Enabling Spatio-Temporal Search in Open Data

Neumaier, Sebastian; Polleres, Axel

DOI:
10.57938/c983a592-c56c-4267-8f42-c0a4c871dfa0

Published: 04/04/2018

Document Version
Publisher's PDF, also known as Version of record

Enabling Spatio-Temporal Search in Open Data

Sebastian Neumaier
Axel Polleres

Arbeitspapiere zum Tätigkeitsfeld
Informationsverarbeitung, Informationswirtschaft und Prozessmanagement
Working Papers on Information Systems, Information Business and Operations

Nr./No. 01/2018
ISSN: 2518-6809
URL: http://epub.wu.ac.at/view/p_series/S1/

Herausgeber / Editor:
Department für Informationsverarbeitung und Prozessmanagement
Wirtschaftsuniversität Wien · Welthandelsplatz 1 · 1020 Wien
Department of Information Systems and Operations · Vienna University of Economics and Business · Welthandelsplatz 1 · 1020 Vienna

Cite as:
Sebastian Neumaier and Axel Polleres. Enabling Spatio-Temporal Search in Open Data. *Journal of Web Semantics (JWS)*, Elsevier, 2019.

Enabling Spatio-Temporal Search in Open Data

Sebastian Neumaier[a,b,1], Axel Polleres[a,c,d,2]

[a] *Vienna University of Economics and Business, Vienna, Austria*
[b] *Vienna University of Technology, Vienna, Austria*
[c] *Complexity Science Hub Vienna, Austria*
[d] *Stanford University, CA, USA*

Abstract

Intuitively, most datasets found on governmental Open Data portals are organized by spatio-temporal criteria, that is, single datasets provide data for a certain region, valid for a certain time period. Likewise, for many use cases (such as, for instance, data journalism and fact checking) a pre-dominant need is to scope down the relevant datasets to a particular period or region. Rich spatio-temporal annotations are therefore a crucial need to enable semantic search for (and across) Open Data portals along those dimensions, yet – to the best of our knowledge – no working solution exists. To this end, in the present paper we (i) present a scalable approach to construct a spatio-temporal knowledge graph that hierarchically structures geographical as well as temporal entities, (ii) annotate a large corpus of tabular datasets from open data portals with entities from this knowledge graph, and (iii) enable structured, spatio-temporal search and querying over Open Data catalogs, both via a search interface as well as via a SPARQL endpoint, available at `data.wu.ac.at/odgraphsearch/`

Keywords: open data, spatio-temporal labelling, spatio-temporal knowledge graph

1. Introduction

Open Data has gained a lot of popularity and support by governments in terms of improving transparency and enabling new business models: Governments and public institutions, but also private companies, provide open access to raw data with the goal to present accountable records [1], for instance in terms of statistical data, but also in fulfillment of regulatory requirements such as, e.g., the EU's INSPIRE directive.[3] The idea to provide raw data, instead of only human-readable reports and documents, is mainly driven by providing direct, machine-processable access to the data, and enable broad and arbitrary (through open licences) reuse of such data [2, 3].

With the advent of *knowledge graphs* traditional web search recently has been revolutionized in that search results can be categorized, browsed and ranked according to well-known concepts and relations, which cover typical search scenarios in search engines. But these scenarios are different for Open Data: in our experience, dataset search needs to be targeted from a different angle than keyword-search (alone). Intuitively, most datasets found in Open Data – as it is mostly regional/national census-based – are organized by spatio-temporal scopes, that is, single datasets provide data for a certain region, and are valid for a certain time period; our goal is to cover exactly these two dimensions which are prevalent in Open Data: Indeed, our approach successfully annotates geospatial information in 75% of the datasets, and temporal information for almost 58% of all datasets (cf. Section 4.3 for the detailed evaluation). Also, Kacprzak et

Email addresses: `sebastian.neumaier@wu.ac.at` (Sebastian Neumaier), `axel.polleres@wu.ac.at` (Axel Polleres)

[1] Sebastian Neumaier's work was funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) through the program "ICT of the Future" by the Austrian Research Promotion Agency (FFG) under the project CommuniData.

[2] Axel Polleres' work was supported under the Distinguished Visiting Austrian Chair Professors program hosted by The Europe Center of Stanford University.

[3] `https://inspire.ec.europa.eu/`

al. [4] recently confirmed the relevance and need of spatio-temporal annotations and search across Open Data portals: They analyzed the query logs of four data portals (including data.gov.uk) wrt. different aspects and characteristic and list temporal and geospatial queries as the top-two query types.

We argue that - just like for regular Web search - knowledge graphs can be helpful to significantly improve search; specifically to our use case we aim at constructing a spatio-temporal knowledge graphs from publicly available sources: In fact, the ingredients for building such a knowledge graph of geographic entities as well as time periods and events exist already on the Web of Data, although they have not yet been integrated and applied – in a principled manner – to the use case of Open Data search.

Herein, we present a scalable approach to (i) construct a spatio-temporal knowledge graph that hierarchically structures geographical entities, as well as temporal entities, (ii) annotate a large corpus of tabular Open Data, currently holding datasets from eleven European (governmental) data portals, (iii) enable structured, spatio-temporal search over Open Data catalogs through this spatio-temporal knowledge graph, available at `http://data.wu.ac.at/odgraphsearch/`.

In more detail, we make the following concrete contributions:

- A detailed construction of a hierarchical base knowledge graph for geo-entities and temporal entities and links between them.
- A scalable labelling algorithm for linking open datasets (both on a dataset-level and on a record-level) to this knowledge graph.
- Indexing and annotation of datasets and metadata from 11 Open Data portals from 10 European countries and an evaluation of the annotated datasets to illustrate the feasibility and effectiveness of the approach.
- A prototypical search interface, consisting of a web user interface allowing faceted and full-text search, a RESTful JSON API that allows programmatic access to the search UI, as well as API-access to retrieve the indexed dataset and respective RDF representations
- A SPARQL endpoint that exposes the annotated links and allows structured search queries.
- Code, data and a description on how to re-run our experiments, which we hope to be a viable basis for further research extending our results, are available for re-use (under GNU General Public License v3.0).[4]

The remainder of this paper is structured as follows: In the following Section 2 we introduce (linked) datasets, repositories and endpoints to retrieve relevant temporal and spatial information. Section 3 provides a schematic description of the construction and integration of these sources into our *base* knowledge graph – a constructed knowledge graph which serves as a basis for annotation and linking of the datasets; its actual realization in terms of implementation details is fully explained in Appendix A. In Section 4 we present the algorithms to add spatio-temporal annotations to datasets from Open Data portals, and evaluate and discuss the performance (in terms of precision and recall based on a manually generated sample) and limitations of our approach. The vocabularies and schema of our RDF data export are explained in Section 5 and the back-end, the user interface and the SPARQL endpoint (including example queries) are presented in Section 6. We provide related and complementary approaches in Section 7, and eventually we conclude in Section 8.

2. Background

When people think of spatial and temporal context of data, they usually think of *concepts* rather than numbers, that is "countries" or "cities" instead of coordinates or a bounding polygon, or an "event" or "time period" instead of e.g. start times end times. In terms of dataset search that could mean someone searching for datasets containing information about demographics for the last government's term (or in other words between the last two general elections).

In order to enable such search by spatio-temporal concepts, our goal is to build a concise, but effective knowledge base, that collects the relevant concepts from openly available data into a coherent, base knowledge graph, for both (i) enabling spatio-temporal search within Open Data portals and (ii) interlinking Open Data portals with other datasets by the principles of Linked Data.

[4] `https://github.com/sebneu/geolabelling/`

The following section gives an overview of datasets and sources to construct the base knowledge graph of temporal- and geo-entities, namely the geo-data sources GeoNames, OpenStreetMap and NUTS, the knowledge bases Wikidata and DBpedia, and the periods/events dataset PeriodO.

GeoNames.org. The GeoNames database contains over 10 million geographical names of entities such as countries, cities, regions, and villages. It assigns unique identifiers to geo-entities and provides a detailed hierarchical description including countries, federal states, regions, cities, etc. For instance, the GeoNames-entity for the city of Munich[5] has the parent relationship "Munich, Urban District", which is located in the region "Upper Bavaria" of the federal state "Bavaria" in the country "Germany", i.e. the GeoNames database allows us to extract the following hierarchical relation for the city of Munich:

$$Germany > Bavaria > Upper\ Bavaria > Munich,\ Urban\ District > Munich$$

The relations are based on the GeoNames ontology[6] which defines administrative divisions (first-order `gn:A`, second-order `gn:A.ADM2`, until `gn:A.ADM5`)[7] for countries, states, cities, and city districts/sub-regions. In this work we make use of an RDF dump of the GeoNames database,[8] which consists of alternative names and hierarchical relations of all the entities.

OpenStreetMap (OSM). OSM[9] was founded in 2004 as a collaborative project to create free editable geospatial data. The map data is mainly produced by volunteers using GPS devices (on foot, bicycle, car, ..) and later by importing commercial and government sources, e.g., aerial photographies. Initially, the project focused on mapping the United Kingdom but soon was extended to a worldwide effort. OSM uses four basic "elements" to describe geo-information:[10]

- *Nodes* in OSM are specific points defined by a latitude and longitude.
- *Ways* are lists of *nodes* that define a line. OSM ways can also define areas, i.e. a "closed" ways where the last node on the way equals to the first node.
- *Relations* define relationships between different OSM elements: They either split long ways into smaller segments (for easier processing), or build complex objects, e.g., a *route* is defined as a relation of multiple ways (such as highway, cycle route, bus route) along the same route.
- *Tags* are used to describe the meaning of any elements, e.g., there could be a tag `highway=residential`[11] (tags are represented as key-value pairs) which is used on a *way* element to indicate a road within settlement.

There are already existing works which exploit the potential of OSM to enrich and link other sources. For instance, in [5] we have extracted indicators, such as the number of hotels or libraries in a city, from OSM to collect statistical information about cities.

Likewise, the software library *Libpostal*[12] uses addresses and places extracted from OSM: it provides street address parsing and normalization by using machine learning algorithms on top of the OSM data. The library converts free-form addresses into clean normalized forms and can therefore be used as a pre-processing step to geo-tagging of streets and addresses. We integrate Libpostal in our framework in order to detect and filter streets and city names in text and address lines.

Sources to obtain Postal codes and NUTS codes. Postal codes are regional codes consisting of a series of letters (not necessarily digits) with the purpose of sorting mail. Since postal codes are country-specific identifiers, and their granularity and availability strongly varies for different countries, there is no single, complete, data source to retrieve these codes. The most reliable way to get the complete dataset is typically via governmental agencies (made easy, in case they publish the codes as

[5] `http://www.geonames.org/6559171/`

[6] `http://www.geonames.org/ontology/ontology\_v3.1.rdf`

[7] Here, `gn:` corresponds to the namespace URL `http://www.geonames.org/ontology#`

[8] `http://www.geonames.org/ontology/documentation.html`, last accessed 2018-01-05

[9] `https://www.openstreetmap.org/`

[10] A detailed description can be found at the OSM documentation pages: `http://wiki.openstreetmap.org/wiki/Main_Page`

[11] cf. `https://wiki.openstreetmap.org/wiki/Tag:highway=residential`

[12] `https://medium.com/@albarrentine/statistical-nlp-on-openstreetmap-b9d573e6cc86`, last accessed 2017-09-12

open data).[13] Another source worth mentioning for matching postal codes is GeoNames: it provides a collection of postal codes for several countries and the respective name of the places/districts.[14]

Partially, postal codes for certain countries are available in the knowledge bases of Wikidata and DBpedia (see below) for the respective entries of the geo-entities (using "postal code" properties). However, we stress that these entries are not complete, i.e., not all postal codes are available in the knowledge bases as not all respective geo-entities are present, and also, the codes' representation is not standardized.

