ODArchive – Creating an Archive for Structured Data from Open Data Portals

Thomas Weber[1(✉)], Johann Mitöhner[1], Sebastian Neumaier[1], and Axel Polleres[1,2]

[1] Vienna University of Economics and Business, Vienna, Austria
thomas.weber@wu.ac.at
[2] Complexity Science Hub Vienna, Vienna, Austria

Abstract. We present ODArchive, a large corpus of structured data collected from over 260 Open Data portals worldwide, alongside with curated, integrated metadata. Furthermore we enrich the harvested datasets by heuristic annotations using the type hierarchies in existing Knowledge Graphs. We both (i) present the underlying distributed architecture to scale up regular harvesting and monitoring changes on these portals, and (ii) make the corpus available via different APIs. Moreover, we (iii) analyse the characteristics of tabular data within the corpus. Our APIs can be used to regularly run such analyses or to reproduce experiments from the literature that have worked on static, not publicly available corpora.

Keywords: Open data · Archiving · Profiling · Reference tables

1 Introduction

The Open Data (OD) movement, mainly driven by public administrations in the form of Open Government Data has over the last years created a rich source of structured data published on the Web, in various formats, covering different domains and typically available under liberal licences. Such OD is typically being published in a decentralized fashion, directly by (governmental) publishing organizations, with data portals, often operated on a national level as central entry points. That is, while OD portals provide somewhat standardized meta-data descriptions and (typically rudimentary, i.e. restricted to metadata only) search functionality, the data resources themselves are available for download on separate locations, as files on specific external download URLs or through web-APIs, again accessible trough a separate URL.

In order to provide unified access to this rich data source, we have been harvesting, integrating and monitoring meta-data from over 260 OD portals for several years now in the Portal Watch project [11,14]. Underlining the increasing importance of providing unified access to structured data on the Web, Google

J. Z. Pan et al. (Eds.): ISWC 2020, LNCS 12507, pp. 311–327, 2020.
https://doi.org/10.1007/978-3-030-62466-8_20

recently started a dataset search [3] facility, which likewise indexes and unifies portal metadata adhering to the Schema.org [5] vocabulary in order to make such metadata searchable at Web scale. In fact, in our earlier works on Portal Watch we demonstrated how the harvested metadata from different OD portal software frameworks, for instance CKAN,[1] can be uniformly mapped to standard formats like DCAT [8] and Schema.org, thereby making the metadata from all indexed portals in Portal Watch available on-the-fly to Google's dataset search.

Yet, while OD *metadata* is well investigated in terms of searchability or quality, the underlying referenced *datasets*, i.e. the actual structured data resources themselves, and their characteristics are still not well understood: What kinds of data are published as OD? How do the datasets themselves develop over time? How do the characteristics of datasets vary between portals? How can search facilities and indexes be built that allow searching within the data and not only the metadata? In order to enable answering such questions, our goal in the present paper is to provide a resource in terms of a dynamically updated corpus of datasets from OD portals, with unified access and filtering capabilities, that shall allow both profiling and scientific analyses of these datasets. To this end we have created, on top of the Portal Watch framework, a dataset crawler and archiver which regularly crawls and indexes OD resources, performs basic data cleansing on known formats, and provides unified access to a large corpus of structured data from OD portals through APIs that allow flexible filtering, e.g. through SPARQL queries over the meta-data, for on-the-fly generation of specific sub-corpora for experiments. We deem this project particularly useful as a resource for experiments on real-world structured data: to name an example, while large corpora of tabular data from Web tables have been made available via CommonCrawl [6], the same is not true for tabular data from OD Portals, for which we expect different characteristics. Indeed, most works on structured OD and its semantics focus on metadata, whereas the structure, properties and linkability of the datasets themselves is, apart from isolated investigations and profiling of adhoc created subcorpora (restricted, for instance, to single data portals), still largely unexplored.

