An Efficient RDF Converter and SPARQL Endpoint for the Complete OpenStreetMap Data

Hannah Bast
University of Freiburg
Freiburg, Germany
bast@cs.uni-freiburg.de

Patrick Brosi
University of Freiburg
Freiburg, Germany
brosi@cs.uni-freiburg.de

Johannes Kalmbach
University of Freiburg
Freiburg, Germany
kalmbach@cs.uni-freiburg.de

Axel Lehmann
University of Freiburg
Freiburg, Germany
lehmann@cs.uni-freiburg.de

ABSTRACT

We present *osm2rdf*, a tool for converting OpenStreetMap (OSM) data to RDF triples, along with an efficient SPARQL endpoint and a convenient user interface for formulating SPARQL queries on that data. Unlike previous tools, osm2rdf retains *all* data provided by OSM, including the complete object geometries. Optionally, the tool can output explicit triples realizing the spatial relations *contains* and *intersects*. We provide weekly updates of the data (for the whole planet and also per continent and per country) on https://osm2rdf.cs.uni-freiburg.de. The tool is publicly available on GitHub. The SPARQL endpoint is realized via the open-source SPARQL engine *QLever*. We extended QLever to enable the efficient geometric filtering of a result by a given axis-parallel rectangle. The *QLever UI* provides interactive context-sensitive autocompletion that helps constructing SPARQL queries without prior knowledge of the details of the data.

CCS CONCEPTS

• **Information systems → Extraction, transformation and loading**; **Search interfaces**.

KEYWORDS

OpenStreetMap, Knowledge Graphs, SPARQL, RDF, QLever, Interactive Search, Autocompletion

ACM Reference Format:
Hannah Bast, Patrick Brosi, Johannes Kalmbach, and Axel Lehmann. 2021. An Efficient RDF Converter and SPARQL Endpoint for the Complete OpenStreetMap Data. In *29th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '21), November 2–5, 2021, Beijing, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3474717.3484256

1 INTRODUCTION

OpenStreetMap (OSM) is a collaborative mapping project, which provides the most widely used open geo dataset in the world. There are several established ways to query the OSM data:

1. Filter the raw files using tools like *osmfilter*, *osmconvert*, *osmium*, or the *libosmium* library; see https://github.com/osmcode .

2. Use the *Overpass API*, a search engine built especially for OSM data, with its own query language; see http://overpass-api.de .

SIGSPATIAL '21, November 2–5, 2021, Beijing, China

ACM ISBN 978-1-4503-8664-7/21/11.
https://doi.org/10.1145/3474717.3484256

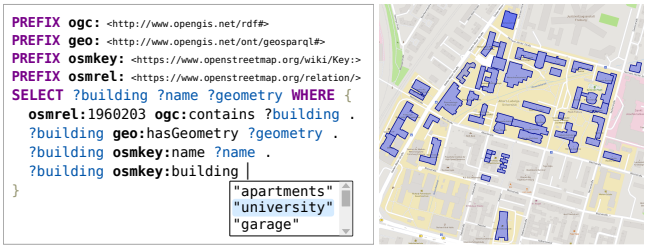

Figure 1: Left: Interactive SPARQL query on the complete OSM data, converted to RDF triples using our tool *osm2rdf*. The query asks for all university buildings in Neuburg (*osmrel:1960203*), a district of the city of Freiburg. The suggestion for "university" is high up in the list of suggestions because the district contains many university buildings. The predicate *ogc:contains* is supported via explicit triples, see Section 4. Right: an excerpt of the query results, displayed on a map.

3. Use a relational database with spatial data support. The most widely used setup here is *PostgreSQL* with the *PostGIS* extension and a loading script called *osm2pgsql*; see https://osm2pgsql.org .