NUTS (French: nomenclature des unitès territoriales statistiques). Apart from national postal codes another geocode standard has been developed and is being regulated by the European Union (EU). It references the statistical subdivisions of all EU member states in three hierarchical levels, NUTS 1, 2, and 3. All codes start with the two-letter ISO 3166-1 [6] country code and each level adds an additional number to the code. The current NUTS classification lists 98 regions at NUTS 1, 276 regions at NUTS 2 and 1342 regions at NUTS 3 level and is available from the EC's Webpage.[15] Also worth mentioning in this context – as an additional source for statistical and topographical maps on NUTS regions – are the basemaps developed at the European level by Eurostat, available as REST services.[16]

Wikidata and DBpedia. These domain-independent, multi-lingual, knowledge bases provide structured content and factual data. While DBpedia [7] is automatically generated by extracting information from Wikipedia, Wikidata [8], in contrary, is a collaboratively edited knowledge base which is intended to provide information to Wikipedia. These knowledge bases already partially include links to GeoNames, NUTS identifier, and postal code entries, as well as temporal knowledge for events and periods, e.g., elections, news events, and historical epochs, which we also harvest to complete our base knowledge graph.

PeriodO. The PeriodO project [9] offers a gazetteer of historical, art-historical, and archaeological periods. The user interface allows to query and filter the periods by different facets. Further, the authors published the full dataset as JSON-LD download[17] and re-use the W3C `skos`, `time` and `dcterms:spatial` vocabularies to describe the temporal and spatial extend of the periods. For instance, the (shortened) PeriodO entry in Figure 1 describes the period of the First World War.

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix skos:<http://www.w3.org/2004/02/skos/core#>
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix time: <http://www.w3.org/2006/time#> .

<http://n2t.net/ark:/99152/p0kh9ds3566>
  dcterms:spatial dbr:United_Kingdom ;
  skos:altLabel "First World War"@eng-latn ;
  time:intervalFinishedBy [
    skos:prefLabel "1918" ;
    time:hasDateTimeDescription [
      time:year "1918"^^xsd:gYear
    ]
  ] ;
  time:intervalStartedBy [
    skos:prefLabel "1914";
    time:hasDateTimeDescription [
      time:year "1914"^^xsd:gYear
    ]
  ] .
```

Figure 1: PeriodO entry for the period of World War I.

3. Base Knowledge Graph Construction

The previous section listed several geo-data repositories as well as datasets containing time periods and event data – some already available as Linked Data via an endpoint – which we use in the following to build up a base knowledge graph: Section 3.1 describes the extraction and integration of geospatial, and Section 3.2 of temporal knowledge. The remaining paper uses an additional color coding of turquoise for introducing temporal and blue for geospatial properties.

Herein, we describe the composition of the graph by presenting *conceptual* SPARQL `CONSTRUCT` queries. This means that (most of) the presented queries cannot be executed because either there is

[13] For instance, the complete list of Austrian postal codes is available as CSV via the Austrian Open Data portal: `https://www.data.gv.at/katalog/dataset/f76ed887-00d6-450f-a158-9f8b1cbbeebf`, last accessed 2018-04-03

[14] `http://download.geonames.org/export/zip/`, last accessed 2018-01-05

[15] `http://ec.europa.eu/eurostat/web/nuts/overview`, last accessed 2018-01-05

[16] `http://ec.europa.eu/eurostat/statistical-atlas/gis/arcgis/rest/services/Basemaps`, last accessed 2018-08-30

[17] `http://perio.do/`, last accessed 2018-03-27.

no respective endpoint available or the query is not feasible and times out. While this section shall serve as a schematic specification of the constructed graph, we detail the actual realization of the queries in Appendix A.

Still, we deem the use of these conceptual SPARQL `CONSTRUCT` useful as a mechanism to declaratively express knowledge graph compilation from Linked Data sources, following Rospocher et al.'s definition, who describe knowledge graphs as "a knowledge-base of facts about entities typically obtained from structured repositories"[10].[18]

3.1. Spatial Knowledge

Our knowledge graph of geo-entities is based on the GeoNames hierarchy, where we extract

- geo-entities and their labels,
- links to parent entities and particularly the containing country,
- coordinates in terms of points and (if available) geometries in terms of polygons,
- postal codes (again, if available), and
- sameAs-links to other sources such as DBpedia, OSM, or Wikidata (again, if available).

The respective SPARQL `CONSTRUCT` query[19] over the GeoNames dataset in Figure 2 displays how the hierarchical data could be extracted from a (currently nonexistent) GeoNames SPARQL endpoint for a selected country `?c`, i.e., if a respective SPARQL endpoint existed to access GeoNames' publishd RDF data,[20] we could get all the relevant data for our knowledge graph per country, by replacing `?c` in this query with a concrete country URI, such as `http://sws.geonames.org/2782113/` (for Austria). The GeoNames RDF data partially already contains external links to DBpedia (using `rdfs:seeAlso`) which we model as equivalent identifiers using `owl:sameAs`. The hierarchy is constructed using the `gn:parentFeature` property. As GeoNames offers various different properties containing names, we extract all official English and (for the moment) German names, as we will use those later on for fueling our search index.

```
CONSTRUCT {                                              knowledge graph model
  ?geoentity rdfs:label ?label ;
     gn:parentFeature ?parent ;
     gn:parentCountry ?c ;
     gn:postalCode ?code ;
     geo:lat ?lat ; geo:lat ?long ;
     owl:sameAs ?external .
} WHERE {                                                labels
  ?geoentity gn:name ?label .
  OPTIONAL { ?geoentity gn:officialName ?label
    FILTER ( LANGMATCHES(LANG(?label), "EN") ||
             LANG(?label) = "" ) }
  OPTIONAL { ?geoentity gn:alternateName ?label
    FILTER ( LANGMATCHES(LANG(?label), "EN") ||
             LANGMATCHES(LANG(?label), "DE") ||
             LANG(?label) = "" ) }

                                                         geospatial
  ?geoentity gn:parentCountry ?c ;
     geo:lat ?lat ; geo:lat ?long .
  OPTIONAL { ?geoentity gn:parentFeature ?parent }
  # external links if available
  OPTIONAL { ?geoentity rdfs:seeAlso ?external }
  # postal code literals
  OPTIONAL { ?wd gn:postalCode ?code }
}
```

Figure 2: Conceptual SPARQL `CONSTRUCT` query to extract hierarchical data for our base Knowledge Graph from GeoNames for a particular country `?c`.

The query in Figure 3 then displays how we integrate the information in Wikidata into our spatial knowledge graph. In particular, Wikidata serves as a source to add labels and links for postal codes (`gn:postalCode`) and NUTS identifiers (`wdt:P605`) for the respective geographical entities. Further, we again add external links (to OSM and Wikidata itself) that we harvest from Wikidata as `owl:sameAs` relations to our graph.

```
CONSTRUCT {                                              knowledge graph model
  ?geoentity owl:sameAs ?wd ;
     gn:postalCode ?code;
     owl:sameAs ?osm ;
     owl:sameAs ?nuts .
  ?nuts wdt:P605 ?n .
} WHERE {                                                geospatial
  ?wd wdt:P1566 ?geoentity .
  # postal code literals
  OPTIONAL { ?wd wdt:P281 ?code }
  # NUTS identifier
  OPTIONAL { ?wd wdt:P605 ?n.
    BIND (CONCAT("<http://dd.eionet.europa.eu/vocabulary
          concept/common/nuts/", ?n,">") AS ?nuts) }
  # OSM relations
  OPTIONAL { ?wd wdt:P402 ?osm }
}
```

Figure 3: SPARQL query to extract Wikidata links and codes – times out on `https://query.wikidata.org`

The query in Figure 4 conceptually shows how and which data we extract for certain OSM entities

[18] As a side remark, such queries could for instance be used to declaratively annotate the provenance trail of knowledge graphs compiled from other Linked Data sources, e.g. expressed through labeling the activity to extract the relevant knowledge with PROV's[11] `prov:wasGeneratedBy` property with a respective SPARQL CONSTRUCT query.

[19] The plain queries are online at `https://github.com/sebneu/geolabelling/tree/master/jws_evaluation/queries`

[20] cf. `http://geonames.org/ontology/`

```
PREFIX : <http://data.wu.ac.at/ns/osm#>

CONSTRUCT {                                              knowledge graph model
  ?geoentity rdfs:label ?label;
    geo:lat ?lat; geo:long ?long ;
    gn:parentFeature ?parent;
    gn:parentCountry ?pc ;
    geosparql:hasGeometry ?geometry .
} WHERE {                                                labels
  [ :display_name ?label ;
                                                         geospatial
    :osm_region ?parent ;
    :osm_id ?id ; :osm_type ?type ;
    :address [ :country ?country ];
    :lat ?lat ; :lon ?long ;
    :geojson [ :coordinates ?geometry] #this is simplifying!
  ] .
  ?pc gn:countryCode ?country .
  BIND(IRI(CONCAT(STR(osm:),?type,"/",?id)) AS ?geoentity)
}
```

Figure 4: Conceptual SPARQL query to extract data from OSM for a particular OSM entity with the OSM numeric identifier `?id`.

into our knowledge graph. We note here that OSM does not provide an RDF or SPARQL interface, but the idea is that we - roughly - perceive and process data returned by OSM's Nominatim API in JSON as JSON-LD; details and pre-processing steps in Appendix A.2 below.

3.2. Temporal Knowledge

As for temporal knowledge, we aim to compile into our knowledge graph a base set of temporal-entities (that is, named periods and events from Wikidata and PeriodO) where we want to extract

- named events and their labels,
- links to parent periods that they are part of, again to create a hierarchy,
- temporal extent in terms of a single beginning and end date, and
- links to a spatial coverage of the respective event or period (if available).

We observe here that temporal knowledge is typically less consolidated than geospatial knowledge, i.e. "important" named entities in terms of periods or events are not governed by internationally agreed and nationally governed structures such as border-agreements in terms of spatial entities. Even worse, cross-cultural differences, such as different calendars or even time zones, add additional confusion. We still believe that the two integrated sources, which cover events of common interest in a multilingual setting on the one hand (Wikidata), and historical periods and epochs from the literature on the other (PeriodO), provide a good starting point.

In the future, it might be useful to also index news events, or recurring periods or points in time, such as public holidays, that occur regularly. However, we did not find any structured datasets available as linked data for that, so, we have to defer these to future work, or respectively, the creation of respective structured datasets as a challenge for the community. One obvious existing starting point here would be the work by Rospocher et al. [10] and the news events datasets they created in the EU Project NewsReader.[21] For the moment, we did not consider this work due to its fine granularity, which in our opinion is not needed in a majority of Open Data search use cases.

Again, we model the knowledge graph extraction and construction in terms of conceptual SPARQL queries: We use the query in Figure 5 to extract events information from Wikidata. Note, that this query times out on the public Wikidata endpoint. Therefore, in order to extract the relevant events and time periods as described in Figure 5, we converted a local Wikidata dump to HDT [12], extracted only the relevant triples for the query, materialized the path expressions, and executed the targeted `CONSTRUCT` query over these extracts on a local endpoint; the full details are provided in Appendix A.3. We do not just extract existing triples from the source, but try to aggregate/flatten the representation to a handful of well-known predicates from Dublin Core (prefix `dcterms:`) and the OWL time ontology (prefix `time:`).

Likewise, we use the query in Figure 7 to extract periods from the PeriodO dataset, again using the same flattened representation. To execute this query, in this case we could simply download the available PeriodO dump into a local RDF store.

Note that in these queries – in a slight abuse of the OWL Time ontology – we "invented" the properties `timex:hasStartTime` and `timex:hasEndTime` that do not really exist in the original OWL time ontology. This is a compromise for the desired compactness of representation in our knowledge graph, i.e. these are mainly introduced as shortcuts to avoid the materialization of unnecessary blank nodes in the (for our purposes too) verbose notation of OWL Time. A proper representation using OWL Time could be easily reconstructed by means of the `CONSTRUCT` query in Figure 6.

[21] http://www.newsreader-project.eu/results/data/