We fill this gap by presenting the Open Dataset Archiver (ODArchive), an infrastructure to crawl, index, and serve a large corpus of regularly crawled structured data from (at the moment) 137 active portals.[2] We describe the challenges that needed to be overcome to build such an infrastructure, including for instance automated change frequency detection in datasets, and make the resource available via various APIs. Moreover, we demonstrate and discuss how these APIs can be used to conduct and regularly update/reproduce various experiments from the literature that have worked on static, not publicly available corpora; as an example we present a detailed profiling analysis on the tabular CSV data in the corpus. Specifically, we make the following concrete contributions:

[1] https://ckan.org/, accessed 2020-08-17.

[2] Overall, historically we monitor and have monitored over 260 portals, however, several of those have gone offline in the meantime or are so-called "harvesting" portals that merely replicate metadata from other portals, for details cf. [14].

- We present a detailed architecture of a distributed and scalable Dataset Archiver. The archiver is deployed at https://archiver.ai.wu.ac.at, and the software is openly available on Github[3] under the MIT license.
- Using the introduced archiver, we regularly collect and archive – based on an approximation of the change rates – a large corpus of datasets from OD sources, and make the whole corpus, including the archived versions available via different APIs, incl. download access to subsets of the corpus configurable via SPARQL queries.
- We focus on the prevalent format in our corpus – tabular data – by presenting a detailed profiling analysis of the CSV files in this corpus, and discuss their characteristics. We heuristically annotate columns in these CSVs, using the CSVWeb metadata standard [16], with base datatypes and type hierarchies in existing Knowledge Graphs (DBpedia or Wikidata). Further, we present an approach to scale finding reference columns in this corpus, i.e. tables that contain one or more columns whose values likely reference foreign keys from another reference table: as we can show, there are significantly more reference tables than links to existing KGs in OD, suggesting that such reference tables in OD tables themselves could be the basis for a knowledge graph on its own.

The remainder of this paper is structured as follows: In the next Sect. 2 we present the architecture of our crawling and archiving framework; in Sect. 3 we discuss how to access and query the archived datasets; after an overview of overall corpus characteristics of our archive (Sect. 4), we present experiments on dataset profiling and analysis of identifying reference tables specifically on tabular (CSV) data in Sect. 5. We discuss related and complementary works in Sect. 6, and eventually conclude in Sect. 7.

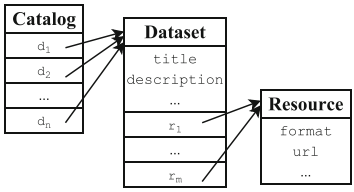

Fig. 1. High-level structure of a data portal.

2 Open Dataset Archive

The datasets that we collect and archive come from the OD Portal Watch project [14]: Portal Watch is a framework for monitoring and quality assessment of (governmental) OD portals, see http://data.wu.ac.at/portalwatch. It monitors

[3] https://github.com/websi96/datasetarchiver.

and archives metadata descriptions from (governmental) portals, however, not the actual datasets. The structure of such a data portal (or *catalog*) is similar to digital libraries (cf. Fig. 1): a *dataset* is associated with corresponding *metadata*, i.e. basic descriptive information in structured format, about these resources, for instance, about the authorship, provenance or licensing of the dataset. Each such dataset description typically aggregates a group of data files (referred to as *resources* or distributions) available for download in one or more formats (e.g., CSV, PDF, spreadsheet, etc.). The focus of the present paper is on how to collect, archive, and profile these data files.

2.1 Architecture

Based on our earlier findings on crawling and profiling a static snapshot of CSV data [9] from OD and about capturing and preserving changes on the Web of Data [18], we expect a large amount of dynamically changing source datasets, spread across different domains: as mentioned above the datasets to be crawled themselves, which are cataloged at OD portals, typically reside on different URLs on servers operated by many different data publishers per portal indexed by Portal Watch and accessible by its SPARQL endpoint residing at http://data.wu.ac.at/portalwatch/sparql.

In order to scalably and regularly crawl these datasets from their sources, we therefore designed an infrastructure with three layers to distribute the workload in an extensible manner: (i) a network layer (handled by Kubernetes and Ingress); (ii) a storage layer (using MongoDB), as well as (iii) a Scheduling and Crawling layer handled by specific components written in JavaScript.