We extend this by a new tool for converting OSM data to RDF triples, along with an efficient SPARQL endpoint and a convenient user interface that allows incremental query construction. Different from the tools above, this allows users to search the OSM data without prior knowledge of an arcane query language or of the details of the data representation. We provide evidence that query times are comparable with or better than those of the engines mentioned above. Very importantly, the "schema-less" RDF data model allows easy combination with other datasets, for example *Wikidata* (many OSM objects are linked to their corresponding Wikidata entities). Different from existing conversion tools to RDF, our tool converts the *complete* OSM data (including all the geometric information), and does so particularly efficiently. Optionally, our tool can add spatial relations between objects as explicit triples. This enables basic geometric queries also for SPARQL engines without explicit GeoSPARQL support. Figure 1 shows a spatial SPARQL query in the process of being built interactively, using our SPARQL engine and user interface.

1.1 OSM Data and the RDF Data Model

OSM objects come in three types: *nodes*, which hold a single coordinate, *ways*, which are ordered lists of nodes and may be used to hold polygonal objects (either open, like streets, or closed, like buildings), and *relations*, which may group arbitrary objects and are also used to model polygonal areas with holes. Each OSM object can have an arbitrary set of key-value pairs, called *tags*.

```
SELECT ?building ?name ?geometry WHERE {
  osmrel:1960203 osm2rdf:contains_area* ?building .
  ?building geo:hasGeometry ?geometry .
  ?building osmkey:name ?name .
  ?building osmkey:building "university"
}
```

```
osmrel:1960203   osm2rdf:contains_area osmway:37486222 ;
                 osm2rdf:contains_area osmway:26290128 ;
                 osmkey:name "Neuburg" .
osmway:37486222  osm2rdf:contains_area osmway:69367089 ;
                 osm2rdf:contains_area osmway:91332395 .
osmway:69367089  geo:hasGeometry "MULTIPOLYGON(...)"^^geo:wktLiteral ;
                 osmkey:name "Mensa Institutsviertel" ;
                 osmkey:building "university" .
osmway:91332395  geo:hasGeometry "MULTIPOLYGON(...)"^^geo:wktLiteral ;
                 osmkey:name "Rechenzentrum" ;
                 osmkey:building "university" .
```

Figure 2: Above: The SPARQL query from Figure 1, with *ogc:contains* rewritten to *osm2rdf:contains_area, as explained in Section 4. Below: An excerpt of the RDF triples produced by *osm2rdf*, which are relevant for the query. The triples are written in Turtle syntax, using a semicolon when the following triple has the same subject. The colors indicate how the triples match the query. Note that *osmrel:1960203* (the district of Neuburg) contains the area *osmway:26290128*, but that area does not contain any university buildings.**

RDF is an elegantly simple data model, where the data is represented as a set of subject-predicate-object statements, called *triples*. Such data can also be considered as a directed graph (then often called a *knowledge graph*), where each distinct subject or object is a node, and each triple corresponds to an edge (from the subject to the object, labeled with the predicate). Two common serialization formats of RDF data are *N-Triples* (one triple per line, with full identifier names, called IRIs) and *Turtle* (a more compact format, which allows abbreviations for repeated prefixes). The standard query language for RDF data is *SPARQL*, which is closely related to the standard database query language SQL. In fact, SPARQL queries can be translated to equivalent SQL queries, and indeed are when using tools like PostgreSQL. For spatial queries, there is a standardized extension called *GeoSPARQL* [7]. Figure 2 gives an example for a spatial SPARQL query, along with the RDF triples (produced by our tool, in Turtle format) relevant for answering the query.