```
CONSTRUCT {
  ?event rdfs:label ?label ; dcterms:isPartOf ?parent ; dcterms:coverage ?geocoordinates ;
         timex:hasStartTime ?startDateTime ; timex:hasEndTime ?endDateTime ; dcterms:spatial ?geoentity .
} WHERE {
  # find events with (for the moment) English, German, or non-language-specific labels:
  ?event wdt:P31/wdt:P279* wd:Q1190554 . ?event rdfs:label ?label .
  FILTER ( LANGMATCHES(LANG(?label), "EN") || LANGMATCHES(LANG(?label), "DE") || LANG(?label) = "" )
  { # restrict to certain event categories, e.g. (for the moment) elections and sports events:   #sports competitions
   { ?event wdt:P31/wdt:P279* wd:Q40231 } UNION { ?event wdt:P31/wdt:P279* wd:Q13406554 }
  }
  {
    { ?event wdt:P585 ?startDateTime . FILTER ( ?startDateTime > "1900-01-01T00:00:00"^^xsd:dateTime) }
    UNION
    { ?event wdt:P580 ?startDateTime. FILTER ( ?startDateTime > "1900-01-01T00:00:00"^^xsd:dateTime)
      ?event wdt:P582 ?endDateTime. FILTER ( DATATYPE(?endDateTime) = xsd:dateTime) }
  }
  BIND(IF(bound(?endDateTime), ?endDateTime, xsd:dateTime(CONCAT(STR(xsd:date(?startDateTime)),"T23:59:59"))) AS ?endDateTime)
  OPTIONAL { ?event wdt:P361 ?parent }
  OPTIONAL { ?event wdt:P276?/(wdt:P17|wdt:P131) ?geoentity }
  OPTIONAL { ?event wdt:P276?/wdt:P625 ?geocoordinates }
}
```

knowledge graph model

labels

filter events

temporal

geospatial

Figure 5: Conceptual SPARQL query to extract event data (from elections and sports competitions) from Wikidata – times out on `https://query.wikidata.org`. (Namespaces can be found in the Apendix in Figure A.18)

```
CONSTRUCT {
   ?X time:hasBeginning [
      time:inXSDDateTime ?startDateTime
   ] ;
      time:hasEnd [
        time:inXSDDateTime ?endDateTime
   ] .
} WHERE {
   ?X timex:hasStartTime ?startDateTime ;
      timex:hasEndTime ?endDateTime
}
```

Figure 6: `CONSTRUCT` query to reconstruct the OWL Time model of our flattened representation `timex:hasStartTime` and `timex:hasEndTime`.

For this purpose we define our own vocabulary extension of the OWL Time ontology, for the moment, under the namespace `http://data.wu.ac.at/ns/timex#`.

4. Dataset Labelling

In this section we first describe the algorithms to add geospatial annotations (Section 4.1) and to extract temporal labels and periodicity patterns (Section 4.2) and subsequently evaluate and discuss the performance – in terms of precision and recall based on a manually evaluated sample – and limitations of our approach in Section 4.3.

In order to add spatial and temporal annotations to Open Data resources we use the CSV files and metadata from the resources' data portals as signals. The metadata descriptions and download links are provided by our Open Data Portal Watch framework [13] which monitors and archives over 260 data portals, and provides APIs to retrieve their metadata descriptions in an homogenized way using the W3C DCAT vocabulary [14]. Regarding the meta-information, we look into several available metadata-fields: we consider the title, description, the tags and keywords, and the publisher. For instance, the upper part of Figure 8 displays an example metadata description. It holds cues in the title and the publisher field (cf. "Veröffentlichende Stelle" - publishing agency) and holds a link to the actual dataset, a CSV file (cf. lower part in Figure 8), which we download and parse.

4.1. Geospatial labelling

The geospatial labelling algorithm uses the different types of labels in our base knowledge graph to annotate the metadata and CSV files from the input data portals.

4.1.1. CSVs

Initially, the columns of a CSV gets classified based on regular expressions for NUTS identifier and postal codes. While the NUTS pattern is rather restrictive,[22] the postal codes pattern has to be very general, potentially allowing many false positives. Basically, the pattern is designed to allow

[22] $[A\text{-}Z]\{2\}[A\text{-}Z0\text{-}9]\{0,3\}$

```
CONSTRUCT {                                                                    knowledge graph model
  ?event rdfs:label ?label ; dcterms:isPartOf ?parent ; dcterms:spatial ?geoentity ;
         timex:hasStartTime ?startDateTime ; timex:hasEndTime ?endDateTime .
} WHERE {
 {                                                                             labels
   { ?event skos:prefLabel ?label } UNION { ?event skos:altLabel ?label } UNION { ?event rdfs:label ?label }
 }
                                                                               temporal
 ?event time:intervalFinishedBy ?end ; time:intervalStartedBy ?start.
 OPTIONAL{ ?end time:hasDateTimeDescription ?endTime .
   OPTIONAL{ ?endTime time:year ?endYear }
   OPTIONAL{ ?endTime periodo:latestYear ?endYear }
 }
 OPTIONAL{ ?start time:hasDateTimeDescription ?startTime .
   OPTIONAL{ ?startTime time:year ?startYear }
   OPTIONAL{ ?startTime periodo:earliestYear ?startYear }
 }
 OPTIONAL{ ?start (!periodo:aux)+ ?startYear. FILTER (isLiteral(?startYear)) }
 OPTIONAL{ ?end (!periodo:aux)+ ?endYear. FILTER (isLiteral(?startYear)) }
 FILTER( ?startYear >= "1900"^^xsd:gYear || xsd:integer(?startYear) >= 1900 ||
         ?endYear >= "1900"^^xsd:gYear || xsd:integer(?endYear) >= 1900 )

 BIND(xsd:dateTime(CONCAT(STR(?startYear),"-01-01T00:00:00")) AS ?startDateTime )
 BIND(xsd:dateTime(CONCAT(STR(?endYear),"-12-31T23:59:59")) AS ?endDateTime )
                                                                               geospatial
 OPTIONAL { ?event periodo:spatialCoverage ?geoentity }
 OPTIONAL { ?event dcterms:spatial ?geoentity }
 OPTIONAL { ?event dcterms:isPartOf ?parent. }
}
```

Figure 7: SPARQL query to extract event data (from historic periods) from PeriodO. (Namespaces as in Figure A.18)

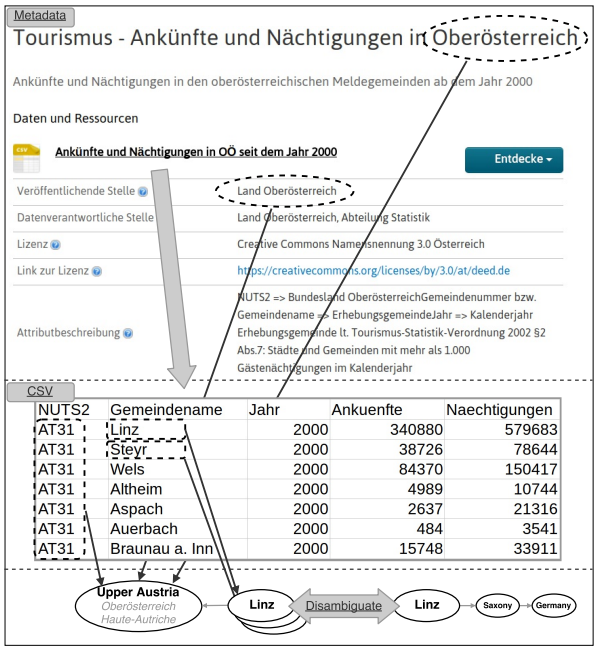

Figure 8: Geo-information in metadata and CSVs. Example dataset from the Austrian data portal: `https://www.data.gv.at/katalog/dataset/4d9787ef-e033-4c4f-8e50-65beb0730536`

all codes in the knowledge graph, and to filter out other strings, words, and decimals.[23]

Potential NUTS column (based on the regular expression) get mapped to the existing NUTS identifier. If this is possible for a certain threshold (set to 90% of the values) we consider a column as NUTS identifier and add the respective semantic labels. In case of potential postal codes the algorithm again tries to map to existing postal codes, however, we restrict the set of codes to the originating country of the dataset. This again results in a set of semantic labels which gets accepted with a 90% threshold.

The labelling of string columns, i.e. set of words or texts, uses all the labels from GeoNames and OSM and is based on the following disambiguation algorithm:

Value disambiguation. The algorithm in Figure 9 shows how we disambiguate a set of string values based on the surroundings. First, the function `get_context(values)` counts all potential parent GeoNames entities for all of the values. To disambiguate a single value we use these counts and select the GeoNames candidate with the most votes from the context values' parent regions; cf. `disamb_value(value)`. The function `get_geonames(value)` returns all potential GeoNames entites for an input string. Additionally, we

[23] $(([A\text{-}Z\backslash d])\{2,4\}|([A\text{-}Z]\{1,2\}.)?\backslash d\{2,5\}(\backslash s[A\text{-}Z]\{2,5\})?(.[\backslash d]\{1,4\})?)$

use the origin country of the dataset (if available) as a restriction, i.e., we only allow GeoNames labels from the matching country.

For instance, in Figure 8 the Austrian "Linz" candidate gets selected in favor of the German "Linz" because the disambiguation resulted in an higher score based on the matching predecessors "Upper Austria" and "Austria" for the other values in the column (Steyr, Wels, Altheim, ...).

