Exposing Points of Interest as Linked Geospatial Data

Kostas Patroumpas
IMSI, Athena Research Center, Greece
kpatro@imis.athena-innovation.gr

Dimitrios Skoutas
IMSI, Athena Research Center, Greece
dskoutas@imis.athena-innovation.gr

Georgios Mandilaras
University of Athens, Greece
gmandi@di.uoa.gr

Giorgos Giannopoulos
IMSI, Athena Research Center, Greece
giann@imis.athena-innovation.gr

Spiros Athanasiou
IMSI, Athena Research Center, Greece
spathan@imis.athena-innovation.gr

ABSTRACT

Point of Interest (POI) data is widely used in many modern applications and services related to navigation, tourism, social networking, logistics, and many more. In this paper, we propose a comprehensive and vendor-agnostic data model to represent multi-faceted and enriched POI profiles. Harnessing the versatility of Linked Data technologies, this semantically rich ontology accommodates and extends existing POI formats for assembling and managing POI data from heterogeneous sources. Furthermore, we have developed the open-source software TripleGeo, which can effectively transform POI data from diverse sources and formats (geographical files, databases, and semi-structured data) to their RDF representations and vice versa. Thus, it is possible to import POI data from various existing systems and products, transfer and address the data integration challenges in the Linked Data domain, and export back the results. Our empirical study confirms the validity and efficiency of this framework for a variety of real-world POI assets and formats, underscoring its robustness to cope with scalable data volumes.

CCS CONCEPTS

• **Information systems → Extraction, transformation and loading**.

KEYWORDS

POI, RDF, ontology, linked geospatial data, transformation

ACM Reference Format:
Kostas Patroumpas, Dimitrios Skoutas, Georgios Mandilaras, Giorgos Giannopoulos, and Spiros Athanasiou. 2019. Exposing Points of Interest as Linked Geospatial Data. In *16th International Symposium on Spatial and Temporal Databases (SSTD '19), August 19–21, 2019, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340964.3340976

1 INTRODUCTION

Points of Interest (POIs) are geospatial entities that represent physical locations or constructs of some particular interest or utility. At a minimum, a POI is characterized by its *name*, a *category*, and a *geolocation* (usually represented by longitude/latitude coordinates). But usually POI entities have many more properties, which either describe additional *thematic attributes* (address, contact details, working hours, etc.) or *relationships* to other entities (e.g., a shop within a mall, a dealer POI related to a manufacturer POI). POIs are valuable resources utilized in our everyday lives (e.g., navigation, social networks, tourism) as well as in various commercial domains (such as logistics, advertising, or geomarketing). Among the Big POI Data assets, HERE Places API[1] offers information about 55 million POIs in 237 countries; Google Places API[2] advertises over 150 million POIs globally; from OpenStreetMap, we have extracted more than 18.5 million POIs regarding specific categories[3]. Increasing data volumes and the evolved POI value chain constantly introduce opportunities for growth, but also intensify complexity relating to quality-assured data integration, enrichment, and sharing.

Despite the importance and high commercial value of POI data, diverse models, formats and identifier schemes are currently adopted. POI profiles are highly diverse in the types of attributes they contain. POI data is semantically diverse and spatiotemporally evolving, representing different entities and associations depending on geographical, temporal, and thematic context or application fields. For instance, tourist guides may offer richer descriptions and accessibility information, whereas Yellow Pages contain mostly business-oriented information (classification by industry, contact details, etc.). Lack of standardization in representing and exchanging POI data combined with industrial competition (e.g., mapping or navigation applications) is another important factor. Earlier limitations in Personal Navigation Devices have also restricted the expressiveness, accuracy and completeness of POI schemata and data.

Current solutions only partially handle such issues on a case-per-case level. Several existing tools or services can perform pairwise mappings and transformations from a source schema or format to a target one. However, nearly all of them are dataset- or use-case specific and are mostly used to convert from one conventional POI format to another, instead of supporting transformations to a global schema. Finally, due to their ad-hoc nature, efficiency against diverse POI models and scalability with larger datasets is problematic.

However, *Linked Data* technologies can provably address current limitations, gaps and challenges in integrating, enriching, and sharing POI data [1, 2]. In this paper, we provide an account of our experience regarding POI data representation and transformation to/from the RDF data model[4] in real-world, industrial settings.

SSTD '19, August 19–21, 2019, Vienna, Austria

ACM ISBN 978-1-4503-6280-1/19/08…$15.00
https://doi.org/10.1145/3340964.3340976

[1] https://developer.here.com/products/geocoding-and-search
[2] https://cloud.google.com/maps-platform/places/
[3] http://slipo.eu/?p=1397
[4] Resource Description Framework (RDF): https://www.w3.org/RDF/

First, we introduce a comprehensive, rich, and vendor-agnostic model for exposing POIs on the Linked Data domain in a consistent and unified way, regardless of their original schemata and formats. Our proposed *POI ontology* reflects characteristics identified in a wide spectrum of datasets and representations, but it is also extensible for specific use cases and applications in the industry.

Second, we provide to stakeholders of the POI value chain a unified framework for performing *transformation* and *mapping* of individual and diverse datasets and schemata into RDF adhering to a common POI ontology. We have been developing TripleGeo, an open source, robust, efficient, and extensible software, which facilitates transformation of POI data from a variety of de facto geospatial formats into RDF triples with minimal overhead. This is possible through adaptable, configurable, and reusable mappings from existing schemata into our POI ontology, whilst there is also support for classification hierarchies in assigning categories to POIs. Besides, this software can be also used for *reverse transformation* of the added-value POI entities back to conventional formats for exploitation by services and products in the industry.

Overall, the proposed model and software make it possible to acquire POI data from various existing systems and products into a *POI data integration lifecycle* (for interlinking, fusion, enrichment, quality assurance) based on Linked Data principles and technologies [1]. Besides, its utilization in POI data integration tasks [2] has been guided by commercial POI stakeholders from diverse domains (navigation, geomarketing, tourism) and assessed on real-world data to confirm its interoperability, extensibility, and scalability.

The remainder of this paper proceeds as follows. In Section 2 we survey related work. In Section 3, we introduce our POI ontology and in Section 4 we present our software for transforming POIs from de facto formats into RDF and back. In Section 6 we evaluate our proposed framework against large real-world POI data. Section 7 summarizes our contribution and outlines future extensions.

2 RELATED WORK

Next, we survey existing POI representations to identify common and special characteristics, as well as issues regarding transformation of geospatial entities (and POIs in particular) to RDF.

POI Representation. Towards facilitating development of large-scale applications and services over linked geospatial data, the Open Geospatial Concortium (OGC) has suggested the GeoSPARQL standard[5]. This provides a concrete ontology for representing features and geometries in RDF as literals according to Well Known Text (WKT)[6] or Geography Markup Language (GML)[7] standards by OGC, while also offering a rich collection of query operators. As this standard does not explicitly intend to model POI features, a W3C group (later jointly with OGC) started developing technical specifications for the representation of POIs on the Web. Although still in progress, this representation[8] interestingly distinguishes locations that may refer to the centroid of the POI, its navigation point (e.g. the POI's entrance or parking lot), an area or other type of geo-reference. For each POI, a category element contains an identifier for the classification scheme and a term within this scheme. A POI may also have one or more time primitives that represent an individual point in time, a span of time, or a recurring time or time span. The model also supports *relationships* between POIs, such as contained-within, contains, or adjacent-to. In a separate approach, the SDI4Apps project[9] has developed a SPARQL endpoint based on a POI data model that also takes care of extra attributes like accessibility and internetAccess information, rights regarding license information, date when a POI was created, as well as relationships between POIs, such as sfWithin, sameAs, or exactMatch.