1.2 Related Work

The most closely related previous work to this paper is the *Sophox* project; see https://sophox.org . Sophox also provides a tool for generating RDF triples from OSM data and a corresponding SPARQL endpoint (using the Blazegraph SPARQL engine). However, tags with non-ASCII characters in their key are discarded, and for each way and relation, the geometric shape information is simplified to a single centroid point. Concerning spatial queries, Sophox provides filtering of only these centroid points by a given axis-parallel rectangle. In contrast, our tool maps *each* key-value pair to a triple and for each OSM object there is a dedicated triple providing the exact geometry of the object. Our SPARQL endpoint enables filtering of the exact geometries by a given axis-parallel rectangle, as well as *contains* and *intersect* queries. An older system for transforming OSM data into RDF triples is described in [8]. They plan to provide support for spatial queries using PostGIS. In [9], the problem of linking OSM data with knowledge graphs is investigated.

There is also work on the general problem of building RDF triples from spatial data (not just OSM). In [5], a conceptually clean sucAh translation from a variety of sources is described; their largest RDF dataset has 480M triples. In [6], the focus is on points of interest (POIs) and the integration of heterogeneous sources. In their evaluation, 7.4M OSM objects are considered as well as a synthetic dataset consisting of 119M objects. We consider our work to be orthogonal to these works: while they focus on the conceptual side and use relatively small datasets, we are interested in converting one particular and very large dataset (OSM, over 5B triples) completely and as efficiently as possible, and we also consider the aspect of searching the resulting RDF data efficiently and conveniently.

Concerning existing work on querying OSM data, we already mentioned the prevailing tools in Section 1. An alternative approach is *OSCAR* [1], which provides a combination of text and spatial search on the OSM data, based on a cell partitioning computed from a triangulation of all OSM regions. SPARQL/SQL-like queries are not supported and neither are geo-relations like *contains*.

The SPARQL engine QLever, which we use in this work, is introduced in [2]. It has been developed significantly further since then; see https://github.com/ad-freiburg/qlever . QLever's context-aware autocompletion feature is described and evaluated in [3]. In [4], various SPARQL engines were tested for their GeoSPARQL capabilities. Their compliance benchmark shows that (full) GeoSPARQL support is still lacking, even in engines claiming support.

1.3 Contributions

We consider the following as our main contributions:

- We present an open-source tool *osm2rdf*, which converts OSM data to RDF triples efficiently and preserving the information of *all* tags and of all *all* geometries.
- We provide a dedicated website https://osm2rdf.cs.uni-freiburg.de with weekly data dumps in RDF Turtle format: for the whole planet, as well as for each continent and each country.
- We also provide on that website an efficient SPARQL endpoint for the whole data, with support for basic spatial queries.
- We also provide a user interface for constructing SPARQL queries on the OSM data using interactive context-sensitive autocompletion and for displaying the results on a map.
- We provide a preliminary evaluation, which compares the features and performance of our engine with Sophox, the Overpass API, and PostgreSQL with PostGIS.

2 CONVERTING OSM DATA TO RDF TRIPLES

To convert OSM data to RDF triples, we have to consider four types of information: (1) the object IDs; (2) the object tags, which are string-based key-value pairs; (3) the geometric data associated with the objects; (4) the member relationships between objects (a node may be part of a way, anything may be part of a relation).

We use the following prefixes: *osmkey:* (for the predicates corresponding to keys), *osmmeta:* (for information that is implicitly contained in the OSM data), *osmnode:*, *osmrel:*, *osmway:* (for information pertaining to one of these object types), and *osm:* (for other information). For each OSM object, we compose a unique IRI by

Table 1: The dimensions of our testing datasets.

	FR	BW	DE	EU	P
nodes	12.42 M	46.71 M	337.22 M	2.72 B	6.50 B
ways	1.82 M	7.72 M	55.03 M	326.43 M	718.51 M
relations	31.60 K	101.67 K	677.84 K	5.51 M	8.38 M
areas	1.24 M	5.43 M	39.62 M	234.80 M	499.37 M

Table 2: The number of triples generated by *osm2rdf* for our testing datasets, and the time needed to generate them.