```
# disambiguate a set of input values
def disamb_values(values, country):
  disambiguated = []
  cont_par = get_context(values)
  for v in values:
    v_id = disamb_value(v, country, cont_par)
    disambiguated.append(v_id)
  return disambiguated

# disambiguate a single value based on
# the parents of the surrounding values
def disamb_value(value, country, cont_par):
  candidates = get_geonames(value)
  c_score = {}
  for c in candidates:
    if country and country != c.country:
      continue
    else:
      parents = get_all_parents(c)
      for p in parents:
        c_score[c] += cont_par[p]
  top = sorted(c_score)[0]
  return top

# counts all parent values
def get_context(values):
  cont_par = {}
  for v in value:
    for c in get_geonames(value):
      parents = get_all_parents(c)
      for p in parents:
        cont_par[p] += 1
  return cont_par
```

Figure 9: Python code fragment for disambiguating a set of input values.

If no GeoNames mapping was found the algorithm tries to instantiate the string values with OSM labels from the base knowledge graph. Again, the same disambiguation algorithm is applied, however, with the following two preprocessing steps for each input value:

1. In order to better parse addresses, we use the Libpostal library (cf. Section 2) to extract streets and place names from strings.
2. We consider the context of a CSV row, e.g., if addresses in CSVs are separated into dedicated columns for street, number, city, state, etc. To do so we filter the allowed OSM labels by candidates within any extracted regions from the metadata description or from the corresponding CSV row (if geo-labels available).

4.1.2. Metadata descriptions

The CSVs' meta-information at the data portals often give hints about the respective regions covering the actual data. Therefore, we use this additional source and try to extract geo-entities from the titles, descriptions and publishers of the datasets:

1. As a first step, we tokenize the input fields, and remove any stopwords. Also, we split any words that are separated by dashes, underscores, semicolon, etc.
2. The input is then grouped by word sequences of up to three words, i.e. all single words, groups of two words, ..., and the previously introduced algorithm for mapping a set of values to the GeoNames labels is applied (including the disambiguation step).

Figure 8 gives an example dataset description found on the Austrian data portal `data.gv.at`. The labelling algorithm extracts the geo-entity "Upper Austria" (an Austrian state) from the title and the publisher "Oberösterreich". The extracted geo-entities are added as additional semantic information to the indexed resource.

4.2. Temporal labelling

Similarly to the geospatial cues, temporal information in Open Data comes in various forms and granularity, e.g., as datetime/timespan information in the metadata indicating the validity of a dataset, or year/month/time information in CSV columns providing timestamps for data points or measurements.

4.2.1. CSVs

To extract potential datetime values from the datasets we parse the columns of the CSVs using the Python dateutil library.[24] This library is able to parse a variety of commonly used date-time patterns (e.g., `''January 1, 2047''`, `''2012-01-19''`, etc.), however, we only allow values where the parsed year is in the range of 1900 and 2050.[25]

[24] `https://dateutil.readthedocs.io/en/stable/`

[25] The main reason for this restriction is that any input year easily yields to wrong mappings of e.g. postal codes, counts, etc.

For both sources of temporal information, i.e. metadata and CSV columns, we store the minimum and maximum (or *start* and *end* time) value so that we can allow range queries over the annotated data.

Datetime periodicity patterns. The algorithm in Figure 10 displays how we estimate any pattern of periodicity of the values in a column for a set of input datetime values. Initially, we check if all the values are the same (denoted as `static` column), e.g., a column where all cells hold "2018". Then we sort the values; however, note that this step could lead to unexpected annotations, because the underlying pattern might not be apparent in the unsorted column.

We compute all differences (`deltas`) between the input dates and check if all these deltas have approximately – with 10% margin – the same length. We distinguish `daily`, `weekly`, `monthly`, `quarterly`, and `yearly` pattern; in case of any other recurring pattern we return `other`.

```
def datetime_pattern(dates):
  # all the dates have the same value
  if len(set(dates)) == 1:
    return 'static'

  # sort the dates and compute the deltas
  dates = sorted(dates)
  deltas = [(d-dates[i-1])
            for i, d in enumerate(dates)][1:]

  for p, l in [('daily', delta(days=1)),
               ('weekly', delta(days=7)),
               ('monthly', delta(days=30)),
               ('quarterly', delta(days=91)),
               ('yearly', delta(days=365))]:
    # add 10% tolerance range
    if all(l-(l*0.1) < d < l+(l*0.1)
           for d in deltas):
      return p

  # none of the pre-defined pattern
  if len(set(deltas)) == 1:
    return 'other'

  # values do not follow a regular pattern
  return 'varying'
```

Figure 10: Python code fragment for estimating the datetime patterns of a set of values.

4.2.2. Metadata descriptions

We extract the datasets' temporal contexts from the metadata descriptions available at the data portals in two forms: (i) We extract the `published` and `last modified` information in case the portal provides dedicated metadata fields for these. (ii) We use the resource title, the resource description, the dataset title, the dataset description, and the keywords as further sources for temporal annotations. However, we prioritize the sources in the above order, meaning that we use the temporal information in the resource metadata rather than the information in the dataset title or description.[26]

The datetime extraction from titles and descriptions is based on the Heideltime framework [15] since this information typically comes as natural text. Heideltime supports extraction and normalization of temporal expressions for ten different languages. In case the data portal's origin language is not supported we use English as a fallback option.

4.3. Indexed Datasets & Evaluation

Our framework currently contains CSV tables from 11 European data portals from 10 different countries, cf. Table 1. We manually selected European governmental data portals (potentially also using NUTS identifier in their datasets) which are already monitored by the Open Data Portal Watch [13]. Note, that the notion of *datasets* on these data portals (wrt. Table 1) usually groups a set of resources; for instance, typically a dataset groups resources which provide the same content in different file formats. A detailed description and analysis of Open Data portals' resources can be found in [13]. The statistics in Table 1, i.e. the number of datasets and indexed CSVs is based on the third week of March 2018. The differing numbers of *CSVs* and *indexed* documents in the table can be explained by offline resources, parsing errors, etc. Also, we currently do not index documents larger than 10MB due to local resource limitations; the basic setup (using Elasticsearch for the indexed CSVs, cf Section 6) is fully scalable.

Table 2 lists the total number of annotated datasets. With respect to the spatial labelling algorithm, we were able to annotate columns of 3518 CSVs and metadata descriptions of 11231 CSVs (of a total of 15k indexed CSVs). For 3299 of the annotated CSVs our algorithm found GeoNames mappings, and for 292 OSM mappings. Regarding the temporal labelling, we detected date/time information in 2822 CSV columns and in 9112 metadata descriptions.

[26] For instance, consider a dataset titled "census data from 2000 to 2010" that holds several CSVs titled "census data 2000", "census data 2001", etc.: This metadata allows to infer that the temporal cues in the CSVs' titles are more accurate/precise than the dataset's title, which gives a more general time span for all CSVs.

portal	datasets	CSVs	indexed
total			15728
govdata.de	19464	10006	5646
data.gv.at	20799	18283	2791
offenedaten.de	28372	4961	2530
datos.gob.es	17132	8809	1275
data.gov.ie	6215	1194	884
data.overheid.nl	12283	1603	828
data.gov.uk	44513	7814	594
data.gov.gr	6648	414	496
data.gov.sk	1402	877	384
www.data.gouv.fr	28401	6038	258
opingogn.is	54	49	41

Table 1: Indexed data portals

	Spatial	*Temporal*	
Columns	Metadata	Columns	Metadata
3518	11231	2822	9112

Table 2: Total numbers of spatial and temporal annotations of metadata descriptions and columns.

Here we focus on evaluating the annotated geo-entities, and neglect the temporal annotations with the following two main reasons: First, the datetime detection over the CSV columns is based on the standard Python library `dateutil`. The library parses standard datetime formats (patterns such as `yyyy-mm-dd`, or `yyyy`) and the potential errors here are that we incorrectly classify a numerical column, e.g., classifying postal codes as years. As a very basic pre-processing, where we do not see a need for evaluation, we reduce the allowed values to the range 1900-2050 (with the drawback of potential false negatives), however, using the distribution of the numeric input values [16] would allow a more informed decision. Second, the labelling of metadata information is based on the temporal tagger Heideltime [15] which provides promising evaluations over several corpora.

Manual inspection of a sample set. To show the performance and limitations of our labelling approach we have randomly selected 10 datasets per portal (using Elasticsearch's built-in random function[27]) and from these again randomly select 10 rows, which resulted in a total of 101 inspected CSVs,[28] i.e. 1010 rows (with up to several dozen columns per CSV). Sampling datasets from different portals allows us to assess and compare the performance for different countries and data publishing strategies. The median percentage of annotated records (i.e. rows) per dataset (across all indexed datasets) is 92%; our sample is representative, in this respect, with a median of 88% annotated rows. The median number of total rows of all indexed datasets is lower (86 rows) than within the evaluated sample (287 rows). However, the overall number of rows varies widely with a mean of 1742 rows across all datasets, which indicates a large variety and non-even distribution of dataset sizes (between 1 and 221k rows).

As for the main findings, in the following let us provide a short summary; all selected datasets and their assigned labels can be found at `https://github.com/sebneu/geolabelling/tree/eu-data/jws_evaluation`.

Initially, we have to state that this evaluation is manually done by the authors and therefore restricted to our knowledge of the data portals' origin countries and their respective language, regions, sub-regions, postal codes, etc. For instance, we were able to see that our algorithm correctly labelled the Greek postal codes in some of the test samples from the Greek data portal `data.gov.gr`,[29] but that we could not assign the corresponding regions and streets.[30] However, as we are not able to read and understand the Greek language (and the same for the other non-English/German/Spanish portals) we cannot fully guarantee any potential mismatches or missing annotations that we did not spot during our manual inspections.

We categorize the datasets' labels by assessing the following dimensions: are there any correctly assigned labels in the dataset (c), are there any missing annotations (m), and did the algorithm assign incorrect links to GeoNames (g) or OSM (o); a result overview is given in Table 3.

[27] `https://www.elastic.co/guide/en/elasticsearch/guide/current/random-scoring.html`, last accessed 2018-04-01

[28] We only selected CSVs with assigned geo-entities – to provide a meaningful precision measure – which resulted in < 10 files for the smaller data portals, e.g., opingogn.is, and therefore in 101 files in total.

[29] E.g., `https://github.com/sebneu/geolabelling/blob/eu-data/jws_evaluation/data_gov_gr/0.csv`, the datasets use "T.K." in the headers to indicate these codes.

[30] The Greek data portal uses the Greek letters in their metadata and CSVs which would require a specialized label mapping wrt. lower-case mappings, stemming, etc.

total	c	m	g	o
101	87	53	12	5

Table 3: Correctly assigned labels (c), missing annotations (m), incorrect links to GeoNames (g) or OSM (o) in the dataset.

Out of 101 inspected datasets we identified in 87 CSVs correct annotations. In particular, for the Spain and the Greek data portal only in 50% of the test samples there were correct links, while for the 9 other indexed data portals we have a near to 100% rate. Regarding any missing annotations, we identified 53 datasets where our algorithm (and also the completeness of our spatial knowledge graph) needs improvements. For instance, in some datasets from the Netherlands' data portal[31] and also the Slovakian portal[32] we identified street names and addresses that could potentially mapped to OSM entries.

Regarding incorrect links there were only 12 files with wrong GeoNames and 5 files with wrong OSM annotations. An exemplary error that we observed here was that some files[33] contain a column with the value "Norwegen" ("Norway"): Since the file is provided at a German data portal, we incorrectly labelled the column using a small German region Norwegen instead of the country, because our algorithm prefers labels from the origin country of the dataset. Another example that we want to consider in future versions of our labelling algorithm is this wrong assignment of postal codes:[34] We incorrectly annotated a numeric column with the provinces of Spain (which use two-digit numbers as postal codes).

Table 4 displays the *precision*, *recall*, and *F1 score* for all sample records, i.e. for all annotated cells of the 101 sample CSVs. We want to emphasize that these results do not say anything about the quality of the data portals themselves. As mentioned in the above paragraph, again, we can see in Table 4 that the Greek (data.gov.gr) and the Spain data portal (datos.gob.es) have a notable drop in precision[35] while for the other portals the *total* precision is still at 86%. The total recall is at 73%, which again shows that our approach needs further improvements in terms of missed annotations and completeness of the spatial knowledge graph.

portal	precision	recall	F_1 score
total	**.86**	**.73**	.79
govdata.de	.89	.67	.77
data.gv.at	1	0.81	0.90
offenedaten.de	0.93	1	0.96
datos.gob.es	**.51**	.91	0.66
data.gov.ie	.98	.86	.92
data.overheid.nl	1	.29	.44
data.gov.uk	.98	.58	.73
data.gov.gr	**.51**	.64	.57
data.gov.sk	.82	.79	.81
www.data.gouv.fr	.98	.68	.81
opingogn.is	1	.72	.84

Table 4: Evaluation of the sample CSVs on record level.

5. Export RDF

We make our base knowledge graph and RDFized linked data points from the CSVs available via a SPARQL endpoint. Figure 11 displays an example extract of the RDF export of the knowledge graph. The sources of the aggregated links between the different entities and literals in our graph are indicated in the figure; we re-use the GeoNames ontology (`gn:`) for the hierarchical enrichments generated by our algorithms (see links `[m]`), and `owl:sameAs` for mappings to OSM relations and NUTS regions, cf. Figure 11.

Annotated data points. We export the linked data points from the CSVs in two ways: First, for any linked geo-entity `<g>` in our base knowledge graph, we add triples for datapoints uniquely linked in CSV resources (that is, values appearing in particular columns) by the following triple schema: if the

[31] E.g.,`https://github.com/sebneu/geolabelling/tree/eu-data/jws_evaluation/data_overheid_nl/4.csv`

[32] E.g., `https://github.com/sebneu/geolabelling/tree/eu-data/jws_evaluation/data_gov_sk/3.csv`

[33] `https://github.com/sebneu/geolabelling/blob/eu-data/jws_evaluation/offenedaten_de/0.csv`

[34] `https://github.com/sebneu/geolabelling/blob/eu-data/jws_evaluation/datos_gob_es/7.csv`

[35] There are streets in OSM that are labelled by an identifier (e.g. "2810 254 527") and, coincidentally, match columns in Greek datasets. Regarding the Spain datasets we incorrectly matched several columns containing the numbers 1-50: We mapped these to the fifty provinces of Spain, which use the numbers 1-50 as ID/zip codes. In future work we plan to include simple rules and heuristics to avoid such simple errors.

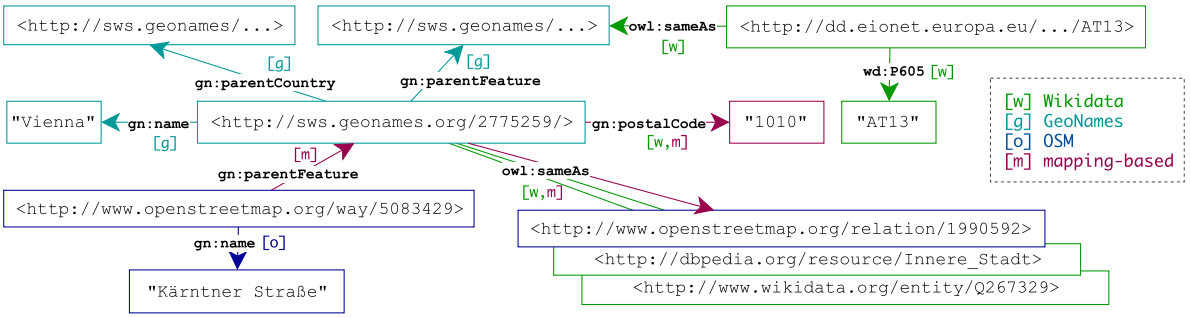

Figure 11: Example RDF export of the geo-entities knowledge graph.

entity `<g>` appears in a column in the given CSV dataset, i.e., the value $VALUE$ in that column has been labeled with `<g>`, we add a triple of the form

`<g>` <u#col> "$VALUE$" .

That is, we mint URIs for each column *col* appearing in a CSV accessible through a URL u by the schema u#col, i.e., using fragment identifiers. The column's name *col* is either the column header (if a header is available and the result is a valid URI) or a generic header using the columns' index `column1`, `column2`, etc. These triples are coarse grained, i.e. they do not refer to a specific row in the data. We chose this representation to enable easy-to-write, concise SPARQL queries like for instance:

```
SELECT ?name ?value
WHERE {
  ?geo <https://www.wien.gv.at/finanzen/ogd/
            hunde-wien.csv#Postal_CODE> ?value ;
       rdfs:label ?name .
}
```

The above query selects all values and their geo-annotations for the selected column named "Postal CODE" in a specific dataset about dog breeds in Vienna per district.

Second, a finer grained representation, which we also expose, provides exact table cells where a certain geospatial entity is linked to, using an extension of the CSVW vocabulary [17]: note that the CSVW vocabulary itself does not provide means to conveniently annotated table cells in column *col* and row n which is what we need here, so we define our own vocabulary extension for this purpose (for the moment, under the namespace `http://data.wu.ac.at/ns/csvwx#`).

We use the CSVW class `csvw:Cell` for an annotated cell and add the row number and value (using `csvw:rownum` and `rdf:value`) to it. We extend CSVW by the property `csvwx:cell` to refer from a `csvw:Column` (using again the fragmented identifier u#col) to a specific cell, and the property `csvwx:rowURL` to refer to the CSV's row (using the schema u#row=n). Here, the property `csvwx:refersToEntity` (cf. the blue line) connects a specific cell to the labelled geo-entity `<g>`. Analogously, in case of available (labelled) temporal information for a cell, we use the property `csvwx:hasTime`; cf. the turquoise line in the following example:

```
@prefix csvwx: <http://data.wu.ac.at/ns/csvwx#> .
@prefix csvw: <http://www.w3.org/ns/csvw#> .
<u#col> csvwx:cell [
  a csvw:Cell ; csvw:rownum n ;
  csvwx:rowURL <u#row=n> ;
  rdf:value "VALUE" ;
  csvwx:refersToEntity <g> ;
  csvwx:hasTime "DATE"8sd:dateTime
] .
```

Moreover, we denote the geospatial scope of the column itself by declaring the type of entities *within* which geographic scope appearing in the column. The idea here is that we annotate – on column level – the least common ancestor of the spatial entities recognized in cells within this column. E.g.,

<u#col> `csvwx:refersToEntitiesWithin` <g_1> .

with the semantics that the entities linked to *col* in the CSV u all refer to entities within the entity g_1 (such that g_1 is the least common ancestor in our knowledge graph.

This could be seen as a shortcut/materialization for a `CONSTRUCT` query as in Figure 12. Obviously, this query is very inefficient and we rather compute these least common ancestors per column during labeling/indexing of each column.

CSV on the Web. In order to complete the descriptions of our annotations in our RDF export, we describe all CSV resources gathered from the annotated Open Data Portals and their columns using the CSV on the Web (CSVW) [17] vocabulary,

```
CONSTRUCT
{ ?UrlCol csvwx:refersToEntitiesWithin ?G_1 }
WHERE {
 ?Col csvwx:cell [ csvwx:refersToEntity ?G ].
 ?G gn:parentFeature* ?G_1 .
 # all elements of this column have to share
 # parent feature ?G_1
 FILTER NOT EXISTS {
  ?Col csvwx:cell [ csvwx:refersToEntity ?G_ ].
  FILTER NOT EXISTS {
   ?G_ gn:parentFeature* ?G_1.
  }
 }
 # this parent feature is the least one that
 # fulfills this property:
 FILTER NOT EXISTS {
  ?G_2 gn:parentFeature ?G_1.
  ?Col csvwx:cell [ csvwx:refersToEntity ?G ].
  ?G gn:parentFeature* ?G_2 .
  # all elements of this column have to share
  # parent feature ?G_2
  FILTER NOT EXISTS {
   ?Col csvwx:cell [ csvwx:refersToEntity ?G__ ].
   FILTER NOT EXISTS {
    ?G__ gn:parentFeature* ?G_2.
   }
  }
 }
}
```

Figure 12: SPARQL `CONSTRUCT` query to materialize the geographic scope of a column.

re-using the following parts of the CSVW schema. Firstly, we use the following scheme to connect our aforementioned annotations to the datasets:

```
@prefix csvw: <http://www.w3.org/ns/csvw#> .
@prefix dcat: <http://www.w3.org/ns/dcat#> .

<d> a dcat:Dataset [ dcat:distribution
          [ dcat:accessURL u ] ].
[] csvw:url u;
   csvw:tableSchema [
     csvw:column (<u#col_1> <u#col_2>... <u#col_n>)] .