Amongst open POI data sources, OpenStreetMap (OSM)[10] is the most prominent. OSM geometry primitives include nodes, ways, and relations, as well as a free tagging system, whereby *tags* associate metadata to the map objects as key-value pairs. Yet, the OSM schema is not specifically tailored to representing POIs and there is no distinction from other types of entities (e.g., road segments, boundaries, rivers, etc.). This makes it significantly harder to extract POI information, since relevant categories must be deduced from tags. In exposing OSM elements (including POIs) to RDF, LinkedGeoData[11] offers a SPARQL endpoint over OSM data with billions of triples interlinked with DBpedia[12] and GeoNames[13]. Wikimapia[14] is another open-content map provider for crowd-sourced *places* (including POIs). It offers extra information on map objects around a selected one (e.g., similar_places, nearest_places) as well as nested places (i.e., smaller places inside a larger one).

Amongst *commercial* applications, Foursquare offers a Web API[15] for searching about locations via its venues endpoint. Apart from standard properties (location, name, address, etc.), Foursquare also lists extra information, such as working hours, popular hours, price tier, rating, likes from users, menu, etc. Besides, Google Places provides a Web API[16] for querying information about places on a variety of categories like establishments, geographic locations, and POIs. Regarding POIs, apart from basic information, it also returns photos, price_level, as well as rating and reviews from users.

POI Transformation. The R2RML Recommendation[17] by W3C specifies a notation for mapping relational tables, views or queries into the RDF data model. Defined as a superset of R2RML, the more generic *RDF Mapping Language* (RML) [4, 5] offers customized mapping rules from heterogeneous data sources (not only relational databases, but also CSV, XML, or JSON formats) to RDF. Regarding transformation tools and scripts, most of them are considered proof-of-concept prototypes [12] that can reuse existing vocabularies and ontologies in transformations from relational databases to RDF. Unfortunately, such tools completely lack support for transforming geospatial data into RDF and thus any special handling of POIs. On the other hand, *Extract-Transform-Load* (ETL) tools like GDAL/OGR[18], GeoKettle[19] or FME Workbench[20] can manage the unique characteristics of spatial data, but offer no RDF support.

[5] OGC GeoSPARQL standard: https://portal.opengeospatial.org/files/?artifact_id=47664
[6] OGC WKT standard: http://portal.opengeospatial.org/files/?artifact_id=25354
[7] OGC GML standard: http://portal.opengeospatial.org/files/?artifact_id=20509
[8] http://www.opengeospatial.org/projects/groups/poiswg
[9] http://sdi4apps.eu/
[10] https://www.openstreetmap.org/
[11] http://linkedgeodata.org
[12] http://dbpedia.org
[13] http://www.geonames.org/
[14] http://wikimapia.org/api/
[15] http://foursquare.com/
[16] https://developers.google.com/places/
[17] https://www.w3.org/TR/r2rml/
[18] http://www.gdal.org/
[19] http://www.spatialytics.org/projects/geokettle/
[20] http://www.esri.com/software/arcgis/extensions/datainteroperability/key-features/spatial-etl

Regarding *geospatial data conversion to RDF*, Geometry2RDF[21] can access data in a few geospatial formats and turn their geometries into RDF [13] with the NeoGeo vocabulary[22]; this not compliant with GeoSPARQL and ignores thematic attributes. The generic DataLift platform[23] offers RDF transformation with a user-selected ontology, and its module GeomRDF [6] supports GeoSPARQL-compliant geometries. Over relational data in Oracle Spatial and Graph, custom RDF views[24] can be created via SQL queries or non-standard mappings (e.g., in R2RML) and the generated RDF triples get physically stored in the DBMS. Instead, GeoTriples [7, 8] extends mapping languages R2RML and RML with new constructs. Although not adhering to a specific geospatial vocabulary, it supports the GeoSPARQL standard. Compared to our software TripleGeo, GeoTriples lacks access to as many input formats, it does not support reverse transformations, and its scalability against large geodatasets may suffer because of the complexity of RML mappings. GeoTriples also lacks any specific support for transformation of POI data (e.g., classification schemes). But mappings generated by GeoTriples can be used by the Ontop-spatial extension [3] of the Ontology-Based Data Access system Ontop [11]. This way, users can view their data sources virtually as linked data through on-the-fly GeoSPARQL-to-SQL translation on top of relational databases using ontologies and mappings. This is similar in spirit with Sparqlify[25] employed in the LinkedGeoData portal, and does not require any transformation of data. To the best of our knowledge, software utilities for converting geospatial features into RDF resources have certain limitations, and certainly none has been suggested so far for direct transformation of POIs to RDF and vice versa.

3 POI DATA MODEL

In this Section, we propose a rich, vendor-agnostic, and extensible *OWL ontology*[26] for representing a large and diverse set of POI characteristics in RDF, as identified in existing POI datasets and models. Figure 1 illustrates a graph with the main classes and properties of the ontology[27]. The top-level classes include:

- POI is the main class for POI features and is modelled as subclass of a spatial Feature in GeoSPARQL, thus directly inheriting properties regarding their geospatial location. It supports multiple geometric representations, and the type of each geometry (e.g., centroid, navigation point, map pin, boundary) may be specified as well. In addition, the model enables specification of point locations according to the Basic Geo (WGS84 lat/long) Vocabulary[28], as well as altitude information (which is not yet included in GeoSPARQL).
- POISet represents a collection of POIs (i.e., a POI dataset).
- POISource: A source of POI data (e.g., a commercial vendor) may be specified using a *Universal Resource Identifier* (URI), title, homepage, detailed description, license, logo, etc.
- Classification: A classification scheme that is applied to a POISet and assigns a category (e.g., restaurant, bar, theater, etc.) to each POI. Classification may be possibly *hierarchical*, having its terms (i.e., categories, subcategories, etc.) in multiple levels with a parent-child relationship. We do not enforce a common, predefined classification to POI data coming from diverse sources, but allow each dataset to retain its own. Matches or conficts of POI categories may be resolved during POI data integration in the Linked Data domain.
- Term can be used to specify a category at any level in a classification scheme. Typically, each major category (e.g., *food*) may be specialized into several subcategories (e.g., *restaurant, fast food, pizza*, etc.).
- AccuracyInfo models accuracy assessment for the properties of POI, including the type of accuracy metric (e.g., positional, geocoding, thematic) and its value.
- Address information for a POI may have diverse representations per region or country. The model is flexible to accommodate addresses with street name, house number, postal code, intersection of streets, as linear reference along a road, etc., or even unstructured addresses as string literals.
- Contact information may include phone number(s), fax, and email address(es), each with an optional characterization (e.g., mobile, direct line for phones).
- Name class supports various naming conventions for POIs: abbreviations, acronyms, phonetic transcriptions, transliterations, etc. as well as characterizations for language and type (e.g., official, alternate, brand, historical).
- LicenseInfo covers license information (attribution, title, URL) either for a POI source or for media objects related to POIs.
- Media may refer to photo, video or audio associated with a POI and models their URL, MIME type, and creation timestamp, as well as their license.
- PaymentMethod indicates whether and which particular forms of payment are accepted (e.g., cash, credit/debit card).
- Rating may be used to model the ranking of a POI (with properties such as the rating value, the number of votes, the rating scale, etc.), but also its priceTier as perceived by users.
- Service indicates whether and what type of service is offered in a POI (e.g., parking, wifi, air conditioning, room service).
- SourceInfo provides information on the data source, including the time this POI was retrieved and its original identifier.
- TimeSlot can capture when a POI is open (openingHours) or mostly visited (popularHours) in various temporal granularities (months, weeks, weekdays, hours), and can also handle special cases (e.g., operation in public holidays).