That is, the whole system is deployed on an extensible Kubernetes-Cluster with an NGINX Ingress Controller, with currently three server nodes, running our Data storage and Crawling/Scheduling components. We additionally use one external server node with a NGINX Reverse Proxy for load-balancing external traffic. The following software packages/frameworks are used:

1. *Kubernetes:* orchestrates our containerized software on the cluster.
2. *NGINX:*
   - *Ingress Controller:* is a HTTP load balancer for applications, represented by one or more services on different nodes.
   - *Reverse Proxy:* is responsible for load-balancing HTTP requests and database connections from external IPs.
3. *MongoDB:* stores all datasets as chunked binaries along with their associated metadata.
4. *Scheduler:* crawling and scheduling component written in Node.js.

In order to scale the system, it is possible to not only plug in additional server nodes but also whole clusters and spread the workload of the datastore and crawling over their provided nodes. Section 2.1 illustrates the architecture components and their interplay in more detail.

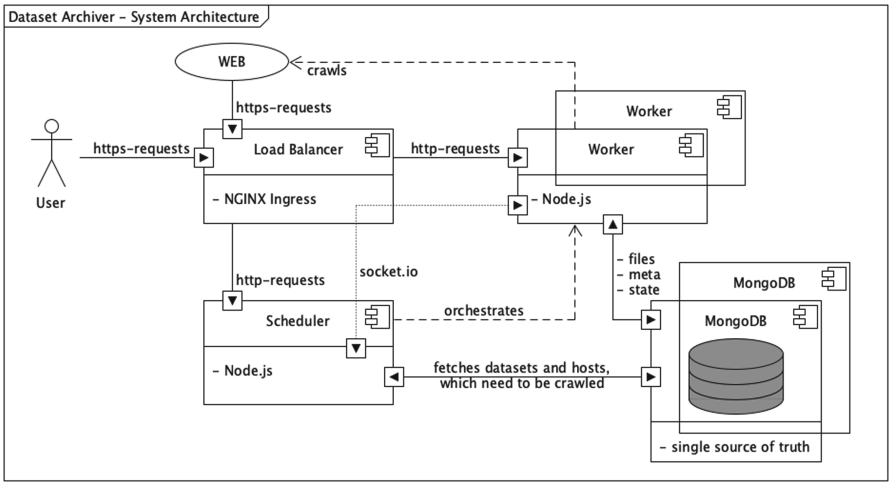

Fig. 2. Archiver architecture

Scheduler. The scheduling component regularly feeds the MongoDB with Resource URLs from the Portal Watch Sparql endpoint. Then it fetches the least-recently downloaded Resource URLs one by each Resource URLs Domain to ensure that our Scheduler does not enforce a denial of service of a domain while distributing the workload to our Crawling Workers via the Load Balancer.

Load Balancer. The Ingress Controller orchestrates the crawling requests by assigning them to worker instances distributed over different nodes/clusters to work in parallel in a Round-Robin fashion. I.e., if there are 3 Workers (w) and 5 requests (r) queued, the requests will be handled by the Workers in following order: $r1 \Rightarrow w1;\ r2 \Rightarrow w2;\ r3 \Rightarrow w3;\ r4 \Rightarrow w1;\ r5 \Rightarrow w2$

The Crawling Workers then download or crawl the requested resources from the Web and store them in MongoDB.

Database. The MongoDB database instances consist of five collections: *datasets, datasets.chunks, datasets.files, hosts* and *sources*. The *datasets* collection stores essential meta- and crawling-information, e.g., available versions, crawl interval, etc. In the *sources* collection we store information about the source of the datasets, e.g., the data portal (obtained from Portal Watch). The remaining collections organize the storage and chunks of the actual files.

2.2 Workload-Management and Scalability

To ensure our system does not overstrain single hosts nor our own underlying network infrastructure, we make use of the "robots.txt" files and also implemented other strategies to distribute the workload and avoid unnecessary re-crawls.

