate nor consistent as they are a reflection of the human perception to the directional relationship between places, such as *Ventura is north of U.S. 101.* A more accurate assessment would be, *U.S. 101 runs through the southern quarter of Ventura.* Not surprisingly, such issues can easily be avoided by using the spatial footprints of both places to accurately compute such relations. This not only provides better accuracy and better consistency, but also substantially improves the completeness of knowledge graphs

and, by extension, their spatial coverage.

However, the approach to use spatial footprints to derive statements raises questions about what should be precomputed and materialized in a knowledge graph versus what should be computed in response to queries, i.e., *on-demand*. One idea could be to ensure that properties which change frequently over space and time are kept in sync with their dependencies. For example, Wikipedia commonly includes census data about places such as the population count and population density. However, population count and a city's spatial extent can vary significantly over time. Statements that depend on population count or spatial extent can therefore be said to be *ephemeral*, implying that they are only assumed true for a limited duration. More generally, we can say data that change frequently over space and/or time are *volatile*. Rather than extracting such information from unstructured texts, one could compute them instead.

No matter which specific properties are selected, the decisions about what to compute as well as how and when to compute them, will inevitably involve trade-offs that first need to be understood.

- **RQ1:** *Which properties of geographic entities and relations between them should be computed on-demand and which should be materialized in the knowledge graph beforehand? Is there a measurable trade-off between both approaches that can be used to guide data providers?*

In order to bring on-demand computation of geodata into knowledge graphs, we must identify a mechanism that allows us to define how to derive dependent data. In

triplestores, built-in support for reasoning allows graph queries to trigger rules defined in an ontology. However, the types of calculations I will perform for these experiments are either not covered by existing ontologies and the reasoners that implement them, or they rely on auxiliary system features such as a persistent geospatial index or frequent network requests. Instead, one can place a transparent SPARQL proxy in between the client and triplestore. The additional layer would allow users to issue SPARQL-compliant queries that appear to retrieve properties that already exist, but effectively act as triggers for rules that compute those properties on-demand.

1.2.2 Precomputing Topological Relations

On the other side of on-demand computation, there is the notion that certain relations will benefit from *precomputation*. Naturally, opportunities and drawbacks exist on the opposite end of the computational spectrum, i.e., the trade-offs of *precomputing* geographic properties. As hypothesized in the previous research question, certain properties will lend themselves better to on-demand computation over comprehensive materialization. Conversely, certain geographic properties will lend themselves better to comprehensive materialization over on-demand computation.

The majority of geographic identifiers on the LOD cloud are represented geometrically as point coordinates which severely limits their potential for spatial analysis. While several efforts have focused on integrating more complex geometries into the LOD cloud, these geometries will be of limited use to linked geospatial data for several reasons.

Firstly, high-resolution geometries incur expensive computational costs to store, transmit, and process. Graph queries that embed spatial operations or spatial filters will not scale well over large, complex geographic datasets such as the USGS Digital Line Graph.

Secondly, geometries are merely a means to an end for spatial analysis and spatial reasoning. The proper geometric representation of real-world geographic entities depends on place type, map scale, and context. Likewise, geometry *informs* topology and not the other way around. However, state of the art solutions such as GeoSPARQL compute topology on-demand and without context such as place type. This leads to ignorance about vagueness and uncertainty principles from Geographic Information Science.

Finally, computing topology from geometries requires pre-processing steps anyway in order to clean them for errors introduced by slight discrepancies in cooordinate precision, survey methods, reference systems, or digitization errors. On the Web of Linked Data, the federation across multiple sources virtually guarantees that geometries will *never* align perfectly enough to be useful for any type of topological queries aside from point-in-polygon tests.

Instead, heterogeneous geographic knowledge graphs that reference high-resolution geometries can be enhanced by mining approximate topological relations, i.e., by pre-computing them, while preserving the provenance of their geometries as well as the parameters used for each spatial operation.

Given that topological relations are expensive to compute for complex features, such as polylines and polygons, deferring them to be calculated at query-time imposes a con-

siderable limitation on the spatial query potential of the knowledge graph.

- **RQ2:** *Rather than expecting users of Linked Data to manually clean the raw, high-resolution geometries from heterogeneous sources in order to perform spatial operations during their query, what can be gained from and what are the challenges of precomputing strict, approximate and metrically-refined topological relations?*

In Geographic Information Systems (GIS), calculating topological relations typically requires a step of cleaning geometries from digitization errors. On the Linked Data Web however, the sharing and reuse of existing resources, along with their provenance, is critical to establishing trust and confidence in data quality. This can easily be overlooked when the primary objective is to provide a previously unavailable geospatial query function to a Linked Data client. A simple remedy to preserve the provenance of a heterogeneous geographic knowledge graph, and therefore maintain users' confidence in the quality of published geodata, is to make those spatial operations transparent.

For instance, a researcher who is seeking to extract topological relations by combining a dataset of water bodies with a transportation network asks for cases where rivers meet roads using the *touches* relation. The researcher instructs their GIS to find approximate topological relations using fuzzy topological space, i.e., by considering the features to have broad boundaries. However, this approach involves the tuning of parameters that affect whether or not a given approximate topological relation is materialized. Performing such operations 'offline' to then only publish the resulting topology, or even just the cleaned geometry as some Linked Data services do, diminishes the reproducibility of

results. Instead, each statement that encodes a topological relation should also include provenance information such as the geometries and parameters used to determine that a given relation holds between two features, providing transparency to end users about how the computation was made.

In order to support spatial reasoning, it is beneficial for geographic knowledge graphs to contain explicit statements about features that are topologically disjoint. However, materializing this relation for all pairwise combinations of features leads to a combinatorial explosion, making it infeasible for large datasets. If the knowledge graph operates under the closed-world assumption however, the system can infer that the omission of all topological relations implies that two features are disjoint. In other words, for any pair of geographic features in our knowledge graph, if a topological relation is not explicitly stated between them, then one can conclude the two features are topologically disjoint. This single axiom spares us from having to materialize a relation for every pairwise combination of features. The remaining pairs include features that touch, overlap, intersect, and so forth.

1.2.3 Storing, Transmitting and Querying Geographic Knowledge Graphs

In the previous two research questions, I explored the trade-offs of computing geographic properties on-demand and the trade-offs of precomputing topological relations. For my third research question, I will explore the computational time and space trade-offs of

storing, transmitting and querying data within geographic knowledge graphs at the system architecture level by taking a closer look at the roles of clients, servers and peers on the Web of Linked Data.

The Web of Linked Data was originally intended to function as a distributed system of intelligent agents crawling the Web and consuming machine-readable data from any number of hosts until accumulating enough relevant information to carry out their own query. On the modern Linked Open Data cloud, hosts have taken over much of the work involved with querying by exposing SPARQL endpoints that allow clients to submit queries for server-side processing. This change in workload also shifted more responsibility onto the server and greatly reduced the barrier to entry for end-users by virtue of simplifying the clients' duties and thereby accelerated the uptake of Semantic Web technologies.

More recently however, accessibility to SPARQL endpoints and query execution times have suffered due to factors of limited resources, limited budgets, a lack of incentives for service providers, large datasets, complex queries, and an excess of concurrent clients [54]. In response to these issues, researchers have encouraged the use of more intelligent clients [105, 107] and vowed to balance query processing between the server and client. While these recent approaches have improved accessibility of endpoints, they do so at the expense of much longer query times and heavier network transactions.

Efforts to create binary serializations of RDF by using *succinct data structures* [32] have revealed interesting use cases for the transmission and querying of datasets. Apply-

ing these data structures on the application front of Web browsers could prove beneficial for the intelligent client model without sacrificing query performance.

With the ongoing advances in Web technologies and Web APIs, a substantial portion of the software stack can now reside in the Web browser. For example, persistent storage, multithreading, network requests, and peer-to-peer communication have all been abstracted by Web standards for the sake of Web applications. The exposed APIs, together with additional third-party libraries that abstract them, let developers focus more heavily on the domain logic of their application rather than the low-level details involved with storage, parallelization and network communication. On the whole, Web applications greatly reduce the friction for developers to target cross-platform compatibility and perform software deployment activities such as release, installation and update, and are gradually superseding the tasks traditionally done exclusively by native applications. While some libraries have been developed to handle RDF storage and querying within the browser [68, 107], none have attempted to operate on binary representations of RDF and thus have been limited to small-scale datasets.

- **RQ3:** *Rather than using a client-server paradigm to answer geographic Linked Data queries, what can be improved by and what are the penalties of enabling clients to directly query compressed, indexed geographic knowledge graphs over peer-to-peer networks through the Web browser?*

In order assess the viability of an intelligent peer in practice, I will conduct an experiment by implementing a query engine in the Web browser that operates on fragments

of a remote, compact binary dataset encoded using efficient data structures with proven benefits from existing literature [32]. I will focus specifically on enhancing the retrieval of Linked Open GeoData by studying the entire architecture of query endpoints including databases, indexing data structures, binary encoding of knowledge graphs, network transactions, system workloads, and spatial graph query optimization. The study will evaluate the storage requirement and query performance of a proof of concept implementation.

1.3 Structure of the Dissertation

This section briefly describes the structure of what follows in my dissertation. Each of my three research questions will be discussed, explored and concluded within their own chapters. This will include a detailed explanation of the problem(s) each question aims to address, related work, an hypothesis, experiments, the results, and a conclusion. Following the chapters, I review the experiments that were conducted and the research contributions therein. I discuss how the results answer the three proposed questions, conclude on the lessons learned, and outline future work.

Chapter 2

Computing the Properties of, and Relations Among, Geographic Features in Knowledge Graphs On-Demand

Peer Reviewed Publication	
Title	VOLT: A Provenance-Producing, Transparent SPARQL Proxy for the On-Demand Computation of Linked Data and its Application to Spatiotemporally Dependent Data
Authors	Blake D. Regalia, Krzysztof Janowicz, Song Gao
Institutions	STKO Lab, Department of Geography, University of California, Santa Barbara, USA
Venue	Proceedings of 'The Semantic Web. Latest Advances and New Domains' at the 13th Extended Semantic Web Conference, 2016 (ESWC 2016)
Location	Anissaras, Crete, Greece
Editors	Harald Sack, Eva Blomqvist, Mathieu d'Aquin, Chiara Ghidini, Simone Paolo Ponzetto, Christoph Lange
Submit Date	December 19, 2015
Accepted Date	March 15, 2016
Presentation Date	June 2, 2016
Copyright	Blake D. Regalia, Krzysztof Janowicz, Song Gao

2.0.1 Abstract

Powered by Semantic Web technologies, the Linked Data paradigm aims at weaving a globally interconnected graph of raw data that transforms the ways we publish, retrieve, share, reuse, and integrate data from a variety of distributed and heterogeneous sources. In practice, however, this vision faces substantial challenges with respect to data quality, coverage, and longevity, the amount of background knowledge required to query distant data, the reproducibility of query results and their derived (scientific) findings, and the lack of computational capabilities required for many tasks. One key issue underlying these challenges is the trade-off between storing data and computing them. Intuitively, data that is derived from already stored data changes frequently in space and time, or is the result of some workflow or procedure, should be computed. However, this functionality is not readily available on the Linked Data cloud with its current technology stack. In this work, we introduce a proxy that can transparently run on top of arbitrary SPARQL endpoints to enable the on-demand computation of Linked Data together with the provenance information required to understand how they were derived. While our work can be generalized to multiple domains, we focus on two geographic use cases to showcase the proxy's capabilities.

2.1 Introduction and Motivation

Linked Data described the paradigm for a Web of densely interconnected yet distributed data. It provided methods and tools that dramatically ease the publication, retrieval,

sharing, reuse, and integration of semantically rich data across heterogeneous sources. Over the last few years, we have witnessed a rapid increase in available data sources on the Linked Data cloud and a fast uptake of the involved technologies in academia, governments, and industry. Nonetheless, several key issues remain to be addressed in order to enable the full potential of Linked Data. One of these issues is the trade-off between storing data and computing them. To give a concrete example, if the population and area of a county are available, should the population density be stored as well or should it be computed on-demand as it depends on already stored properties? Storing such data is often problematic or even impossible for multiple reasons. Keeping the population density in sync with a changing population is just one example. Consequently, such statements should be computed. However, this functionality is not readily available on the Linked Data cloud and is not fully supported by existing query languages, endpoints, or APIs.

Recently, a variety of approaches [100, 1, 63, 98, 61] have been proposed to address this and related issues. Here, we argue why these approaches alone are not sufficient and propose a framework inspired by a combination of their findings. Essentially, we propose a proxy[1] that can transparently run on top of any SPARQL 1.1 compliant endpoint while providing a framework for the on-demand computation and caching of Linked Data. Going beyond existing work, our approach also provides the provenance information required to make sense of the (cached) results, thereby improving reproducibility. Essentially, all (derived) data together with the procedures used to compute them are stored as RDF in separate graphs.

[1] A working VOLT proxy prototype is available at: http://demo.volt-name.space/

In the following, and as space permits, we highlight key aspects of the VOLT[2] proxy and framework by example. Instead of focusing on technical (implementation) aspects alone, we showcase VOLT's capabilities by discussing two use cases in detail. These use cases also serve as the evaluation of our work, e.g., they demonstrate how to improve the data quality of DBpedia and reduce storage size at the same time. While our work can be generalized to multiple domains, both use cases focus on geo-data. We believe that the challenges introduced by spatiotemporal data are ideal for discussing the need for provenance information on the procedural (workflow) level, the difficulties resulting from keeping *dependent* data in sync, and the problem that allegedly *raw* data was created by using some latent assumptions that now hinder reproducibility and thus interoperability.

2.2 The VOLT Framework and Proxy

Work that aims at bringing API-like features to the Semantic Web typically does so by either suggesting ways to extend SPARQL or by providing additional functionality outside of the typical Semantic Web layer cake; see Section 2.4. Implementing such solutions often requires a custom SPARQL engine or the adoption of future W3C recommendations. Furthermore, running non-standard SPARQL engines threatens Linked Data interoperability and reusability of federated and non-federated queries alike. For these reasons, we often fail to see widespread use of experimental technologies. Finally, most of these technologies are not transparent, i.e., they require additional knowledge

[2]VOLT: **V**OLT **O**ntology and **L**inked data **T**echnology

or at least awareness by the end user. To overcome these issues, we strive to develop a transparent framework that embraces the existing technology stack without any changes to SPARQL. Our approach functions as a *transparent proxy* [59] to any existing SPARQL 1.1 engine and thereby acts as a legitimate endpoint. When a query is issued to the proxy, it triggers a series of interactions with the underlying, encapsulated SPARQL endpoint before forwarding the results back to the client. In other words, the client does not notice any difference to a regular endpoint.

In this section we introduce the general VOLT architecture, highlight important aspects such as transparency, and give an overview of the implementation.

2.2.1 SPARQL as an API

The idea of using triples within the basic graph pattern of a query to invoke computation is referred to by iSPARQL as *virtual triples*[63]. A virtual triple uses the predicate to identify a procedure and effectively treats each subject and object of the triple as an input or output to the procedure. Like virtual triples and the *magic properties*[3] of Apache's ARQ, we use the triple's predicate as a way to identify a user-defined procedure. However, we make a distinction between the various ways in which these special patterns are used in our framework:

Firstly, *computable properties* simply represent an existential relation between two named entities. For example, consider a computable property named `udf:intersects` that tests for the spatial intersection between two individuals. A client may trigger

[3] https://jena.apache.org/documentation/query/extension.html#property-functions

computation on the individuals `:A` and `:B` by issuing a SPARQL ASK query with the basic graph pattern `:A udf:intersects :B`. Alternatively, a client may find all things that intersect with `:A` via a SELECT query `:A udf:intersects ?other`, where the object of the previous triple has been replaced by the variable `?other`. Yet another style allows the client to test multiple computable properties on the same triple by using a variable in place of the predicate along with a triple that constrains the variable to a specific `rdf:type`. For instance, a client may discover all topological relations between two particular regions by issuing the SELECT query `:A ?relation :B. ?relation a udf:RegionConnectionRelation`. In this variation, the triple that constrains the variable `?relation` functions as a *computable property trigger*. It indicates the client's intention to invoke testing on an entire class of computable properties.

Secondly, *functional triples* act as interfaces for calling user-defined procedures with named inputs and outputs. To invoke a procedure, functional triples expect a primary *root triple* where the subject is anonymous and the object is a blank node. The blank node object acts as a hashmap for both the input arguments and output variables to the procedure. Whereas EVT[4] functions accept an ordered list of input arguments and return a single RDF term, functional triples accept an unordered set of named input arguments, are capable of returning multiple output bindings, and allow both inputs and outputs to be either RDF terms or RDF graphs. Once the functional triple call is executed, the entire graph constructed within the blank node is saved (either temporarily or persistently) to a graph in the triplestore. Doing so enables auxiliary pattern groups

[4]Extensible Value Testing

within the same query to work as if the entire functional triple's blank node was matched to an existing set of triples. The subject of a root triple must be an unbounded variable or a top-level blank node (that is, anonymous) in the query as the entire functional triple will be materialized and the subject will become a URI suffixed by a UUID[5]. A functional triple example will be shown in Listing 8.

Thirdly, *pattern rewriters* perform special expansions to the SPARQL query at run-time for patterns that may be otherwise impossible to write in a single query, such as subqueries that construct RDF graphs. A pattern rewriter is invoked by a functional triple which identifies the rewriter's procedure along with its input arguments. A group of query patterns gets associated to the rewriter by exploiting the GRAPH keyword in SPARQL. Consider an example where we want to select only the first valid object matched by a list of acceptable predicates that are semantically equivalent (in a certain context). Say we want to count the sum of populations given by the DBpedia `dbp:population` predicate for some distinct places. Since we do not want to count the same subject twice, we only want to match a single value for each subject. If a subject does not have a valid numeric literal belonging to the primary predicate `dbp:population`, then we opt for a secondary predicate, `dbp:populationTotal`. One can perform this in a regular SPARQL query as depicted in Listing 1.

```
select (sum(?population) as ?totalPopulation) where {
    {   ?s dbp:population ?population .
        filter(isNumeric(?population))
    } union {
        ?s dbp:populationTotal ?population .
```

[5]Universally Unique Identifier

```
        filter(isNumeric(?population))
        filter not exists {
            ?s dbp:population ?primary_population .
            filter(isNumeric(?primary_population))}}}
```

Listing 1: Select the sum of population counts using a preferred order of predicates in a query to a regular SPARQL endpoint.

As the number of predicates to test for increases, so does the number of `FILTER NOT EXISTS` blocks in each new `UNION` group. Furthermore, if we wanted to use a list of predicate IRIs from an RDF collection found in a triplestore, then this selection would be impossible to perform in a single query. Employing a pattern rewriter, we can automate building such queries in addition to having their bindings projected onto the surrounding query level; see Listing 2.

```
select (sum(?population) as ?totalPopulation) where {
    ?matcher volt:firstMatch [
        input:onVariable "?p"^^volt:Variable ;
        input:useValuesFrom (dbp:population dbp:populationTotal) ;
        input:sampleFromVariables ("?population"^^volt:Variable)  ] .
    graph ?matcher {
        ?s ?p ?population .
        filter(isNumeric(?population))   }}
```

Listing 2: The `?matcher` variable can be thought of as binding to the URI of a named, transient graph. In reality, the pattern rewriter's procedure will transform the patterns within the `GRAPH` group into a new subquery. This code snippet along with the expanded query can be seen in its entirety at https://git.io/v2Nxb .

2.2.2 Transparency and Reproducibility

A key limitation of previous approaches has been with the client's inability to inspect the source code behind an API function. Functions are not always trivial and their algorithms may overlook cornercases or depend on undocumented assumptions – leading to a breakdown in *semantic interoperability*. Our approach is to make the source code for all procedures readily accessible to the client by storing everything in the triplestore as RDF. Each procedure is serialized according to the VOLT ontology[6] and stored in the *model graph*. In order to execute a procedure, the proxy downloads a segment of the model graph and evaluates each step from the procedure's sequence of instructions. A simple example of an instruction is the assignment of a variable to an expression, e.g., `?x = ?y + ?z`, which applies the addition operator to the values stored in the variables `?y` and `?z`, then puts the result in the locally-scoped variable `?x`. In the model graph, this expression is serialized as an abstract syntax tree; shown in Listing 3.

```
... [ a volt:Assignment ;
    volt:name "?x"^^volt:Variable ;
    volt:gets [ a volt:BinaryOperation ;
        volt:operator "+"^^volt:Operator ;
        volt:lhs "?y"^^volt:Variable ;
        volt:rhs "?z"^^volt:Variable ]] ...
```

Listing 3: Abstract syntax tree of an assignment instruction for a VOLT procedure.

In taking this approach, we are able to statically evaluate the validity of a procedure's RDF serialization by using an ontology. Another example of a procedural instruction

[6] https://github.com/blake-regalia/volt

might be a SPARQL query, which has the benefit of referential integrity in its serialized form. This implies that a client can discover a procedure that depends on a particular IRI by querying the model graph for that IRI in the object position of a triple. E.g., we can discover any procedures that depend on the `geo:geometry` predicate by using the query shown in Listing 4.

```
describe ?procedure from named volt:graphs where {
    graph volt:graphs { ?modelGraph a volt:ModelGraph }
    graph ?modelGraph {
        ?procedure rdf:type/rdfs:subClassOf volt:Procedure .
        ?procedure (!</>)+ geo:geometry .  }}
```

Listing 4: Discover any VOLT procedures that depend on the `geo:geometry` predicate by using the nexus property path `(!</>)+`.

Thus, VOLT is transparent in two ways: (1) the proxy sits on top of a regular endpoint without a client noticing any difference, i.e., computed Linked Data behave as if they were stored in the underlying triplestore [59], and (2) procedures (defined by users or providers) are open for inspection.

2.2.3 Provenance

During execution of a procedure, all SPARQL queries and function calls are analyzed and recorded. Any information used during the evaluation of these transactions gets serialized as RDF triples and stored into a *provenance graph*. Those details are used to associate a cached triple to the inputs and expected outputs of SPARQL queries and function calls which led to that result. This offers two advantages: (1) the provenance

of a cached triple is stored and remains available for inspection by which a client has the means to review source information that led a procedure to its conclusion and (2) it enables the invalidation of *stale cache*.

2.2.4 Caching and Cache Invalidation

To improve the performance of matching query patterns against computable properties and functional triples, we make use of caching. When caching is enabled by the proxy's host, each cacheable result is diverted to a persistent *output graph* instead of a temporary *results graph*. The input query is ultimately executed on the union of the source graph(s), results graph, and output graph, known collectively as the *content graph*. Determining whether or not a result should be cached depends on the ontological definition of the procedure that was used. Caching will only take place on a result when the procedure allows it. However, a client can bypass caching any results for the entire duration of a query's execution by including `optional {[] volt:ignoreCache true}` in the input query. Using the OPTIONAL keyword ensures that the query is reusable against arbitrary SPARQL engines, e.g., ones that do not run the proxy.

Each time a new triple is cached for a computable property, that triple runs the risk of being obsolete for future queries if the contents of its original source graph were to change. To detect this issue and protect against stale cache, we embed a cache invalidation feature within the framework. For procedures that use simple SPARQL queries, this may just involve confirming the existence of triples. In these cases, a cached result may be validated

by a single query directed at the actual SPARQL engine. However, procedures that use more complex queries can employ patterns such as property paths or aggregate functions which can only be validated by executing those queries in full. We realize the need for an ontology that enables serializing, with various levels of complexity, methods of result validation for outputs of function calls and SPARQL queries given their inputs.

2.2.5 VOLT Procedures

The VOLT framework supports several types of user-defined procedures; each type serves a different purpose. In the model graph, a user may define procedures for EVT functions, computable properties, functional triples, and pattern rewriters. For each of these mediums, there is an ontological class that defines how an associated procedure must be encoded as RDF in the user-defined model graph. For example, a VOLT EVT function must have at least one member of type `volt:ReturnStage` in the RDF collection object pointed to by the `volt:stages` predicate within the procedure's set of defining triples.

The user-defined model exists as an RDF graph which encodes each procedure definition as a sequence of instructions. These instructions are limited to the basics, such as: operational expressions, control flow, SPARQL queries, and so on. To provide developers with the full flexibility of a programming language, the user-defined model can be extended by scripting plugins. Plugins are ideal for handling tasks such as complex calculations, networking, and I/O. They are treated as namespaced modules. A single plugin may host an array of functions. For instance, we created a plugin that uses

```
prefix postgis: <http://stko.geog.ucsb.edu/volt-plugins/postgis/#>
select ?angle where {
        dbr:Santa_Barbara geo:geometry ?wktFrom .
        dbr:Ventura geo:geometry ?wktTo .
        bind( postgis:azimuth(?wktFrom, ?wktTo) as ?angle ) }
```

Listing 5: Calls the user-defined EVT function 'azimuth' in the PostGIS plugin.

PostGIS[7] to handle geographic calculations; see Listing 5 for a call to the EVT function `postgis:azimuth`.

Under the hood, each plugin registers a specific namespace with the proxy by inserting RDF statements about itself into the model graph. This information includes metadata such as the path of the binary to execute, the path or URL of the source code if available, the namespace IRI, and process-related configuration. Anytime an EVT function call has a registered namespace, it will trigger the corresponding plugin. If the plugin is not already running, the proxy will load it into memory by spawning a child process. Once a plugin is running, the proxy pipes the function name and input arguments, serialized as JSON over stdin, to that child process. A single process may be used to run multiple tasks in series and multiple process of a plugin may be spawned in order to run tasks in parallel. Idle and busy processes may be terminated at the discretion of the proxy.