Overall, this ontology aims at capturing properties of POIs that are commonly found in different types of POIs in various domains and applications. Other domain or application-specific properties for particular types of POIs (e.g., for representing electric vehicle charging stations) may still be added, extending the ontology to fit the particular needs of a given application.

4 TRANSFORMING POIS TO/FROM RDF

POI data may be available from multiple sources as files, maintained in DBMSs, retrieved via Web APIs, etc. Providers may also offer a wealth of POI data assets, possibly employing diverse geometry representations, different attribute schemata, and possibly assigning

[21] http://mayor2.dia.fi.upm.es/oeg-upm/index.php/en/technologies/151-geometry2rdf
[22] http://geovocab.org/doc/neogeo
[23] http://datalift.org/
[24] https://docs.oracle.com/en/database/oracle/oracle-database/12.2/rdfrm/rdf-views.html
[25] https://github.com/AKSW/Sparqlify
[26] Web Ontology Language (OWL): https://www.w3.org/2001/sw/wiki/OWL
[27] Our complete POI ontology in OWL is available at https://github.com/SLIPO-EU/poi-data-model
[28] http://www.w3.org/2003/01/geo/

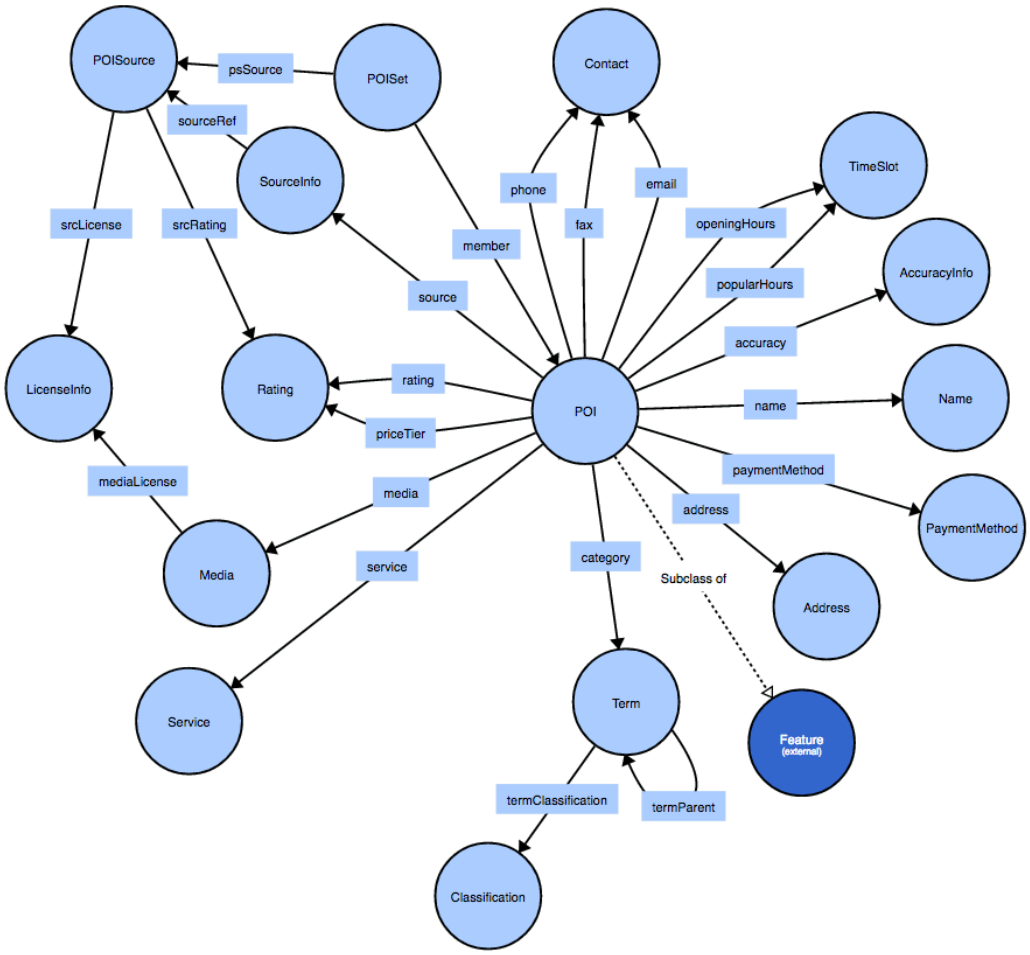


Figure 1: Main classes and object properties in the POI ontology.

categories to POIs under varying classification schemes. However, by transforming original POI data into an RDF representation according to a consistent, extensible, and rich ontology, all challenging tasks involved in POI data integration can be addressed using Linked Data technologies. As explained next, we have customized TripleGeo, our software that can turn any type of geospatial data into RDF [9], towards *transformation* of conventional POI profiles according to the proposed ontology. However, it should be stressed that transformation is actually a two-way process that should also allow *reverse transformation* of linked POI data back into conventional POI formats, thus enabling existing products, systems, and services to exploit the integrated POI datasets. Next, we present the main features of TripleGeo[29] as well as its deployment on multiple concurrent threads for advanced efficiency.

4.1 Transformation to RDF

Figure 2 illustrates the flow diagram employed by TripleGeo for transforming geospatial POI features into RDF triples. A *configuration file* sets properties that control the various stages of transformation: how input source will be accessed, which data to retrieve, what geometric representation to apply, whether geometries must be georeferenced in another system, the output format, etc.

Input data may be obtained from vector geospatial files either *structured* (e.g., ESRI shapefiles, CSV, GeoJSON, GPX) or *semi-structured* (in XML, GML, or KML), as well as from *geospatially-enabled DBMSs* (IBM DB2, Microsoft SQL Server, MySQL, Oracle Spatial, PostGIS, SpatiaLite, or ESRI personal geodatabases). Connectors to source data are used to provide access to geometric features: this is possible either via JDBC drivers for DBMSs or the GeoTools library for vector geographical files.

Especially for *structured* geospatial data from files or retrieved from a DBMS, a *feature iterator* consumes each input record (i.e., all attributes concerning a POI) and converts its geometry into a suitable representation according to user specifications. TripleGeo supports not only points, but all OGC primitives for 2-dimensional geometries like (Multi)Point, (Multi)LineString, or (Multi)Polygon, and even complex geometries (Geometry Collection). Optionally, *reprojection* of geometries between coordinate reference systems (CRS) is possible thanks to the integrated GeoTools library.