	FR	BW	DE	EU	P
triples	22.05M	89.51M	598.73M	3.38B	4.67B
time	19.80s	1.38min	10.21min	1.08h	2.32h

concatenating the suitable prefix and the OSM ID of the object. For example, the Eiffel tower in Paris becomes `osmnode:8690332214`, La Rambla in Barcelona becomes `osmway:126336685`.

2.1 Tags and Member Relationships

For each key-value pair of each OSM object, a triple is generated: the subject is the IRI of the OSM object, the predicate is the concatenation of *osmkey:* and a grammar-conform escaping of the key, and the object is the value as an RDF string literal. Since key and value can be arbitrary strings, care has to be taken to obtain valid IRIs. Our tool also considers several special cases in the OSM data. For example, semicolons are sometimes used to define a set of values and *osm2rdf* offers the option to produce multiple triples for such values for given keys. For explicit relations between OSM objects in the OSM data, we use the special `osmrel:member` predicate.

2.2 Spatial Data

For each OSM object with a geometry, we add a triple with the IRI of the object as subject, the predicate *geo:hasGeometry*, and the geometry as object. The object is a string literal written in the Well-Known Text format (WKT), a standard way to serialize geometry objects. This task is nontrivial because in the OSM data, only nodes have explicit point geometries, while ways are represented via sequences of node IDs. A way is interpreted as an open or closed polygon, depending on certain conditions[1]. We also consider (multi-)polygons described by OSM relations. We optionally add a triple connecting the OSM object to its bounding box via the *osmmeta:envelope* predicate. This allows for the efficient filtering of a result by a given axis-parallel rectangle; see Section 4.

2.3 Performance and Weekly Updates

On https://osm2rdf.cs.uni-freiburg.de, we provide weekly updated data dumps in RDF Turtle format, for the whole planet, as well as for each continent and each country. To get a feeling for the size of the raw data, Table 1 provides the number of OSM objects (by type) in the following five regions: the city of Freiburg (FR), the state of Baden-Württemberg (BW), Germany (DE), Europe (EU), and the entire planet (P). For each of these datasets, Table 2 provides the number of triples and the processing time required by *osm2rdf*, without the spatial triples described in Section 4. Processing times were measured on a machine with an AMD Ryzen 7 3700X processor, 128 GB of RAM, and a 1.7 TB SSD. Our tool is carefully engineered in all its components and uses multi-threading where useful. The conversion of the OSM data of the whole planet to almost 5 billion triples takes only about 2.5 hours.

[1] https://wiki.openstreetmap.org/wiki/Area

3 SPARQL ENDPOINT AND UI VIA QLEVER

On https://osm2rdf.cs.uni-freiburg.de, we also provide a SPARQL endpoint via the QLever SPARQL engine [2]. QLever maps each entity and literal from the given RDF data to a unique integer ID. The query is then processed entirely in this ID space. QLever always materializes all intermediate results during query processing. This is different from other SPARQL engines, where results are typically produced one element at a time. QLever is generally much faster for difficult queries with large (intermediate) results.

QLever also provides efficient support for context-sensitive autocompletion, as described in [3]. At each point in the query, an autocompletion query is launched (itself a SPARQL query, to the same dataset) that provides suggestion for subjects, predicates, or objects (depending on the position in the query). The suggestions are always meaningful continuations of the query typed so far, in the sense that they lead to a query with a non-empty result. In the example query of Figure 1, the suggested objects are only building types, and more than that, only building types that actually occur in the region specified by the preceding part of the query.

4 SUPPORT FOR SPATIAL QUERIES

We have extended QLever by a *FILTER* operation that takes a variable, say *?x*, that matches an arbitrary set of entities with a geometry, and an axis-parallel rectangle *R*. The operation is realized by adding the triple *?x osmmeta:envelope ?x_env* (see Section 2.2), and then checking for each binding of *?x_env* whether the geometry lies in *R*. Through a compact and clever representation of the bounding boxes, this approach can filter around 50M entities per second.