2.2.6 Query Flow Overview

To give a brief overview of how the proxy works, we examine VOLT's computable property feature. In Case Study I, we will demonstrate the use of such a property to determine the cardinal direction between Santa Barbara and Ventura. Figure 2.1 depicts the process of

[7] http://postgis.net/

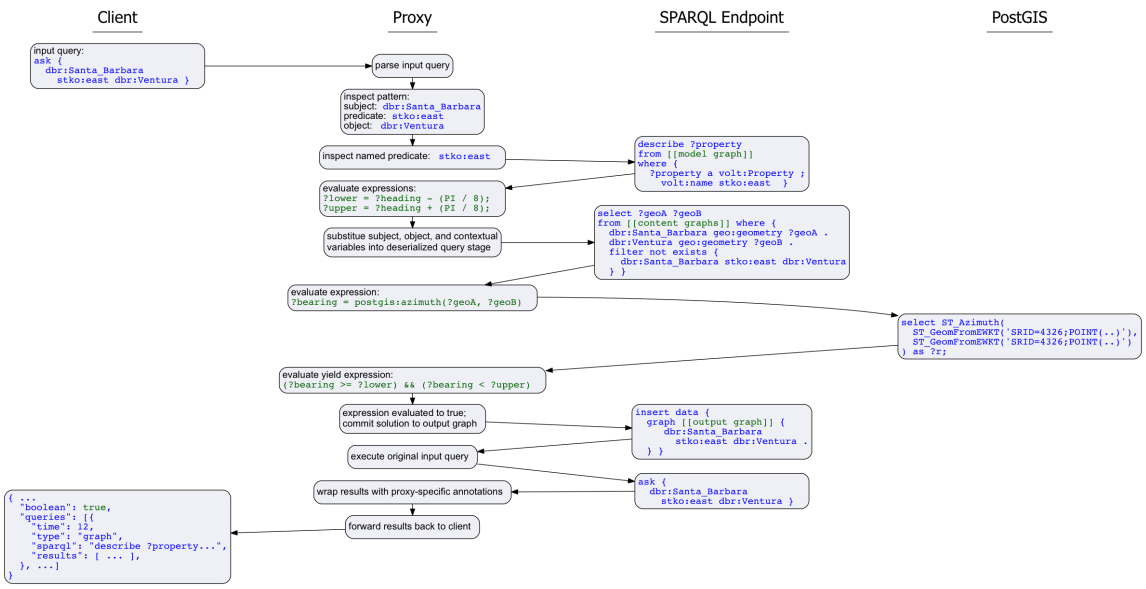

Figure 2.1: The execution of computable property `stko:east` represented by a flowchart.

executing the procedure for `stko:east` as a flowchart.

2.3 Case Studies

This section discusses two geographic use cases to showcase VOLT in action. Each use case highlights a different capability of the framework.

2.3.1 Case Study I: Cardinal Directions

The four cardinal directions North (N), South (S), East (E), and West (W) are among the most common means to express directional relations. The equal directional divisions of a compass rose are known as the four intercardinal directions, i.e., Northeast (NE), Southeast (SE), Southwest (SW) and Northwest (NW). In this section, *cardinal directions*

will refer to all eight directions. Figure 2.2 shows how the bearing span ω for a cardinal direction is represented. The directionality is determined by testing if the azimuth between the point geometries of two places falls within ω from the primary angle of the direction. For the 8 cardinal directions, ω is set to $\pi/8$. For example, SE (*stko:southeast* here) covers the range $5\pi/8$ to $7\pi/8$ which is measured from the positive y-axis.

According to a SPARQL query for all resources of type `dbo:Place` or having a `geo:geometry`, there are over 1 million places in DBpedia.[8] Nearly 35,000 of them are associated to at least one triple with a cardinal direction predicate, leading to a total of 108,818 distinct triples involved. While this number is large, it is only a small portion ($\approx$ 1.2%) of the potential amount of cardinal direction relations among all places if merely storing a single triple per direction, e.g., only storing the nearest place to the North, South, and so forth. Trying to store all cardinal directions between all places would lead to a combinatorial explosion.

The entities contained in these triples vary widely and include macro-scale types such as *Mountain Range* or *Country*, meso-scale types, such as *City* or *River*, and micro-scale place types such as *Hospital*. Interestingly, types such as *Person* also show up, likely confusing persons with the places where they were buried; e.g., `dbr:Saint Mechell is dbp:north of dbr:Tref Alaw`. Intuitively, and leaving cases such as headlands and meandering rivers aside, there should be only one cardinal direction relation between two places. Surprisingly, there are 3,411 places (involving $\approx$ 17,000 triples) with more than one cardinal direction to the same entity. For instance, Chicago, IL is both *dbp:east*

[8] All queries & experiments were performed on the stable DBpedia 2015-04 version.

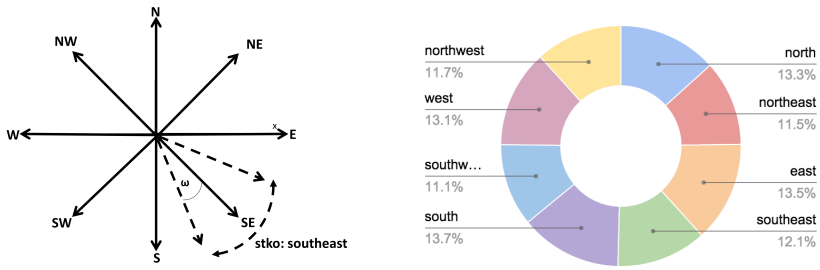

Figure 2.2: The eight primary/inter-cardinal directions and their range (left) and the proportional distribution of mismatched directions normalized by categorical count (right).

and *dbp:west* of *Lincolnwood, Rosemont, and Schiller Park*, which is controversial. Consequently, we are compelled to test the accuracy of cardinal directions in DBpedia.

In order to compute the cardinal direction accuracy between entities of type `dbo:Place` that have one or more `geo:geometry` property in DBpedia, we selected all combinations of geometries between two places[9]. Our selection yielded 136,964 results of which 91,890 matched correctly, leaving 45,084 rows (33%) marked as incorrect. To validate that our computational representation of the cardinal directions does not introduce bias, we show in Figure 2.2 that each of the eight cardinal directions have roughly equal portions of incorrect relations. If we consider all 133,941 cardinal direction triples in DBpedia, we find that 55,928 (42%) of them have a subject or object lacking `geo:geometry`, or are not of type `dbo:Place`. In fact, 17,957 triples have cardinal direction relations to RDF literals, 537 of which are of datatype `xsd:integer`. Most importantly, our argument is that given the few correct existing cardinal direction triples, a Linked Data user has to wonder why these specific relations are present in DBpedia and not a comprehensive set of cardinal

[9]Please note that some places have more than one geometry.

directions between all places. This, however, would far exceed the total number of triples in DBpedia today. The imbalanced cardinal direction distribution becomes immediately clear by inspecting Table 2.1.

Table 2.1: Cardinal direction accuracies of the top 20 places with the most relations.

Place	Matches	Total	Accuracy	Place	Matches	Total	Accuracy
Wrexham	47	71	0.66	Karimnagar	27	38	0.71
Dolgellau	49	58	0.84	Ranchi	27	38	0.80
Ruthin	27	57	0.47	Shrewsbury	34	35	0.97
Bradford	29	53	0.55	Brothertoft	30	34	0.88
Bala, Gwynedd	22	47	0.47	Burton-upon-Trent	34	34	1.0
Orlando, Florida	27	47	0.57	Boston, Lincolnshire	25	33	0.76
Lichfield	43	46	0.93	Kirkby	29	33	0.88
Corwen	26	43	0.60	Ford, Shropshire	17	31	0.55
Aberystwyth	31	43	0.72	Mansfield	26	31	0.84
Derby	22	42	0.52	Glensanda	24	30	0.80

Another challenging issue is the computation of cardinal directions between polygonal representations of places. It is straightforward to compute point-to-point cardinal direction results on-the-fly if centroids are taken as the representations of regions. Depending on the polygons and the representativeness of centroids, there may be a varying degree of uncertainty associated with a cardinal direction relation between two regions. For example, according to DBpedia, the city of *Ventura* is linked to the city of *Santa Barbara* via the `dbp:northwest` relation, i.e., Ventura should be located to the *southeast* of Santa Barbara. This may be true for a certain point-feature representation of the cities but is not correct for all points inside the city boundaries. In fact, by taking the OpenStreetMap polygons for Santa Barbara and Ventura and defining a regular point grid of 1x1 km, we can compute the probability of grid points contained in Ventura to locate in the *southeast* of Santa Barbara (grid points). We filter out those points which

are either outside of the city boundary or in the ocean. In total, we get 79 representative points for Santa Barbara and 88 points for Ventura. As depicted in Figure 2.3, we compute cardinal direction relations between 6,952 pairs of points in total. Our result shows that *southeast* is only the correct relation in 7.6% of the cases while it is *east* in 92.4% of the cases. That is to say that the DBpedia statement of Ventura being southeast of Santa Barbara is merely true for 527 point pairs, while east is the correct relation for 6,425 other pairs. The situation would be even more complex if we consider fuzzy-set representation typically used for cognitive regions, e.g., *downtown*.

The last issue that remains to be discussed is performance. Clearly, computing cardinal directions takes longer than retrieving stored triples. A SPARQL query for all cardinal directions of the top 20 places takes about 3.3s on DBpedia's public endpoint. A *cold*, i.e., non-cached, VOLT prototype computes the same relations and returns its results (but does not yield erroneous data as does DBpedia) in about 18s on a modern laptop. This number should be taken into perspective by comparing it to the cache-enabled VOLT which takes only 6.9s after an initial run. Finally, it is important to remember that queries typically ask for the cardinal direction between a place and other geographic features and not for hundreds of directions among 20 random places. In such real-world cases, however, the overhead introduced by computation is relatively small.

Summing up, DBpedia currently only stores a very small, and from an end user's perspective, arbitrary fraction of cardinal direction relations. Approximately 33% of these relations are defective and many others need an understanding of the involved

uncertainties to make use of them in a reproducible setting. For instance, there is no way for a user to understand what is returned by a SPARQL query for cardinal directions: are the results about the closest entity in a given direction, multiple entities, entities of the same type (e.g., the city north of LA), and so forth. Using the VOLT proxy, cardinal directions between all places can be computed on-demand along with provenance records that document how the computation was done and based on which formal definitions.

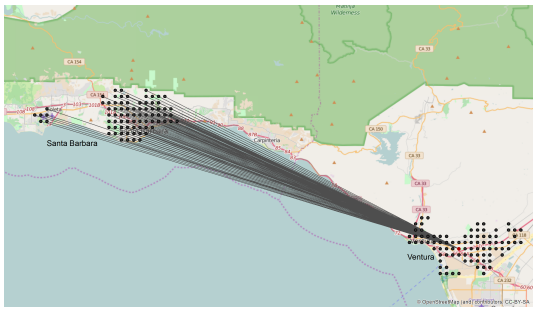

Figure 2.3: Uncertainty in cardinal directions for Santa Barbara and Ventura.

Figure 2.4: Union of coastal counties computed as adjacent to Pacific Ocean.

2.3.2 Case Study II: Counting Regional Population

For the second case study, let us assume that a client wants to count the total population of California's coastal counties. She discovers the DBpedia resources for: North Coast of California, Central Coast of California and South Coast of California; each of which embodies counties along the coast. Intuitively, the user expects these three regions to be spatially disjoint and after inspecting the page for the Central Coast, naively devises the SPARQL query shown in Listing 6.

```
select (sum(?regionalPopulation) as ?coastalPopulation) where {
    ?region dbp:population ?regionalPopulation .
    values ?region {
        <http://dbpedia.org/resource/North_Coast_(California)>
        <http://dbpedia.org/resource/Central_Coast_(California)>
        <http://dbpedia.org/resource/South_Coast_(California)> }}
```

Listing 6: Select the sum of population counts for all three CA coastal regions.

At the time of this writing, the query from Listing 6 returns a `?coastalPopulation` of 2,249,558 - the same number as the population property given by the DBpedia resource for the Central Coast. In fact, the South Coast was not included since its population value is the literal "~ `20 million`" and the North Coast does not have a population property to begin with. Therefore, since the query does not check if each region was matched to a triple, and since the `sum` aggregate function in SPARQL *silently* ignores non-numeric values, the result of this query is misleading. Even more, the three coastal regions are neither continuous nor disjoint. For example, there are two coastal counties, San Francisco County and San Mateo County, which do not belong to any of the three coastal regions in California; they break the continuity of these regions by making a gap in between the North Coast and the Central Coast. The regions are also not disjoint because the Central Coast and the South Coast both include Ventura County; this could lead to counting the population of Ventura County twice.

Clearly, the client needs a better way to select the coastal counties of California and should be able to validate the accuracy of their operation by inspecting the provenance of constituent population values. By modifying our data to be GeoSPARQL-conformant, we can build a better query as shown in Listing 7.

```
# count the population of coastal counties in California
```

```
select (sum(?countyPopulation) as ?coastalPopulation) where {
    # get geometry of Pacific Coast as WKT
    data:PacificCoast geo:hasGeometry/geo:asWkt ?pacificCoastWkt .
    # use a subquery to group by place; avoid counting same place twice
    { select ?county (sample(?population) as ?countyPopulation) {
        # select all California counties and geometries as WKT
        ?county a yago:CaliforniaCounties .
        ?county geo:hasGeometry/geo:asWKT ?countyWkt .
        # make sure the county geometry is a polygon
        filter(regex(?countyWkt, '^(<[^>]*>)?(MULTI)?POLYGON', 'i'))
        # filter for coastal counties only
        filter(geof:sfTouches(?countyWkt, ?pacificCoastWkt))
        # get population of each county using best valid property name
        {   # best property to use is `dbo:populationTotal'
            ?county dbo:populationTotal ?population .
            filter(isNumeric(?population))
        } union {
            # next best property is `dbp:populationTotal'
            ?county dbp:populationTotal ?population .
            filter(isNumeric(?population))
            # block counties that have the preferred property
            filter not exists {
                ?county dbo:populationTotal ?best_population .
                filter(isNumeric(?best_population))
            }}} group by ?county }}
```

Listing 7: Use GeoSPARQL to count the population of California's coastal counties.

While the GeoSPARQL query is more likely to yield an accurate result, the user cannot perform aggregate spatial operations. In order to check if the entire coast was accounted for, she would have to issue a separate query in which `?countyWkt` is selected without any aggregate functions and then plot each geometry on a map. With the VOLT framework however, we provide namespaced aggregate functions that construct

temporary RDF graphs in the SPARQL query from a list of results for a single variable. By keeping only the county selection patterns in the subquery and aggregating those counties into an RDF Set, we can then call the user-defined `stko:sumOfPlaces` method to sum the values of the population properties as shown in Listing 8. Additionally, the user-defined method can construct a single geometry feature that is the union of all coastal counties in California. We then plot this geometry feature on a map to inspect the areas included in our population count, as shown in Figure 2.4.

```
prefix volt: <http://volt-name.space/ontology/>
prefix input: <http://volt-name.space/vocab/input#>
prefix output: <http://volt-name.space/vocab/output#>
prefix stko:  <http://stko.geog.ucsb.edu/vocab/>

# count the population of coastal counties in California
select ?population ?area where {
    # in a subquery, aggregate all California's coastal counties into a set
    { select (volt:cluster(?county) as ?setOfCounties) {
        # select only California counties
        ?county a yago:CaliforniaCounties .
        # ...that are `along' the Pacific Coast (refers to a computable property)
        ?county stko:along data:PacificCoast . } }
    # let `sumOfPlaces' method compute the total population of coastal counties
    [] stko:sumOfPlaces [
        input:places ?setOfCounties ;
        input:propertyList (dbo:populationTotal dbp:populationTotal) ;
        output:sum ?population ;
        output:coveredArea ?area ;  ]  }
```

Listing 8: Computes the sum of values for the first valid numeric property from `dbo:populationTotal` or `dbp:populationTotal` for all coastal counties in California.

The `stko:sumOfPlaces` method is stored in the model graph as RDF triples. To

simplify the process of programming user-defined procedures in RDF, we developed the VOLT syntax and its compiler [6]. The language allows inline embedding of SPARQL query fragments, dynamically-scoped variables, operational expressions, and basic flow control. The VOLT source code for the `stko:sumOfPlaces` method is shown in Listing 9. At runtime, the population example will cause this method to generate the SPARQL query shown in Listing 10. Note that the VOLT language does not invalidate our claim of the proxy being transparent and only depending on well established W3C technologies. The language is only used to simplify the production of ontologically-compatible RDF statements which define custom functions and optionally their connections to external systems such as PostGIS. This language is not used for querying or any other functionality exposed to the client. As explained before, each procedure is serialized to RDF and stored in the model graph where it is available for public inspection.

```
method stko:sumOfPlaces {
    input ?places decluster into ?place
    input ?propertyList(list)
    select ?sum=sum(?value) ?placeGeomsWkt=volt:collect(?placeWkt) {
        ?matcher volt:firstMatch [
            input:forVariable "?property"^^volt:Variable ;
            input:useValuesFrom ?propertyList ;
            input:sampleFromVariables ("?value"^^volt:Variable)  ] .
        graph ?matcher {
            ?place ?property ?value .
            filter(isNumeric(?value)) }
        ?place geo:hasGeometry/geo:asWKT ?placeWkt }
    output ?sum  # shorthand for `output [output:sum ?sum]'
    if object has output:placeGeometries {
        output [output:placeGeometries ?placeGeomsWkt] }
    if object has output:coveredArea {
```

```
        ?coveredArea = postgis:union(?placeGeomsWkt)
        output ?coveredArea }
    if object has output:overlap {
        ?overlap = postgis:union(postgis:intersectionAmong(?placeGeomsWkt))
        output ?overlap } }
```

Listing 9: User-defined `sumOfPlaces` method in VOLT syntax. It accepts two inputs: (1) a set of places whose properties should be summed and (2) a list of property IRIs ordered by the most preferred property value to match each distinct `?place`.

```
select (sum(?value) as ?sum)
    (group_concat(?_n3_placeWkt; separator='\n') as ?placeGeomsWkt)
where {
    # volt:firstMatch for variable ?property, use values from: (dbo:populationTotal dbp:populationTotal).
    #   sample from variable ?value
    { select ?place ?property (sample ?_sample_value as ?value)
        where {
            {   ?place ?property ?_sample_value .
                filter(isNumeric(?_sample_value))
                values ?property { dbo:populationTotal }
            } union {
                ?place ?property ?_sample_value .
                filter(isNumeric(?_sample_value))
                values ?property { dbp:populationTotal }
                filter not exists {
                    ?place dbo:populationTotal ?_0_value .
                    filter(isNumeric(?_0_value))
                }}} group by ?place ?property }
    ?place geo:hasGeometry/geo:asWKT ?placeWkt .
    # decluster ?places into ?place
    values ?place { dbr:Alameda_County   dbr:Contra_Costa_County  dbr:Del_Norte_County ... }
    # volt:collect(?placeWkt)
    bind( if(isBlank(?placeWkt), concat('_:', struuid()),
        if(isIri(?placeWkt), concat('<', str(?placeWkt), '>'),
        if(isLiteral(?placeWkt),
            concat('"',
                replace(
                    replace(str(?placeWkt), '"', '\\\\"'),
                    '\n', '\\\\n' ), '"',
                if(lang(?placeWkt) = '',
```

```
            concat('^^<', str(datatype(?placeWkt)), '>'),
            concat('@', lang(?placeWkt)) )),
    concat('?', struuid()) )))
    as ?_n3_placeWkt   ) }
```

Listing 10: A SPARQL query issued by the proxy on behalf of the client's input query. The client invokes the `stko:sumOfPlaces` method that substitutes values and subquery selection results into its own SELECT stage, ultimately yielding this SPARQL query.

Summing up, this second use case highlights the difficulties in naively querying Linked Data and the misleading results that commonly result from such queries. We use it to showcase VOLT's capabilities with respect to user (or provider) defined methods and the provenance information that allows others to inspect how the returned query results came to be.

2.4 Related Work

In this section we introduce work that is either related in terms of common goals, similar technological approaches, similar target domain, i.e., geo-data, or inspired and informed our thinking while developing the VOLT framework.

Linked Data Services (LIDS) [98] describes a formalization for connecting SPARQL queries to RESTful Web APIs by enabling a service layer behind query execution. Service calls have named inputs and outputs in the query. VOLT provides functional triples which also use named inputs and outputs in the query to make API calls to registered plugins. Plugins execute asynchronously and may perform networking tasks such as requests to RESTful Web APIs.

Linked Open Services (LOS) [81] sets forth the principles on how to establish interoperability between RESTful resources and Linked Open Data by semantically lifting flat content to RDF.

The **Linked Data API** (LDA) [94] is used to create RESTful APIs over RDF triple stores to streamline the process of web applications consuming Linked Data. Similar to LDA, VOLT also runs as a SPARQL proxy and dynamically generates SPARQL queries on behalf of the client.

Linked Data Fragments (LDF)[105] highlight the role of clients for scaling query engines by offloading partial execution to the web browser. Since our prototype is implemented in JavaScript, the proxy also runs as a standalone instance in the browser. The framework only needs a connection to a SPARQL endpoint over HTTP, or a locally emulated one such as the LDF client. In this regard, we aim to achieve Web-Scale querying as described by Verborgh[105].

SCRY[100] is a SPARQL endpoint that allows a client to invoke user-defined services by using named predicates in SPARQL queries. It simply identifies which service to execute and forwards the appropriate arguments given by the associated triple. SCRY's current implementation requires services to be implemented as Python modules or as command-line executables. Compared to VOLT, it does not provide the means for a client to inspect the source of user-defined services.

iSPARQL is a *virtual triple approach*[63] to invoking custom functions similar to the concept of *magic properties*. It extends the SPARQL grammar with a *SimilarityBlock-*

Pattern production to distinguish between basic graph pattern triples and triple-like function calls having the form *?v apf:funct ArgList*[63].

The **SPIN**[1] framework generates entailments by issuing SPARQL queries to perform inferencing. The framework consists of a set of vocabularies that enable the serialization of user-defined rules, input as SPARQL queries, directly into an RDF graph; a technique that preserves IRI referential integrity. VOLT also serializes SPARQL fragments and graph patterns into RDF to use as inference rules. However, SPIN requires use of a proprietary extension of the SPARQL language to explicitly invoke computation while VOLT is designed to automatically recognize the need for computation on regular SPARQL queries that are issued as if the patterns are simply being matched to existing triples.

Logical Linked Data Compression[61] proposes a lossless compression technique which benefits large datasets when storage and sharing may be an issue. Similar to their compression, VOLT reduces the number of triples by using procedures to generate statements that can be deduced from source triples. However, our approach increases the total size of a dataset when caching is enabled. With caching disabled, one can instead opt for computing such statements on-demand thus saving storage space at the cost of query execution time.

2.5 Conclusions

In this work we introduced the transparent VOLT proxy for SPARQL endpoints. We outlined its core features, highlighted selected implementation details, and presented use cases that demonstrate the proxy's capabilities in addressing key shortcomings that we believe prevent the wide usage of Linked Data in science. Instead of storing triples that depend on already stored data, we propose to compute results on-demand and then cache them. Our work goes beyond merely reducing the amount of stored triples but also addresses quality issues as the *dependent* triples have to be kept in sync with their source data, e.g., when storing population densities in addition to population and areal data. We also address issues of provenance and the reproducibility of results by making the VOLT functions available and inspectable and by storing all data and procedures that were used to arrive at certain results in a separate graph. Finally, we discuss two use cases to demonstrate the difficulty in querying Linked Data, quality issues in Linked Data, and the need for the implemented VOLT capabilities.

Future work will focus on improving our current prototype and making it easier to extend and customize by others. We will also work on improving the proxy's performance and an alignment of our provenance and model graphs with ontologies such as PROV-O [66] and (semantic) workflow models in general [40].

Chapter 3

Prospects of Precomputing Topological Relations in Geo Knowledge Graphs

Peer Reviewed Publication	
Title	Computing and Querying Strict, Approximate, and Metrically-Refined Topological Relations in Linked Geographic Data
Authors	Blake D. Regalia, Krzysztof Janowicz, Grant D. McKenzie
Institutions	STKO Lab, Department of Geography, University of California, Santa Barbara, USA
DOI	10.1111/tgis.12548
Venue	Transactions in GIS. 23(3) 601-619, Wiley Press
Editors	John P. Wilson
Submit Date	February 4, 2019
Accepted Date	April 3, 2019
Published Date	June 25, 2019
Presentation Date	July 9, 2019
Copyright	Blake D. Regalia, Krzysztof Janowicz, Grant McKenzie

3.0.1 Abstract

Geographic entities and the information associated with them play a major role in Web-scale knowledge graphs such as Linked Data. Interestingly, almost all major datasets represent places and even entire regions as point coordinates. There are two key reasons for this. First, complex geometries are difficult to store and query using the current Linked Data technology stack to a degree where many queries take minutes to return or will simply time out. Secondly, the absence of complex geometries confirms a common suspicion among GIScientists, namely that for many everyday queries place-based relational knowledge is more relevant than raw geometries alone. To give an illustrative example, the statement that the White House is in Washington DC is more important

for gaining an understating of the city than the exact geometries of both entities. This does not imply that complex geometries are unimportant but that (topological) relations should also be extracted from them. As Egenhofer and Mark put it in their landmark paper on naive geography, *topology matters, metric refines*. In this work we demonstrate how to compute and utilize strict, approximate, and metrically-refined topological relations between several geographic feature types in DBpedia and compare our results to approaches that compute result sets for topological queries on-the-fly.