<u#col_1> a csvw:name "col_1" ; csvw:datatype d_col_1 .
<u#col_2> a csvw:name "col_2" ; csvw:datatype d_col_2 .
```

Then, we enrich this skeleton with further CSVW annotations that we can extract automatically from the respective CSV files. Figure 13 displays an example export for a CSV resource. The blank node `_:csv` represents the CSV resource which can be downloaded at the URL at property `csvw:url`. The values of the properties `dcat:byteSize` and `dcat:mediaType` are values of the corresponding HTTP header fields. The dialect description of the CSV can be found via the blank node `_:dialect` at property `csvw:dialect` and the columns of the CSV are connected to the `_:schema` blank node (describing the `csvw:tableSchema` of the CSV).

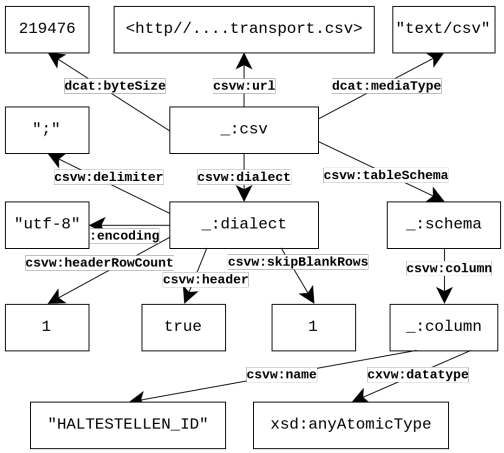

Figure 13: Example export of CSVW metadata for a dataset.

6. Search & Query Interface

Our integrated prototype for a spatio-temporal search and query system for Open Data currently consists of three main parts: First, the *geo-entities DB* and *search engine* in the back end (Section 6.1), second the *user interface* and *APIs* (Section 6.2), and third, access to the above described RDF exports via an *SPARQL endpoint* (Section 6.3).

6.1. Back End

All labels from all the integrated datasets and their corresponding geo-entities are stored in a look-up store, where we use the NoSQL key-value database MongoDB. It allows an easy integration of heterogeneous data sources and very performant look-ups of keys (e.g., labels, GeoNames IDs, postal codes, etc. in our case).

Further, we use Elasticsearch to store and index the processed CSVs and their metadata descriptions. In our setup, an Elasticsearch document corresponds to an indexed CSV and consists of all cell values of the table (arranged by columns), the potential geo-labels for a labelled column, metadata of the CSV (e.g., the data portal, title, publisher, etc.), the temporal annotations, and any additional labels extracted from the metadata.

The different components all have an impact on the performance and efficiency of the system. The indexing performance depends on the MongoDB database for label look-ups, the time-tagger Heideltime, and, Elasticsearch for storing the datasets. To show the efficiency and scalability of our approach, we timed the indexing of a sample of 2160

datasets, with an average file size of ∼50kB. The total processing time for all dataset was 16.8 hours – deactivated parallelization, including download, parsing, and processing time – whereof 8 hours were consumed by the labelling algorithms. Notably, the median total time for indexing a dataset is only 1.2 seconds, with a median time of 0.7 seconds for the labelling algorithms.[36]

6.2. User interface

The user interface, available at `http://data.wu.ac.at/odgraphsearch/`, allows search queries for geo-entities but also full-text matches. Note, that the current UI implements geo-entity search using auto-completion of the input (but only suggesting entries with existing datasets) and supports full-text querying by using the "Enter"-key in the input form. The screenshot in Figure 14 displays an example query for the Austrian city "Linz". The green highlighted cells in the rows below show the annotated labels, for instance, the annotated NUTS2 code "`AT31`" in the second result in Figure 14.

Also, we add facets to filter datasets relevant to a particular period either by auto-completion in a separate field to filter the time period by a period/event label, or by choosing start and end dates via sliders (cf. Figure 14). The users can decide to apply this filter to temporal information in title and description of the dataset, or the CSV columns.

By separating the search at these two levels we do not mix dates within the data and the metadata level. For instance, the metadata could have date/time that refers to the present such as created, modified, etc. while in the datasets there can be a mixture of dates referring to temporal information or events (e.g., a column of birth dates).

The chosen geo-entities and durations which are fixed via these lookups in our search index through the UI are passed on as parameters as a concrete geo-ID and/or start&end-date to our API, which we describe next.

Additionally, the web interface provides APIs (`http://data.wu.ac.at/odgraphsearch/api`) to retrieve the search results, all indexed datasets, and the RDF export per dataset. To allow programmatic access to the search UI we offer the following HTTP GET API:

[36] We deliberately discuss the median since the shape and size of the datasets can vary widely, which heavily influences the total and mean values.

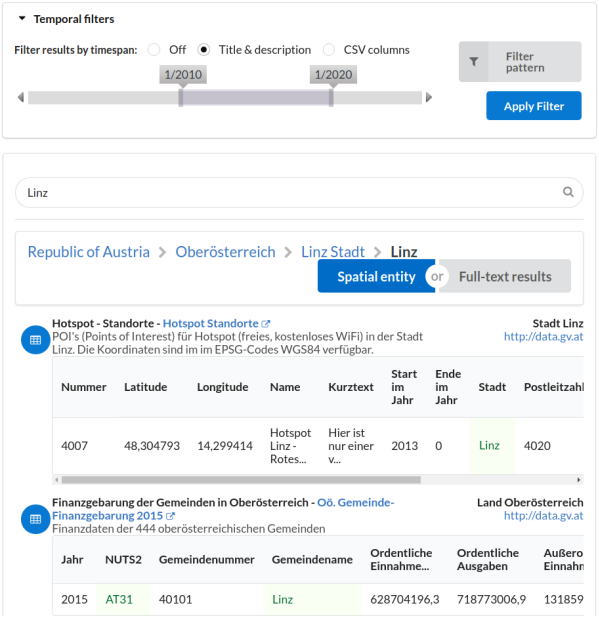

Figure 14: Screenshot of of an example search at the UI.

`/api/v1/get/datasets?l=`{*GeoIDs*}
`&limit=`{*limit*}`&offset=`{*offset*}
`&start=`{*startDate*}`&end=`{*endDate*}
`&mstart=`{*startDate*}`&mend=`{*endDate*}
`&periodicity=`{*dateTimePattern*}
`&q=`{*keyword*}

The API takes multiple instances of geo IDs, that is, `GeoNames` or OSM IDs (formatted as `osm:{ID}`) using parameter `l`, a `limit` and an `offset` parameter, which restricts the amount of items (datasets) returned, and an optional white space separated list of keywords (`q`) as full-text query parameter to enable conventional keyword search in the metadata and header information of the datesets. To restrict the results to a specific temporal range we implemented the parameters `mstart`, `mend` (for filtering a time range from the metadata-information), and `start`, `end` (for the min and max values of date annotations from CSV columns). The parameter `periodicity` allows to filter for datetime periodicity patterns such as "yearly", "monthly", or "static" (in case there is only a single datetime value in this column), cf. Section 4.2.1 for a detailed description of the periodicity patterns.

The output consists of a JSON list of documents that contain the requested GeoNames/OSM IDs or any tables matching the input keywords.

6.3. SPARQL endpoint

We offer a SPARQL endpoint at `http://data.wu.ac.at/odgraphsearch/sparql` where we provide the data as described in Section 5. Currently, as of the first week of April 2018, the endpoint contains 88 million triples: 15 million hierarchical relations using the `gn:parentFeature` relation, 11768 CSVs (together with their CSV on the Web descriptions), where we added a total of 5 million geo-annotations using the `csvwx:refersToEntity` property, and 1.3 million datetime-annotations using the `csvwx:hasTime` annotation.

Example queries. The first example in Figure 15 lists all datasets from Vienna, using the `csvwx:refersToEntity` metadata annotation, and only lists CSVs where there exists a column with dates within the range of the last Austrian legislative period, using the Wikidata entities of the past two elections.

```
SELECT ?d ?url WHERE {
  # dates of the past two elections in Austria
  wd:Q1386143 timex:hasStartTime ?t1 .
  wd:Q19311231 timex:hasStartTime ?t2 .

  ?d dcat:distribution [
    dcat:accessURL ?url ;
    # the min and max date values
    timex:hasStartTime ?start ;
    timex:hasEndTime ?end
  ] .
  # filter datasets about Vienna
  ?d csvwx:refersToEntity
      <http://sws.geonames.org/2761369/> .

  FILTER((?start >= ?t1) && (?end <= ?t2))
}
```

Figure 15: Example SPARQL query using the spatial property `csvwx:refersToEntity` and the temporal properties `timex:hasStartTime` and `timex:hasEndTime`.

The next example query in Figure 16 combines text search for time periods with a structured query for relevant data; it looks for information of datasets about a time period before the 2nd World War, called the "Anschluss movement" (i.e., the preparation of the annexation of Austria into Nazi Germany) and queries for all available CSV rows where a date within this period's range (1918-1938, according to PeriodO), and a geo-entity within the period's spatial coverage location (i.e. Austria) occurs.

GeoSPARQL. GeoSPARQL [18] extends SPARQL to a geographic query language for RDF data. It

```
SELECT ?d ?url ?rownum WHERE {
  # get the "Anschluss movement"
  ?p rdfs:label ?L.
  FILTER (CONTAINS(?L, "Anschluss movement") ) .
  ?p timex:hasStartTime ?start ;
       timex:hasEndTime ?end ;
       dcterms:spatial ?sp .
  # find the GeoNames entities
  ?spatial owl:sameAs ?sp .
  ?d dcat:distribution [ dcat:accessURL ?url ] .
  [] csvw:url ?url ;
     csvw:tableSchema ?s .
  # find a cell where date falls in the range
  # of the found period
  ?s csvw:column ?col1 .
  ?col1 csvwx:cell [
    csvw:rownum ?rownum ;
    csvwx:hasTime ?cTime
  ]
  FILTER((?cTime >= ?start) &&
         (?cTime <= ?end))
  # find another cell in the same row where the
  # geo-entity has the spatial coverage area of
  # the found period as the parent country
  ?s csvw:column ?col2 .
  ?col2 csvwx:cell [
    csvw:rownum ?rownum ;
    csvwx:refersToEntity [
      gn:parentCountry ?spatial
    ]
  ]
}
```

Figure 16: Example SPARQL query combining text search for a time period with a structured query for datasets within the period's temporal and spatial coverage.

defines a small ontology to represent geometries (i.e., points, polygons, etc.) and connections between spatial regions (e.g., contains, part-of, intersects), as well as a set of SPARQL functions to test such relationships. The example query in Figure 17 (namespaces as in Figure A.18) uses the available polygon of the Viennese district "Leopoldstadt" to filter all annotated data points within the borders of this district.

While we do not yet offer a full GeoSPARQL endpoint for our prototype yet (which we leave to a forthcoming next release), our RDFized datasets and knowledge graph is GeoSPARQL "ready", i.e. having all the geo-coordinates and polygons in the endpoint using the GeoSPARQL vocabulary; an external GeoSPARQL endpoint could already access our data using the SERVICE keyword and evaluate the GeoSPARQL specific functions locally, or simply import our data.

7. Related Work

The European Union identified the issue of insufficient description of public datasets and conducted several activities towards metadata stan-

```
SELECT ?d ?url ?rownum WHERE {
  # get the geometry of the Viennese district "Leopoldstadt"
  <http://sws.geonames.org/2772614/>
              geosparql:hasGeometry ?polygon .

  ?d dcat:distribution [ dcat:accessURL ?url ] .
  [ csvw:url ?url ; csvw:tableSchema ?s ].
  # select the geometries of any annotated cells
  ?s csvw:column ?col .
  ?col csvwx:cell [ csvw:rownum ?rownum ;
    csvwx:refersToEntity [geosparql:hasGeometry ?geoentity]]