Regarding *thematic* (i.e., non-spatial) attribute values (e.g., name, address, contact information) of a POI feature, TripleGeo emits

[29] Java source code, full documentation, and examples at https://github.com/SLIPO-EU/TripleGeo

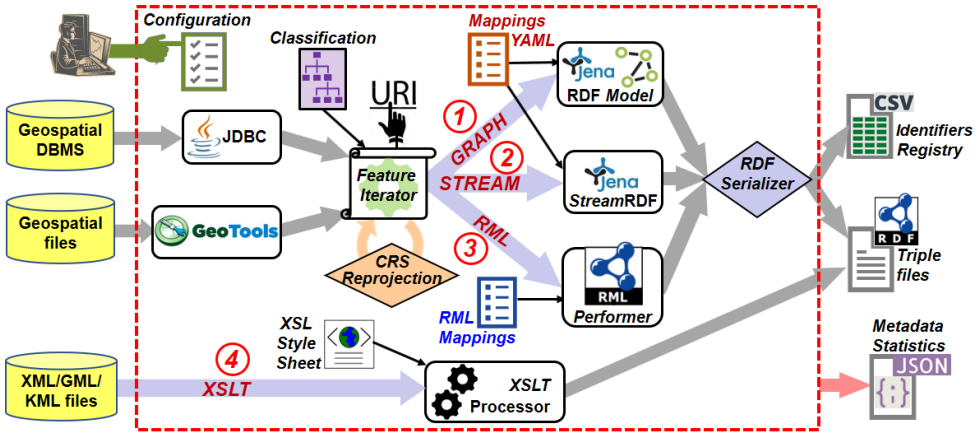


Figure 2: Processing flow for transformation to RDF with TripleGeo.

properly formatted literals as defined in user-specified *attribute mappings*. Such mappings can be prescribed in two alternative representations that both reflect the underlying POI ontology. In particular, an *RML mapping* [4] specifies rules concerning how input data will be represented in RDF as triples. This is achieved by specifying TriplesMap constructs, which control the generation of RDF nodes, i.e., URIs or blank nodes or literals, according to constants, references to original attribute values or templates, as well as based on cross-references between TriplesMaps. Although RML offers enough expressiveness for defining mappings, it may seem cumbersome for regular users without RDF skills. So, we introduced a simplified, *custom mapping* in *YAML* format that offers full support for mapping POI attributes specifically to our ontology. During transformation, TripleGeo identifies the YAML mapping defined for a given attribute and accordingly creates RDF triples.

In case that input data follows a (possibly multi-level) *classification scheme* into categories, subcategories, etc. assigned to POI features, this can be also utilized in the transformation by assigning URIs to these categories. The classification scheme is also transformed into triples (according to the hierarchical structure defined in the POI ontology), and TripleGeo introduces extra links between a POI and its respective category URI under this scheme.

TripleGeo assigns an HTTP URI to each processed feature according to a configurable scheme, so that data owners have enough flexibility and full control over creating and managing their own POI identifiers, while still adhering to a uniform format. Namespaces for classes can also be defined in the configuration and used in the transformation according to the mappings. The default URI pattern is based on *Universally Unique Identifiers* (UUID), i.e., 128-bit long numbers generated on-the-fly during transformation. We avoid reusing original POI identifiers or assigning auto-incremented integer values, since UUIDs can almost safely serve as unique identifiers without the need of a central registration authority or other means of coordination among involved parties. TripleGeo intentionally avoids creating blank nodes by inheriting the URI of the main feature to all its object properties with a suitable suffix based on the property name (e.g., /address, /contact, etc.). Note that TripleGeo does not check for duplicate POI identifiers either in the input data or in the assigned URIs. Instead, we intentionally opted to separate POI transformation from POI deduplication, allowing possibly multiple identifiers for the same POI to be created (e.g., a POI available from multiple providers), which may be interlinked at a later stage.

TripleGeo can perform RDF transformation in various *modes*:

(1) GRAPH materializes a *disk-based* Jena model[30] for collecting all triples transformed according to YAML mappings, and eventually exports them to a file. This persistent, queryable RDF graph essentially maintains disk-based indices regarding the produced triples; once triples for each POI are added, the respective indices must be updated, which gets costlier with time. Even though the Jena model does not support spatial data types and operations, POI geometries can be stored as GeoSPARQL-compliant WKT literals without problems.

(2) STREAM applies *in-memory* conversion with prompt creation of triples per input POI feature, making use of the high performance writers of the Jena RIOT API. By iterating over each POI feature, every attribute (geometry, thematic, classification) is transformed into triples according to a given YAML mapping to the ontology. Once transformation of a given POI is completed, the resulting triples can be readily written to the output file. The only restriction is their serialization, because not all RDF formats are suitable for streamlined writing to files, except for N-Triples/N-Quads.

(3) RML mode applies attribute mappings specified in RML language on each input feature. For integration with TripleGeo, we modified the RML processor[31] to work in a streaming fashion. Instead of materializing triples in a SAIL repository, which may easily become a bottleneck when transforming large datasets, we created a wrapper over RML performers for iterating over each input POI and producing transformed triples according to user-defined RML mappings. In case of cross-referencing, execution of any dependent TriplesMap is triggered by the appropriately defined parent TriplesMap. Each batch of triples is generated in-memory and serialized to a file before the iterator fetches the next POI.

(4) XSLT is exclusively used against *semi-structured* (GML, KML, XML) geographical data and metadata. Parsing is based on XSL

[30] https://jena.apache.org/
[31] http://rml.io/

style sheets that define application profiles for transforming the input data into RDF triples.

Serialization of output triples is performed by the Jena API into several triple formats (RDF/XML, N-Triples, N3, TTL, etc.).

An extra CSV file is issued after transformation, containing basic attributes per POI (URI, name, category, and geometry) for its inclusion in an *Identifiers Registry*. This is a directory of identifiers created and associated with key metadata about each POI processed within the POI integration lifecycle. It can serve as a lookup service for retrieving basic information about a previously imported POI or for searching for already registered POIs and their identifiers.

Finally, *metadata* and *statistics* compiled during transformation are written into a JSON file. This metadata concern execution time, input and output size, number of transformed values per attribute, and the spatial extent of the dataset.

4.2 Reverse Transformation from RDF

TripleGeo supports reverse transformation of RDF POI data into conventional geospatial formats (currently, CSV and ESRI shapefiles) and works according to user-specified configuration settings. Input RDF data may be obtained from files with standard serializations. Multiple RDF files may be specified (with the same serialization and conforming to the same ontology) in order to reconstruct a single geospatial file with all (geometric and thematic) information. In effect, TripleGeo creates an intermediate disk-based Jena model that stores all input triples (equivalent to the GRAPH mode in transformation). In the configuration, the user must specify a SELECT query in SPARQL that will be used to retrieve results (records) from the constructed model according to the underlying POI ontology. Once query results are returned, a feature iterator consumes each one and recreates a record from it; each attribute specified in the SELECT clause becomes a column in the resulting file. Reprojection of geometries into another coordinate reference system (CRS) is also an option. In case that data types (e.g., date or numeric) are known for RDF literals, the respective attributes may be suitably defined in the resulting file; otherwise, these are stored as strings.