3.1 Motivation and Research Contribution

Places and the information associated with them are among the most interlinked types of entities on the global Linked Data cloud [50]. Within such Web-scale, cross-domain knowledge graphs, places act as pivotal vertices connecting events, people, and objects. Repositories that contain large collections of geographic identifiers are among the most central and densely interlinked hubs on the Linked Data cloud. For instance, *named* places are the second most frequent entities within DBpedia and collectively contribute millions of properties to the dataset, including some of the most common property types such as birthplaces of historic figures and administrative subdivision names.

Nonetheless, the vast majority of geographic identifiers are represented in the simplest of all possible spatial representations, namely point coordinates. While such representation is appropriate for many everyday information retrieval tasks, e.g., finding nearby restaurants, it is not suitable for the plethora of operations performed by scientists, gov-

ernment agencies, and industry professionals using geographic information systems and spatial analysis more broadly. To some extent, these demands could be met by simply providing the more complex geometries, e.g., polylines and polygons, for places in raw form as Linked Data. However, such approaches overlook the key underlying issues. (1) Querying high-resolution geometries, e.g., the areal extent of a river, by using on-demand spatial extensions to triplestores, such as GeoSPARQL, does not scale well over large datasets. (2) Real world applications for complex geometries and semantically empowered queries require preprocessing steps, e.g., to handle so-called sliver polygons, that are not currently supported by any Linked Data based framework. (3) The ultimate purpose of spatial analysis is often concerned with topological information, e.g., whether a river runs through a city, thereby turning geometries into a "means to an end" for acquiring the topological relation between entities. (4) Finally, the proper geometric representation of real-world entities varies by place type, scale, and task, often leading to unintended consequences when operating on raw, precomputed geometries alone. For instance, representing a state park as a point-feature may be sufficient to get a general sense of its location, but the representation does not support queries for adjacent water bodies. An unintended consequence may be that the centroid of the park is in one county but the extent of the park actually spans two or more counties leading to improper topological results.

With the advent of GeoSPARQL [89] and other means to perform spatio(temporal) queries [64] over Linked Data, complex geometries are becoming more popular across

several datasets. The LinkedGeoData project [99], 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, patterns, and convex hulls.

Based on the presented argumentation, we conjecture that replacing the simple geometries that dominate knowledge graphs and search engines today with more complex geometries will be of limited use for many everyday applications. Instead, we believe that knowledge graphs and Linked Data more concretely will benefit further from topological relations. One could now 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 [37, 101], our argument will focus on the role of domain knowledge, vagueness, and uncertainty [8] and not on computational issues. The fact that simple point geometries are sufficient for the most frequent Point Of Interest (POI) queries has been sufficiently demonstrated by major search and map engines, POI repositories, and place-based social networks, Wikipedia, and so on. Therefore, we will only consider places that are of sufficient spatial extent to result in substantial inaccuracies when modeled as point features alone. Examples include rivers, roads, counties, parks, and so on.

To understand how topology is handled in GIS, it is important to note that data collection, modeling, and preprocessing take about 80% of the entire 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.[1] Neither the preprocessing steps nor the domain-specific topology rules are available when computing topological relations on-demand using GeoSPARQL over Linked Data. Also, the datasets used for any given GIS task that involve topological relations are orders of magnitude smaller than querying such relations over Linked Data hubs such as DBpedia. Hence, queries such as finding cities along the Mississippi River or counties that run along state borders cannot be effectively answered over Linked Data today.

Consider the following illustrative example. Lynchburg, Tennessee is a consolidated city-county whose boundaries coincide with Moore County. Using Region Connection Calculus 8 (RCC8) [18], the true topological relation between the city and county should be *equal* however computing the relation using GeoSPARQL returns *partially overlaps*; see Fig. 3.1. The reason is due in large part to digitization errors, i.e., the double-digitized boundaries problem. While differences in granularity are common sources of errors, difficulties arising from uncertainty and vagueness are even more troublesome. Whereas uncertainty stems from lack of precise knowledge, vagueness is caused by intrinsically under-determined concepts that do not have clear borders [8]. For example, the true shape of a city can be determined in theory although measurement accuracy, timeliness (the city may grow or shrink), administrative definitions, and so forth, impact the results.

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

In contrast, the shape of a mountain or forest cannot be exactly determined in practice nor theory as the transition zones between a mountain and a valley, as well as a forest and isolated trees, are *conceptually* vague.

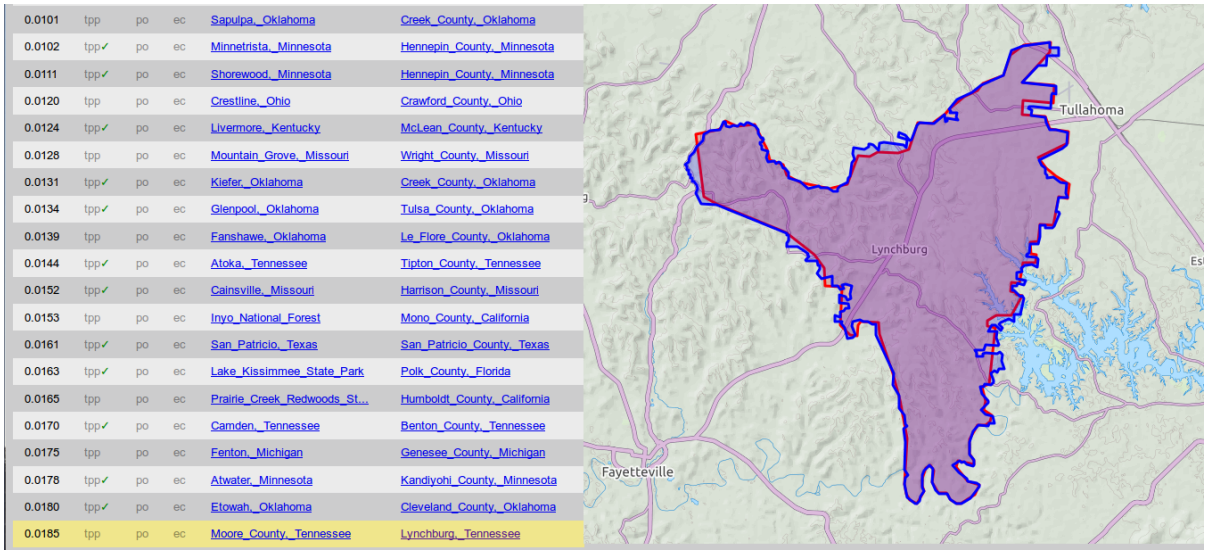

Figure 3.1: Lynchburg, Tennessee is a consolidated city-county whose boundaries coincide with Moore County. The expected topological RCC8 relation should be *equal* (**EQ**), however computing the relation solely given the geometries will return *partial overlap* (**PO**).

Problem statement: Following Egenhofer and Mark's slogan that *topology matters, metric refines* [29], knowledge graphs will benefit from explicit topological relations in addition to (complex) geometries and other place-specific properties. Computing such relations, e.g., using GeoSPARQL, based on geometry alone is not currently possible in the context of Linked Data.

To address this problem, we propose to combine techniques from GIS and the Semantic Web then demonstrate how to derive strict, approximate, and metrically-refined

topological relations, how to use background knowledge in the form of RDF triples that, while strictly speaking are not topological, can be used to infer topological relations, how to define an ontology to distinguish between geographic feature types that have broad boundaries versus those that do not, and finally, how to integrate the aforementioned methods into a multi-layered topological relations framework to enrich DBpedia.

In terms of a bigger picture, this work is about exploring one of three major trade-offs to bring the full Digital Line Graph data from the USGS National Map to the Linked Data cloud. The first trade-off is the decision about which relations to compute on-the-fly and which to materialize [92]. For instance, dependent properties such as population densities should be computed if the population count and area are already stored as triples. Similarly, while DBpedia stores select cardinal direction triples, storing all of them would lead to a combinatorial explosion. The work presented here takes a complementary perspective by looking at relations that cannot be easily computed on-the-fly, and, thus, should be precomputed and materialized instead. We will show that queries which include topological relations often cannot be effectively answered using GeoSPARQL. The third tradeoff is about balancing client-side versus server-side queries [91].

The research contributions of this work are as follows:

- To demonstrate the feasibility of the proposed methods, we present a linked dataset of topological relations derived from the geometries of cities, counties, parks, streams, and roadways for the contiguous United States. We selected these feature types

since they cover both strict boundaries (e.g., administrative boundaries) and broad boundaries (e.g., streams) as well as the pairwise relations between regions-to-regions, regions-to-polylines and polylines-to-polylines.

- In addition to the strict topological relations based on a subset of RCC8, we also include approximate topological relations [15], and additional topological relations with metric refinements [26]. To the best of our knowledge, these extended topological relations have never before been used in the context of Semantic Web research and are neither part of any linked dataset nor ontology.

- We demonstrate how to derive these relations by applying methods known from geographic information systems to features matched between DBpedia and OpenStreetMap. We show how Semantic Web technology can be leveraged to discover latent properties within a heterogeneous geographic dataset by applying topological reasoning. One example would be the creation of a `coastal city` class, defined as a city that has the `(broadly) touches` relation to a feature of the class `ocean`. We will discuss a more complex example about the topological relation of parks and county borders.

- We present example queries based on the resulting dataset and compare them to using GeoSPARQL for qualitative spatial reasoning.

- Finally, we show how our work can help in detecting and clearing erroneous place type definitions in DBpedia based on implausible topological relations.

In this work, we discuss the primary challenges to computing and representing topological relations solely from geometries, demonstrate how to use ontologies and multi-layered topological relations to overcome these challenges, and produce a preprocessed and cleaned dataset of topologically linked places derived from DBpedia and OpenStreetMap.

The remainder of the paper is structured as follows. In Section 3.2, we describe the process of preparing data collected from DBpedia and OpenStreetMap in order to compute topological relations. In Section 3.3, we describe how we compute the relations, including for crisp boundaries, broad/approximate boundaries, and metrically-refined topological relations. In Section 3.4, we provide an overview of the resulting dataset, show a comparison to computing topological relations using GeoSPARQL, and demonstrate the utility of our dataset by example. Finally, we conclude the paper and point to directions for future work.

3.2 Data Preparation

In this section, we discuss the procedure for constructing a spatially-enabled database in preparation for computing topological relations. The database combines and resolves RDF resources from DBpedia with spatial elements[2] from OpenStreetMap.

[2]From OSM's conceptual data model terminology: https://wiki.openstreetmap.org/wiki/Elements

3.2.1 Data Integration

The goal of this work is to enrich DBpedia with topological relations by producing a dataset of RDF triples. Since DBpedia places are only represented as point coordinates, our first task is to match as many places from DBpedia with their corresponding polygon or polyline geometries in OpenStreetMap. This type of coreference resolution task presents a number of challenges, most notably those discussed by Sehgal et al. [95] and Ngomo [80]. Existing approaches include the use of string similarity measures [76] and spatial signatures [110], to name a few. In this paper however, we focus on topological methods and the accuracy of resulting topological relations.

Therefore, we rely on existing meta-level links between the two datasets, i.e., matching normalized Wikipedia and Wikidata URIs through the (a) objects from DBpedia triples linked via `owl:sameAs` and `foaf:isPrimaryTopicOf` (b) "wikidata" and "wikipedia"/"wikipedia:en" tag values from OSM elements in order of precedence. We show an example for Yosemite National Park in Listings 11 and 12. Approximately $90k$ OSM elements in North America have such links to DBpedia.

It is important to note that compared to OpenStreetMap which strives for comprehensive geographic coverage, DBpedia exhibits a sparser coverage yet contains a greater depth of information per feature. This is a natural consequence of the fact that Wikipedia, DBpedia's data source, is primarily driven by community members writing articles about topics of societal significance such as cities, national parks, important historic landmarks, and so on. Consequently, coverage is not a primary concern since our goal is to produce

an RDF dataset for the Linked Open Data cloud of which DBpedia is the central hub. It's also worth mentioning that the relatively sparser coverage does not jeopardize our ability to compute topology since we are only interested in materializing relations between existing resources when they available. In other words, we envision our approach as being able to adapt to varying degrees of data availability.

```
# http://dbpedia.org/resource/Yosemite_National_Park
dbr:Yosemite_National_Park owl:sameAs wikidata:Q180402 ;
    foaf:isPrimaryTopicOf wikipedia-en:Yosemite_National_Park .
```

Listing 11: An example of the meta-level links that exist for Yosemite National Park in an RDF document from DBpedia http://dbpedia.org/data/Yosemite_National_Park.ttl

```
<!-- https://www.openstreetmap.org/relation/1643367 -->
<osm>
    <relation id="1643367">
        <tag k="wikidata" v="Q180402"/>
        <tag k="wikipedia" v="en:Yosemite National Park"/>
        <!-- polygon geometry and other tag nodes... -->
    </relation>
</osm>
```

Listing 12: An example of the meta-level links that exist for Yosemite National Park a relation element from OpenStreetMap. In this case, the feature has both links so the Wikidata entity id "Q180402" is used for resolution and the Wikipedia URI is used to validate the resolution.

In order to store the geometries and compute topological relations between all pairs of geographic features, we use the *PostGIS* spatial extension to *PostgreSQL*. We use Overpass[3] to query for all *ways* and *relations* that have a `wikidata`, `wikipedia` or `wikipedia:en` tag. These features are loaded into a spatially-indexed PostgreSQL table with their id, Wikipedia URL suffix, and geometry.

[3]Overpass Query API: https://wiki.openstreetmap.org/wiki/Overpass_API

3.2.2 Entity Selection

In order to derive meaningful strict, approximate, and metrically-refined topological relations requires tuning place-type-specific parameters. For example, the exact buffer radius to use in order to derive a polygon's broad boundary should differ when comparing two cities versus comparing two national parks for the *approximately adjacent* relation, even assuming all polygons are of similar size. In other words, broad boundaries cannot depend on geometry alone. Therefore, we focus our efforts on a subset of features by selecting those of specific place types. We select cities, counties, and parks, which are represented by multipolygon geometries, as well as roadways and streams, which are represented by polyline geometries.

A final collection of places within the contiguous United States have the following essential properties: A DBpedia resource URI, an OpenStreetMap element URI, some geometry ((multi)polygon or polyline), and a place type tag such as city, county, park, roadway, or stream. A composite overview of these are shown in Figure 3.2.

3.2.3 Cleaning Digitization Errors

As a first step in deriving topological relations from the noisy geometries we collect from OpenStreetMap, we define a set of metrics that measure various characteristics of the interaction between two geometries. These metrics are initially defined from a top-down perspective and supported through manual inspection of the data. A custom map-enabled interface is used for inspection, providing a broad overview of possible threshold values

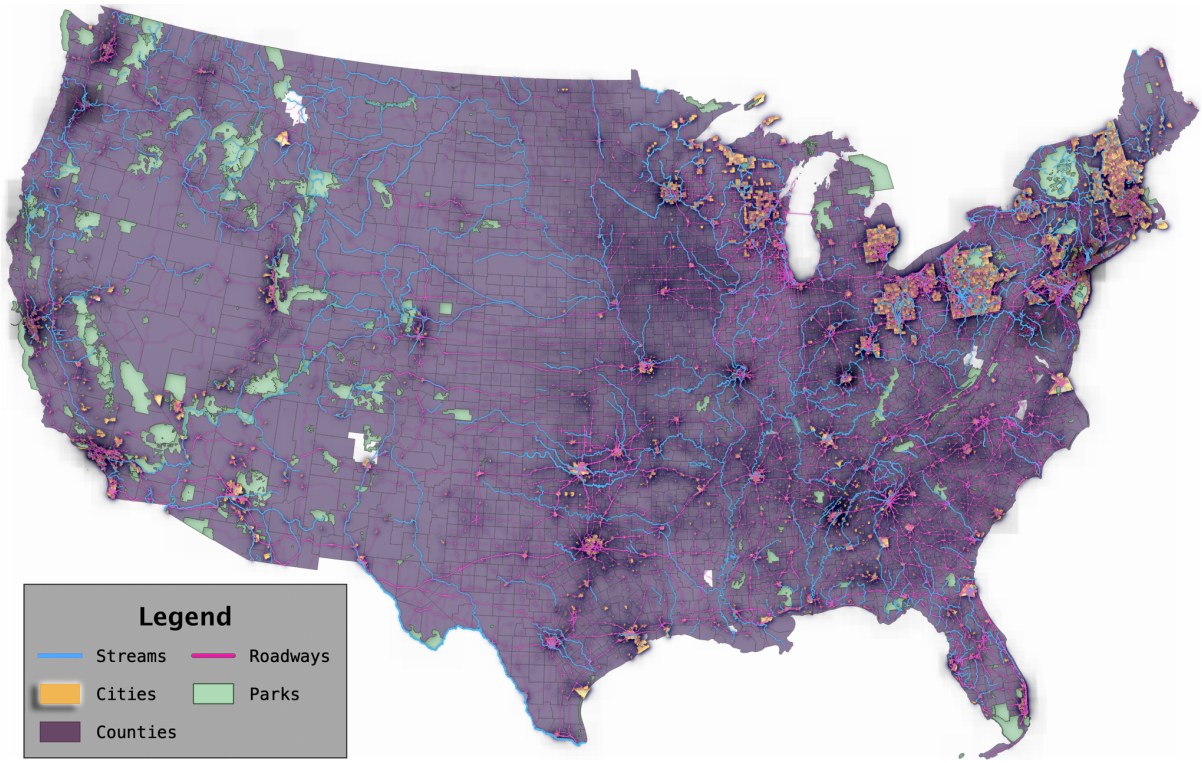

Figure 3.2: A map showing a composite overview of all places, categorized by place type, that were matched between DBpedia and OpenStreetMap which we use to compute topological relations. Notice that the 'Cities' symbol uses a drop-shadow effect to reveal the density of small polygon features at this macro-scale.

for identifying and labeling proper topological relations.

Digitization errors are handled through manual exploration of the data in order to identify a conservative threshold that will coerce relations arising from poor geometric alignment into their *correct* relation. For example, intuitively one might expect that the City of Santa Barbara would be completely contained by Santa Barbara County. In a strictly topological sense, however, the two regions in OpenStreetMap partially overlap, as shown in Figure 3.3. The area of their difference though is only $11.3m^2$, clearly the

result of digitization error. In order to help identify such cases, we construct a range of metrics for each relation during the initial computation of strict topological relations. Relations that result in high values for these measure are intended to signify an increased likelihood of a digitization error. We designed a visual interface to inspect these measures case-by-case, one at a time, alongside a map view of the geometries (Figure 3.1).

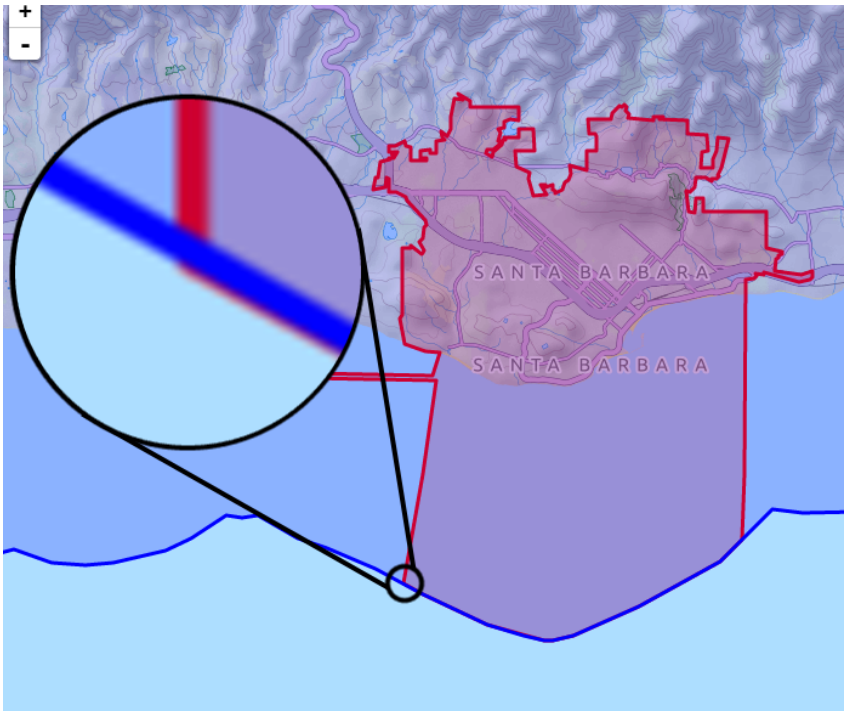

Figure 3.3: A rendered image of the geometries for the city of Santa Barbara (red) strictly overlapping Santa Barbara County (blue). The inset zoom bubble shows the $11m^2$ difference of the two geometries.

Egenhofer and Dube [26] define a set of nine splitting measures for *polygon-overlaps-polygon* relations in order to support metrically refined topology. Here, we apply their Inner Area Splitting (IAS) measure by computing the area of intersection divided by the area of the smaller polygon, i.e., $\frac{Area(L \cap S)}{Area(S)}$. This metric enables us to identify an

appropriate threshold to separate cases that should actually be labeled as *tangential proper part* from those that are indeed *overlapping*. This same metric is also used to correct relations erroneously identified as *partial overlaps* to the more suitable *externally connected* (EC). In our analysis, we also measured Inner Traversal Splitting (ITS) and Outer Traversal Splitting (OTS) yet found IAS to be the strongest measure for separating region overlap cases on this dataset.

For all strictly *disjoint* cases, we focus only on those relations that are clearly digitization errors. Adhering to the conceptual neighborhood graph, we can only obtain EC relations from those that start strictly as *disjoint* (DC). Using our custom map-enabled dataset interface, we manually inspected all DC region-region pairs sorted by separation distance in ascending order until reaching cases that were no longer unquestionably disjoint. We then settled on using the conservative distance threshold value of 20 meters or less between polygon geometries to clean digitization errors by coercing them to the EC relation. In other words, we manually labelled all 375 cases of DC region-region pairs that were corrected to EC. Furthermore, pairs of geometries that are of a distance greater than or equal to 20 meters apart are later used as candidates for the *nearly meets* broad boundary relations. We plan to add more measures such as Expansion Closeness from Egenhofer and Dube [26], in future work.

3.3 Computing Topological Relations

In this section we provide details about the selected strict, approximate, and metrically-refined topological relations and their computation.

We start by computing an index of all spatially disjoint features to avoid redundant calculations since every pairwise combination between features must be considered for each topological relation. For instance, to discover that a city and a nearby river are broadly touching, we first need to compute that they are strictly disjoint, yet close enough that their boundaries might overlap. We then proceed by checking topological relations on the remaining pairwise combinations. In fact, we continue this pattern of propagating result sets from each computed topological relation onto the next task in the series to substantially reduce the overall processing time. We provide an example of this technique for the polygon-to-polygon relations procedure in Listing 13.

```
# 'cps' stands for 'compute_pairwise_self'
non_interacting := cps_non_interacting(all)
disjoint := non_interacting + cps_disjoint(all - non_interacting)
interacting := all - disjoint
touches := cps_touches(interacting)
intersecting := interacting - touches
overlaps := cps_overlaps(intersecting)
within := cps_within(intersecting - overlaps)
tangential_proper_part := cps_tpp(within)
non_tangential_proper_part := within - tangential_proper_part
```

Listing 13: Pseudocode summarizing the procedure for computing the polygon-to-polygon topological relations in series by reusing and subtracting results sets from previous computations, an application of the conceptual neighborhood graph. Notice in this Listing, we use "+" to represent the union of two sets and "−" to represent the relative complement.

3.3.1 Strict Topological Relations

As opposed to relations between features with broad boundaries, i.e., *approximate* relations, we use the term *strict* to refer to relations about polygons with crisp boundaries and polylines.

Egenhofer and Franzosa [27] initially defined a framework for the description of topological spatial relations based on the intersections of boundaries and interiors between two sets in $\mathbb{R}^2$. Clementini et al. [17] extended the 9-Intersection Model [30] for topological interactions between spatial regions with the Dimensionally Extended 9-Intersection Model (DE-9IM). Cohn et al. [18] provide a family of first-order logical calculi known as Region Connection Calculus which treats spatial regions as primitives in order to support reasoning about spatial entities with *connections*. Most notably, RCC8 is a set of eight relations that are jointly exhaustive and pairwise disjointed.

In RCC8, TPPi and NTPPi are inverse relations of TPP and NTPP, respectively. Consequently, the inverse relations are reserved to be inferenced by the RDF reasoner during query execution. In fact, we omit materializing any triples that would be handled by basic reasoning on inverse properties and transitive properties. We also do not materialize *disjoint* relations as this is not only infeasible from a storage perspective but also unnecessary for operating under the Open World Assumption (OWA). Because Linked Data operates under the OWA, dataset publishers may choose to exclude any sets of relations without introducing inconsistencies among their enriched dataset. In total, for region-region relations, we compute *equals* (EQ), *externally connected* (EC), *partially*

overlapping (PO), *tangential proper part* (TPP) and *non-tangential proper part* (NTPP).