Dynamic Crawl Frequency. [18] proposes to implement the crawling scheduler as an adaptive component in order to dynamically adapt the crawl frequency per URLs based on estimated content change frequency from earlier crawls. We accordingly base our implementation on a comparison sampling method – which we evaluated in [12] – and take into account the Nyquist sampling theorem [20]: to recreate a frequency from unknown source, the sampling rate must at minimum be twice as high as the frequency itself. We monitor a fixed amount of past versions (concretely, we store the last up to 10 intervals between downloads in seconds and a boolean declaring if a file has changed or not) in order to schedule/predict the best next crawl timestamp. From the mean change interval per dataset we pick half as the newly proposed interval, to ensure that our re-crawl/sampling rate remains on average twice as high as the actual change rate. We also set a maximum of every 6 months and a minimum of every 6 hours as upper and lower bounds for the crawl rate.

Scalability. For additional scalability we rely on MongoDB's sharding capabilities[4] and Kubernetes' container orchestration functionality[5] to horizontally scale across multiple machines: we currently use three nodes totaling 377 GB of memory and 72 CPU cores to distribute all our workload. Each shard contains a subset of the data and each query is routed by a MongoDB instance, providing an interface between client applications and the sharded cluster. A shard key defines which node stores file chunks: we shard by dataset id plus the version number as shard key to keep all chunks of single files on the same node.

The combination of Ingress, Kubernetes and MongoDB connected through micro-services can by extended dynamically, by adding more nodes, when needed.

```
PREFIX arc: <https://archiver.ai.wu.ac.at/ns/csvw#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX csvw: <http://www.w3.org/ns/csvw#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX dc: <http://purl.org/dc/elements/1.1>
INSERT {
  <https://offenedaten.de/dataset/be8c1bf6-50cf-4fab-8ea3-179ca947652a>
     dcat:accessURL <https://www.berlin.de/daten/liste-der-kfz-kennzeichen/kfz-kennz-d.csv> .
  <https://www.berlin.de/daten/liste-der-kfz-kennzeichen/kfz-kennz-d.csv>
     dcat:mediaType "text/csv" ;
     dc:title "kfz-kennz-d.csv" ;
     dc:hasVersion <https://archiver.ai.wu.ac.at/api/v1/get/file/id/5e863ee2b511a4001191dcf8_0> ;
     dc:hasVersion <https://archiver.ai.wu.ac.at/api/v1/get/file/id/5e863ee2b511a4001191dcf8_1> .
  <https://archiver.ai.wu.ac.at/api/v1/get/file/id/5e863ee2b511a4001191dcf8_0>
     dc:identifier "0eec56f69acbda76b375ee982dbd4d7e" ;
     dc:issued "2020-04-06T22:09:56.336Z" ;
     dcat:byteSize 12642 .
  <https://archiver.ai.wu.ac.at/api/v1/get/file/id/5e863ee2b511a4001191dcf8_1>
     dc:identifier "74f78308cb653142663c057744cde84b" ;
     dc:issued "2020-04-12T22:09:56.336Z" ;
     dcat:byteSize 12642 . }
```

Fig. 3. Example `INSERT` statement to add the dataset meta-information.

[4] https://docs.mongodb.com/manual/sharding/#shard-keys, accessed 2020-05-22.
[5] https://kubernetes.io/, accessed 2020-05-22.

3 Data Access and Client Interface

SPARQL Endpoint. We make the metadata of the collected and archived datasets queryable over SPARQL by providing the corresponding meta-information in a triple store; the endpoint is available at https://archiver.ai.wu.ac.at/sparql. To describe the datasets we make use of the Data Catalog vocabulary (DCAT) [8] for all crawled datasets (`dcat:Dataset`) to specify links to the portal (`dcat:Catalog`) where datasets were published, as well as `dcat:accessURL`s of *resources* and their respective format (`dcat:mediaType`). Additionally, for tabular data resources, we provide metadata using the CSV on the Web vocabulary (CSVW) [16]: CSVW provides table-specific properties, such as `csvw:tableSchema` and `csvw:datatype`s per column. Figure 3 shows an example of the meta-information stored for an archived dataset.