Admittedly, there exists an impedance mismatch in this reverse direction, given that the POI ontology is semantically more expressive than conventional POI schemata. Presently, we have tested TripleGeo with SPARQL specifications that can successfully retrieve the same attributes as in original POI data given for transformation to RDF. But generally, POI attributes, relations and metadata in RDF representation will be richer than those supported by conventional file formats, as we intend to explore this issue in the future.

4.3 Extra Features

Since its inception, TripleGeo has been implemented to be *standards-compliant*. This mostly concerns RDF geometries, fully compliant with the OGC GeoSPARQL standard (and indirectly to other OGC standards). Moreover, transformation of INSPIRE-aligned data and metadata[32] is also supported [10], thus abiding by the EU Directive for interoperable Spatial Data Infrastructures across Europe.

TripleGeo can accept mappings to other ontologies as well as diverse classification schemes. Beside the ontology in Section 3, we have also performed transformations of POIs using a data model for representing places[33]. Although this collaborative model differs a lot from our own ontology, it is rather straightforward to specify suitable mappings and perform transformations. Furthermore, using TripleGeo we have launched a free download service[34] that offers global POI data extracted from OpenStreetMap and transformed into RDF format. To select features representing POIs from the entire OSM database, a custom filtering has been applied involving tags that signify POIs of various categories. In the resulting triples, a data property is generated for each key under the OpenStreetMap namespace, hence retaining all original tags according to the native OSM data model, while also creating resolvable, machine-readable URIs per POI.

We are currently working on extending TripleGeo towards *semi-automatic workflows* to assist and guide users in creating attribute mappings for new datasets. We have built a Machine Learning utility that suggests new mappings from a corpus of previously specified ones, available from the various use cases of POI data handled so far. This utility also analyzes the contents of each attribute in a new POI dataset, based on its data type (string, numeric, etc.), formatting (e.g., phone numbers, postal codes), as well as the presence of special characters. Users can then verify or modify these recommended mappings through a graphical interface before applying them for transforming their POI data into RDF.

5 SCALABLE TRANSFORMATION WITH POI DATA PARTITIONING

Assuming no relationships between POIs, each one may considered as an autonomous entity with its own properties, so its generated triples have no links to other POIs. In this case, a large POI dataset may be *partitioned* into a number of disjoint subsets, each one ideally having an equivalent number of POIs. Then, a separate RDF transformation task may be employed for each partition. Naturally, the optimal number of partitions highly depends on the available resources of the cluster or the standalone machine that must perform the RDF transformation. Of course, more partitions mean that each task has to convert a smaller chunk of the original data, but this may not always lead to an overall faster execution. Typically, POI data partitioning may be carried out in several fashions:

- *Evenly*, by splitting the dataset into a specified number of partitions, each one holding the same number of POIs.
- *Hash-based*, by employing a hashing algorithm over a partitioning key (e.g., the URIs). A good choice of hashing algorithm can evenly distribute POIs among partitions, giving partitions of approximately the same size.
- *Spatially*, by uniform grid, quadtree, or other tessellations.
- *Thematically* after grouping POIs by a specific attribute, e.g., per category or country of origin. In this case, ranges of values may be defined (e.g., over dates), and each such range may be used to map data into a separate partition.

Depending on the strategy, partitioning may yield subsets of varying sizes. For instance, distrubution of POIs in cells of a grid partitioning may not be even, as the original data may be skewed (e.g., more dense POIs in urban areas). But, in any case, each chunk may be processed separately and produce its own RDF output.

[32] EU INSPIRE Directive 2007/2/EC, https://inspire.ec.europa.eu/

[33] https://schema.org/Place

[34] http://download.slipo.eu/results/osm-to-rdf/

Multi-threaded Execution. As a first approach towards scalable RDF transformation, we have adjusted TripleGeo to run concurrently in *multiple Java threads*. This method requires a preprocessing step that subdivides the input data and stores each partition into a separate file having the same schema as the original. For each such chunk, a separate thread of TripleGeo can be launched for its transformation, which proceeds totally isolated from the rest. Each thread abides by the same configuration settings, e.g., applies the same classification scheme, attribute mappings, namespaces, etc., as if it worked alone. Finally, these partial results may be merged into a single one, or loaded into a triple store to create a unified RDF representation for the original data.

Parallelized Execution over Spark. For parallelized POI data transformation in *cluster* infrastructures, we have also extended TripleGeo to use Apache Spark[35] and GeoSpark[36] as its underlying partitioning mechanism. As an distributed, cluster-computing framework, Apache Spark provides APIs in many programming languages, offers a stack of data management libraries (SQL, DataFrames, and Datasets), while also supporting machine learning, graph processing, and streaming applications. Spark abstracts the data in *Resilient Distributed Datasets* (RDD), an immutable distributed collection of data elements that can be stored in memory or disk across a cluster of machines. Besides, GeoSpark [14] is an extension of Spark core and supports spatial data types, indices (R-tree, Quadtree, grid, etc.), and topological operations at scale. Enriched with a set of out-of-the-box Spatial RDDs (SRDDs), it can efficiently load, process, and analyze large-scale spatial data across machines.

This parallelized extension for TripleGeo is available for several spatial formats. In case of GeoJSON and CSV, input POI information is parsed into a Spark DataFrame that contains all original attributes. Information from ESRI shapefiles is read using GeoSpark into a *Spatial RDD*, which represents records along with their geometries.

This parallelized process has two stages. First, (Geo)Spark loads the input dataset and partitions it into a user-specified number of partitions. The input data is read and stored in partitions on HDFS accessible by different worker nodes. In the second step, each Spark worker performs an independent transformation task by invoking its own TripleGeo instance against its assigned subset of the POI data. In particular, each input POI instance from the Spatial RDD is mapped to a collection of (key, value) pairs (one key per attribute value) and this is forwarded to TripleGeo for RDF transformation according to the global configuration. Note that there is no need for reshuffling data between workers, since partial transformation results produced by each worker come from disjoint chunks of the original POI data and have no associations to other POIs.

As our experiments is Section 6 testify, employing multiple concurrent threads or Spark-based partitioning for transforming disjoint pieces of large POI datasets can offer orders of magnitude performance gains compared to standalone execution and testifies the robustness and scalability of TripleGeo.

6 EXPERIMENTAL EVALUATION

Next, we report results from a comprehensive validation of the transformation process against *open POI data* extracted from OSM and stored in various spatial repositories[37]. Based on this study, we have also setup a SPARQL endpoint[38] for searching and querying according to our ontology against an RDF graph of open POI data in Europe, available under the Open Database License (ODbL).