For strict topological relations between line-region, we compute *touches* (TCH), *passes through* (PTH), and *includes* (INC) [28, 35].

3.3.2 Approximate Topological Relations

Conceptually there is a disconnect between what is clearly a strict relation and an *approximate* relation. Approximate topological relations [15] are used to describe broad boundaries [24] between spatial features. For example, a river may border a city on one side, but topologically the river does not coincide with the border in all sections; rather it *approximately* follows part of the city boundary before continuing on. In such cases, one can argue there is a topological relationship between the two features as the concrete geometric representations of features and their accuracy depend on time, scale, measurement conventions, and so forth. This is particularly the case when both fiat and bona fide boundaries are involved [97]. However, what exactly constitutes a *broad boundary* compared to two features simply being *nearby* requires further exploration.

To determine the radius by which to buffer a polygon's boundary, the 0.05 percentile of the cumulative distribution function of ordered minimum distances between pairs is used as the maximum *broad boundary* measure. We then multiply this by the isoperimetric quotient shown in Equation 3.1 to account for the observation that features which have very specific (fiat or bona fide) boundaries can be thought of as having a lesser degree of

uncertainty as compared to those features that have simple shapes/boundaries.

$$IQ = \frac{4\pi A}{p^2} \tag{3.1}$$

Finally, the geometric boundaries for each feature pair in our set of disjoint relations are buffered using the above approach. Those feature pairs whose buffered boundaries intersect get assigned an approximate topological relation. For approximate topological relations, we compute *nearly contains* (nCt) and *nearly equals* (nE) for region-region relations, and *nearly meets* (nM) for both polyline-region and region-region relations.

3.3.3 Metrically-Refined Topological Relations

Here we briefly explain the concept of metrically-refined topological relations and then provide an explanation of four relations: *mostly within* (mW), *barely touches* (bT), *connects* (CON), *runs along* (RAL), and *runs alongside* (RAS).

Metrically-refined topology opens the door to a wide range of potential relations that distinguish more detail about relations between spatial entities than purely qualitative topological ones [26]. This includes predicates that may be conceptually vague and difficult to represent especially as a binary relation. Therefore, we attempt to capture the semantics of concepts that are obvious to human perception, such as a highway running along the ocean, even if they are occasionally inconsistent from a strictly topological point of view [29] .

To start off with a straightforward demonstration of metrically-refined topology, we refine the EC relation for region-region by defining *barely touches* (bT) as when the

length of the boundary connection is less than $10m$. Although this threshold is not data-driven, we emphasize that the primary goal of metrically-refined topological relations is to provide some meaningful distinction to users, oftentimes allowing domain experts to transparently impose a top-down perspective on the refined relations.

Next, we refine the PO relation for region-region by defining *mostly within* (mW) as when the area of the intersection is greater than or equal to 80% of the smaller polygon's area. This metric is based on the Inner Area Splitting (IAS), one of the nine splitting measures for *region-overlaps-region* relations from Egenhofer and Dube [26].

For relations involving polylines, we measure the area of intersection between the buffered regions of the two features. If X_D is the minimum bounding diameter of polyline X in meters, we define the buffer radii X_R as $\ln(X_D)$. The buffered polygon X_B is then used to calculate a polyline's interactions with other buffered features for the RAL and RAS relations. In Equation 3.2, we define the inequality for T, the threshold value for which the *runs along* RAL relation holds between two polylines X and Y. The *runs alongisde* relation applies a similar metric to line-region DC relations by using the buffered boundary and buffer radius of each feature.

$$\frac{Area(X_B \cap Y_B)}{(min(X_D, Y_D))^2} \geq T \tag{3.2}$$

For the CON relation, we select cases that approximately match the 16-intersection matrix code strings `0*0**0*10*11*111` (e) and `0*1**0*20*02*111` (h) for polyline-polyline relations from Formica et al. [36], with a relaxation of $\mathcal{E} = 10m$ as the maximum endpoint 'snapping' distance for which a polyline's endpoint is allowed to move in order

to intersect the other polyline. This threshold value was selected following the same process described in Section 3.2.3.

3.4 Application and Evaluation

In total, we produce 120,681 distinct RDF statements covering various topological relations among features of the selected place types within the contiguous United States. We provide a breakdown of these relations for polygons-to-polygons in Table 3.2, polylines-to-polygons in Table 3.3, and polylines-to-polylines Table 3.4. Next, we demonstrate how to use this dataset to correct place-type classification errors in DBpedia, validate existing adjacency relations in DBpedia, and perform topological queries over Linked Data. We also compare the performance of our materialized relations to querying them on-the-fly using GeoSPARQL.

3.4.1 DBpedia Error Correction

We compute topological relations between *all* combinations of features, regardless of their place type. However, certain place type combinations should exclude some topological relations by virtue of their ontological axiomatization. For example, two administrative regions of the same class cannot overlap by definition, so no county should ever be contained by another county. Our experiment yields cases that would appear to violate such rules, including the 10 combined county-county proper part relations, the 239 combined city-city proper-part relations, and the 48 city-city partial overlaps relations; see

tables 2-4. Manual inspection of these cases reveals that the features involved with these relations are in fact misclassified by DBpedia. Namely, the 10 county-county relations involve places that are actually cities, and the 287 city-city relations mostly involve places that are not cities but actually a variety of place types including cemeteries, airports, buildings, and so on. Many of these DBpedia resources also include `rdf:type` relations to their proper classes but DBpedia does not prevent the aforementioned class violations, e.g., by performing validation on the TBox statement that `Airport` and `City` are disjoint classes. Defining such disjointness axioms, however, for all place types combinations a-priori is not feasible due to many cases that can arise in reality such as cities spanning two counties. The same is true for constraints in the form of SHACL shapes.

It is worth noting that additional complications can arise from the fact that NTTP and EC can be easily confused both in terms of geometric errors and conceptually. For instance, one could naively assume that the village of Birmingham, Missouri is inside (NTTP) of Kansas City, Missouri while, in fact, it is entirely surrounded by it (EC). In contrast, a city is really contained by a county and not externally connected to it. Put differently, the area of Kansas City is determined by its polygon's area minus the holes represented by inner rings, while a city inside a county does not form such an inner ring. In everyday language, however, we typically do not make such distinctions.

3.4.2 Validating DBpedia's Adjacency and Partonomy Relations

While DBpedia itself does not aim at providing any robust topological relations between places, there do exist avenues for structured and semi-structured data from Wikipedia to make their way into topologically significant relations in DBpedia. For example, the primary cardinal direction relations, `dbp:north`, `dbp:east`, `dbp:south` and, so on, are generated via natural language processing on Wikipedia article abstracts, as well as from a special "Adjacent Communities" wiki template[4]. Ostensibly, these cardinal direction relations in DBpedia encode some meaningful topological relation, namely adjacency, between places. However, as one might suspect, relations are sometimes made to well-known places that are not remotely adjacent simply because they serve as a geographic reference or are in some way significant to the history or function of a place. For instance, Flint, MI has a southwest relation to Chicago, IL even though the two cities are more than $350km$ apart. Nonetheless, triples that make use of such cardinal direction relations offer an opportunity to deploy our materialized topological dataset for comparison.

In fact, for each of the $82,973$ distinct combinations between places that interact topologically according to our dataset, we query DBpedia for all relations that exist between each pair then rank the set of involved predicates by the number of times they appear in a triple. As we can see in Figure 3.4, cardinal direction relations make up nearly half of all relations, followed by nearly a third belonging to the `dbo:isPartOf`

[4] https://en.wikipedia.org/wiki/Template:Adjacent_communities

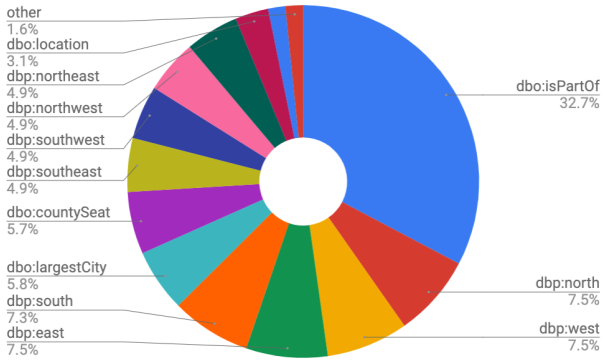

Figure 3.4: Relative frequencies of the most common predicates that relate two places to each other on DBpedia, excluding `dbo:wikiPageWikiLink`, for all features that exhibit any topological interaction within our dataset. Collectively, cardinal direction relations constitute nearly 50% of all such triples.

predicate. This allows us to assess which cardinal directions in DBpedia coincide with topological relations and which do not. As we have demonstrated in previous work [92], approximately 33% of the cardinal direction relations in DBpedia are defective and many other require additional information about the involved uncertainties to become reproducible. We compare all EC, TPP, and NTPP triples and find a majority of statements from DBpedia to be potentially accurate, see Figure 3.5. Based on the results shown above, such cases should be replaced with topological relations instead, particularly if they have been extracted from Wikipedia's adjacency template.

3.4.3 Relation to GeoSPARQL Queries

A key utility of our resulting dataset is to support topological queries over Linked Data. Given that users can already perform topological queries over Linked Data using GeoSPARQL, in this subsection, we demonstrate the shortcomings of computing

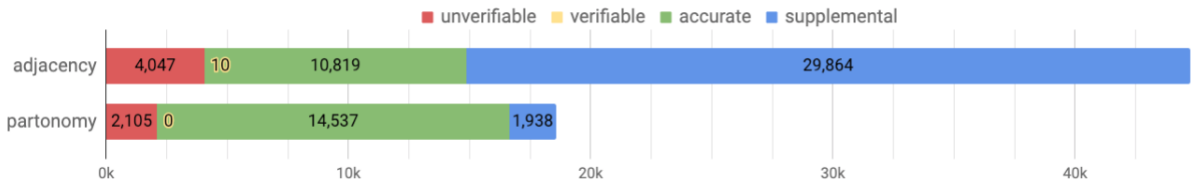

Figure 3.5: Stacked bar chart showing how many place pair relations are (a) **unverifiable** based on their absence from our materialized dataset, which suggests potentially inaccurate topological triples on DBpedia, (b) **verifiable** based on their presence in both datasets *with the stipulation* that the topological relation(s) observed in our dataset does not align with the topological relation inferred from DBpedia, (c) **accurate** based on their presence in both datasets *and* the condition that the topological relations align, which supports the topological accuracy of such triples on DBpedia, and (d) **supplemental** based on their absence from DBpedia, which demonstrates the volume of our contribution towards enhancing the LOD cloud. Cardinal direction relations are represented by the *adjacency* label and `dbo:isPartOf` by *partonomy*.

topology *on-the-fly*, i.e., in response to queries, illustrate the limitations of using purely crisp boundary topology, and show why the Web of Linked Data needs *cleaned* geometry data and *precomputed* topology encoded with domain knowledge, e.g., for applying the correct relations.

Consider, for example, a query for *how many other counties does each county share a border with?*; shown in Listings 14 and 15. Using GeoSPARQL, we are able to obtain 3,074 results in 176 seconds, compared to our approach which yields 3,080 results in about 9 seconds. Both queries run on the same *cold*, i.e., uncached, triplestore. The difference in performance is expected since the GeoSPARQL approach must compute topology on-the-fly whereas ours is already materialized.[5]

However, this bordering counties example was carefully chosen in order to be able

[5]Hence, this experiment should not be confused for a runtime performance evaluation but is supposed to demonstrate the feasibility (or lack thereof) of computing with complex geometries on-the-fly.

to compare our approach against GeoSPARQL since most other interesting use cases are simply unfeasible for a GeoSPARQL triplestore to handle on-the-fly, i.e., they either timeout or run out of memory, due to the computational cost for each topological relation combined with the large number of pairwise combinations between geometric features in such a dataset. To illustrate, the *city-touches-city* relation we (pre)computed for our dataset took over 12 hours to running on 56 2.1 GHz cores in parallel, while other relations, such as *road-nearlyMeets-road*, took more than 35 hours, combining topological and metric queries in PostGIS.

On the other hand, there are also discrepancies between the two result sets. Out of the $3,074$ counties that both result sets have in common, our approach finds between 1 and 4 **additional** bordering counties in 42 cases where GeoSPARQL does not register EC relations due to sliver polygons. Even more compelling, our approach returns 6 results that do not appear at all in the GeoSPARQL result set due to the fact that their geometries do not exhibit perfectly precise common boundaries with adjacent features. For instance, due to sliver polygons that are imperceptible to the human eye, GeoSPARQL finds 0 bordering counties for Houston County, Georgia, and, thus, it is not included in the result set, whereas our approach yields all 8 bordering counties; see Figure 3.6. Furthermore, our dataset also materializes a supplementary *barely touches* relation to one of Houston County's bordering counties, Crawford County, Georgia, in order to metrically refine the EC relation. In total, GeoSPARQL failed to capture 78 EC relations between counties.

Finally, querying the broader Web of Linked Data for topological relations by computing them on-the-fly will, at some point, inevitably involve geometries combined from heterogeneous datasets, e.g., by using federated querying or Linked Data aggregators such as the LOD Laundromat[6]. However, this approach to computing topology is fraught with limitations and potential errors due to *dirty* geometries, e.g., the fact that no two sources will digitize the exact same boundaries, and misaligned ontological concepts due to different understandings or modeling decisions about *place types*.

```
# Using our precomputed topological dataset
select ?countyA ?borderingCounties where {
    select ?countyA (count(?countyB) as ?borderingCounties) {
        ?countyA a experiment:County . ?countyB a experiment:County .
        { ?countyA agt:touches ?countyB }
        union { ?countyB agt:touches ?countyA }
    } group by ?countyA
} order by desc(?borderingCounties)
```

Listing 14: Query for all bordering counties using GeoSPARQL's *extensible value testing* function `geof:sfTouches`, which computes the **EC** topological relation on-the-fly.

```
# Using our precomputed topological dataset
select ?countyA ?borderingCounties where {
    select ?countyA (count(?countyB) as ?borderingCounties) {
        ?countyA a experiment:County . ?countyB a experiment:County .
        { ?countyA agt:touches ?countyB }
        union { ?countyB agt:touches ?countyA }
    } group by ?countyA
} order by desc(?borderingCounties)
```

Listing 15: Query for all bordering counties using our `agt:touches` predicate, which represents the **EC** topological relation that was materialized by precomputing topology for all features.

[6] http://lodlaundromat.org/

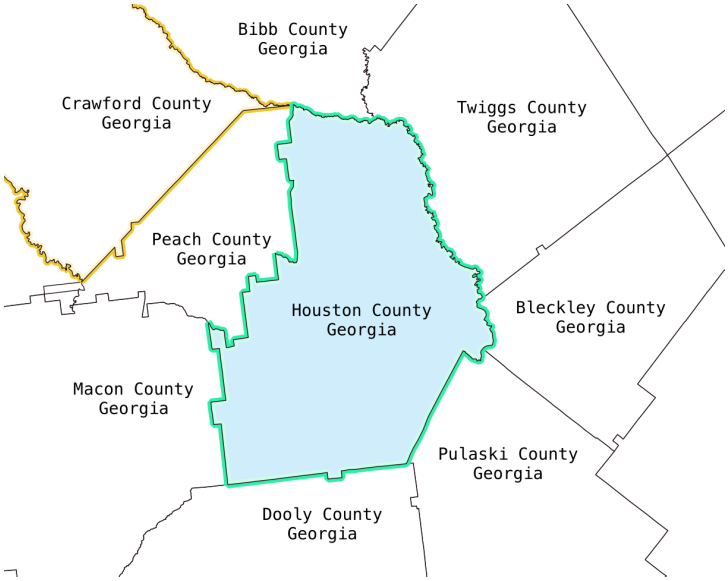

Figure 3.6: Houston County, Georgia shares a border with 8 counties, none of which are captured by GeoSPARQL's `geof:sfTouches` topological operator function due to tiny sliver polygons. Our approach also materializes the `agt:barelyTouches` relation to Crawford County, Georgia.

3.4.4 Topological Queries over Linked Data

There is another benefit to precomputing and materializing topological relations for use in Web-scale knowledge graphs that is less obvious than performance trade-offs and scalability. The fact that topological relations are materialized as object properties in RDF allows users to define custom axioms, such as class assertions, in order to perform topological and subclass reasoning on a geographic dataset. In this section, we provide an example, created to reflect a potential scenario from our dataset, that illustrates the capabilities of topological reasoning as it applies to Linked Data.

In this example, we wish to create a class that identifies parks which would require a traveler to entirely cross through the interior of one or more counties in order to reach the

park's region from a starting location on the containing state's boundary. Conceptually, we assume that the traveler cannot traverse along the zero-width, one-dimensional edges of county boundaries and must therefore be within exactly one county at any given location. We define the axioms in Equation 3.3, starting with the assertion that counties cannot overlap with, nor be within, other counties. From the jointly exhaustive and pairwise disjoint set of relations from RCC8, this implies that counties must either be DC or EC to other counties. We then apply a similar assertion to US states, followed by the axiom for non-tangential counties (`NTC`), which defines counties that are NTPP to a state. Finally, we define our target class, `ParksInNTC` which identifies parks that either only have PO relations to `NTC`s or are a NTPP of an `NTC`.

$$\begin{aligned} \text{County} \sqcap (\exists\ \text{PO.County} \sqcup \exists\ \text{NTPP.County} \sqcup \exists\ \text{TPP.County}) &\sqsubseteq \bot \\ \text{State} \sqcap (\exists\ \text{PO.State} \sqcup \exists\ \text{NTPP.State} \sqcup \exists\ \text{TPP.State}) &\sqsubseteq \bot \\ \text{NTC} \equiv \text{County} \sqcap \exists\ \text{NTPP.State} & \\ \text{ParksInNTC} \equiv \text{Park} \sqcap (\forall\ \text{PO.NTC} \sqcup \exists\ \text{NTPP.NTC}) & \end{aligned} \tag{3.3}$$

Another practical use for Semantic Web technology on topological relations can be to support question answering systems, which typically involve conceptually vague relations to begin with, such as *nearby*. In this example scenario, we translate the question *are there important figures who were born in one city along the Mississippi River and died in a different city along the Mississippi River, and if so, who are they and which cities where involved?* into a query. Here, we attempt to model the relation *along* with a metrically-refined topological relation, *runs alongside* (RAS), defined in Section 3.3.3, in an effort to implement a *naive geographic model* which Egenhofer and Mark [29] propose

as the first stage in a feedback loop that ideally aligns formal models with intuitive human perception. The SPARQL query is shown in Listing 16, while the result is illustrated graphically in Figure 3.7.

```
select ?person ?placeBorn ?placeDied where  {
    ?placeBorn a :City .  ?placeDied a :City .

    dbr:Mississippi_River ?interactsA ?placeBorn .
    values ?interactsA { agt:touches agt:crosses agt:nearlyMeets }

    dbr:Mississippi_River ?interactsB ?placeDied .
    values ?interactsB { agt:touches agt:crosses agt:nearlyMeets }

    filter(?placeBorn != ?placeDied)

    service <http://dbpedia.org/sparql/> {
        ?person a dbo:Person ;
            dbo:birthPlace ?placeBorn ;
            dbo:deathPlace ?placeDied .
    }
}
```

Listing 16: SPARQL query for persons who were born in a city along the Mississippi River and died in a different city along the Mississippi River using a *Federated Query* to combine our topological dataset with DBpedia's knowledge graph.

3.5 Conclusions and Further Work

Publishing massive geographic datasets with complex geometries as Linked Data requires balancing several trade-offs. One family of trade-offs is concerned with the question of which properties to compute on-the-fly, i.e., during query time, and which to store in pre-computed form. In this work, we argued why topological relations (and

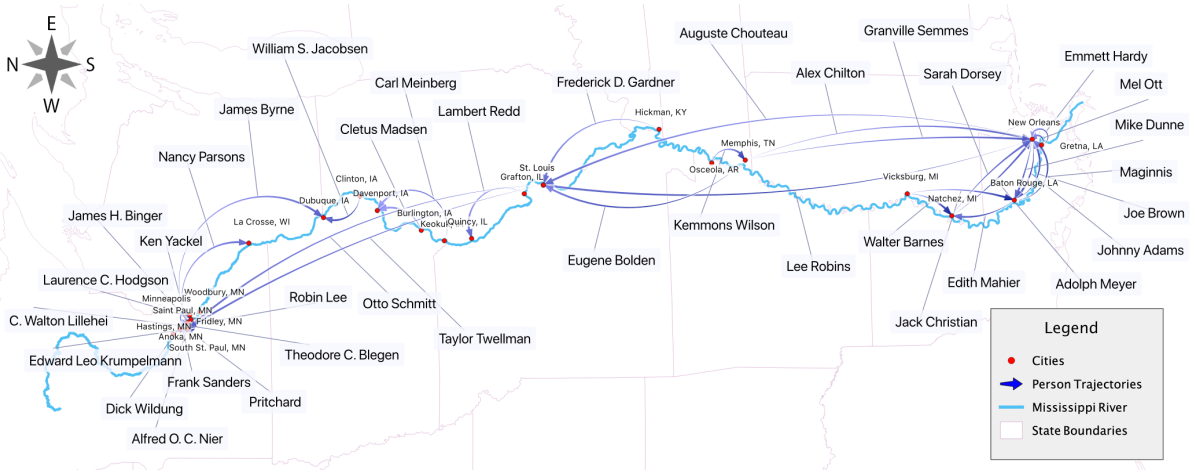

Figure 3.7: A map of the trajectories of persons who were born in a city along the Mississippi River and died in a different city along the river.

queries involving them) often cannot be computed during query time, despite being supported in theory by GeoSPARQL. Additionally, GeoSPARQL and related approaches only support a subset of relationships relevant for everyday queries. Following Egenhofer and Mark's slogan that *topology matters, metric refines*, we compute polygon-polygon, polygon-polyline, and polyline-polyline topological relations for several feature types such as cities, parks, and roadways in DBpedia. As DBpedia does not contain complex geometries but merely points, we derive the geometries from aligning DBpedia entities with OpenStreetMap. On top of strict topological relations, here RCC8, we also compute approximate and metrically-refined relations. Interestingly, both approximate and metrically-refined relations have not been studied in the geospatial semantics literature before and no ontologies or datasets have been published. We present a variety of interesting findings such as how to detect classification errors in DBpedia and

how to validate existing adjacency rations. Finally, we give examples for queries enabled by our approach and compare their runtime and results with GeoSPARQL. From a big picture perspective, our work contributes to finding the right balance between cases where complex geometries should be made available as Linked Data and cases where providing point data enriched by topological relations computed based on these complex geometries is sufficient. We provide the source code to our custom computational framework at https://github.com/blake-regalia/awesemantic-geo, along with a live SPARQL endpoint of the materialized dataset which can be queried using a web interface at http://yasgui.org/short/Lp1v0cYL4.

Future work will focus on computing topological relations for the full USGS Digital Line Graph dataset and publishing them as Linked Data. We also hope to integrate the current dataset with DBpedia. We also aim at developing a full ontology for strict, approximate, and metrically-refined relations, an addition to the subset presented in the current work. Finally, as it is difficult to find a context-independent definition for the range of broad boundaries and even more so for metrically-refined topological relations, we plan to introduce a second provenance graph, e.g., using PROV-O with additional axioms that model uncertainty, that enables users of topologically linked data to understand the individual design decisions that went into creating the data.

Code	Types	Description
	L	Refers to (Multi)Polyline geometry types.
	G	Refers to (Multi)Polygon geometry types.
	E	Refers to *either* of the two aforementioned geometry types.
		Crisp Boundary Relations for G/G pairs – RCC8 [18]
DC	G/G	Disconnected
EC	G/G	Externally Connected
PO	G/G	Partially Overlaps
EQ	G/G	Equals
TPP/i	G/G	Tangential Proper Part ∪ Tangential Proper Part Inverse
NTPP/i	G/G	Non-Tangential Proper Part ∪ Non-Tangential Proper Part Inverse
		Crisp Boundary Relations for L/G pairs – as used by Formica [35].
TCH	L/E	Touches
PTH	L/G	Passes Through
INC	E/L	Inclusion
		Crisp Boundary Relations for L/L pairs [36].
CRS	L/L	Crosses
TCS	L/L	Touch Crosses ⊂ *TCH*
		Broad Boundary Relations – as defined by Clementini et al. [16].
nM	E/G	Nearly Meets ⊂ *DC*
nCt	G/G	Nearly Contains ⊂ *PO*
nE	G/G	Nearly Equals ⊂ ($PO \cup TPP/i \cup NTTP/i$)
		Metrically-Refined Topological Relations
mW	G/G	Mostly Within ⊂ *PO*: the area of intersection is greater than or equal to 80% of P_1's area.
bT	G/G	Barely Touches ⊂ *EC*: the spheroidal length of the intersecting boundary is less than $10m$.
RAL RAS	L/E	Runs Along (L/L), Runs Alongside (L/G): the area of intersection between the features' broad boundary buffers is greater than some threshold value as described in Section 3.3.2.
CON	L/L	Connects ⊂ *TCH*: at least one of the points where the polylines intersect is colocated with one of the points that either polyline starts or ends.