In this case, as the dataset is a CSV, we also insert CSVWeb metadata as shown in Fig. 4: for these CSVs we heuristically detect the encoding, delimiters, as well as column datatypes of a CSV table, and provide this information using the `csvw:dialect` property. We further try to detect if the CSV provides a header row, to extract column labels. Details on these heuristic annotations are given in our preliminary work [9]. Additionally, as discussed in more detail in Sect. 5 below, we annotate – where possible – column types as well as basic statistics such as selectivity per table column.

```
INSERT {
  _:csv csvw:url <https://archiver.ai.wu.ac.at/api/v1/get/file/id/5e863ee2b511a4001191dcf8_0> ;
    arc:rows 403 ;
    arc:columns 3 .
  _:csv csvw:dialect [
    csvw:encoding "utf-8" ;
    csvw:delimiter "," ;
    csvw:header true ] .
  _:csv csvw:tableSchema [
    csvw:column <https://archiver.ai.wu.ac.at/api/v1/get/file/id/5e863ee2b511a4001191dcf8_0#1> ;
    csvw:column <https://archiver.ai.wu.ac.at/api/v1/get/file/id/5e863ee2b511a4001191dcf8_0#2> ] .
  <https://archiver.ai.wu.ac.at/api/v1/get/file/id/5e863ee2b511a4001191dcf8_0#1>
    csvw:name "Stadt bzw. Landkreis" ;
    csvw:datatype "string" ;
    rdfs:range <http://dbpedia.org/ontology/Place> .
  <https://archiver.ai.wu.ac.at/api/v1/get/file/id/5e863ee2b511a4001191dcf8_0#2>
    csvw:name "Bundesland" ;
    csvw:datatype "string" ;
    rdfs:range <http://dbpedia.org/ontology/PopulatedPlace> . }
```

Fig. 4. `INSERT` statement of example CSV meta-information.

API Endpoints. We provide the following API endpoints to interact with the Dataset Archiver. The API is devided into a publicly available API for searching and retrieving our crawled OD resources and a private API used for maintanance, requiring resp. credentials.

Public API.

`/stats/basic` – Basic statistics on the data stored in the crawler's database.
`/get/dataset/{URL}` – Returns a JSON object of a dataset description by its referencing URL.
`/get/datasets/{domain}` – Returns a JSON object of all dataset descriptions provided by the same domain.
`/get/dataset/type/{TYPE}` – Returns a JSON object of all dataset descriptions which offer resources with the specified filetype e.g. "text/csv" or just "csv".
`/get/file/{URL}` – Returns a resource (crawled file) by its referencingURL (i.e., for `dc:accessURL`s the latest downloaded version is retrieved, or, resp. a concrete `ds:hasVersion` URL can be provided directly).
`/get/files/type/{TYPE}` – Returns a zip file containing all versions of the specified filetype e.g. "text/csv" or just "csv".
`/get/files/sparql?q={QUERY}` – Returns a zip file of the resource versions specified by a SPARQL query, that is, all the files corresponding to (version or dataset) URLs that appear in the SPARQL query result cf. detailed explanations below.

Private API.

`/post/resource?secret=SECRET` – Adds a new resource to the crawler by posting a JSON object containing the URL of the resource, the URL of the portal and the format e.g. 'text/csv' or 'csv'. Only the URL of the resource is mandatory and a secret key credential is needed to post resources.
`/post/resources?secret=SECRET` – Adds several resources at once in batch, using the same parameters as above.
`/crawl?id=ID&domain=DOMAIN&secret=SECRET` – Tells the workers which resource has to be crawled. It is used by the master scheduler; a crawl can also be enforced with this endpoint.

Detailed usage examples of the different APIs are documented on our Webpage at https://archiver.ai.wu.ac.at/api-doc.