6.1 Experimental Setup

We extracted all POI data across Europe from the OpenStreetMap (OSM) database, including their *detailed geometry* (i.e., not just long/lat point locations, but also polygons, linestrings, etc.) and all their *tags* as a list of key-value pairs. Based on the tags available for each OSM element, we categorized the extracted records according to a two-tier classification scheme[39] with 15 *categories* and 167 *types* (subcategories). OSM elements not qualifying as POIs to any term in this classifcation scheme were ignored (e.g., road segments). The filtered POI dataset contains 7,447,697 records over Europe and was stored in a PostGIS database. We specified extra filters with SQL queries over the set of OSM tags in order to isolate particular attributes concerning POIs as listed in Table 1. This base real dataset D (which occupies about 1.9GB on disk in CSV format) is referred to as *OSM 7.4M POIs* in the sequel. For scalability tests over Spark, we synthetically generated extra datasets, by replicating base dataset D several times (×2, ×4, ×8, ×16). For other efficiency tests, we have also taken a subset S of the base OSM dataset by randomly selecting one million records (coined as *OSM 1M POIs*). Details on the contents of all datasets are listed in Table 2. To check performance when accessing POI data from various spatial repositories, apart from PostGIS, we have also stored dataset S in three other formats: CSV, ESRI shapefile, and a commercial geospatially-enabled DBMS (hereafter referred to as X-DBMS).

Experiments regarding standalone and multi-threaded execution of TripleGeo were conducted on a Virtual Machine (VM) running Debian Linux 3.16.0 on an Intel Core i7-3820 CPU with 4MB cache at 2.2 GHz. This VM was given 8GB RAM, 1GB swap, 4 (virtual) CPU cores and 300GB disk space. All data was locally available on the VM, so no network delays were involved.

[35] https://spark.apache.org/
[36] http://geospark.datasyslab.org/

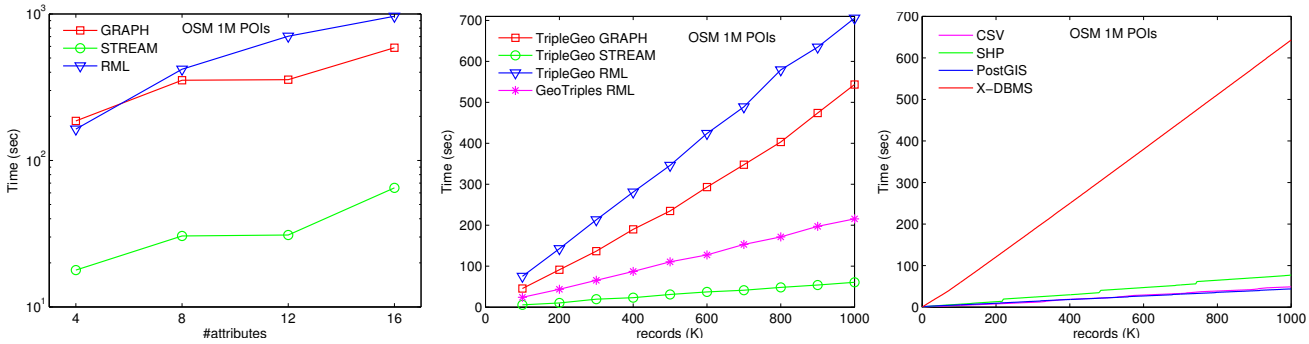
[37] In the context of the SLIPO project, we have transformed several POI datasets available from commercial vendors for Europe, but these results cannot be published due to copyright.
[38] http://geoknow-server.imis.athena-innovation.gr:11480/pois.html
[39] Used in https://github.com/MorbZ/OsmPoisPbf for extracting POIs from OSM into CSV format.

Table 1: POI attributes extracted from OSM.

Basic	Address	Contact	Other
OSM id	street	phone	international name
name	number	fax	category
type	zipcode	email	country
geometry	city	webpage	opening hours

Table 2: POIs in the real and synthetic OSM datasets.

Dataset	Type	Total #POIs
$D: 7.4M\ POIs$	*original*	7,447,697
$S: 1M\ POIs$	*sample*	1,000,000
$D \times 2$	*synthetic*	14,895,394
$D \times 4$	*synthetic*	29,790,788
$D \times 8$	*synthetic*	59,581,576
$D \times 16$	*synthetic*	119,163,152



(a) Varying attributes by transformation mode (b) Increasing data volumes by transformation mode (c) Increasing data volumes from various spatial repositories

Figure 3: Performance of POI transformation (single-thread executions).

Tests with TripleGeo over Spark were conducted on top of a YARN/Hadoop cluster consisting of 7 processing nodes using an HDFS cluster for storage (with a disk capacity of 1.2TB); by default, the HDFS block size is 64MBs. Each node has Intel® Xeon® CPU E5-2650 v3 @ 2.30GHz and can allocate 7 vcores and 15GB RAM for running YARN tasks. All nodes in the cluster are connected on a rack-local network with a Gigabit Ethernet switch and run Linux Ubuntu 16.04 LTS with Hadoop 2.9, Spark 2.2.3, and GeoSpark 1.1.2.

Each experiment was executed in *cold runs*, i.e., invoking TripleGeo immediately after all caches of the operating system are cleared, the DBMS is re-started (if used), and no data is loaded in main memory. We primarily measure the end-to-end *clock time* (in seconds) required to transform a given dataset to RDF or the cost of *reverse transformation* to de facto POI formats. For transformation modes of TripleGeo that work in a streaming fashion (i.e., STREAM and RML), we also provide results regarding the average *throughput*, i.e., the rate in triples/sec at which TripleGeo generates RDF triples as it progressively consumes the input dataset. Execution tests on top of Spark also indicate the data partitioning time.

6.2 Performance Results

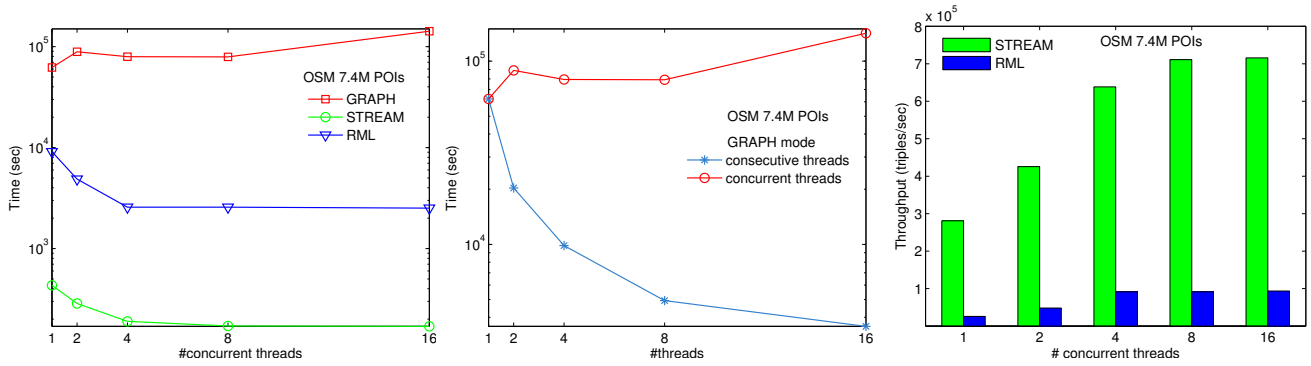
Standalone POI Transformation to RDF. Figure 3(a) depicts the overall cost for transforming one million POIs to RDF with TripleGeo, when each POI includes a *varying number of attributes*, as listed in columns in Table 1. Clearly, the more the attributes available per input record, the more the resulting RDF triples; hence, the cost should increase linearly with the number of attributes (note the logarithmic scale along the time axis). This is exactly the case with the RML mode, since the RML performers employed in transformation must examine each attribute value irrespective of NULL values. But notice that costs for STREAM and GRAPH modes seem practically unaffected when dealing with 12 attributes instead of 8, i.e., when also examining contact-related attributes, because such values are missing from most POI records. As YAML mappings are not applied at all in such cases, this saves in processing cost. The STREAM mode is manifestly the most efficient, at least an order of magnitude faster than GRAPH, and even more faster than RML for any number of input attributes. The only case where RML fares slightly better than GRAPH is when only the four basic attributes are examined. Clearly, TripleGeo can efficiently handle a varying number of thematic attributes per record with linear (worst-case) or sublinear (amortized) increase in transformation cost.