Table 3.1: Topological operator codes as defined by related works as well as our custom *metrically-refined* operator codes. P_1 refers to the polygon with lesser area and P_2 the polygon with greater area.

												avg. area of...	
region-region		**EQ**	**EC**	**PO**	**TPP/i**	**NTPP/i**	**nE**	**nM**	**nCt**	**mW**	**bT**	*smaller polygon*	*larger polygon*
park-park		1	220	9	10	49	0	84	0	3	4	$477km^2$	$3,952km^2$
park-city		0	283	160	79	740	0	120	14	47	291	$22km^2$	$617km^2$
park-county		0	516	512	135	1,645	0	15	1	74	439	$411km^2$	$4,971km^2$
city-city		0	11,827	48	58	189	0	386	0	20	27	$65km^2$	$170km^2$
city-county		1	6,768	1,046	3,397	12,496	0	84	5	280	880	$40km^2$	$2,694km^2$
county-county		0	9,117	0	1	9	0	25	0	0	0	$2,048km^2$	$3,302km^2$

Table 3.2: Number of region-to-region relations materialized for each place type combination by row, and each topological relation by column using codes defined in Table 3.1.

									avg. length/area of...	
polyline-region		**PTH**	**TCH**	**INC**	**nM**	**bT**	**RAS**		*polyline*	*polygon*
road-park		137	984	155	13,928	10	11		$316km$	$1,169km^2$
road-city		3,072	17,302	3,303	19,528	106	100		$425km$	$137km^2$
road-county		7,041	5,597	3,751	4,579	213	9		$383km$	$2,739km^2$
stream-park		156	220	123	1,303	3	5		$293km$	$4,180km^2$
stream-city		708	2,516	285	2,973	106	382		$408km$	$258km^2$
stream-county		1,502	1,491	1,718	828	118	241		$418km$	$4,221km^2$

Table 3.3: Number of polyline-to-region relations materialized for each place type combination by row, and each topological relation by column using codes defined in Table 3.1.

								avg. length of...	
polyline-polyline		**CRS**	**TCS**	**CON**	**nM**	**RAL**		*shorter polyline*	*longer polyline*
road-road		9,861	2	658	100,790	65		$20km$	$127km$
road-stream		4,109	0	84	7,922	94		$79km$	$556km$
stream-stream		12	0	237	4,573	2		$5km$	$24km$

Table 3.4: Number of polyline-to-polyline relations materialized for each place type combination by row, and each topological relation by column using codes defined in Table 3.1.

Chapter 4

Low-Cost Publishing and Decentralized Querying of Linked Open Data using Intelligent Peers

Submitted for Peer Review	
Title	The Poor Man's Endpoint: Low-Cost Publishing and Decentralized Querying of Linked Open Data using Intelligent Peers
Authors	Blake D. Regalia
Institutions	STKO Lab, Department of Geography, University of California, Santa Barbara, USA
Venue	IEEE Transactions on Knowledge and Data Engineering
Editors	Xuemin Lin, Lei Chen
Submit Date	March 26, 2020
Copyright	Blake D. Regalia

4.0.1 Abstract

The vision for decentralized Linked Data suggests that any contributor should be able to make an equally important impact on the cloud of Linked Open Data. In practice however, many attempts to contribute valuable, long-lasting datasets have gone by unnoticed or unmaintained due to the technical and financial challenges associated with initializing and maintaining public endpoints. The Geospatial and Semantic Web communities have largely grown accustomed to service-oriented architectures for hosting, curating, and querying datasets. This has placed the expectation on contributors of original datasets to host their own endpoint for others to use free of charge, often placing burden on the publishers to seek or provide hosting in order to ensure their work has a chance to gain traction in the community. In this work, we propose a low-cost, decentralized solution for preparing, hosting, downloading and querying large-scale knowledge graphs on the 'client' and without the need for a hosted service. Specifically, our use case will focus on *geographic* knowledge graphs that contain many spatial objects. We replace both servers and clients with one class of agent — the *intelligent peer* — which uses peer-to-peer communication to publish and download datasets while incidentally enhancing dataset availability as demand scales. We complement our approach with a novel binary encoding framework that leverages Linked Open Data to share and reuse encoding schemes. Finally, we demonstrate the feasibility of our approach with a proof of concept implementation on geospatial knowledge graphs.

4.1 Introduction

OpenStreetMap (OSM) is a free and open, community-driven project that collaboratively produces a constantly updated vector-based street map of the world. All of the project's datasets, services, and wikis are hosted entirely on donated resources. In 2019, the OpenStreetMap Foundation spent approximately £46,000 on technical operations such as hardware, hosting and connectivity [87]. OSM's map tile content delivery network, which serves rendered map tiles of the dataset's latest stable version to end-users, collectively delivered approximately 500 terabytes of data each month [86]. Although this project has now achieved critical mass to ensure its continued funding and support, the service-oriented architecture it is based on poses many challenges to newcomer projects aimed at publishing datasets that represent, make use of, or link to geographic information, especially in the form of knowledge graphs.

For instance, the most frequent access of Linked Data on the Web for non-human consumption is made through the use of SPARQL endpoints [77], which has been widely regarded as a prominent yet problematic infrastructure for querying Linked Data in practice for several reasons. In reality, a majority of endpoints suffer from extremely limited availability and reliability. One survey found that out of 427 sampled public endpoints, only one third showed an availability rate above 99%, and nearly one quarter were "always down"[13]. A common source of these issues can be traced back to the fact that many open data publishers operate under limited budgets and with limited personnel. Unfortunately, the attention needed to initialize and maintain public endpoints, to host

and disseminate datasets, and the technical difficulties surrounding the deployment of scalable web services can quickly expend the resources that data publishers are capable of allocating.

To make matters worse, existing enterprise solutions typically require high-end resources to process spatial graph queries and are limited in terms of the query loads, i.e., the number of concurrent users combined with the complexity of their spatial graph queries, that they can reliably handle. Cloud computing offers a solution here, with elastic instances that grow or shrink dynamically depending on service demands. However, this approach depends on IT professionals in order to set up and deploy such services and requires a substantial budget if the providers intend to meet high demands. Otherwise, ordinary endpoints will eventually experience low availability and poor reliability, leading to frequent query and network timeouts. Furthermore, the presumption that endpoints that see limited use are not at risk of such issues is nullified by the fact that such endpoints remain a single point of failure. Power outages, network interruptions, hardware faults and software glitches are all very real crises that threaten the availability of a single system.

Finally, the Geospatial Semantic Web has focused entirely on service-oriented architectures for hosting, curating, and querying geodatasets along with their metadata. However, these solutions do not scale easily and their public-facing services are frequently challenged by availability. Public server-side web APIs have historically offered very simple query interfaces to clients. For example, typically REST APIs for data repositories

allow users to query tabular data by specifying value filters on certain fields. These types of query interfaces are well-defined and have a predictable, linear impact on server workload. SPARQL on the other hand grants users the ability to issue arbitrarily complex graph queries making it much more difficult to anticipate server workload. Spatial graph queries in particular seem to be either too difficult to optimize properly or too expensive for most query engines to reliably handle on large geographic knowledge graphs that contain many high-resolution geometries [93, 56]. To give a running example, we show that the query *show me the 10 nearest features to Lake Tahoe that are of type 'stream'* can pose a significant burden on geospatial triplestores representing the USGS National Map as Linked Data, a large-scale geographic repository with many high-resolution geometries.

In this paper, we expand on previous work [91] by exploring an alternative to the aforementioned service-oriented architecture, one that offers low-cost publishing while preserving scalable querying of Linked Open GeoData. We compare and contrast this architecture to *intelligent clients* as popularized by the authors of Linked Data Fragments [106, 107], an approach that vastly improves the availability of endpoints by shifting intelligence to the client. However, rather than relying on the intelligence of a remote system (the server in the case of Linked Data Fragments), an *intelligent peer* is capable of processing queries all on its own. Rather than sending queries to a single source, intelligent peers download fragments of a dataset from any number of *seeds*, which may be a mix of static file servers and fellow peers.

The novel research contributions of this paper are as follows:

- **Purely Peer-to-peer Prepare, Publish, and Process Workflow (P5 Workflow)**. We introduce a workflow solution that promotes the low-cost publishing and decentralized querying of Linked Open Data using an *intelligent peer architecture*.
- **Linked Open Encodings**. We create a novel binary encoding framework and complementary ontology that enables a network of open and extensible data serialization structures to encode knowledge graphs. The ontology is used to define and annotate the byte-level composition of data serialization structures used in the framework, as well as to link them to encoder and decoder implementations.
- **Specialized Data Tables**. We devise a compression and consumption pattern for creating compact, self-indexing, user-defined serializations of datatypes in order to optimize querying, reduce storage size and offer custom query operators.

In short, we study how an entirely peer-to-peer based infrastructure for publishing and querying Linked Data would reduce costs for data publishers and improve the accessibility and reliability of query endpoints for data consumers. To realize this goal, we develop a set of open-source JavaScript modules and their RDF descriptors, following the Linked Open Encodings framework, for creating and consuming binary encoded knowledge graphs entirely within the Web browser over peer-to-peer networks.

This paper is organized as follows. In Section 4.2, we discuss the merits of a decentralized architecture for preparing, hosting, downloading, and querying Linked Open Data

in a novel peer-to-peer workflow that uses networks of *intelligent peers*. Next, in Section 4.3, we propose our novel binary encoding framework for compressing knowledge graphs into queryable *binary bundles*. Then, in Section 4.4, we put those ideas into practice by instantiating a proof of concept which includes the `brisk` implementations for creating and consuming binary bundles, specialized data tables for efficiently handling large amounts of datatyped RDF literals, a comparison to other approaches, and a brief evaluation. Finally, we review our contributions, conclude on the viability of our approach, and enumerate the research directions left for future work in Section 4.5.

4.1.1 A Note on Mutability

It is important to note that one drawback to our approach is that it does not readily support modifying the original dataset, e.g., `UPDATE` queries, since there is no longer a single, centralized host. However, in the wild, we observe virtually all public endpoints disabling such mechanisms since they must control for quality and fend off vandalism by employing techniques such as user access control, version control, and replication. We therefore focus on the publication of open data, and specifically consider the use-case for Linked Open GeoData, which does allow users to download, modify and republish a dataset autonomously. Consumers are therefore free to issue `UPDATE` queries against their own copies and then share those revisions with peers. This is typical for any peer-to-peer data management system which usually employs *forking* to track and maintain changes to a dataset. We intend to study options for update mechanisms in future work.

4.2 Intelligent Peer Architecture

In this section, we describe the mechanics of our proposed decentralized architecture. Namely, we discuss how compressing, hosting, downloading, and querying geospatial knowledge graphs can be realized entirely using provisional swarms of *intelligent peers*.

4.2.1 Peers, Swarms and Seeds

Peer-to-peer (P2P) communication refers to the exchange of information between active, interconnected nodes on a network. A collection of peers cooperatively sharing a specific file or distributing a certain workload is called a *swarm*. An application may distribute its workload evenly across the swarm, or may use the swarm to boost storage potential or network bandwidth for its participants.

In P2P file sharing, if all peers were to disconnect from a swarm, then the subject file would disappear from the network and subsequent new peers would not be able to download the resource. For this reason, it is common to support a file with many redundant *seeds*, which are peers that remain on the network indefinitely for the sole purpose of providing initial upload to new peers when the swarm is sparse.

In our novel intelligent peer architecture, we no longer have a client and a server since all actors are capable of fulfilling the same role. Instead, all actors are considered peers. Peers are capable of participating equally in the tasks of uploading, downloading, and querying datasets. In practice, some peers will persist online for the sole purpose of providing initial upload for new peers when the swarm is sparse. Such peers take on

the designation of *seeds*. Seeds are not servers. Servers provide clients an entrypoint to a centralized service through an endpoint. Seeds on the other hand do not provide a service and are architecturally decentralized, e.g., no seed acts as the master node.

In order to establish a network of intelligent peers for distributing content, it is necessary to maintain at least one seed which remains online to initiate the swarm between periods of inactivity. However, the seed itself does not need to be intelligent. The sole responsibility of a seed in our proposed architecture is to host a static file that can be served over HTTP/S or Websocket. Compared to a service-oriented architecture, not only does this avoid the need for setting up and maintaining a hosted service, but it also opens the door for publishers to take advantage of the many free cloud storage services on the Web. In the BitTorrent community, these are commonly referred to as *Web seeds*.

Given the number of free or low-cost cloud storage options available, it is also reasonable for a publisher to create multiple persistent seeds to redundantly support the same dataset and thus overcome the risks of a single point of failure. We provide a diagram of our proposed architecture in Figure 4.1.

One of the key benefits to our approach over a service-oriented architecture is that the availability of a dataset scales as its demand increases. For hosted services on centralized systems, demand has the opposite effect on availability. On a P2P network, the higher the demand is for a particular dataset, the more bandwidth the content delivery network will have. Since demand for a particular dataset will vary significantly over time, and the fact that swarms are formed independently without coordination from the data publisher,

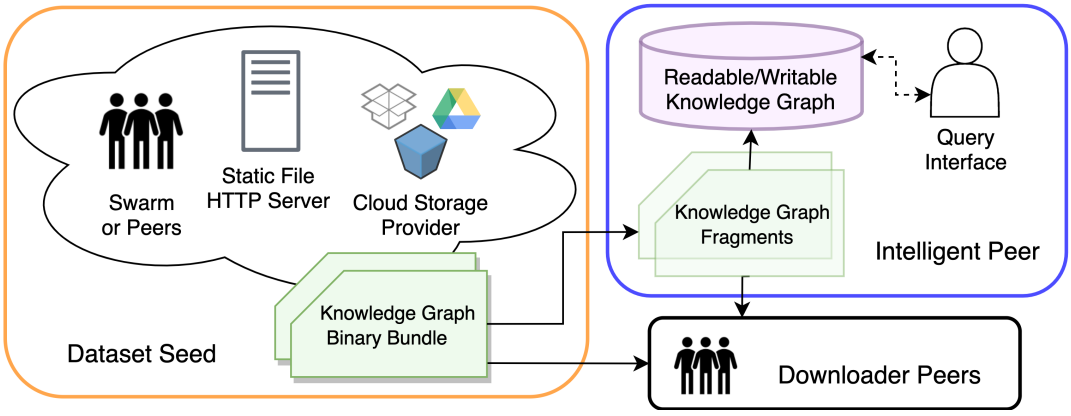

Figure 4.1: Diagram of the intelligent peer architecture for publishing, downloading, sharing, and querying a knowledge graph. Seeds upload the dataset to new peers, whom in turn upload the fragments they downloaded to other peers.

these swarms of intelligent peers can be said to be *provisional*. Overall, an intelligent peer architecture drastically reduces the cost required to publish and maintain a public query interface.

4.2.2 P5 Workflow

We envision a novel workflow for the creation and consumption of knowledge graphs called the **P**urely **P**eer-to-peer **P**repare, **P**ublish and **P**rocess Workflow (P5 Workflow). The diagram is shown in Figure 4.2

- It is said to be **Purely Peer-to-peer** since there are no longer actors in the *client* or *server* roles, but only *peers* who are architecturally decentralized and equally capable of carrying out the same range of tasks.

- The **Prepare** phase refers to the compression and encoding of a knowledge graph into a binary bundle intended for querying, incorporating special indexes for classes

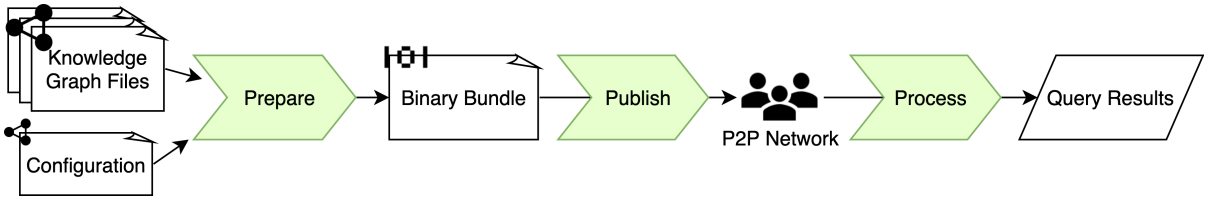

Figure 4.2: Diagram of the **P**urely **P**eer-to-peer **P**repare, **P**ublish and **P**rocess Workflow (P5 Workflow) which follows the path of a knowledge graph from its conception to obtaining query results.

of datatyped information such as spatial objects. Along with the other phases, the tools needed for the Prepare phase must be accessible to the lay user, e.g., via an *intelligent* Web application that takes a set of RDF files and a configuration as inputs then creates a binary bundle as output.

- The **Publish** phase refers to the hosting and subsequent downloading of datasets via peer-to-peer communication.

- Finally, the **Process** phase refers to the data retrieval and query processing executed on the peer, either entirely on the consumer device or via a distributed query mechanism.

In short, the envisioned P5 workflow promises to drastically reduce publishing costs associated with hosting since a P2P content delivery network operates at a fraction of the cost compared to the equivalent centralized approach. Furthermore, it provides the means to overcome the single point of failure problem and improve the reliability of obtaining query results.

4.2.3 Browser-Based Interfaces

With the ongoing advances in Web technologies and Web APIs, a substantial portion of the software stack can now reside in the Web browser. For example, persistent storage, multithreading, network requests, and peer-to-peer communication have all been abstracted by Web standards for the sake of Web applications. The exposed APIs, together with additional third-party libraries that abstract them, let developers focus more heavily on the domain logic of their application rather than the low-level details involved with storage, parallelization and network communication. On the whole, Web applications greatly reduce the friction for developers to target cross-platform compatibility and perform software deployment activities such as release, installation and update, and are gradually superseding the tasks traditionally done exclusively by native applications. While some libraries have been developed to handle RDF storage and querying within the browser [68, 107], none have attempted to operate on binary representations of RDF and thus have been limited to small-scale datasets.

According to OSM's self-reported database statistics as of November 2019, the project sees roughly 2,000,000 new points added per day [83]. It sees roughly 40,000 active users contributing [84], and more than 10,000 new users joining [82], each month. With such a large and active community, it is interesting to note the market share of editing softwares by platform. Recent OSM editor usage statistics report that Web-based editors hold the majority of, and continue to see an increase in, market share by number of users and number of changesets when compared to mobile and desktop platform editing software

[85]. These figures suggest that new users prefer OSM's web application *iD* over native desktop software when it comes to visualizing and editing geospatial data. According to a 2010 analysis[1], more than 78% of HTTP requests to tile servers come from self-reported Web browsers (based on the user-agent HTTP header), meaning that the vast majority of tile usage traffic is serving Web client interfaces. While there are certainly many factors to consider, including the cross-platform advantage that Web applications have, by and large, users of geographic data prefer Web applications over native desktop software since they are able to start using the interface right away, i.e., Web applications don't require any download, install and setup phases which normally incur some burden on the end-user [22, 5]. The lesson here is that users commonly prefer Web applications for visualizing and editing geographic data, and that Web applications offer a streamlined deployment processes for collaborative, open-source development on the Web. In addition, the large ecosystem of browser-based tooling[2] that support the use of Semantic Web technologies provide much of the foundation for map-based geospatial Linked Data interfaces. For these reasons, we believe that a spatial knowledge graph application should ideally support 100% of the tasks, such as preparing, hosting, downloading, and querying, within the Web browser.

We envision our approach to ultimately exist as an underlying software platform for Web applications by replacing the functions of distant SPARQL endpoints. Currently, we have implemented a minimal proof-of-concept that is capable of consuming a spatial

[1] https://wiki.openstreetmap.org/wiki/Talk:Tile_usage_policy#User_Agent_2
[2] https://github.com/semantalytics/awesome-semantic-web

knowledge graph bundle in fragments and executing partial queries. In future work, we intend to implement a full SPARQL query engine and mutable triplestore situated in the browser.

4.3 Linked Open Encodings

In this section, we propose a binary encoding framework for knowledge graphs that allows for a configurable assortment of open and extensible data serialization structures. The framework leverages Linked Open Data to define and describe the byte-level composition of each data serialization structure, link to dereferenceable implementations that can encode or decode them, and reuse existing structures or interfaces by nesting them as fields.

4.3.1 Binary Encoding Motivation

In general, *binary* formats simply refer to those not represented in human-readable text. Not to be confused with structured versus unstructured data, text documents may encode information that is both human-readable and machine-readable, such as XML, JSON, YAML, and so forth. Whether binary or textual, all machine-readable encoding schemes rely on two software components: an encoder and a decoder [69]. Depending on the type of data and its application, these software components are also sometimes referred to as serializer and deserializer, writer and reader, stringifier and parser, or creator and consumer [46].

In pursuit of our research objective to study the costs and benefits of intelligent peer query interfaces, one of the earliest questions we are faced with is how to store and transmit the information necessary to carry out some arbitrary query on a knowledge graph. In a client-server architecture, the client is typically requesting the server to send data that matches some given pattern, and the server is responding with a selection of data or query results. In an intelligent peer architecture, whether or not query processing is distributed, each peer needs to be able to carry out the full range of query processing tasks including data retrieval. Data retrieval is the part of the query process concerned with obtaining select data from a structured data source. Indexes are often used to enhance query performance by making access and search operations more efficient during data retrieval.

While designing an intelligent peer query interface for our proof of concept, one of our primary objectives is to attain query performance comparable to that of traditional endpoints. To realize this objective however, the mechanics of data exchange are no longer merely a means to an end like they typically are in a client-server architecture. Rather, the specific structures used to serialize data, and the strategies used to retrieve them, take on a much more pivotal role.

For one thing, the shape of the data being stored and the usage patterns of the target application are two variables that affect which data serialization *structures* will make for optimal data retrieval [74, 31]. On the Web of intelligent peers, there are several additional variables that affect optimal data retrieval such as network connection qual-

ity, device memory, and device processing power. These variables affect which retrieval *strategies* will perform best under the given circumstances. In order to account for all of these variables, rather than searching for a one-size-fits all solution, **we propose an encoding framework that allows publishers to configure the structures used to serialize knowledge graph data for downstream querying while allowing consumers to independently and dynamically configure their retrieval strategy**.

The intended utility of such an encoding framework is to create binary files that bundle together several self-indexing, compact data structures to encode a knowledge graph for downstream querying by intelligent peers. We therefore distinguish between two parties, one party prepares a dataset and another party consumes it. This setup allows the encoder and decoder to function independently. For example, in order to create a binary bundle, a dictionary encoder might employ the trie data structure, a.k.a., a prefix tree, before serializing all values into a front coded, a.k.a., incrementally encoded, dictionary. However, the corresponding decoder will not need to employ the same data structures in order to read values from and make use of the serialized data. In fact, the decoder may even only need to access fragments of the whole data structure in order to carry out some task. In these circumstances, we prefer to use the terms *creator* and *consumer*, which more accurately describe the roles of the encoder and decoder in our application. To recap, the *creator* is responsible for preparing, compressing, and ultimately serializing a knowledge graph to a binary bundle based on some encoding

scheme, whereas the *consumer* is responsible for carrying out operations defined by some programming interface by accessing and interpreting fragments of the binary bundle based on some encoding scheme.

4.3.2 Container Format

On the practice of designing binary encoding schemes, there are many factors that influence the pace of obsolescence. New features, new techniques, and new hardware instructions all contribute to an ever-changing format that quickly inundates developers with layers of backwards compatibility logic [102, 11]. To further complicate the matter, the process of selecting the most efficient combination of data structures to encode a knowledge graph even depends on the shape of the data and the usage patterns of the target application [74, 31].

In order to adapt to the shape of the data or the demands of the target application, an encoding scheme will often use some reserved bits of control information to identify which data structure is being used out of some limited set of options supported by the format. This same 'reserved control bits' technique is also used to identify different versions of an evolving data structure [32]. As we pointed out earlier, this technique can burden developers with tracking and maintaining backwards compatibility logic on the decoding side. More importantly however, it prescribes encoders with a limited set of data structures to choose from when in reality there could be many more alternatives that better suit the needs of an individual data publisher.

On the other hand, some binary formats employ the concept of *containers*, which simply wrap and describe some payload(s) whose specific encoding scheme can vary. For example, media file formats use containers to embed various configurable codecs for audio and video tracks [55, 79]. As a result, certain types of codecs are interchangeable and new codecs can be deployed without updates to the container format.

In computer science, a graph is an abstract data type that can be implemented using many different data structures [75]. When it comes to the serialization of knowledge graphs for downstream query tasks, one must not only encode the graph structure but also its metadata and the contents of IRIs and datatype properties. In order to support the unlimited range of potential data structures that can encode knowledge graphs, as well as their myriad combinations, **our encoding framework uses a simple container-based binary format. The purpose of the container is simply to provide a consistent, data-agnostic mechanism for describing the size and structure of some serialized payload**.