Data Download via SPARQL. Besides the APIs to directly access files from our crawler and the SPARQL interface to query metadata, we also offer a way of directly downloading data parameterized by SPARQL queries, i.e., for queries that include any URLs from the subject (*datasetURL*) or object (*versionURL*) of the `dc:hasVersion` property in our triple store, we provide a direct, zipped, download of the data: here *versionURL*s will directly refer to concrete dowloaded file versions, whereas any *datasetURL* will retrieve the resp. latest available version in our corpus.

For instance, the query in Fig. 5 selects all archived resources from a specific *data portal* (data.gv.at),[6] collected after a certain *time stamp*, with a specific

[6] To filter datasets by certain data portals we enriched the descriptions by information collected in the Portal Watch (https://data.wu.ac.at/portalwatch/): we use

HTTP media type (in this case CSV files); executing this query at our SPARQL user interface (https://archiver.ai.wu.ac.at/yasgui) gives an additional option to retrieve the specific matching versions directly as a zip file. Alternatively, given this query to the `/get/files/sparql?q={QUERY}` API mentioned above, will retrieve these without the need to use the UI.

```
SELECT ?versionURL WHERE {
  ?datasetURL arc:hasPortal ?Portal ;  # ?datasetURL: the download URL of a specific resource
                                       # ?Portal: a dcat:catalog indexed in Portal Watch
    dc:hasVersion ?versionURL ;        # ?versionURL: a crawled version of the resource
    dcat:mediaType ?mediaType .        # ?mediaType: media type as per HTTP response.
  ?versionURL dc:issued ?dateVersion . # ?dateVersion: crawl time.

  FILTER (?Portal = <http://data.gv.at> &&
          ?mediaType = "text/csv"       &&
          strdt(?dateVersion, xsd:dateTimeStamp) >= "2020-05-10T00:00Z"^^xsd:dateTimeStamp) }
```

Fig. 5. Example query to get a set of URLs of archived datasets.

4 Overall Corpus Characteristics

Table 1 shows an overview of our overall current ODArchive corpus – as of week 21 in 2020: we regularly crawl a total of ∼800k resource URLs of datasets from 137 OD portals; over a time of 8 weeks we collected a total of 4.8 million versions of these datasets. Resource URLs origin from 6k different (pay-level) domains, collected from 137 OD portals, which demonstrates the spread of actual data-providing servers and services indexed by OD portal catalogs. The latest crawled versions of all datasets amount to a total of 1.2 TB uncompressed, and the total of all stored versions sums up to around 5.5 TB. Additionally, Table 1 shows the top-5 most common data formats across the most recent crawl.

Table 1. Total number of URLs of datasets, archived versions, domains/portals, and size of the corpus (left); top-5 most frequent HTTP media types (right).

#Resource URLs	798,091
#Versions	4,833,271
#Domains	6,001
#Portals	137
Latest Versions Corpus Size	1.2 TB
Total Corpus Size	5.5 TB

Media type	Count
text/html	187,196
text/csv	116,922
application/json	102,559
application/zip	93,352
application/xml	76,862

`arc:hasPortal` to add this reference. More sophisticated federated queries could be formulated by including the Portal Watch endpoint [14] which contains additional metadata.

Table 2 shows the main sources of our data corpus: in the left table we provide the ten most frequent domains of the resource URLs in the corpus, whereas the right table shows the top-10 data portals.

Table 2. Top-10 most frequent domains of the resource URLs in the archiver, and the most frequent source portals.

Resource URL Domain	Count
wab.zug.ch	77,436
data.opendatasoft.com	63,481
clss.nrcan.gc.ca	59,519
services.cuzk.cz	40,504
abstimmungen.gr.ch	36,604
www.geoportal.rlp.de	26,275
www150.statcan.gc.ca	20,295
archiv.transparenz.hamburg.de	19,321
cdn.ruarxive.org	17,242
www.dati.lombardia.it	15,743

Data Portal	Count
europeandataportal.eu	282,541
open.canada.ca	118,949
data.opendatasoft.com	63,481
offenedaten.de	38,348
datamx.io	32,202
dados.gov.br	31,961
data.gov.ie	20,826
hubofdata.ru	19,783
edx.netl.doe.gov	19,379
data.gov.gr	18,687

Note that these numbers can easily be computed through our SPARQL endpoint in an always up-to-date manner, and also over time, by restricting to the most recent versions before a certain date, with queries analogous to those shown in Sect. 3. For instance, the following query produces the statistics given in Table 1: https://short.wu.ac.at/odarchiverquery1.