The next test compares performance of TripleGeo in its various modes and also against GeoTriples [8]. This concerns *increasing volumes* of input POIs in CSV format, up to one million records involving all 16 attributes in Table 1. As Figure 3(b) indicates, the cost grows linearly with the input size in all three transformation modes. Although GRAPH involves a disk-based RDF repository that collects all triples before issuing any results, it is still better than the RML mode of TripleGeo for such moderate data sizes. This is due to cross-referencing among properties in the POI ontology (Fig. 1), hence dependent RML mappings (e.g., for names or contacts) get triggered by their parent ones (i.e., for POIs). In effect, each input POI is examined against every defined RML mapping, which is unnecessary in case of missing attribute values. Instead, GeoTriples [8] optimizes the RML processor and extends it with inherent support for geometries, hence its better performance. Nevertheless, under its STREAM mode TripleGeo is always superior by far, as it handles each record in isolation and, thanks to its lightweight YAML mappings, it can promptly emit the resulting triples.

Figure 3(c) plots the performance of TripleGeo in STREAM mode when the same input data (1 million POIs) is retrieved from different repositories: either de facto file formats (CSV, ESRI shapefile) or two geospatially-aware DBMS (PostGIS, X-DBMS). Clearly, there is some divergence in performance amongst repositories, yet this is unrelated to transformation. Indeed, when input data comes from CSV or PostGIS, execution proceeds rapidly. In contrast, the JDBC driver for X-DBMS provides records at a much slower pace, hence the significant slowdown in performance. When input is in shapefile format, each record must first be parsed by the GeoTools library, which almost doubles the cost compared to CSV. So, the POI data repository affects the rate at which input records are accessed and has a strong impact on overall cost.

Overall, these experiments testify that TripleGeo is orders of magnitude performance gains compared to its original release [9], and can efficiently transform large POI datasets faster than GeoTriples even without any sophisticated data partitioning schemes.

Scalability with Multiple Execution Threads. These experiments are applied against the much larger OSM dataset of *7.4M*



(a) Concurrent threads (b) Threading in GRAPH mode (c) Throughput in streaming modes

Figure 4: POI transformation when deploying TripleGeo **on multiple execution threads.**

POIs to examine performance of transformation with TripleGeo on multiple concurrent threads. As discussed in Section 5, input data is split into several equi-sized parts (in these tests, up to 16 CSV files) in advance, and a separate thread per subset transforms it to RDF under the same mode (GRAPH, STREAM, or RML).

Figure 4(a) illustrates the time (in logarithmic scale) needed to complete transformation of the entire dataset with a varying number of *concurrent* threads. Observe that STREAM is consistently an order of magnitude faster than RML mode, and almost two orders of magnitude faster than GRAPH mode. It is no wonder that transformation cost drops with extra threads for STREAM and RML modes, as each thread handles separately a smaller chunk of the data and takes out the most of available system resources (memory, CPU). Of course, this performance gain gets less pronounced when invoking more than 8 threads, as the system cannot resourcefully sustain all of them concurrently and context switching inevitably ensues. Instead, the cost of applying GRAPH mode increases with multiple concurrent threads. This is primarily due to high I/O interaction, as each thread in this mode has to concurrently maintain its own disk-resident RDF model with frequent updates on their ever growing indices. To verify this effect, we also tested the GRAPH mode with *consecutively* triggered threads, i.e., one at a time and not concurrently, thus not competing each other over system resources. Figure 4(b) confirms that such a scheduling can boost performance of GRAPH mode, as RDF models in Jena are created faster for smaller input chunks. Even then, the disk-based GRAPH mode still trails behind the two streaming ones (STREAM, RML), as these latter work entirely in main memory. However, the GRAPH mode may still be useful in practice, since a persistent RDF graph is created for free and can support SPARQL queries.

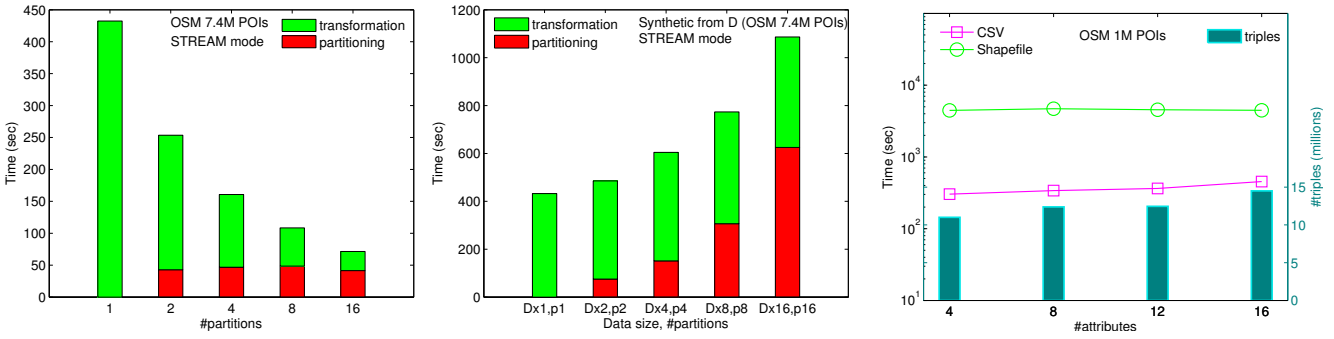
Figure 4(c) depicts the *average throughput* when TripleGeo works in streaming modes (STREAM, RML). As expected, throughput is increasing with extra threads, but the effect diminishes when reaching the ceiling of available system resources (in our setting, when more than 8 threads are used). Again, the STREAM mode readily applies YAML mappings tailored for each attribute and can generate up to 715,000 triples/sec. RML is much less efficient since it has to iterate over all RML performers in order to identify the one with the RML mapping suitable for a given attribute value.

Scalability with Execution over Spark. This set of experiments examines the scalability of TripleGeo when running in STREAM mode on top of a Spark/GeoSpark cluster infrastructure.

Figure 5(a) depicts execution time by fixing the data size D (7.4 million POIs in CSV format) and varying the number of its partitions. Each partition is assigned to a processing node for transformation. Clearly, the worst case is when no partitioning is involved, so a single node has to consume and transform the entire D. By doubling the number of partitions, the transformation cost drops by half, as each data chunk is processed separately and no shuffling is involved. Of course, partitioning incurs some overhead, but this depends on the input size and not on the number of partitions. This test clearly shows the "scale-up" efficiency of TripleGeo and its ability to handle large datasets in a distributed setting making the best utilization of the available system resources (HDFS, CPU cores).