4.3.3 Data Serialization Structures and Interfaces

From the application perspective, serialized data has two important properties: the types of values stored, and the set of operations to be supported on those values [46]. In many cases, the specific data serialization structures used to encode data are concealed by APIs that abstract them, allowing application logic to operate agnostic of the serialization details [70]. To give an example, the structure of the basic `brs:Dataset.DQ` encoding

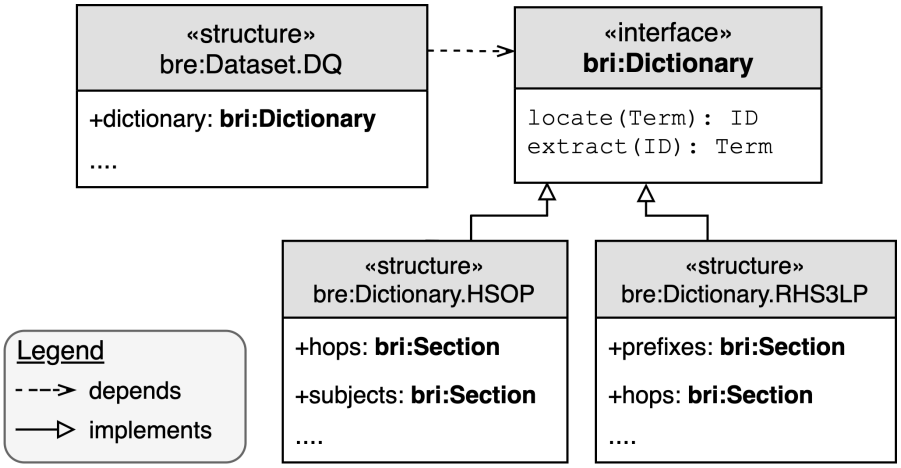

Figure 4.3: Class model diagram representation of the encoding structure for the `brs:Dataset.DQ` which contains a *dictionary* member of type `bri:Dictionary`. This creates a dependency relation to the `bri:Dictionary` interface, which is implemented by the two interchangeable structures `brs:Dictionary.HSOP` and `brs:Dictionary.RHS3LP`.

scheme contains a dictionary and quadruples member. While the specific data serialization structure used to encode the dictionary member can vary, the types of values it stores and the operations supported by the dictionary decoder must remain fixed. In fact, the `brs:Dataset.DQ` encoding simply defines its structure by composing the interface definitions of its members. An interface definition describes some set of requisite properties and procedures. In this example, the dictionary member depends on the `bri:Dictionary` interface, meaning that the implementing structure must be able to store the full range of RDF term types and its corresponding decoder must support the `extract` and `locate` procedures.

It is important to note that these abstractions exist at the data model level, i.e., for defining the structure of the encoding scheme, and are consequently reflected in the programming environment by the encoder and decoder implementations. In an object-

```
|--------- Header ---------|---- Payload ----|
 [1:LPU8-String] [2:VUint]   [3:ByteSequence]
```

Figure 4.4: The serialization structure of the Linked Open Encodings container is represented here as a sequence of 'primitive' datatypes. **1:** the IRI that identifies the payload's data serialization structure, e.g., "https://brisk-rdf.io/structure/Dataset.DQ". **2:** the byte length of the upcoming byte sequence, e.g., '261250'. **3:** the byte sequence payload for the given data structure.

oriented programming environment, these abstractions would manifest as procedural data abstractions (i.e., abstract classes or interfaces), whereas in a functional programming environment they would manifest as abstract data types.

To describe the byte-level structure of our container format, we first define two encoding primitives: a variable-width unsigned integer (VUint), and a length-prefixed UTF-8 encoded string (LPU8-string). The LPU8-string is composed of a single VUint at its start, which describes the number of bytes occupied by the string that follows, and then the string itself using the UTF-8 character encoding. The physical structure of the container format is shown in Listing 4.4, which comprises a header section immediately followed by the payload. The header is comprised of two elements: (1) an IRI encoded as an LPU8-string that uniquely identifies the encoding of the embedded payload and (2) a non-negative integer encoded as a VUint that describes the size of the payload in bytes.

The encoding scheme of each payload is modeled in RDF using the Linked Open Encodings (LOE) ontology, visualized in Figure 4.6. The model defines and annotates the byte-level composition of the data serialization structure. This includes the position,

data type, byte size range, and human-readable description of each field. Aside from primitive data types such as strings, integers, arrays, bitfields, etc., another acceptable field type is the LOE *interface*. In those cases, the data model links a field descriptor to an interface which defines the properties and procedures that the implementing structure must support.

In addition to uniquely identifying the encoding of the embedded payload, the IRI also functions as a dereferenceable location to fetch metadata about the payload's encoding scheme in RDF. The machine-readable resource defines its encoding structure, describes the types of its fields, and links to the default encoder and decoder implementations. In theory, this design pattern allows consumers to fetch a suitable decoder implementation (e.g., as a JavaScript module) for a particular data structure at the same time they are decoding the bundle, i.e., without knowing them beforehand. This novel feature decouples the data in each binary bundle from the software that can interpret them. For example, Bob can publish his dataset using structures authored and hosted by Charlie. Later, Alice, who has never known of Charlie's decoders before, finds Bob's dataset and is able to interpret it using Charlie's decoders by dereferencing the URIs embedded in the bundle. We illustrate this example scenario in Figure 4.5.

The data structures are therefore *extensible* in the sense that anyone can create and deploy a new encoding scheme and corresponding decoder within the Linked Open Encodings framework. We see this as a vitally important feature to an open binary format for encoding knowledge graphs, enabling researchers and developers to experiment with

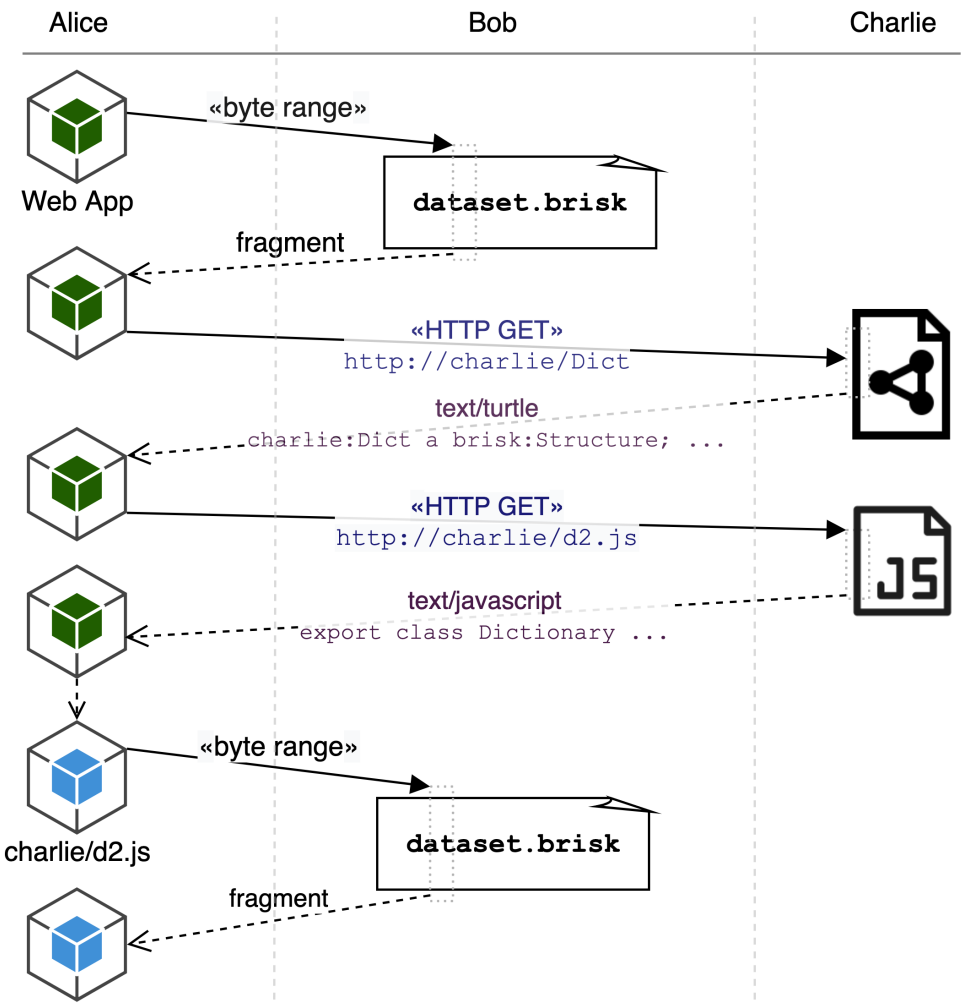

Figure 4.5: An example scenario illustrating the use of Linked Open Encodings, whereby Alice discovers, within Bob's published dataset, a serialization structure developed and hosted by Charlie (`charlie:Dict`). Alice then downloads and subsequently applies a linked implementation that can decode that structure (`charlie/d2.js`).

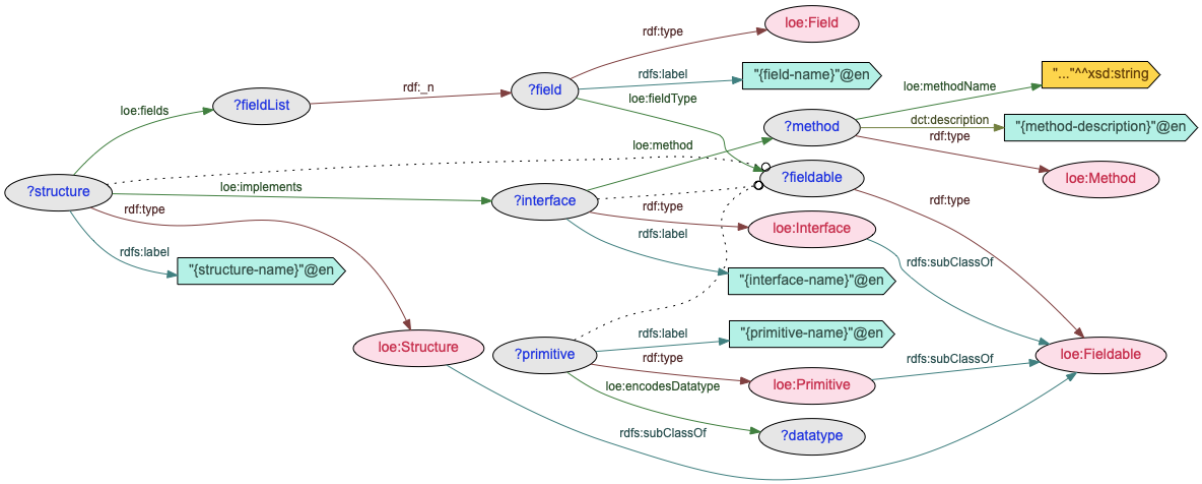

Figure 4.6: Visual overview of the Linked Open Encodings (LOE) ontology which relates data serialization structures to their fields which may either be an interface, a primitive datatype, or another structure.

new storage techniques or data structures and subsequently publishing them for reuse.

Configurability is one of the principles of our proposed container format. We encourage developers to compartmentalize their data structures into logical units in order to maximize reuse and configurability. This ethos provides data publishers options when selecting which data structure to use for a particular interface. For example, the optimal encoding of a dictionary can depend on its intended use. Data publishers can select which specific data structure or encoding scheme to use to best fit their needs as long as the options abide by some common interface. Likewise, clients can select which decoding consumer to use for a particular encoding scheme, e.g., one implementation may prioritize search and query speed, while another is optimized for unmarshalling the data into a mutable store to enable editing.

4.3.4 HDT, HDT-FoQ and WaterFowl

HDT, and its query-enhanced variant **HDT-FoQ** [32, 73], is a binary RDF format designed for the efficient storage, exchange, and consumption of large RDF datasets. HDT-FoQ followed up on the original work by introducing optional indexes and a few alternative data structures to optimize querying. HDT encodes triples and the contents of RDF terms by employing a collection of *succinct data structures* which are data structures that store information using close to as few bits as possible while still allowing direct access to its elements or for efficient query operations. Through the use of succinct data structures, HDT successfully developed the first practical means to query a compact, read-only data dump of an RDF dataset. **WaterFowl** [21] further improved upon aspects of HDT-FoQ by prioritizing query performance and query capability above compression. Most notably, WaterFowl encodes T-Box statements such as class hierarchies using a dedicated index, allowing for efficient inferencing at query time.

In the experiment section of this paper, our primary objective is to enable efficient 'client'-side querying of large geospatial knowledge graphs. To this end, we take some liberties with the data structures developed and tested by previous works with the intention of improving query performance under the unique 'serverless' circumstances. In other words, since our application performs all query execution logic in the user's web browser, we are constrained by the resources of client devices and the limitations of their web browsers. This leads to a different set of assumptions and priorities for the decoding consumer compared to those taken on by HDT and WaterFowl. In those approaches,

it is assume that the entire binary file is available in memory, or at least on local disk via paging. However, our approach demands that each peer be able to carry out queries by operating on only fragments of a remotely stored file which incurs nontrivial network costs for each access request. We therefore modify or extend aspects of the data structures proposed by previous works with the goal of optimizing query performance on remotely accessed fragments. These modifications and extensions are driven by heuristics and although we do not provide a thorough evaluation of their performance by comparing them to their alternatives, we remind readers that the research contribution of this section is about the generalized container format and the Linked Open Encoding framework. The encoding schema presented here is merely an instantiation of this framework which openly allows for potentially more efficient structures to replace them.

4.3.5 A Default Encoding Schema

Having established the generic binary container format for encoding arbitrary data structures, our next step is to define a complete set of default encoding schemes, i.e., the default *encoding schema*, along with their corresponding decoder implementations. Keep in mind that we call this the *default* schema since we simply need an initial set of data serialization structures to encode the entire contents of a knowledge graph as proof of concept, and that future work will evaluate the various trade-offs between different data structures in the context of an intelligent peer architecture (which again can depend on the shape of the data being compressed and the usage patterns of the target application).

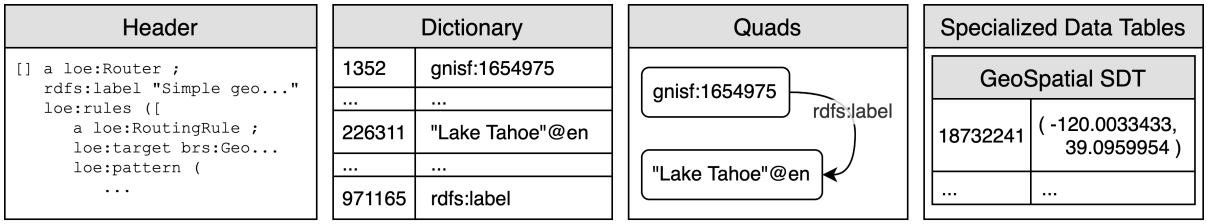

Figure 4.7: The layout of the four members in the `brs:Dataset.HDQS` container, shown here with excerpts from our running example of Lake Tahoe. The Header includes a router definition in RDF for determining where a given term is stored, the Dictionary stores RDF terms by ID, the Quads member stores information about triples using term IDs, and the GeoSpatial SDT stores GeoSPARQL point coordinates in a k-d index by ID.

In this section, we describe a selection of data structures used in our default schema which are primarily based on those proposed by the authors of HDT.

We begin with root structure `brs:Dataset.HDQS`, which encodes the following members using nested LOE containers: **H**eader information describing metadata about the dataset, **D**ictionary for encoding RDF terms, **Q**uads for encoding the graph structure, and finally, **S**pecialized data tables. In Figure 4.7, we shown an illustration of this structure using our running example of Lake Tahoe. This layout follows HDT very closely with the exception of the additional specialized data table member which we describe in detail in Section 4.4.1.

The first major difference between our approach and HDT is in the encoding of the dictionary, which stores all RDF terms present in the dataset and maps them to an ID space. More specifically, we further compress all IRIs using prefix codes and a lookup table to speed up string comparison during binary searches, drop the labels of blank nodes to reduce the number of access requests during query translation and result binding, and divide RDF literals into three separate sections, *datatyped*, *languaged*, and *simple*, instead

of using a single section for all literals. The authors of HDT also in fact divided literals into three sections in a precursor to their HDT publication [72]. Furthermore, we encode the language tags and datatype IRIs at the beginning of the string for those literals, resulting in terms with the same language or datatype occupying contiguous ranges in the sorted list. This lends well to indexing by language or datatype as the index only needs to store the ID range of such literals.

The `locate` operation on a dictionary accepts an RDF term object (i.e., the type of term and its value) as input and returns the corresponding ID if the term exists in the dataset, or `0` if it does not. In a plain front-coded dictionary data structure, the `locate` operation performs a binary search on a sequence of blocks, comparing the search target string to the block's first word in each test. The string comparison must take place by comparing each character one at a time from the beginning of the string in order to determine if the binary search should branch left or right. For a dictionary consisting entirely of IRIs with many of them sharing common prefixes, these lengthy string comparisons can incur an unnecessary, nontrivial cost. In order to reduce the time it takes to make each IRI string comparison, we compress all IRIs using a simple pattern to split the string into a prefix and suffix at the last occurrence of the '/' or '#' character (see the regular expression in Lst. 17). Next, assign or lookup an ID for the prefix string, encode it using a fixed-width byte sequence, and then concatenate the suffix string to this byte sequence. IRIs that do not match our compression pattern are assigned the reserved prefix ID 0. We refer to the prefix byte sequence as the *prefix code*

and store the associations between prefix strings and their code in a separate table. Upon reviewing the literature, we found that the authors of TripleBit [108] employ a nearly identical approach of splitting IRIs into a prefix and suffix with the distinction that our split pattern uses the last occurrence of either the '/' or '#' character. Compared to TripleBit's approach which splits at the last occurrence of only the '/' character, our pattern simply captures more prefixes following the common namespace convention.

```
/^(.+?)([^\/#]+)$/
```

Listing 17: The pattern used to split IRI strings into a prefix and suffix, represented here as a regular expression in PCRE syntax. The expression translates to: *split the string after the last occurrence of the '/' or '#' character.*

In practice, the number of unique prefixes contained by an RDF dataset is relatively small compared to the number of distinct IRIs. For this reason, we instruct peers to proactively cache the entire prefix table as a hash map into memory upon initialization, meaning that `extract` operations on the prefix table occur in constant time $O(1)$. Furthermore, we advise applications to create a mapping from user-defined prefix strings to dataset-defined prefix codes, allowing the `locate` operation to bypass prefix expansion in most cases, avoiding the $O(\log n)$ binary search that would normally be required to match a namespace IRI string.

Since the number of prefixes contained by dataset is indeterminate, we also encode the global byte width of a dictionary's prefix code so that the `extract` algorithm on IRI sections knows how many bytes to slice when reconstructing the full IRI string.

Similar to HDT, we create separate sections for terms based on their role. For ex-

ample, nodes (i.e., the union of IRIs and blank nodes) that appear in both the subject and object position among the dataset's triples are stored in a 'common subject-object' section, which we simply refer to as *hops* (analogous to routing nodes in a computer network). This approach allows query engines to immediately deduce if a discovered node in triples data has incoming or outgoing triples by comparing the ID value to the cached table of ID ranges.

In order to conserve bits, HDT assigns predicates to a separate ID space than all other terms, and shares the subject and object ID space. Since these ID spaces never co-occur in the same data structures used for serialization, this practice normally does not lead to any conflicts. However, the same does not hold true during querying. Since any type of term can bind to a query variable, the data structures used to store *intermediate* query results must either resolve every term ID to a term object, or introduce an additional field to distinguish which ID space a binding result ID corresponds to. To get around this issue, we simply concatenate these shared ID spaces into a global ID space and call it the Localized Term Index (LTI), which associates a unique ID to every term in the dataset over the range `[1, |T|]`, where `T` covers all terms in the dataset. The LTI is used at API borders to pass and return references to terms. Internally and on the wire, predicates retain their bit-conserving ID space over the range `[1, |P|]` within the triples data. The same can be said for subjects and objects sharing an ID space. The LTI is therefore not applicable to the serialization structure but rather to the query API. The LTI space is organized as follows:

- **Hops** (H) occupy the ID space [1, |H|] covering all terms that appear in both the *subject* and *object* positions.

- **Subjects** (S) occupy the ID space [|H|+1, |H|+|S|] covering all terms that appear exclusively in the *subject* position.

- **Object Nodes** (O) occupy the ID space [|H|+|S|+1, |H|+|S|+|O|] covering all *named nodes* and *blank nodes* that appear exclusively in the *object* position.

- **Literals Simple** (M) occupy the ID space [|H|+|S|+|O|+1, |H|+|S|+|O|+|M|] covering all *simple literals* which are literals that have the datatype IRI *http://www.w3.org/2001/XMLSchema#string*.

- **Literals Languaged** (G) occupy the ID space [|H|+|S|+|O|+|M|+1, |H|+|S|+|O|+|M|+|G|] covering all *literals with a language tag.*

- **Literals Datatyped** (D) occupy the ID space [|H|+|S|+|O|+|M|+|G|+1, |H|+|S|+|O|+|M|+|G|+|D|] covering all remaining *literals with an explicit datatype.*

- **Predicates** (P) occupy the ID space [|H|+|S|+|O|+|M|+|G|+|D|+1, |H|+|S|+|O|+|M|+|G|+|D|+|P|] covering all terms that appear in the *predicate* position, regardless of whether or not they appear in the subject or object position as well.

As we mention above, another discrepancy between our serialization structure and HDT's is that ours drops the labels of blank nodes during compression. Instead, we store

two VUints to delimit the range of each blank node section, i.e., one for each hop, subject and object role. During the `extract` operation, the label of a blank node is created by using its LTI thus providing a unique label to each blank node in order to avoid label collisions. For example, `extract(3)` might return the blank node `'_:b3'`, and similarly, `locate('_:b3')` would return `3`. This practice reduces the number of access requests during query translation and result binding since blank node labels do not need to be retrieved.

Apart from the aforementioned differences, the rest of our serialization structures are virtually identical to those proposed by HDT. Namely, we employ *Bitmap Triples* [32] using AdjacencyLists and Bitmap375s [43] to encode triples data.

4.4 Proof of Concept

So far, we have proposed a binary encoding framework for compressing knowledge graphs and an intelligent peer architecture for preparing, publishing and processing datasets. In this section, we demonstrate those approaches by instantiating a proof of concept that puts these ideas into practice.

4.4.1 Specialized Data Tables

In an effort to support the efficient storage and retrieval of geospatial data within a binary knowledge graph encoding, as well as to support geospatial query operations on those data at query time, we developed a novel technique to interface with datatyped RDF

literals while allowing them to be stored and indexed by an arbitrary structure suited for their intended usage. We refer to these extensible components as *specialized data tables* (SDTs), as they are specially designed to store and search a specific type of data. In this experiment, we create a specialized data table for geospatial point features, but remind readers that this approach can be generalized to handle any type of data.

Functionally, SDTs implement the `bri:Dictionary` interface and separately store RDF terms that are not covered by the main dictionary. In other words, SDTs are responsible for carrying out the `locate` and `extract` operations for the RDF terms that they store. This design has two negative consequences: (1) the dataset performs roughly one extra comparison on average in order to determine how to `extract` an RDF term given its ID, i.e., whether to forward the `locate` call onto the main dictionary or to one of the specialized data tables, and (2) the dataset performs roughly one extra comparison on average to determine how to `locate` the ID of a given RDF term, i.e., by routing datatyped RDF literals based on their datatyped IRI using a hash table in $O(1)$ time complexity. Given that the `locate` and `extract` operations only occur during query translation, result binding, and in some cases advanced filters (e.g., matching regular expressions on the contents of literals), we accept this marginal performance hit under the assumption that the overall query performance gains are far more cost effective in the long term. Keep in mind that the overall query performance gains also depend on the specific types of data structures used by the specialized data table and how those compare to the naive structures employed by the main dictionary for encoding and searching text

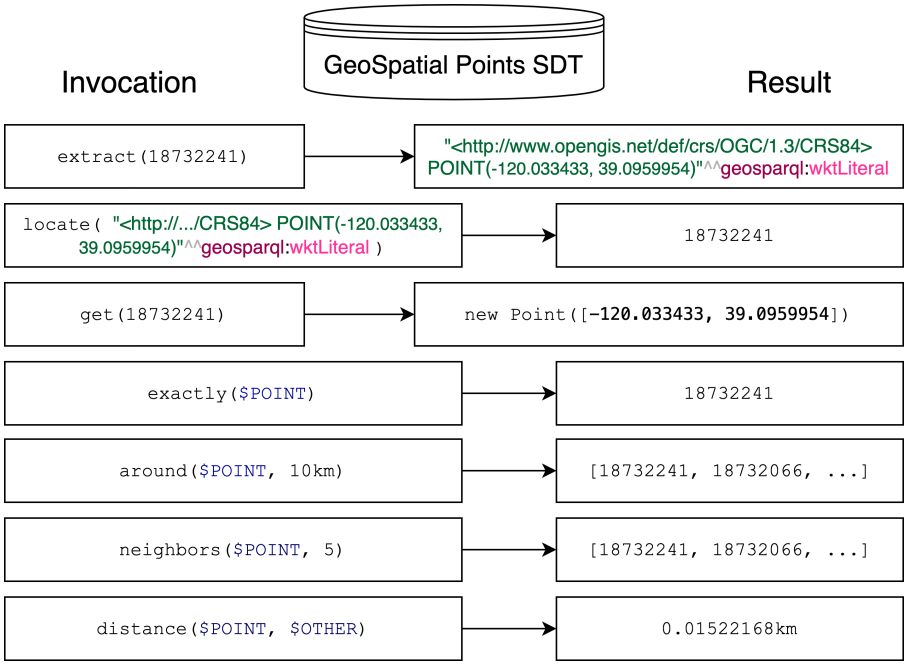

Figure 4.8: An illustration of the operations that can be executed on our geospatial SDT using the ID, RDF term, and spatial object representation of the geometry for Lake Tahoe.

strings.