To enable *ogc:contains* queries (as shown in Figures 1 and 2) even on SPARQL engines without explicit support for such queries, *osm2rdf* has an option to generate explicit triples for this relation. We distinguish between *areas* (all ways and relations in the OSM data that can be represented as a multi-polygon, that is, a closed polygon or a set of such polygons) and all other objects. We compute two predicates: *osm2rdf:contains_area* (whether an area is contained in another area, excluding containments which follow by transitivity) and *osm2rdf:contains_nonarea* (whether a non-area is contained in an area which is minimal with respect to the partial ordering defined by *osm2rdf:contains_area*). The predicate *ogc:contains* is then realized by the SPARQL predicate path *osm2rdf:contains_area*/osm2rdf:contains_nonarea?* . Note that this mechanism avoids a single very large predicate for *ogc:contains*. Figure 2 gives an example, where only the *osm2rdf:contains_area** part is needed. The *ogc:intersects* relation is realized analogously. The pre-computation of these predicates is currently slow (16 hours for Germany, 48 days for the whole planet). However, once they are computed, SPARQL queries using them are fast.

Table 3: Running time of the four example queries described in Section 5.2. The — means that the query timed out. Query Q3 on Sophox only considers the centroids (*).

	QLever	Overpass API	PostGIS	Sophox
Q1	61ms	—	7min	8s
Q2	390ms	—	6min	—
Q3	391ms	169ms	100ms	4.5s*
Q4	134ms	300ms	188ms	n/a

5 COMPARISON TO EXISTING ENGINES

We briefly compare the features and performance of our extended version of QLever to the three engines mentioned in Section 1.2 (PostGIS, with data imported by osm2pgsql, the Overpass API, and Sophox) on the two datasets DE (Germany) and P (the whole planet) from Table 1. We did not compare to OSCAR because their (text-search) query language does not support the precise semantics needed for Q2 - Q4.

5.1 Features Comparison

Tags: Our tool *osm2rdf* retains all tags for all nodes, ways, and relations. Overpass also retains this information. Sophox discards tags, where the keys have non-ASCII characters. The PostGIS database also stores only a subset of the keys. The reason is that *osm2pgsql* stores tags in a table with one column per key, and PostgreSQL has a limit of 1,600 columns per table; see https://www.postgresql.org/docs/current/limits.html. The complete OSM data currently uses about 90,000 different keys.

Geometries: *osm2rdf* includes all the geometries included in the raw OSM data, as does Overpass and *osm2pqsql*. Sophox only stores a centroid point for each entity.

Query Language: QLever, Sophox, and PostGIS can be used with a standard query language (SPARQL or SQL, respectively). Overpass uses its own (rather arcane) query language.

5.2 Performance Comparison

We compare all engines on four typical queries. This is not an exhaustive evaluation; we leave this to future work. Still, this comparison gives a hint at which engine works how well (or not) with which kind of queries. For each engine, we formulated each query in the respective query language, asking for both the object IDs and their geometries. We measured the query processing times.

(Q1) All university buildings
(Q2) All university buildings in the bounding box of Germany
(Q3) All university buildings in the bounding box of Freiburg
(Q4) All university buildings in Freiburg

The evaluation for the Overpass API and *osm2pgsql*/PostGIS was run on a machine with two Xeon E5649 processors and 96 GB of RAM. The evaluation for QLever was run on a machine with one Xeon E5-1650 processor and 256 GB of RAM. The latter machine was almost twice as fast, which is why in Table 3, for the Overpass API and PostGIS, we state half the running time that we actually measured. The Overpass API was run with relaxed constraints for query time and memory usage. For PostGIS, we used a dedicated PostgreSQL instance with default configuration and the default settings of *osm2pgsql*. In this setting, indices are built only for the primary key columns (IDs). To allow arbitrary index-backed queries, we would have to build an index for *every* column, which is unrealistic (additionally, the number of OSM keys greatly exceeds the maximum number of columns, as described above). For Sophox we used the endpoint provided at https://sophox.org . Queries Q1-Q3 were run on the complete OSM data (P), and for all four engines. Query Q4 was run on the OSM data of Germany (DE), but not for Sophox because it does not support queries involving spatial relationships between objects. It should be noted that Sophox can query an Overpass API endpoint via the SPARQL *SERVICE* keyword, but we already compare to Overpass explicitly.