In the next experiment, we measure execution time by varying both the data size and the number of its partitions. More specifically, we synthetically increase the data size (multiples of original dataset D in CSV format) and equally increase the number of partitions, so that their ratio is fixed in each test. As Figure 5(b) testifies, the partitioning cost escalates for growing data sizes, since it takes more time to split and distribute larger datasets. But afterwards, transformation tasks can run in parallel for each partition at their assigned nodes. As each data partition has the same size, their transformation ends up almost at the same time (we plot the maximum duration among transformation tasks in all nodes). Obviously, the impact of partitioning gets more pronounced for larger datasets, while transformation cost is practically stable for equi-sized data chunks. Note that the overall cost increases sublinearly with the data size, provided that more system resources are allocated. Indeed, processing dataset $D \times 16$ (issuing over 2 billion RDF triples) can be carried out with only a $\times 2.5$ increase in execution time compared to D, underscoring the "scale-out" capabilities of TripleGeo.

Reverse Transformation from RDF. We consider RDF data previously transformed from the original OSM data concerning a *varying number of attributes* (i.e., the process examined in Figure 3(a)). To assess the efficiency of *reverse transformation*, the linear plots in Figure 6 show the time (in logarithmic scale on the left y-axis)



(a) Scale up via partitioning

(b) Scale out with increasing data volumes

Figure 5: Scalability of TripleGeo when executed on top of Spark.

Figure 6: Reverse transformation from RDF with varying numbers of attributes.

required to reconstruct POI data in CSV and ESRI shapefile formats from those RDF triples. As indicated with the bar plots, the more the attributes (i.e., extra columns in Table 1), the more the generated RDF triples. In reverse transformation, it takes more time to restore the triples into a disk-based RDF model and subsequently reconstruct records by linking all available properties per POI entity in the model. But, even with 16 attributes per POI, a CSV dataset is restored in almost 7.5 minutes, even though it has to compile this information over 14.5 million triples. Conversion to shapefiles costs an order to magnitude more, because the GeoTools library must first create a feature record, assign the respective attribute values along with validity tests on geometries, before storing this record into the shapefile. We have also compared the reconstructed files with the original OSM data, verifying that we received the same number of records with no extra NULL values in any attribute. All this confirms that the reverse transformation functionality in TripleGeo can swiftly reconstruct original POI data with no information loss.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a generic, vendor-agnostic, and extensible model for representing POI data as linked geospatial data. This model covers a rich set of attributes in de facto POI assets, allows representation of complex POI metadata and relations, and adhers to well-established standards (RDF, GeoSPARQL). Furthermore, we developed transformation software TripleGeo that can access POI data from various spatial repositories and return their RDF representation according to the suggested model along with user-specified attribute mappings and classification schemes. In a comprehensive evaluation against millions of POIs in real and synthetic data, their transformation is concluded in a few minutes, confirming the robustness and versatility of our software and testifying its scalability to handle large POI datasets. Employing data partitioning schemes and parallelization in modern cluster infrastructures, TripleGeo has confirmed its ability to transform massive collections of POIs and generate the resulting RDF triples with minimal latency.

In the future, we plan even more improvements. First, our model can be extended with additional properties regarding POI data evolution and provenance. We also wish to accommodate information about Areas of Interest (e.g., shopping or entertainment hotspots) and POI associations (e.g., a shop within a shopping center). Finally, making estimates about suitable partitions based on input samples without user interaction, taking into account available system resources (CPU, memory, disk), as well as statistics over data characteristics (spatial distribution, number of attributes, etc.) would be also advantageous for even more advanced scalability.

Acknowledgements. This work was partially supported by the EU project *SLIPO* (H2020-ICT-2016-1-731581) and the NSRF 2014-2020 project *HELIX* (grant #5002781). Many thanks to Michalis Alexakis for his precious support in setting up the cluster infrastructure.

REFERENCES

[1] S. Athanasiou, G. Giannopoulos, D. Graux, N. Karagiannakis, J. Lehmann, A. C. Ngonga Ngomo, K. Patroumpas, M. A. Sherif, and D. Skoutas. Big POI Data Integration with Linked Data Technologies. In *EDBT*, pp. 477–488, 2019.
[2] S. Athanasiou, M. Alexakis, G. Giannopoulos, N. Karagiannakis, Y. Kouvaras, P. Mitropoulos, K. Patroumpas, and D. Skoutas. SLIPO: Large-Scale Data Integration for Points of Interest. In *EDBT*, pp. 574–577, 2019.
[3] K. Bereta and M. Koubarakis. Ontop of Geospatial Databases. In *ISWC*, pp. 37-52, 2016.
[4] A. Dimou, M. van der Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *LDOW*, 2014.
[5] A. Dimou, D. Kontokostas, M. Freudenberg, R. Verborgh, J. Lehmann, E. Mannens, S. Hellmann, and R. Van de Walle. Assessing and Refining Mappings to RDF to Improve Dataset Quality. In *ISWC*, pp. 133-149, 2015.
[6] F. Hamdi, N. Abadie, B. Bucher, and A. Feliachi. GeomRDF: A Geodata Converter with a Fine-Grained Structured Representation of Geometry in the Web. In *GeoLD*, 2014.
[7] K. Kyzirakos, I. Vlachopoulos, D. Savva, S. Manegold and M. Koubarakis. GeoTriples: a Tool for Publishing Geospatial Data as RDF Graphs Using R2RML Mappings. In *Terra Cognita*, 2014.
[8] K. Kyzirakos, D. Savva, I. Vlachopoulos, A. Vasileiou, N. Karalis, M. Koubarakis, and S. Manegold. GeoTriples: Transforming Geospatial Data into RDF Graphs Using R2RML and RML Mappings. *Journal of Web Semantics*, 52-53: 16–32, 2018.
[9] K. Patroumpas, M. Alexakis, G. Giannopoulos, and S. Athanasiou. TripleGeo: an ETL Tool for Transforming Geospatial Data into RDF Triples. In *LWDM*, 2014.
[10] K. Patroumpas, N. Georgomanolis, T. Stratiotis, M. Alexakis, S. Athanasiou. Exposing INSPIRE on the Semantic Web. *Journal of Web Semantics*, 35: 53-62, 2015.
[11] M. Rodriguez-Muro and M. Rezk. Efficient SPARQL-to-SQL with R2RML mappings. *Journal of Web Semantics*, 33: 141-169, 2015.
[12] J. Unbehauen, S. Hellmann, S. Auer, and C. Stadler. Knowledge Extraction from Structured Sources. In *Search Computing*, pp. 34-52, 2012.
[13] L.M. Vilches-Blázquez, B. Villazón-Terrazas, V. Saquicela, A. de León, O. Corcho, and A. Gómez-Pérez. GeoLinked Data and INSPIRE through an Application Case. In *ACM SIGSPATIAL*, pp. 446-449, 2010.
[14] J. Yu, Z. Zhang, and M. Sarwat. Spatial Data Management in Apache Spark: the GeoSpark Perspective and Beyond. *GeoInformatica*, https://doi.org/10.1007/s10707-018-0330-9, 2018.