In this experiment, we create a specialized data table for geospatial point features that supports nearest neighbor search. Following the GeoSPARQL standard for representing geometries in RDF, our geospatial SDT employs a k-d tree to index the coordinates of all point features. During the `extract` procedure, the SDT reconstructs the GeoSPARQL-compatible well-known text (WKT) string of the point feature using the EPSG 4326 coordinate reference system (CRS) based on WGS84. An example is shown in Fig 4.8.

A single GeoSPARQL point feature would normally occupy approximately 30 bytes on average in the main dictionary as a front-coded RDF literal, but occupies only 16 bytes (two 64-bit double precision floating point numbers) in our geospatial SDT. The

storage savings would be much more significant for complex spatial feature types such as polylines and polygons.

In addition to the requisite `locate` and `extract` dictionary operations, our geospatial SDT also supports spatial operations for finding nearby points using a distance radius or k nearest neighbors, ordering results by ascending geographic (spheroidal) distance. This helps answer spatial queries such as, *show me all streams within 10km of Lake Tahoe*. All operations are asynchronous in case the data need to be fetched from the remote resource. The `around` and `neighbors` operations yield results via asynchronous iterators, allowing the user to pull results one at a time, indefinitely, and without the use of a list to buffer results.

The keen observer might notice that SDTs create a possible dilemma for the `locate` operation since the query engine must either try searching both the main dictionary and each specialized data table, or have some rules for determining where to search for a given term. In our `bri:Dataset.HDQS` implementation, we exercise the latter by enabling the data publisher to define a *router*, i.e., a list of rules, which describes which specialized data table of those present, if any, should store a given RDF term. For example, our geospatial SDT is designed exclusively for storing geospatial point features represented as GeoSPARQL WKT literals. Notice how this rule must also exclude WKT literals which are not points, e.g., polylines and polygons. This rule therefore not only matches the datatype IRI of a given RDF literal, but also checks the string contents of the WKT literal to match the POINT keyword. The router is defined in RDF and serialized

into the header section of the `brs:Dataset.HDQS`. An example router that makes use of our geospatial SDT can be seen in Listing 18. In order to avoid redundancies and ID conflicts, the router is required during both dataset creation and consumption. At the time of writing, we only support routing datatyped RDF literals based on their datatype IRI and literal contents.

```
@prefix loe: <https://linked-open-encodings.net/ontology/> .
@prefix loe-dt: <https://linked-open-encodings.net/datatype/> .
@prefix brs: <https://brisk-rdf.io/structure/>

[] a loe:Router ;
  rdfs:label "Simple geospatial example router"@en ;
  rdfs:description "Routes GeoSPARQL WKT literals"@en ;
  loe:rules ([
      a loe:RoutingRule ;
      loe:target brs:GeoSpatialPoints ;
      loe:pattern (
        [ loe:termType loe:DatatypedLiteral ]
        [ loe:datatype geosparql:wktLiteral ]
        [ loe:contents "/^\\s*<http:\\/\\/www\\.opengis\\.net\\/def\\/crs\\/OGC\\/1\\.3\\/
      ↪ CRS84>\\s*POINT\\s*\\(/i"^^loe-dt:regexp
        ]
      ) ;
  ]) .
```

Listing 18: An example router in RDF, shown here using the Turtle syntax. This router uses a single rule to test if a given RDF term is a datatyped literal, has the `geosparql:wktLiteral` datatype IRI, and has literal contents matching a regular expression that indicate a GeoSPARQL WKT string for a point feature in the EPSG:4326 CRS. If a term passes all those tests, it is routed to the `brs:GeoSpatialPoints` SDT during both dataset creation and consumption.

In future work, we plan on using SDTs to provide cost estimates for given invocations during the query planning phase, i.e., in order to provide the query planner with statistics

that help determine the most efficient join order. For example, this component is crucial to the query performance of spatial graph queries that combine spatial operations with basic graph patterns in SPARQL.

4.4.2 Intelligent Peers vs Intelligent Clients

Linked Data Fragments (LDF) and **Triple Pattern Fragments** (TPF) [106, 107, 104] have shown that shifting more intelligence to the client drastically improves service availability by lowering query processing loads on the server. The goals of our approach are very closely aligned with those of Linked Data Fragments, as well as our use-cases, but our approach operates under a fundamentally different paradigm. LDF's *intelligent client* architecture can have many modes and thus the client can take on varying levels of query processing workloads. However, no matter which mode the client operates in, the architecture remains dependent upon a single centralized source of information, i.e., the server. In the case of federated querying, the situation may even be worse since the client is depending upon multiple disjoint centralized sources of information, meaning that if one of the sources is unavailable then the whole query may be prone to fail. While it certainly reduces server workload and improves availability, the intelligent client approach alone does not solve the single point of failure problem and remains susceptible to reliability issues.

The authors of LDF point out that shifting intelligence all the way to client, e.g., by querying data dumps, may have better performance but in doing so loses the ability to

query the Web *live*. However, there are two major weaknesses to this argument. First of all, the average update cycle of any given dataset on the Linked Open Data cloud is rather infrequent. While there are many exceptions including DBpedia Live, a plurality of users seem to be more comfortable with the stability of querying and linking data from snapshots. Secondly, the criticism assumes a legacy workflow of users downloading a dataset to their machine manually and does not take into account a management system capable of delivering and acting upon notifications about update events. Finally, one could argue that compared to LDF, the preprocessing steps required by our approach to publish a dataset hinder the update cycle. However, one of the highest performance options for a TPF backend that ensures improved server availability operates on static HDT files. In this scenario, the publisher needs to regenerate the HDT files anyway and on a rolling basis in order to keep the LDF clients up-to-date. Our approach essentially cuts out the middleman TPF server and asks directly for the compressed dataset file.

As we previously mentioned, in our intelligent peer architecture, we no longer have a client and a server since all actors are capable of fulfilling the same role. Instead, all actors are considered peers. Peers are capable of participating equally in the tasks of uploading, downloading, and querying datasets. Instead of relying on another system, all actors hold the ability to carry out the full range of query processing tasks. Note that this architecture does not *preclude* distributed querying whereby less capable devices ask others to assist with query processing. Rather, all actors maintain the *ability* to carry out their own query processing. In practice, persistent seeds are needed to provide

initial upload for new peers when the swarm is sparse. Seeds are not servers. Servers provide clients an entrypoint to a centralized service through an endpoint. Seeds on the other hand do not provide a service and are architecturally decentralized, e.g., no seed acts as the master node. Architectural decentralization is the fundamental difference between intelligent clients and intelligent peers, as it makes it possible for an intelligent peer architecture to overcome the single point of failure problem and thus substantially improve query reliability.

Another collection of approaches worth mentioning are the number of open-source JavaScript libraries for storing and querying RDF triples in the Web browser[3]. These libraries are certainly capable of serving an application in an intelligent peer architecture and we share their enthusiasm for the cause. However, the libraries provide the functionality of a read-write triplestore and consequently face severe limitations in terms of the number of triples they can reliably handle, in terms of the query performance they can deliver, or both. Since the vast majority of public endpoints disable anonymous updates, or have an entirely separate mechanism for creating, updating or generating their knowledge graphs, we see these libraries as serving a different set of use-cases than endpoints meant for querying large-scale, read-only knowledge graphs.

[3] https://www.w3.org/community/rdfjs/wiki/Comparison_of_RDFJS_libraries#Data_storage_libraries

4.4.3 Query API

We implement our proof of concept as a JavaScript module for the Web browser. The module exposes an API that allows users to match triple patterns, perform merge joins on chain-shaped graph patterns, conduct indexed searches on specialized data, and filter or transform results with predefined or user-defined functions. These capabilities are foundational to creating a fully-developed RDF query engine and triplestore. However, further work is required to support all the query mechanics needed to evaluate SPARQL.

Since the success of such an approach hinges on the computational resources of end user devices, we design our data retrieval implementation (and by extension the query engine) to function without requiring the entire dataset to be downloaded. The application therefore only needs to download those fragments of the dataset that help answer their query. This strategy enables peer devices to query datasets much larger than they could normally store in main memory while still bearing the entire query processing workload.

The use of self-indexing succinct data structures is critical in this regard as it not only compresses the knowledge graph to reduce transmission costs and increase storage density, but it also provides the means to search the dataset using a series of byte range requests. For example, in our implementation, the *locate* operation performs a binary search on a sorted list to find which block contains the given search term. The algorithm only needs to fetch enough characters from the beginning of a block to determine if the search term is located to the left or right on each iteration. Since the block positions are indexed by a complementary data structure in the dictionary, the peer knows the

exact byte ranges to request from their seed(s) in order to carry out the search. Critics will point out that this strategy demands a high number of sequential network requests which may incur substantial overhead or lag due to the fact they depend on the results of previous requests. However, there are many configurable factors that determine the performance of such systems and these must be evaluated circumstantially. For example, many of the search algorithms can be tuned to make fewer requests at the cost of higher memory consumption or longer processing time by prefetching the next values before they are evaluated (e.g., prefetching both the left and right child nodes during a binary search, effectively reducing the number of requests by a factor of 2). Furthermore, graph patterns allow for highly concurrent matching engines [65] which compensate for idle time lost to asynchronous data access.

Another factor that influences performance caused by network effects is the underlying communication protocol. For example, HTTP range requests are substantially less cost-effective for frequent back-and-forth communication than WebSockets. The bottom line is that while our approach indeed requires more network requests than a simple client-server query endpoint paradigm, it does not preclude us from achieving comparable performance to those systems under real-world circumstances. As we imply above, the scale of network effects can be dynamically tuned by each peer on a case-by-case basis, such as decreasing the number of network requests by using a prefetch strategy during binary searches of sorted lists.

4.4.4 Evaluation

We provide a brief evaluation of our proof of concept by comparing the query performance and storage consumption to two other open-source geospatial triplestores. Apache Jena, the popular Java framework for RDF, recently implemented an extension for GeoSPARQL 1.0 with full compatibility [3]. The project ships an executable triplestore that exposes a SPARQL endpoint over HTTP, named *Apache Jena GeoSPARQL Fuseki (AJGF)*[4]. The other triplestore we use is the GeoSPARQL branch of *Apache Marmotta*[5], which also implements the full GeoSPARQL 1.0 standard and exposes a SPARQL endpoint over HTTP.

We use the GNIS-LD[6] dataset, a Linked Data version of the entire USGS GNIS gazeteer, and a set of queries (two spatial, and one non-spatial) to compare the query performance and storage consumption of each query engine. The GNIS-LD consists of 2.28 million geographic features (each accompanied by point coordinates) spread across 37.4 million triples.

We run all triplestores on the same machine and query them from localhost. The machine has $32.7GB$ of memory, and 12 Intel hyperthreaded processor cores running @$2.90GHz$ each.

[4] https://jena.apache.org/documentation/geosparql/geosparql-fuseki
[5] https://marmotta.apache.org/kiwi/geosparql.html
[6] https://gnis-ld.org

Storage Consumption

For the best possible performance, we instruct the AJGF instance to use an in-memory database for its triplestore, which it allocates on the JVM heap. Once the GNIS-LD is loaded into the AJGF database, the process occupies $30.5GB$ of RAM which includes an index for each permutation of triples as well as a spatial index for spatial objects.

Apache Marmotta uses the relational database, PostgreSQL[7], with the geospatial extension PostGIS[8], as a backend to store triples data and spatial objects with indexes for each. Upon loading the GNIS-LD dataset, the PostgreSQL database occupies $21.5GB$ on disk.

In our approach, we prepare our database by compressing the knowledge graph into a binary bundle which includes the indexed triples data as well as the indexed spatial objects. The resulting binary bundle is $415.2MB$, less than 2% the size of Marmotta's database and less than 1.5% the memory footprint of AJGF's process. Part of the reason for this enormous discrepancy is the fact that the other two databases do not make as strong assumptions about the immutability of their records. In other words, our binary bundle is optimized for read-only querying while theirs must account for the ability to insert or delete triples, resulting in data structures that require much larger storage. Additionally, our approach follows HDT's clever three 'triples data' indexing solution (SPO, OP-S and PS-O), whereas the other two use twice the number of indexes to cover all permutations of triples. Our binary bundle results in a rough storage density of 9,000

[7] https://www.postgresql.org/
[8] https://postgis.net/

triples per MB, or 5,500 features per MB.

Another important metric we can report is the cumulative size of binary fragments that our application ended up downloading in order to answer each test query. This measure has no equivalent to compare against the other triplestores we evaluated since they do not cache results on the client. The application downloaded less than $40KB$ worth of fragments to answer Q1 and Q3, less than 1% of the total dataset, while Q2 downloaded approximately $177KB$ or 4.3% of the total dataset. Given that the average modern smartphone has more than $2GB$ of RAM, **these storage consumption results for querying dataset fragments demonstrate the potential for limited devices to efficiently and reliably query datasets much larger than their memory capacity**.

Query Performance

We author a set of queries, shown in Listing 19, to test two spatial queries including a thematic search for k-nearest neighbors (Q1) and a thematic search combined with a geospatial buffer query (Q2), as well a non-spatial geographic query that uses partonomy (Q3).

The query performance times are shown in Table 4.1 and discussed here. AJGF excels at the non-spatial graph query (Q2), but performs worse than Marmotta for the spatial graph queries (Q1 and Q3). Our implementation outperforms both triplestores for spatial queries (Q1 and Q3) by a wide margin, approximately 15-40x better for cold cache, and 5-30x better for warm cache. AJGF outperforms ours for the non-spatial graph query

	AJGF	Marmotta	Ours
Q1 Cold	38.26s	24.92s	**0.96s**
Q1 Warm	14.8s	10.67s	**0.45s**
Q2 Cold	**5.83s**	60.31s	11.94s
Q2 Warm	**0.3s**	47.67s	5.93s
Q3 Cold	31.38s	11.37s	**0.81s**
Q3 Warm	2.8s	1.72s	**0.41s**

Table 4.1: A comparison of the query performance of Q1, Q2 and Q3, first with a cold cache and then a warm cache, between Apache Jena GeoSPARQL Fuseki (AJGF), the GeoSPARQL branch of Apache Marmotta, and our implementation 'brisk'.

(Q2) by about 2x better time on cold cache, and about 20x better time on warm cache. However, ours still outperforms Marmottta for Q2 by about 5-7x better time.

Since the query performance times will be inevitably slower and vary with greater network latency than our evaluation which was tested on the loopback interface, further research needs to be conducted in order to determine the real-world performance of such systems. Overall, we believe the query performance results shown here demonstrate that intelligent peers may indeed be capable of attaining comparable query performance to that of traditional endpoints.

```
# Q1: 10 nearest streams to Lake Tahoe in California
select ?stream_label ?distance_m {
  ?lake_tahoe rdfs:label "Lake Tahoe"@en ;
    gnis:state gnisf-alias:CA ;
    ago:geometry ?lake_tahoe_geom .

  ?stream a cegis:Stream ;
    rdfs:label ?stream_label ;
    ago:geometry ?stream_geom .

  bind(geof:distance(?lake_tahoe_geom, ?stream_geom, opengis:meter) as ?distance_m)
```

```
} order by asc(?distance) limit 10


# Q2: All summits in Mesa County, Colorado
select * {
  ?feature a cegis:Summit ;
    rdfs:label ?label ;
    gnis:county gnisf-alias:Colorado.Mesa ;
    gnis:elevation [ qudt:unit unit:FT ;
      qudt:numericValue ?elevation ] ;
    ago:geometry ?geom .

  ?geom geosparql:asWKT ?wkt .
}


# Q3: Summits within 10km of Yosemite National Park
select ?summit_label ?distance_m {
  ?yosemite rdfs:label "Yosemite National Park"@en ;
    gnis:state gnisf-alias:CA ;
    ago:geometry ?yosemite_geom .

  ?summit a cegis:Summit ;
    rdfs:label ?summit_label ;
    ago:geometry ?summit_geom .

  bind(geof:distance(?yosemite_geom, ?summit_geom, units:meter) as ?distance_m)
  filter(?distance_m < 10e3)
}
```

Listing 19: The set of queries we use to evaluate the performance of each query engine, represented here using the SPARQL 1.1 language. **Q1.** Selects the 10 nearest streams to Lake Tahoe in California, along with their label and relative distance. **Q2.** Selects all summits, along with their label, elevation, and location, in Mesa County, Colorado. **Q3.** Selects all summits within 10km of Yosemite National Park.

4.5 Conclusions and Future Work

In this work, we set out to investigate a solution for low-cost publishing and decentralized querying of Linked Open Data by rethinking the need for a client-server architecture. We explored what is required to create a system for querying knowledge graphs that pushes intelligence all the way onto the client while working within the capabilities of lower-end devices such as limited memory and processing power. We outlined our vision for the novel Purely Peer-to-peer Prepare, Publish and Process Workflow, in which the preparation, hosting, downloading, and querying of knowledge graphs takes place across provisional swarms of intelligent peers. This architecture significantly lowers the costs associated with publishing a dataset since a P2P content delivery network operates at a fraction of the cost compared to the equivalent centralized approach, i.e., publishers are only concerned with the cost of initial seeds. Additionally, the decentralized architecture circumvents the challenges associated with maintaining a public endpoint including availability since each intelligent peer is capable of carrying out their own query processing tasks.

We proposed a novel binary encoding framework for knowledge graphs, Linked Open Encodings, that allows publishers to configure the data serialization structures used to encode their knowledge graphs in order to better fit the needs of their target application. Consumers are also able to choose which decoder implementation to use for a given serialization structure depending on their needs, e.g., querying a read-only dataset versus transforming a read-write one. Furthermore, consumers are able to dynamically tune

their retrieval strategy depending on the demands of the application and the capabilities of their query-processing system. The framework includes an ontology for describing the byte-level composition of data serialization structures, as well as for linking them to their corresponding encoder and decoder implementations. This allows for the discovery and dereferencing of decoders on-the-fly as peers begin downloading and querying a remote dataset.

Following the Linked Open Encodings framework and ontology, we modeled a set of default serialization structures in RDF and implemented their corresponding encoders and decoders as JavaScript modules for the Web browser. We emphasize that by making all aspects of the P5 workflow accessible from the Web browser, we improve the ease-of-use for a wider range of users, such as for those with less technical skills normally required to deploy their own public endpoint.

We especially considered an approach that would be able to meet the challenges brought on by storing and querying large geospatial databases and created specialized data tables, a novel compression and consumption pattern for creating compact, self-indexing, user-defined serializations of special datatypes. We showed how this approach reduces the storage requirement of the queryable binary bundle and thus improves the size of datasets that end devices can reliably handle. In addition, we showed how this approach enables applications to perform efficient, arbitrary searches on specialized data such as for searching nearest neighbors using a spatial index.

In our evaluation, we demonstrated the feasability of our approach using a proof of

concept that performs all phases of the P5 workflow in the Web browser. We implemented and annotated a complete set of data serialization structures, their encoders and decoders for compressing and indexing geospatial knowledge graphs by leveraging specialized data tables. We showed that our application was able to outperform two popular open-source GeoSPARQL triplestores for spatial graph queries, and exhibited comparable performance for a non-spatial graph query. We conclude that our approach may indeed be able to attain comparable query performance to traditional endpoints under real-world circumstances and thus be a viable competitor.

We believe that the intelligent peer architecture will eventually find traction in the community due to the low-cost solution it offers to publishers and since it does not suffer from the same availability or reliability issues that plague client-server architectures due to its inherent decentralization.

In future work, we intend to overhaul our proof of concept into a stable prototype that supports the SPARQL query language. This would also entail supporting SPARQL UPDATE queries which would require either downloading and storing all triples into mutable triplestore, or an update strategy to record pending edits to the binary knowledge graph bundle, e.g., WaterFowl has proposed one such strategy [20]. We intend to perform a full evaluation of our prototype by comparing the SPARQL query performance to more triplestores using benchmarks such as LUBM[47] and Geographica[39] for geospatial queries. We also intend to evaluate the prototype under real-world circumstances such as longer network latencies, limited device capabilities, and knowledge graphs beyond

1 billion triples. We believe at that time we will be able to more thoroughly compare our approach to Linked Data Fragments including the implications of the intelligent peer architecture such as intelligent peer workload and how the availability of a dataset scales with increased swarm sizes.

We believe another important direction for future work is to scale workloads horizontally on intelligent clients and intelligent peers by delving into parallel computing, including the use of general-purpose computing on graphics processing units (GPGPU). Several works have studied parallel and distributed processing mechanisms for large-scale RDF data [67, 42, 90, 41] including over P2P networks [51, 14, 52, 2], but none have been evaluated in the context of a Web browser environment which presents unique challenges and opportunities such as workload adaptability, fault tolerance to browser eviction, dynamic module loading, and progressive web applications.

In another paper, we hope to evaluate the trade-offs between various data serialization structures in the context of the intelligent peer architecture, i.e., which structures perform best under specific circumstances. We also intend to implement Annotated Triples and Annotated Graphs by HDTQ [33] which adds the ability to support *quads* by storing the additional graph component of each triple in compressed space.

Chapter 5

Conclusions and Future Work

In this dissertation, I motivated the need to rethink how geographic data is facilitated on the Semantic Web, namely, how it is stored, computed, materialized, transmitted, and ultimately queried. I started with the assumption that data publishers and data consumers are hindered by the current state of the Geospatial Semantic Web, and that there exist at least three trade-offs that affect the computational time and space needed to handle geographic knowledge graphs. I studied these trade-offs by proposing a research question for each one. I then answered each research question in its own chapter. I made several novel research contributions that help mitigate the problems outlined by my research questions. Overall, there are many opportunities to optimize the storage, retrieval, and querying of geographic information on the Semantic Web that are not currently being used in practice. In this dissertation, the results of my experiments demonstrate that the proposed solutions to each trade-off can significantly outperform

existing approaches for managing geographic knowledge graphs. I do not advocate for a one-size-fits-all solution and approaches that factor the various needs of publishers and consumers into account will likely see much greater traction within the community. In the following sections, I discuss the conclusions of each chapter and how they answer the proposed research questions.

5.1 Computing Geographic Properties and Relations On-Demand

To answer **RQ1**, we conducted an experiment on computing the properties of, and relations among, select geographic features from DBpedia in an on-demand fashion. We designed an architecture that allows for a transparent SPARQL proxy to sit in between a client and an encapsulated triplestore in order to perform dynamic query rewriting. We designed a novel language for writing procedural-style rules that allow user-defined functions to create so-called magic properties in response to queries. Our approach enabled users to author graph patterns for triples as if they already existed when in reality they may be computed and returned on-demand, i.e., *virtual triples*. We also took advantage of this pattern to construct and annotate computed results in an ad-hoc RDF graph, allowing our proxy to automatically handle caching and cache invalidation.

What we learned is that computing certain geographic properties on-demand will substantially benefit the accuracy, consistency and completeness of statements within

knowledge graphs. Namely, triples that are derived by computing them from adjacent dependent statements can be replaced with rules that describe how to derive them. Not only does this approach generally improve the accuracy of such statements, but it also promises to provide better coverage for datasets with inherent data sparsity such as those derived from extracted texts. We also learned that triples which change frequently over space and time, i.e., volatile statements, should be considered for replacement with rules.

An important quality often overlooked by most related work is the need for transparency when evaluating on-demand computational query results. To ensure reproducibility, users need to be able to inspect the rules used to derive computed results as well as the transactional history of their invocations. This also includes annotating derived statements with provenance information. Finally, we also showed how computable properties offer coverage for relations that can exist between all pairwise combinations of geographic features. The alternative, i.e., materializing all possible relations, leads to a combinatorial explosion in terms of the number of triples, making it infeasible.

5.2 Precomputing Topological Relations

To answer **RQ2**, we conducted an experiment on a heterogeneous geographic linked dataset in order to simulate the real-world circumstances of integrating data across different sources. We combined complex, high-resolution geometries from OSM with the rich feature types and relations sourced from DBpedia. We then computed a set of topological relations for several top-level feature types including populated places, rivers,

roads, and parks. Rather than computing topology using only the strict boundaries of regions, we also applied approximate topological relations using broad boundaries as well as metrically-refined topological relations. We created an ontology, AGO, in order to describe these sets of relations in RDF. Finally, we used the resulting knowledge graph to demonstrate the utility of our topological relations by evaluating a set of queries that normally cannot possibly be answered using GeoSPARQL, including the ability to employ qualitative spatial reasoning.

Contrary to what the current standards for the Geospatial Semantic Web would suggest, geographic question answering relies more heavily on topology than geometries as users are more interested in the relations between features than the properties of their geometric representations. With respect to knowledge graphs, geometries merely serve as a means to an end for deriving higher level statements about the features or the relations between them. However, topology cannot be computed from geometries alone as they often depend on contextual knowledge such as place type. Even in ideal circumstances, computing topology from geometries requires preprocessing in order to correct for geometric discrepancies within heterogeneous geodatabases. Even more compelling, none of the current approaches take into account vagueness and uncertainty principles which have been well understood by Geographers for decades. Topological knowledge needs to support approximate relations by allowing features to have broad boundaries that grow or shrink depending on place type, the type and attributes of the features they are being compared to, as well as the semantics of the relation they are being computed for.