  # filter all annotated data points
  # within the polygon of Leopoldstadt
  FILTER(geof:sfWithin(?g, ?polygon))
}
```

Figure 17: Example GeoSPARQL query over using the available geometries – not yet available via the endpoint.

dards across European portals: The DCAT Application Profile for Data Portals in Europe (DCAT-AP)[37] aims towards the integration of datasets from different European data portals. In its current version (v1.1) it extends the existing DCAT schema [14] by a set of additional properties, e.g., it allows to specify the version and the period of time for a dataset. Going one step further, the INSPIRE directive[38] and the GeoDCAT-AP specification[39] have more restrictive requirements for spatial metadata, i.e., they model spatial coverage either as a bounding box, or using a geographic identifier; notably, the specification also mentions GeoNames as potential identifiers. The main barrier with these approaches is a lacking adoption: We could not see a broad use of these standards across the portals (neither in terms of vocabulary nor in complete spatial descriptions) and therefore could not further use them. In principle, our approach distinguishes from these activities by not only having the spatio-temporal descriptions but also interlinking the datasets to external sources, i.e. to GeoNames, Wikidata, and OSM. Also, these standards only allow descriptions on datasets level, whereas we annotate the data on record level as well.

The 2013 study by Janowicz et al. [19] gives an overview of Semantic Web approaches and technologies in the geospatial domain. Among the Linked Data repositories and ontologies listed in the article we also find the GeoNames ontology (cf. Section 2), the W3C Geospatial Ontologies [20], and the GeoSPARQL Schemas [18]. However, when looking into the paper's listed repositories, most of them (6/7) were not available, i.e. offline, which seems to indicate that many efforts around Geo-Linked data have unfortunately not been pursued in a sustainable manner.

The 2012 project LinkedGeoData [21] resulted in a Linked Data resource, generated by converting a subset of OpenStreetMap data to RDF and deriving a lightweight ontology from it. In [22] the authors describe their attempts to further connect GeoNames and LinkedGeoData, using string similarity measures and geometry matching. However, while LinkedGeoData is also listed in [19] as a geospatial Linked Data repository, unfortunately, it was not available online at the time of writing this paper. Also, this work was complementary to ours, as we do not focus on matching and entity alignment, but rather on the integration of existing structured entities from different geo and temporal (Linked) Data sources. The recent effort "Sophox"[40] can be seen as a conceptual continuation of the LinkedGeoData project: actually intended as a cleanup tool, it allows SPARQL queries over OSM elements and tags. In the future we could also consider directly using the SPARQL interface of Sophox.

The GeoKnow project [23] is another attempt to provide and manage geospatial data as Linked Data. GeoKnow provides a huge toolset to process these datasets, including the storage, authoring, interlinking, and geospatially-enabled query optimization techniques.

The project PlanetData (2010 to 2014), funded by the European Commission, released an RDF mapping of the NUTS classifications[41] [24] using the GeoVocab vocabulary.[42] This dataset models the hierarchical relations of the regions, provides labels and polygons. Unfortunately, the project does not include external links to GeoNames, or Wikidata, except for the country level, i.e. there are only 28 links to the corresponding GeoNames entries of the EU member states.

Complementary to our approach to access street addresses via OSM, Open Addresses[43] is a global collection of address data sources, which could be considered for future work as an additional dataset to feed into our base knowledge graph. The manually collected and homogenized dataset consists of a

[37] https://joinup.ec.europa.eu/release/dcat-ap-v11

[38] https://inspire.ec.europa.eu/

[39] https://joinup.ec.europa.eu/release/geodcat-ap/v101

[40] https://wiki.openstreetmap.org/wiki/Sophox, last accessed 2018-09-03

[41] http://nuts.geovocab.org/, last accessed 2018-01-05

[42] http://geovocab.org/, last accessed 2018-01-05

[43] https://openaddresses.io/, last accessed 2018-04-01

total of 478M addresses; street names, house numbers, and post codes combined with geographic coordinates, harvested from governmental datasets of the respective countries.

A conceptually related approach, although not focusing on geo-data, is the work by Taheriyan et al. [25], who learn the semantic description of a new source given a set of known semantic descriptions as the training set and the domain ontology as the background knowledge.

In [26] Paulheim provides a comprehensive survey of refinement methods, i.e., methods that try to infer and add missing data to a graph, however, these approaches work on graphs in a domain independent setting and do not focus on temporal and spatial knowledge. Still, in some sense, we view our methodology of systematic Knowledge Graph aggregation from Linked Data sources via declarative, use-case specific, minimal mappings as potentially complementary to the domain-independent methods mentioned therein, i.e., we think in future works, such methods should be explored in combination.

Most related wrt. the construction of the temporal knowledge graph is the work by Gottschalk and Demidova [27] (2018): they present a temporal knowledge graph that integrates and harmonizes event-centric and temporal information regarding historical and contemporary events. In contrast to [27] we additionally integrate data from PeriodO [9] and focus on periods in a geospatial context. This work is built upon [28] where the authors extract event information from the Wikipedia Current Events Portal (WCEP). In future work we want to connect the resource from [27], since the additional data extracted from the WCEP and WikiTimes interface is in particular interesting for our framework. Similar to [27], [29] gather temporal information from knowledge bases, and additionally from the Web of documents. The extracted facts get then mapped and merged into time intervals.

In [10], Rospocher et al. build a knowledge graph directly from news articles, and in [30] by extracting event-centric data from Wikipedia articles. These approaches work over plain text (with the potential drawback of noisy data) while we integrate existing structured sources of temporal information; therefore these are complementary/groundwork to our contributions.

Modelling and querying geospatial information has also been discussed conceptually in the literature: [31] present an ontology design pattern derived from time geography, and [32] discuss the requirements of a geospatial search platform and present a geospatial registry.

8. Conclusions

Governmental data portals such as Austria's `data.gv.at` or the UK's `data.gov.uk` release local, regional and national data to a variety of users (citizens, businesses, academics, civil servants, etc.). As this data is mainly collected as part of census collections, infrastructure assessments or any other, secondary output data, these resources almost always contain or refer to some kind of geographic and temporal information; for instance, think of public transport data, results of past elections, demographic indicators, etc. Search across these dimensions seems therefore natural, i.e., we have identified the spatial and temporal dimensions as the crucial, characterizing dimensions of datasets on such data portals.

In order to enable such search and to integrate these datasets in the LOD cloud (as they are mainly published as CSVs [13]) we have achieved the following tasks in this work:

- We have described a hierarchical knowledge graph of spatial and temporal entities in terms of SPARQL queries, as well as the integration of temporal information and its interlinkage with the geospatial-knowledge from various Linked data sources (GeoNames, OSM, Wikidata, PeriodO), where our general approach is extensible to adding new sources; further details of the construction are provided in the Appendix.
- We have described algorithms to annotate CSV tables and their respective metadata descriptions from Open Data Portals and we have annotated datasets and metadata from 11 European data portals.
- To demonstrate the performance and limitations of our spatio-temporal labelling we have evaluated the annotations by manual inspection of a random sample per data portal, where we identified correct geo-annotations for around 90% of the inspected datasets.
- To access and query the data, we offer an user interface, RESTful APIs and a SPARQL endpoint, which allows structured queries over our spatio-temporal annotations.

To the best of our knowledge, this is the first work addressing a spatial-temporal labelling and allowing structured spatio-temporal search of datasets based on a knowledge graphs of temporal and geo-entities.

To further improve geo-labelling, we are currently working on parsing coordinates in datasets. To do so we have to consider that the long/lat pairs potentially come in column groups, i.e., one column per coordinate, which we might need to combine. Having all the geometries for the geo-entities and data points, we want to integrate a visual representation and search interface for datasets by displaying and filtering the datasets/records on a map.

While CSV is a popular and dominant data-publishing format on the Web [13], we also want to extend our indexing to other popular Open Data formats (such as XLS and JSON). Additionally, we want to test how well our approaches could be applied to unstructured or semi-structured data and other domains such as tweets or web pages (e.g., newspaper articles), or complementarily, we could use our knowledge graph, along with known methods for temporal and geo-labelling of such unstructured sources, to link them to (supporting) data, to enable for instance fact checking. The applications of Open Data sources searchable and annotated in such a manner seem promising and widespread.

References

[1] J. Attard, F. Orlandi, S. Scerri, S. Auer, A systematic review of open government data initiatives, Government Information Quarterly 32 (4) (2015) 399 – 418. doi:https://doi.org/10.1016/j.giq.2015.07.006.
URL http://www.sciencedirect.com/science/article/pii/S0740624X1500091X

[2] M. B. Gurstein, Open data: Empowering the empowered or effective data use for everyone?, First Monday 16 (2).

[3] S. Kubler, J. Robert, S. Neumaier, J. Umbrich, Y. L. Traon, Comparison of metadata quality in open data portals using the analytic hierarchy process, Government Information Quarterly-doi:https://doi.org/10.1016/j.giq.2017.11.003.
URL http://www.sciencedirect.com/science/article/pii/S0740624X16301319

[4] E. Kacprzak, L. M. Koesten, L.-D. Ibáñez, E. Simperl, J. Tennison, A query log analysis of dataset search, in: J. Cabot, R. De Virgilio, R. Torlone (Eds.), Web Engineering, Springer International Publishing, Cham, 2017, pp. 429–436.

[5] M. Posada-Sánchez, S. Bischof, A. Polleres, Extracting geo-semantics about cities from openstreetmap, in: Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016), Leipzig, Germany, September 12-15, 2016., 2016.
URL http://ceur-ws.org/Vol-1695/paper39.pdf

[6] International Organization on Standardization, ISO 3166-1, Codes for the representation of names of countries and their subdivisions (2013).
URL https://www.iso.org/standard/63545.html

[7] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, C. Bizer, Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia, Semantic Web 6 (2) (2015) 167–195. doi:10.3233/SW-140134.
URL https://doi.org/10.3233/SW-140134

[8] D. Vrandečić, M. Krötzsch, Wikidata: A free collaborative knowledgebase, Commun. ACM 57 (10) (2014) 78–85. doi:10.1145/2629489.
URL http://doi.acm.org/10.1145/2629489

[9] P. Golden, R. B. Shaw, Nanopublication beyond the sciences: the periodo period gazetteer, PeerJ Computer Science 2 (2016) e44. doi:10.7717/peerj-cs.44.
URL https://doi.org/10.7717/peerj-cs.44

[10] M. Rospocher, M. van Erp, P. Vossen, A. Fokkens, I. Aldabe, G. Rigau, A. Soroa, T. Ploeger, T. Bogaard, Building event-centric knowledge graphs from news, J. Web Sem. 37-38 (2016) 132–151. doi:10.1016/j.websem.2015.12.004.
URL https://doi.org/10.1016/j.websem.2015.12.004

[11] D. McGuinness, T. Lebo, S. Sahoo, The PROV Ontology (PROV-O), http://www.w3.org/TR/prov-o/ (Apr. 2013).

[12] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, M. Arias, Binary RDF representation for publication and exchange (HDT), J. Web Sem. 19 (2013) 22–41. doi:10.1016/j.websem.2013.01.002.
URL https://doi.org/10.1016/j.websem.2013.01.002

[13] S. Neumaier, J. Umbrich, A. Polleres, Automated quality assessment of metadata across open data portals, J. Data and Information Quality 8 (1) (2016) 2:1–2:29. doi:10.1145/2964909.
URL http://doi.acm.org/10.1145/2964909

[14] F. Maali, J. Erickson, Data Catalog Vocabulary (DCAT), http://www.w3.org/TR/vocab-dcat/ (Jan. 2014).

[15] J. Strötgen, M. Gertz, Multilingual and cross-domain temporal tagging, Language Resources and Evaluation 47 (2) (2013) 269–298. doi:10.1007/s10579-012-9179-y.

[16] S. Neumaier, J. Umbrich, J. X. Parreira, A. Polleres, Multi-level semantic labelling of numerical values, in: International Semantic Web Conference, Springer, 2016, pp. 428–445.
URL https://doi.org/10.1007/978-3-319-46523-4_26

[17] R. Pollock, J. Tennison, G. Kellogg, I. Herman, Metadata Vocabulary for Tabular Data, https://www.w3.org/TR/2015/REC-tabular-metadata-20151217/, W3C Recommendation (Dec. 2015).

[18] M. Perry, J. Herring, OGC GeoSPARQL - A geographic query language for RDF data, OGC Implementation Standard. Sept.

[19] K. Janowicz, S. Scheider, B. Adams, A geo-semantics

flyby, in: Reasoning web. Semantic technologies for intelligent data access, Springer, 2013, pp. 230–250.
[20] J. Lieberman, R. Singh, C. Goad, W3c geospatial ontologies, Incubator group report, W3C.
[21] C. Stadler, J. Lehmann, K. Höffner, S. Auer, Linkedgeodata: A core for a web of spatial open data, Semantic Web 3 (4) (2012) 333–354.
[22] S. Hahmann, D. Burghardt, Connecting linkedgeodata and geonames in the spatial semantic web, in: 6th International GIScience Conference, 2010.
[23] J. Lehmann, S. Athanasiou, A. Both, A. García-Rojas, G. Giannopoulos, D. Hladky, J. J. Le Grange, A.-C. N. Ngomo, M. A. Sherif, C. Stadler, et al., Managing geospatial linked data in the geoknow project. (2015).
[24] A. Harth, Y. Gil, Geospatial data integration with linked data and provenance tracking, in: W3C/OGC Linking Geospatial Data Workshop, 2014, pp. 1–5.
[25] M. Taheriyan, C. A. Knoblock, P. Szekely, J. L. Ambite, A graph-based approach to learn semantic descriptions of data sources, in: The Semantic Web – ISWC 2013, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 607–623.
[26] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, Semantic Web 8 (3) (2017) 489–508. doi:10.3233/SW-160218.
URL https://doi.org/10.3233/SW-160218
[27] S. Gottschalk, E. Demidova, Eventkg: A multilingual event-centric temporal knowledge graph, in: The Semantic Web - 15th International Conference, ESWC 2018, 2018.
[28] G. B. Tran, M. Alrifai, Indexing and analyzing wikipedia's current events portal, the daily news summaries by the crowd, in: Proceedings of the 23rd International Conference on World Wide Web, WWW '14 Companion, ACM, New York, NY, USA, 2014, pp. 511–516. doi:10.1145/2567948.2576942.
URL http://doi.acm.org/10.1145/2567948.2576942
[29] A. Rula, M. Palmonari, A.-C. Ngonga Ngomo, D. Gerber, J. Lehmann, L. Bühmann, Hybrid acquisition of temporal scopes for rdf data, in: V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin, S. Staab, A. Tordai (Eds.), The Semantic Web: Trends and Challenges, Springer International Publishing, Cham, 2014, pp. 488–503.
[30] A. Spitz, M. Gertz, Terms over LOAD: leveraging named entities for cross-document extraction and summarization of events, in: Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016, 2016, pp. 503–512. doi:10.1145/2911451.2911529.
URL http://doi.acm.org/10.1145/2911451.2911529
[31] C. Keßler, C. J. Farmer, Querying and integrating spatial–temporal information on the web of data via time geography, Web Semantics: Science, Services and Agents on the World Wide Web 35 (2015) 25 – 34, geospatial Semantics. doi:https://doi.org/10.1016/j.websem.2015.09.005.
URL http://www.sciencedirect.com/science/article/pii/S1570826815000888
[32] P. Corti, B. G. Lewis, T. Kralidis, J. Mwenda, Implementing an open source spatio-temporal search platform for spatial data infrastructures, Tech. rep., PeerJ Preprints (2016).
[33] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, N. Lindström, Json-ld 1.0: A json-based serialization for linked data, W3c recommendation, W3C (2014).
URL https://www.w3.org/TR/json-ld/
[34] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub, The geojson format, RFC 7946, IETF (Aug. 2016).
URL https://tools.ietf.org/html/rfc7946
[35] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, P. Colpaert, Triple Pattern Fragments: a low-cost knowledge graph interface for the Web, Journal of Web Semantics 37–38 (2016) 184–206. doi:doi:10.1016/j.websem.2016.03.003.
URL http://linkeddatafragments.org/publications/jws2016.pdf

Appendix A. Realizing the Queries from Section 3

As mentioned in Section 3, we extract the relevant RDF Data for constructing our knowledge graph from different Linked Data Sources, which provide RDF[44] data either in the form of dumps or via SPARQL endpoints, where we presented the respective SPARQL queries that theoretically should suffice to extract the data relevant for us in Section 3. A common problem with these sources is however that either such a SPARQL endpoint is not available or does not support complex queries. To this end, we discuss in this appendix how such limitations could be circumvented in the specific cases. We note that we expect the presented workaround could be similarly applied to other use cases for extracting relevant data from large RDF dumps or public endpoints, so we hope the discussion herein might be useful also for others.

Figure A.18 lists all the namespaces that are used here in the Appendix and throughout the paper.

Appendix A.1. Extracting postal codes and NUTS identifier from Wikidata

Due to the fact that the query in Figure 3 resulted in timeouts at the Wikidata SPARQL endpoint we split the query in sub-queries.[45] The task of extracting the NUTS identifier provides mappings for 1316 (out of 1716) NUTS codes. The missing 400 codes are NUTS regions where no Wikidata

[44] We note OSM here as an exception; the JSON-data we extract from OSM is not directly in an RDF serializtation, but the provided JSON can be easily converted to JSON-LD.

[45] `SELECT ?s ?nuts ?geonames WHERE {?s wdt:P605 ?nuts. ?s wdt:P1566 ?geonames}` to get the NUTS-to-GeoNames mappings. Similarly for the postal code property `wdt:P281`.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX csvw: <http://www.w3.org/ns/csvw#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX geosparql: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX periodo: <http://n2t.net/ark:/99152/p0v#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX gn: <http://www.geonames.org/ontology#>
PREFIX osm: <https://www.openstreetmap.org/>
PREFIX timex: <http://data.wu.ac.at/ns/timex#>
PREFIX csvwx: <http://data.wu.ac.at/ns/csvwx#>
```