For QLever and Sophox, Q1 is a simple join, but QLever processes joins much more efficiently. Overpass and PostGIS scan all ways (around 689 million) and filter them by `building=university`.

Q2 is Q1 plus the efficient bounding box filter described in Section 1.2 for QLever. For Overpass and PostGIS it is the same problem as for Q1, but filtered to the bounding box of Germany. Sophox first computes all objects in the given bounding box, which times out if the bounding box is too large.

For QLever, Q3 is comparable to Q2. Overpass and PostGIS can handle this query by first computing all objects in the (small) bounding box, and so does Sophox.

QLever computes Q4 using the property paths described in Section 4. For Overpass and PostGIS, the query processing is similar as for Q3, except that the filtering is now by a complex geometry.

6 CONCLUSIONS AND FUTURE WORK

We have presented *osm2rdf*, a tool for efficient conversion of the complete OSM data to RDF, as well as a SPARQL endpoint for the efficient and convenient querying of this data. We showed how explicit spatial triples can support basic spatial queries, also for SPARQL engines without GeoSPARQL support. However, the precomputation of these spatial triples is currently very slow and we consider it a very interesting and relevant open problem to speed this up. We also plan to conduct a more extensive evaluation and comparison of the four engines.

REFERENCES

[1] Daniel Bahrdt, Stefan Funke, Rick Gelhausen, and Sabine Storandt. 2017. Searching OSM Planet with Context-Aware Spatial Relations. In *GIS 2017, Redondo Beach, CA, USA, November 7-10.*
[2] Hannah Bast and Björn Buchhold. 2017. QLever: A Query Engine for Efficient SPARQL+Text Search. In *CIKM 2017, Singapore, November 06 - 10.*
[3] Hannah Bast, Johannes Kalmbach, Theresa Klumpp, Florian Kramer, and Niklas Schnelle. 2021. Efficient SPARQL Autocompletion via SPARQL. *CoRR* abs/2104.14595 (2021). https://arxiv.org/abs/2104.14595
[4] Milos Jovanovik, Timo Homburg, and Mirko Spasic. 2021. A GeoSPARQL Compliance Benchmark. *CoRR* abs/2102.06139 (2021). https://arxiv.org/abs/2102.06139
[5] Kostis Kyzirakos, Dimitrianos Savva, Ioannis Vlachopoulos, Alexandros Vasileiou, Nikolaos Karalis, Manolis Koubarakis, and Stefan Manegold. 2018. GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings. *J. Web Semant.* 52-53 (2018), 16–32.
[6] Kostas Patroumpas, Dimitrios Skoutas, Georgios M. Mandilaras, Giorgos Giannopoulos, and Spiros Athanasiou. 2019. Exposing Points of Interest as Linked Geospatial Data. In *SSTD 2019, Vienna, Austria, August 19-21, 2019.*
[7] Matthew Perry and John Herring. 2012. OGC GeoSPARQL - A Geographic Query Language for RDF Data. https://portal.ogc.org/files/?artifact_id=47664
[8] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. 2012. LinkedGeoData: A core for a web of spatial open data. *Semantic Web* 3, 4 (2012), 333–354.
[9] Nicolas Tempelmeier and Elena Demidova. 2020. Linking OpenStreetMap with Knowledge Graphs - Link Discovery for Schema-Agnostic Volunteered Geographic Information. *CoRR* abs/2011.05841 (2020). https://arxiv.org/abs/2011.05841