5.3 Storing, Transmitting and Querying Geographic Knowledge Graphs

Finally, we investigated **RQ3** by creating a decentralized query engine for geographic Linked Open Data in the Web browser. We advanced the idea of shifting intelligence to the client by removing the server all together and replacing them with intelligent peers. We envisioned a workflow that enables the low-cost publishing and decentralized querying of geographic Linked Open Data using peer-to-peer communication. We supported this architecture with a novel binary encoding framework that allows the creation and reuse of open, extensible data serialization structures to encode geospatial knowledge graphs. We applied these concepts by developing a set of default serialization structures along with their corresponding encoder and decoder implementations. We also proposed a compression and consumption pattern for creating compact, self-indexing serialization structures for specialized data, such as large collections of spatial objects. We showed how this technique reduces the storage requirements and offers efficient custom search and filter functionality, such as the ability to perform spatial operations, which drastically improves overall query performance. Finally, we demonstrated the feasibility of our approach by evaluating a proof of concept that outperforms existing open-source solutions.

As we hypothesize, an intelligent peer architecture significantly reduces costs associated with publishing since peer-to-peer content delivery networks operate at a fraction

of the cost compared to service-oriented architectures such as those provided by cloud computing infrastructure. Decentralization creates an opportunity to overcome the single point of failure problem and thus achieve greater reliability than centralized client-server architectures typically manifested for Linked Data in the form of public SPARQL endpoints.

The tools for deploying compact, queryable binary data dumps need to provide publishers with the ability to choose which data serialization structures best fit their needs. There is no one-size-fits-all binary serialization format for RDF given the variety in both the shape of data as well as the usage patterns of target applications. Likewise, consumers need to have the freedom to choose which implementation to use for decoding a given serialization structure depending on end-user or application needs, such as whether or not they intend to modify datasets or if they are just interested in querying them. Furthermore, an intelligent peer architecture requires that consumers be able to dynamically adapt their data retrieval strategy depending on factors that are indeterminate to publishers such as network connectivity, device memory, and processing power.

Finally, geospatial knowledge graphs do not receive enough attention when it comes to searching them efficiently in intelligent client interfaces. We proposed one approach that answers this challenge with satisfactory performance for point coordinates, but insist that the problem needs further research to evaluate the trade-offs for more complex geometries and a broader family of spatial queries.

5.4 Future Work

Finally, in this section, I discuss several directions for future work that follow the research conducted in my dissertation.

5.4.1 Provenance of Derived Statements

Several of the experiments in this dissertation revealed the need for a comprehensive ontology that is capable of annotating derived statements, i.e., computed statements, with provenance metadata about how the computations were made, such as: the source triples that the derived statements are based on, the operations that were used, the values of the parameters supplied to those operations, and so on. The vocabularies we developed for the experiments were designed to annotate aspects of derived statements but did so with limited depth. Ideally, such annotations should provide enough transparency to end users so that they are able to reproduce the derived statements. Further work should also align these vocabularies with existing standardized ontologies for provenance such as PROV-O [66] and function definitions such as the Function Ontology [23].

5.4.2 Heterogeneous Geographic Linked Data

While the research innovations presented in this dissertation took steps towards realizing the vision for a Linked Data portal that connects a collection of rich geographic datasets such as the USGS National Map, more tooling needs to be developed around supporting the process of aligning, cleaning, and publishing large geographic Linked Datasets that

combine multiple sources. From the experience of combining OSM, DBpedia, and the USGS Digital Line Graph data, it is clear that there are many opportunities to improve the tooling that researchers and publishers use to align and clean geographic features from multiple sources. Aside from the tooling, I also expect there will be many more optimizations to be discovered from applying the proposed methods at larger scales, i.e., macro-level effects. For example, a dataset that combines daily snapshots of OSM with DBpedia Live will likely present several new challenges. Given the computational complexity required to align and clean features, how will data publishers be able to estimate the growing costs associated with maintaining an up-to-date heterogeneous product? What can be done to improve the efficiency and turnaround time of the update process, such as by using incremental updates? How can providers assess the reliability that their geospatial graph query services assure end users when the queries may be arbitrarily complex and involve any number of features for large datasets?

5.4.3 Tooling for Virtual Triples

The work on VOLT included the development of a language for expressing how to derive statements on-demand. This included procedural definitions for computing triples, as well as expressions for determining which queries from the user should *trigger* those procedures. However, our approach relied on SPARQL query language constructs and consequently, the *triggers* are not defined for all possible types of query patterns, e.g., those with variable predicates. The same can be said for related work. There have

been some improvements on the topic of rule-based virtual triples since the work done in VOLT, but there has been very little development for tooling to support authoring procedural-style rules, especially for users who need to perform computations outside of the SPARQL framework (which is mostly limited to string manipulation and basic arithmetic). Overall, the process for authoring, sharing, and reusing rules for generating virtual triples will benefit from a dedicated programming language that compiles down to RDF and is capable of interfacing with external programs, such as the VOLT language. However, further work is needed to expand the capabilities of the language as well as its translation to RDF.

The best path forward for future work should focus on ensuring that users are aware of the capabilities and limitations of query systems that compute statements on demand. In fact, at least one such solution involves updating the query results interchange format so that results can be accompanied by warnings and disclaimers from the query system in a machine-readable format. For example, several triplestores have exhibited the problematic behavior of returning partial results after a timeout error without any indication that the query was not able to finish. On one hand, the server was able to complete part of the query before timing out and it is useful to obtain partial results rather than none. On the other hand, the user is led to believe that they have a complete answer set since the results did not return an error. Ideally, the system would be able to notify the user of the timeout error in addition to returning partial results. Likewise, in a virtual triple context, the system should be able to indicate to users which parts of their query

triggered inferencing or on-demand computation, as well as which results are based on derived statements.

5.4.4 An Ontology for Fuzzy Topology

Finally, an important direction for future work is to develop an ontology for describing approximate and metrically-refined topological relations. No other approaches on the Geospatial Semantic Web have considered fuzzy topology, but many of the statements extracted from articles on the Web that relate geographic features behave as proxies for fuzzy topological relations. For example, we showed that a majority of the statements relating any two places on DBpedia involve cardinal directions or partonomy, which serve as proxies for the topological relations of adjacency and containment, respectively. Part of the reason we have not seen other attempts to express fuzzy topology may be due to the fact that such relations are difficult to represent since they must encode measures of uncertainty. Furthermore, the semantics of metrically-refined topological relations need to capture vagueness which makes them difficult to model. For example, how exactly should an ontology formally define the meaning of the predicate in the statement *U.S. Route 101 runs along the Pacific Coast*?

Bibliography

[1] SPIN - SPARQL Inferencing Notation, 2011.

[2] Ibrahim Abdelaziz, Essam Mansour, Mourad Ouzzani, Ashraf Aboulnaga, and Panos Kalnis. Query optimizations over decentralized rdf graphs. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 139–142. IEEE, 2017.

[3] Gregory L Albiston, Taha Osman, and Haozhe Chen. Geosparql-jena: Implementation and benchmarking of a geosparql graphstore.

[4] Dean Allemang and James Hendler. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier, 2011.

[5] Ryosuke Aoki, Akihiro Miyata, Shunichi Seko, Ryo Hashimoto, Tatsuro Ishida, Masahiro Watanabe, and Masayuki Ihara. An information display system with information scrapping user interface based on digital signage terminals and mobile devices for disaster situations. In *International Conference on Human-Computer Interaction*, pages 353–363. Springer, 2016.

[6] Sören Auer, Jens Lehmann, and Sebastian Hellmann. Linkedgeodata: Adding a spatial dimension to the web of data. *The Semantic Web-ISWC 2009*, pages 731–746, 2009.

[7] Andrea Ballatore, Michela Bertolotto, and David C Wilson. Grounding linked open data in wordnet: The case of the osm semantic network. In *International Symposium on Web and Wireless Geographical Information Systems*, pages 1–15. Springer, 2013.

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

[9] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[10] Yaser Bishr. Semantic aspects of interoperable gis. ITC, 1997.

[11] Amrit Kumar Biswal and OBADA AL MALLAH. Analytical assessment of binary data serialization techniques in iot context (evaluating protocol buffers, flat buffers, message pack, and bson for sensor nodes). 2019.

[12] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia-a crystallization point for the web of data. *Web Semantics: science, services and agents on the world wide web*, 7(3):154–165, 2009.

[13] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. Sparql web-querying infrastructure: Ready for action? In *International Semantic Web Conference*, pages 277–293. Springer, 2013.

[14] Hyunsik Choi, Jihoon Son, YongHyun Cho, Min Kyoung Sung, and Yon Dohn Chung. Spider: a system for scalable, parallel/distributed evaluation of large-scale rdf data. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 2087–2088, 2009.

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

[16] Eliseo Clementini and Paolino Di Felice. A spatial model for complex objects with a broad boundary supporting queries on uncertain data. *Data & Knowledge Engineering*, 37(3):285–305, 2001.

[17] Eliseo Clementini, Paolino Di Felice, and Peter Van Oosterom. A small set of formal topological relationships suitable for end-user interaction. In *International Symposium on Spatial Databases*, pages 277–295. Springer, 1993.

[18] Anthony G Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 1(3):275–316, 1997.

[19] Max Craglia, Michael F Goodchild, Alessandro Annoni, Gilberto Camara, Michael Gould, Werner Kuhn, David Mark, Ian Masser, David Maguire, Steve Liang, et al. Next-generation digital earth: A position paper from the vespucci initiative for the advancement of geographic information science. *International Journal of Spatial Data Infrastructures Research*, 3(3):146–167, 2008.

[20] Olivier Curé and Guillaume Blin. An update strategy for the waterfowl rdf data store. 2014.

[21] Olivier Curé, Guillaume Blin, Dominique Revuz, and David Célestin Faye. Waterfowl, a compact, self-indexed rdf store with inference-enabled dictionaries. In *European Semantic Web Conference*, pages 302–316. Springer, 2014.

[22] Giulia de Andrade Cardieri and Luciana Martinez Zaina. Analyzing user experience in mobile web, native and progressive web applications: A user and hci specialist perspectives. In *Proceedings of the 17th Brazilian Symposium on Human Factors in Computing Systems*, pages 1–11, 2018.

[23] Ben De Meester, Anastasia Dimou, Ruben Verborgh, and Erik Mannens. An ontology to semantically declare and describe functions. In *European Semantic Web Conference*, pages 46–49. Springer, 2016.

[24] Shihong Du, Qimin Qin, Qiao Wang, and Haijian Ma. Reasoning about topological relations between regions with broad boundaries. *International Journal of Approximate Reasoning*, 47(2):219–232, 2008.

[25] Max J Egenhofer. Toward the Semantic Geospatial Web. In *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, pages 1–4. ACM, 2002.

[26] Max J Egenhofer and Matthew P Dube. Topological relations from metric refinements. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 158–167. ACM, 2009.

[27] Max J Egenhofer and Robert D Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information System*, 5(2):161–174, 1991.

[28] Max J Egenhofer and David M Mark. Modelling conceptual neighbourhoods of topological line-region relations. *International journal of geographical information systems*, 9(5):555–565, 1995.

[29] Max J Egenhofer and David M Mark. Naive geography. In *International Conference on Spatial Information Theory*, pages 1–15. Springer, 1995.

[30] Max J Egenhofer, Jayant Sharma, and David M Mark. 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, 1993.

[31] David C Faye, Olivier Cure, and Guillaume Blin. A survey of rdf storage approaches. 2012.

[32] Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary rdf representation for publication and exchange (hdt). *Journal of Web Semantics*, 19:22–41, 2013.

[33] Javier D Fernández, Miguel A Martínez-Prieto, Axel Polleres, and Julian Reindorf. Hdtq: managing rdf datasets in compressed space. In *European Semantic Web Conference*, pages 191–208. Springer, 2018.

[34] Frederico T Fonseca and Max J Egenhofer. Ontology-driven geographic information systems. In *Proceedings of the 7th ACM international symposium on Advances in geographic information systems*, pages 14–19, 1999.

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

[36] Anna Formica, Mauro Mazzei, Elaheh Pourabbas, and Maurizio Rafanelli. Approximate answering of queries involving polyline–polyline topological relationships. *Information Visualization*, 17(2):128–145, 2018.

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

[38] Mark N Gahegan. Characterizing the semantic content of geographic data, models, and systems. In *Interoperating Geographic Information Systems*, pages 71–83. Springer, 1999.

[39] George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A benchmark for geospatial rdf stores (long version). In *International semantic web conference*, pages 343–359. Springer, 2013.

[40] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, et al. Examining the challenges of scientific workflows. *IEEE computer*, 40(12):26–34, 2007.

[41] José M Giménez-García, Javier D Fernández, and Miguel A Martínez-Prieto. Hdt-mr: A scalable solution for rdf compression with hdt and mapreduce. In *European Semantic Web Conference*, pages 253–268. Springer, 2015.

[42] François Goasdoué, Zoi Kaoudi, Ioana Manolescu, Jorge-Arnulfo Quiané-Ruiz, and Stamatis Zampetakis. Cliquesquare: Flat plans for massively parallel rdf queries. In *2015 IEEE 31st International Conference on Data Engineering*, pages 771–782. IEEE, 2015.

[43] Rodrigo González, Szymon Grabowski, Veli Mäkinen, and Gonzalo Navarro. Practical implementation of rank and select queries. In *Poster Proc. Volume of 4th Workshop on Efficient and Experimental Algorithms (WEA)*, pages 27–38, 2005.

[44] Michael Goodchild, Max J Egenhofer, Robin Fegeas, and Cliff Kottman. *Interoperating geographic information systems*, volume 495. Springer Science & Business Media, 2012.

[45] Michael F Goodchild. Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4):211–221, 2007.

[46] Konrad Grochowski, Michał Breiter, and Robert Nowak. Serialization in object-oriented programming languages. In *Software Design and Modelling*. IntechOpen, 2019.

[47] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.

[48] Olaf Hartig. Reconciliation of rdf* and property graphs. *arXiv preprint arXiv:1409.3288*, 2014.

[49] Francis Harvey. Designing for interoperability: Overcoming semantic differences. In *Interoperating Geographic Information Systems*, pages 85–97. Springer, 1999.

[50] Tom Heath and Christian Bizer. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, 1(1):1–136, 2011.

[51] Felix Heine. Scalable p2p based rdf querying. In *Proceedings of the 1st international conference on Scalable information systems*, pages 17–es, 2006.

[52] Felix Heine, Matthias Hovestadt, and Odej Kao. Processing complex rdf queries over p2p networks. In *Proceedings of the 2005 ACM workshop on Information retrieval in peer-to-peer networks*, pages 41–48, 2005.

[53] Charles Herring. An architecture of cyberspace: Spatialization of the internet. *US Army Construction Engineering Research Laboratory: Champaign, IL, USA*, 1994.

[54] Aidan Hogan, Pascal Hitzler, and Krzysztof Janowicz. Linked dataset description papers at the semantic web journal: A critical assessment. *Semantic Web*, 7(2):105–116, 2016.

[55] J Scott Houchin and David W Singer. File format technology in jpeg 2000 enables flexible use of still and motion sequences. *Signal Processing: Image Communication*, 17(1):131–144, 2002.

[56] Theofilos Ioannidis, George Garbis, Kostis Kyzirakos, Konstantina Bereta, and Manolis Koubarakis. Evaluating geospatial rdf stores using the benchmark geographica 2. *arXiv preprint arXiv:1906.01933*, 2019.

[57] Krzysztof Janowicz. The role of space and time for knowledge organization on the semantic web. *Semantic Web*, 1(1, 2):25–32, 2010.

[58] Krzysztof Janowicz, Yingjie Hu, Grant McKenzie, Song Gao, Blake Regalia, Gengchen Mai, Rui Zhu, Benjamin Adams, and Kerry Taylor. Moon landing or safari? a study of systematic errors and their causes in geographic linked data. In *The Annual International Conference on Geographic Information Science*, pages 275–290. Springer, 2016.

[59] Krzysztof Janowicz, Sven Schade, Arne Bröring, Carsten Keßler, Patrick Maué, and Christoph Stasch. Semantic enablement for spatial data infrastructures. *Trans. in GIS*, 14(2):111–129, 2010.

[60] Krzysztof Janowicz, Simon Scheider, Todd Pehle, and Glen Hart. Geospatial semantics and linked spatiotemporal data–past, present, and future. *Semantic Web*, 3(4):321–332, 2012.

[61] Amit Krishna Joshi, Pascal Hitzler, and Guozhu Dong. Logical linked data compression. In *The Semantic Web: Semantics and Big Data*, pages 170–184. Springer, 2013.

[62] Vipul Kashyap and Amit Sheth. Semantic heterogeneity in global information systems: The role of metadata, context and ontologies. *Cooperative information systems: Current trends and directions*, pages 139–178, 1998.

[63] Christoph Kiefer, Abraham Bernstein, and Markus Stocker. *The Fundamentals of iSPARQL: A Virtual Triple Approach For Similarity-Based Semantic Web Tasks.* Springer, 2007.

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

[65] Alexander Krause, Annett Ungethüm, Thomas Kissinger, Dirk Habich, and Wolfgang Lehner. Asynchronous graph pattern matching on multiprocessor systems. In *European Conference on Advances in Databases and Information Systems*, pages 45–53. Springer, 2017.

[66] Timothy Lebo, Satya Sahoo, Deborah McGuinness, Khalid Belhajjame, James Cheney, et al. Prov-o: The prov ontology. *W3C Recommendation*, 30, 2013.

[67] Chang Liu, Jacopo Urbani, and Guilin Qi. Efficient rdf stream reasoning with graphics processingunits (gpus). In *Proceedings of the 23rd International Conference on World Wide Web*, pages 343–344, 2014.

[68] Antonio Maccioni and Matteo Collina. Graph databases in the browser: using levelgraph to explore new delhi. *Proceedings of the VLDB Endowment*, 9(13):1469–1472, 2016.

[69] Kazuaki Maeda. Comparative survey of object serialization techniques and the programming supports. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 5(12):1488–1493, 2011.

[70] Kazuaki Maeda. Performance evaluation of object serialization libraries in xml, json and binary formats. In *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, pages 177–182. IEEE, 2012.

[71] David M Mark. Toward a theoretical framework for geographic entity types. In *European Conference on Spatial Information Theory*, pages 270–283. Springer, 1993.

[72] Miguel A Martínez-Prieto, Javier D Fernández, and Rodrigo Cánovas. Compression of rdf dictionaries. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 340–347, 2012.

[73] Miguel A Martínez-Prieto, Mario Arias Gallego, and Javier D Fernández. Exchange and consumption of huge rdf data. In *Extended Semantic Web Conference*, pages 437–452. Springer, 2012.

[74] Ruslan Mavlyutov, Marcin Wylot, and Philippe Cudre-Mauroux. A comparison of data structures to manage uris on the web of data. In *European Semantic Web Conference*, pages 137–151. Springer, 2015.

[75] Kurt Mehlhorn, Stefan Naher, and Stefan Näher. *LEDA: a platform for combinatorial and geometric computing*. Cambridge university press, 1999.

[76] Martin Michalowski, Jose Luis Ambite, Snehal Thakkar, Rattapoom Tuchinda, Craig A Knoblock, and Steve Minton. Retrieving and semantically integrating heterogeneous data from the web. *IEEE Intelligent Systems*, 19(3):72–79, 2004.

[77] Knud Möller, Michael Hausenblas, Richard Cyganiak, Gunnar Aastrand Grimnes, and Siegfried Handschuh. Learning from linked open data usage: Patterns & metrics. *Proceedings of the WebSci10: Extending the Frontiers of Society On-Line*, 159, 2010.

[78] OPEN STREETMAP SEMANTIC NETWORK. Osm semantic network–openstreetmap wiki [online] htt. *wiki. o enstreetma. org/wiki/OSM Semantic Network [Accessed: 30 june 2015]*.

[79] Jan Newmarch. Sound codecs and file formats. In *Linux Sound Programming*, pages 11–14. Springer, 2017.

[80] Axel-Cyrille Ngonga Ngomo. On link discovery using a hybrid approach. *Journal on Data Semantics*, 1(4):203–217, 2012.

[81] Barry Norton and Reto Krummenacher. Consuming dynamic linked data. In *COLD*, 2010.

[82] OpenStreetMap contributors. Openstreetmap database statistics. new contributors per month. https://wiki.openstreetmap.org/wiki/File:New_contributors_month.png, 2020. Accessed: 2020-01-25.

[83] OpenStreetMap contributors. Openstreetmap database statistics. number of nodes added each day. https://wiki.openstreetmap.org/wiki/File:Osmdbstats7A.png, 2020. Accessed: 2020-01-25.

[84] OpenStreetMap contributors. Openstreetmap database statistics. number of users editing per month. https://wiki.openstreetmap.org/wiki/File:Osmdbstats4A.png, 2020. Accessed: 2020-01-25.

[85] OpenStreetMap contributors. Openstreetmap editor usage statistics. https://wiki.openstreetmap.org/wiki/Editor_usage_stats, 2020. Accessed: 2020-01-25.

[86] OpenStreetMap contributors. Openstreetmap servers/tile cdn: Traffic estimates. https://wiki.openstreetmap.org/wiki/Servers/Tile_CDN#Traffic_estimates, 2020. Accessed: 2020-01-25.

[87] OpenStreetMap Foundation. Finances/treasurer's report for the december 2019 agm. https://wiki.osmfoundation.org/wiki/Finances/Treasurer%27s_Report_for_the_December_2019_AGM, 2020. Accessed: 2020-01-25.

[88] Kostas Patroumpas, Giorgos Giannopoulos, and Spiros Athanasiou. 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*, pages 301–310. ACM, 2014.

[89] Matthew Perry and John Herring. OGC GeoSPARQL - a geographic query language for rdf data. *OGC Implementation Standard. Sept*, 2012.

[90] Padmashree Ravindra, HyeongSik Kim, and Kemafor Anyanwu. An intermediate algebra for optimizing rdf graph pattern matching on mapreduce. In *Extended semantic web conference*, pages 46–61. Springer, 2011.

[91] Blake Regalia. Decentralizing the persistence and querying of rdf datasets through browser-based technologies. 2017.

[92] Blake Regalia, Krzysztof Janowicz, and Song Gao. Volt: a provenance-producing, transparent sparql proxy for the on-demand computation of linked data and its application to spatiotemporally dependent data. In *International Semantic Web Conference*, pages 523–538. Springer, 2016.

[93] Blake Regalia, Krzysztof Janowicz, and Grant McKenzie. Revisiting the representation of and need for raw geometries on the linked data web. In *LDOW@ WWW*, 2017.

[94] Dave Reynolds, Jeni Tennison, and Leigh Dodds. Linked Data API, 2012.

[95] Vivek Sehgal, Lise Getoor, and Peter D Viechnicki. Entity resolution in geospatial data integration. In *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pages 83–90. ACM, 2006.

[96] Amit P Sheth. Changing focus on interoperability in information systems: from system, syntax, structure to semantics. In *Interoperating geographic information systems*, pages 5–29. Springer, 1999.

[97] Barry Smith and David M Mark. Ontology and geographic kinds. 1998.

[98] Sebastian Speiser and Andreas Harth. Integrating linked data and services with linked data services. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC'11)*, pages 170–184. Springer, 2011.

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

[100] Bas Stringer, Albert Meroño-Peñuela, Antonis Loizou, Sanne Abeln, and Jaap Heringa. To SCRY Linked Data: Extending SPARQL the Easy Way. In: Diversity++; ISWC 2015. 2015.

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

[102] Ralf Vandenhouten and Thomas Kistel. A model-driven concept for the automatic integration of legacy protocols to distributed component-oriented software systems. 2012.

[103] Andrej Vckovski. Interoperability and spatial information theory. In *Interoperating geographic information systems*, pages 31–37. Springer, 1999.

[104] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle. Querying datasets on the Web with high availability. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock,

Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble, editors, *Proceedings of the 13th International Semantic Web Conference*, volume 8796 of *Lecture Notes in Computer Science*, pages 180–196. Springer, October 2014.

[105] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-scale querying through linked data fragments. In *Proceedings of the 7th Workshop on Linked Data on the Web*, 2014.

[106] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-scale querying through Linked Data Fragments. In Christian Bizer, Tom Heath, Sören Auer, and Tim Berners-Lee, editors, *Proceedings of the 7th Workshop on Linked Data on the Web*, volume 1184 of *CEUR Workshop Proceedings*, April 2014.

[107] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple pattern fragments: a low-cost knowledge graph interface for the web. *Journal of Web Semantics*, 37:184–206, 2016.

[108] Pingpeng Yuan, Pu Liu, Buwen Wu, Hai Jin, Wenya Zhang, and Ling Liu. Triplebit: a fast and compact system for large scale rdf data. *Proceedings of the VLDB Endowment*, 6(7):517–528, 2013.

[109] Chuanrong Zhang, Tian Zhao, and Weidong Li. Current and future challenges of geospatial semantic web. In *Geospatial Semantic Web*, pages 167–189. Springer, 2015.

[110] Rui Zhu, Krzysztof Janowicz, Bo Yan, and Yingjie Hu. Which kobani? a case study on the role of spatial statistics and semantics for coreference resolution across gazetteers. In *International Conference on GIScience Short Paper Proceedings*, volume 1, 2016.