Figure A.18: Namespaces used throughout the paper.

and/or GeoNames entry exists because, strictly speaking, there is no such corresponding administrative region. For instance, the Austrian NUTS regions AT126 and AT127 are called "Wiener Umland/Nordteil" and "Wiener Umland/Südteil", however, these are no political districts, but statistical entities grouping a *set* of districts Wikidata/-GeoNames entity to map.

To complement the set of postal codes in Wikidata we use the extra postal code dataset by GeoNames[46] which consists of a total of 1.1M entries from 84 countries. For each code it provides a *place name*, and (depending on the country) several parent region/subdivion names. Based on these names we use a simple heuristic to map the postal codes to GeoNames entities: We split place names in the dataset by separators (white spaces, "-", "/")[47] and try to find GeoNames entries, in the respective country, with matching names.

Appendix A.2. Extracting Spatial Data from OSM

Since there exists – to the best of our knowledge – no up-to-date and integrated linked data version of OSM, we extract OSM relations, ways and nodes and map these to our spatial knowledge graph. To do so we perform the following steps on a local extract of OSM:[48]

1. OSM provides different administrative levels for their relations, e.g., the relation which represents the states of a country, their subdivisions, and so on.[49] We use the alignment of these administrative levels with the previously introduced NUTS identifier to add the mappings to GeoNames: We perform lookups with the GeoNames labels of the NUTS 1, 2, and 3 regions at OSM's Nominatim service.[50] This service returns a set of potential candidate OSM relations for a given label. We select the correct relation (i.e. OSM region) by choosing the OSM relation at the same administrative/NUTS level as the corresponding GeoNames region.
2. Having the mapping for the countries' regions we again use OSM Nominatim to get the polygons for all sub-regions. These polygons can be used to extract any street names, places, etc. from a OSM data extract.[51]
3. We introduce relations between the extracted OSM entities and their parent GeoNames regions (which we get from the Nominatim mappings). This hierarchical relations are indicated using the `:osm_region` property in the conceptual SPARQL query in Figure 4, Section 3.1.

The OSM polygons returned by OSM's Nominatim service in Item 2 are not available as RDF, so we try to interpret the JSON from Nominatim as JSON-LD. This could be done relatively straightforwardly by adding to the JSON you get by e.g. calling `https://nominatim.openstreetmap.org/reverse?osm_id=1990594&osm_type=R&polygon_geojson=1&format=json` for obtaining the data for OSM id `1990594` (i.e. Vienna's district "Leopoldstadt", and extending the returned JSON with a JSON-LD [33] context:

```
"@context": {
  "@vocab": "https://data.wu.ac.at/ns/osm#"
}
```

However, the query from Figure 4 still would not work "as is", since OSM returns the coordinates of its entities as GeoJSON [34], which due to the way that GeoJSON represents geometries as nested

[46] `http://download.geonames.org/export/zip/`, last accessed 2018-03-28

[47] We add this preprocessing step because there are many translated place names separated by slash or comma.

[48] We use Geofabrik, `http://download.geofabrik.de/`, to download extracts of OSM on a country level.

[49] `http://wiki.openstreetmap.org/wiki/Tag:boundary\%3Dadministrative`

[50] `http://nominatim.openstreetmap.org`

[51] OSM provides a tool, Osmosis `http://wiki.openstreetmap.org/wiki/Osmosis`, to process polygons on OSM data dumps

JSON arrays, is incompatible with JSON-LD.[52] We therefore pre-convert GeoJSONs nested way of representing polygon's to the format compatible with GeoSPARQL [18], by replacing JSON attributes of the form:

```
"geojson": {
   "type":"Polygon",
   "coordinates":
       [[[lat_1,long_1], ... , [lat_n,long_n]]]
}
```

with:

```
"geojson": {
   "type":"Polygon",
   "coordinates":
       "POLYGON(lat_1 long_1, ... , lat_n long_n)"
}
```

and extend the context to:

```
"@context": {
   "@vocab": "https://data.wu.ac.at/ns/osm#",
   "coordinates": {
      "@type":
"http://www.opengis.net/ont/geosparql#wktLiteral"
   }
}
```

in a simple pre-processing step. The query in Figure 4 works as expected on this respectively preprocessed data from Nominatim.

Appendix A.3. Extracting Temporal Data from Wikidata

The query to extract event and time period data from Wikidata is shown in Figure 5; however as mentioned above, this query times out on the public endpoint. We note that Wikidata contained (at the time of writing) 4.8b RDF triples, so retrieving a dump and trying to extract the relevant information by setting up a local SPARQL endpoint also didn't seem an attractive solution. Rather, we propose a combination of

1. extracting relevant triples to answer the query via HDT [12] and
2. executing targeted `CONSTRUCT` queries to the full SPARQL endpoint for specific sub-queries in order to materialize path expressions.

As for Item 1, we downloaded the complete Wikidata dump,[53] converted it locally to HDT [12] and executed the following triple pattern queries over it to collect all data to match non-property-path triple patterns in Figure 5. We note that alternatively, we could have used Wikidata's Triple Pattern Fragment API [35] at `https://query.wikidata.org/bigdata/ldf` similarly.

We then executed the following extraction queries separately on the dump, to extract the necessary component data:

`CONSTRUCT WHERE {?S wp:P17 ?O}` → 6613664 *triples*
`CONSTRUCT WHERE {?S wp:P131 ?O}` → 3928939 *triples*
`CONSTRUCT WHERE {?S wp:P276 ?O}` → 697238 *triples*
`CONSTRUCT WHERE {?S wp:P580 ?O}` → 26354 *triples*
`CONSTRUCT WHERE {?S wp:P582 ?O}` → 19241 *triples*
`CONSTRUCT WHERE {?S wp:P585 ?O}` → 91509 *triples*
`CONSTRUCT WHERE {?S wp:P625 ?O}` → 4158225 *triples*

In order to retrieve the remaining triples, that are instances of (subclasses of) the Wikidata classes of elections (`wd:Q40231`) and sports competitions (`wd:Q13406554`), we executed the following queries against the Wikidata SPARQL endpoint:

```
CONSTRUCT {
  ?S a wd:Q13406554. ?S rdfs:label ?label.
} WHERE {
  ?S wdt:P31/wdt:P279* wd:Q13406554.
  ?S rdfs:label ?label.
  FILTER( LANG(?label) = "en" ||
          LANG(?label) = "de" ||
          LANG(?label) = "" )
}
```
→ 418136 *triples*

```
CONSTRUCT {
  ?S a wd:Q40231. ?S rdfs:label ?label.
} WHERE {
  ?S wdt:P31/wdt:P279* wd:Q40231.
  ?S rdfs:label ?label.
  FILTER( LANG(?label) = "en" ||
          LANG(?label) = "de" ||
          LANG(?label) = "" )
}
```
→ 46899 *triples*

We then loaded these triples into a local triple store and executed the query in Figure A.19 on it, which is equivalent to the query in Figure 5, Section 3.2.

[52] There is ongoing work to fix it, which, however points to the same problem as an outstanding issue, cf. `https://github.com/json-ld/json-ld.org/issues/397`, retrieved 2018-03-29.

[53] `https://www.wikidata.org/wiki/Wikidata:Database_download`

```
CONSTRUCT {
  ?event rdfs:label ?label ;
    dcterms:isPartOf ?Parent ;
    timex:hasStartTime ?StartDateTime ;
    timex:hasEndTime ?EndDateTime ;
    dcterms:coverage ?geocoordinates ;
    dcterms:spatial ?geoentity .
} WHERE {
  ?event rdfs:label ?label .
  {   # with a point in time or start end end date
    { ?event wdt:P585 ?StartDateTime.
     FILTER(?StartDateTime >
            "1900-01-01T00:00:00"^^xsd:dateTime)
    }
    UNION
    { ?event wdt:P580 ?StartDateTime.
      FILTER(?StartDateTime >
             "1900-01-01T00:00:00"^^xsd:dateTime)
      ?event wdt:P582 ?EndDateT.
      FILTER(DATATYPE(?EndDateT) = xsd:dateTime)}
  }
  OPTIONAL { ?event wdt:P361 ?Parent. }
  # specific spatialCoverage if available
  OPTIONAL {
    ?event wdt:P276?/(wdt:P17|wdt:P131) ?geoentity
  }
  OPTIONAL {
    ?event wdt:P276?/wdt:P625 ?geocoordinates
  }
  BIND ( if(bound(?EndDateT), ?EndDateT,
xsd:dateTime(concat(str(xsd:date(?StartDateTime)),
                    "T23:59:59")))
    AS ?EndDateTime )
}
```

Figure A.19: SPARQL query on local Wikidata extract - Namespaces as in Figure A.18