5 CSVs: Column Types from KGs and Reference Tables

In order to demonstrate the potential use of our data collection, we herein discuss reproducible profiling experiments we conducted on a subcorpus of tabular data: the experiments focus on CSV files (116,922, as per Table 1 in the most recent crawl) from our corpus, as the most prominent structured format in OD portals. Also, as mentioned in the introduction, while tabular data on the Web is a popular subject of investigations, the particular characteristics of tabular data from OD portals have thus far not been the main focus of these investigations.

Tables	Header	Columns	Avg Rows	Avg Cols
67974	53279	685276	195.5	10.1

5.1 Labelling Columns with Types from KGs

In the first experiment section, we focus on scalably annotating columns in our CSV table corpus to classes in existing knowledge graphs (KGs), specifically DBpedia [2] and Wikidata [19]. To this end, we distinguish by column datatypes between "textual" and "numeric" columns; we herein specifically focus on scaling named entity recognition (NER) by textual labels in columns to our corpus. As for numeric colums, we note that labeling numeric data in tabular OD corpora with references to KGs has its own challenges and remains a topic of active research, cf. for instance [13] for our own work in this space.

Our basic idea here is to build a NE gazetteer from DBpedia and Wikidata labels, along with references of labels to their associated types (i.e., `rdf:type` links in DPpedia, or `wdt:P31` in Wikidata, resp.). The base assumption here is that, despite potential ambiguities, columns containing labels of predominantly same-typed entities, can be associated with the respective DBpedia/Wikidata type(s). To this end, we extracted label and type information and as well as the transitive class hierarchy (using `rdfs:subClassOf`, or `wdt:P279` links, resp.) from both KGs.[7]

In order to scale, rather than relying on SPARQL, we have constructed our gazetteer by simply compiling the extracted data into huge Python dictionaries (one for the types of a given label, and another for the labels of a given type). This conceptually simple approach is further complicated by two main scalability challenges, which we discuss in the following.

1. Synonyms and Homonyms: A given entity in the NE sources is often associated with a number of ambiguous labels and a given label can be associated with a number of entities, and therefore an even larger number of types; e.g., the entity Abraham Lincoln in the sense of the 16th president of the United States is assigned the types dbpedia.org/ontology/Person, xmlns.com/foaf/0.1/Person, www.w3.org/2002/07/owl#Thing, schema.org/Person, dbpedia.org/ontology/-Agent, and many others within DBpedia.

However, since many more entities share the name of the great president, including bridges, ships, high schools, and universities, the number of types that can be assigned to the *label* 'Abraham Lincoln' is in fact much larger. In addition, the president is also known under a number of other labels, such 'Abe Lincoln', 'Honest Abe', and 'President Lincoln', each of which is also assigned (among others) the types listed above.

2. Multi-linguality: Labels and types are available in various languages; at the moment, we limit ourselves to English and German labels, implementing multi-linguality only in principle and not in any sense exhaustively which would of course still be limited to the language versions available in the NE sources. Restricting to those two languages was also useful to significantly reduce size of the extracted gazetteer from the raw DBPedia and Wikidata HDT dump files containing all language labels. Still, while English labels and types form the

[7] The resp. information has been extracted from the most recent DBpedia and Wikidata HDT [4] dumps available at http://www.rdfhdt.org/datasets/.

largest part of the NE sources, we assume many other languages in e.g. nationally operated OD portals, which we do not cover yet. This label-type information was then imported into Python dictionaries for efficient access to all types of labels and vice versa, fitting in memory, provided that the available RAM is sufficient; in this case roughly 30 GB.

CSV Table Pre-Processing. We assume that very large files would not significantly contribute to the results, and therefore only consider files <100 KB in the analysis, resulting in 71,787 CSVs (∼60% of all CSV files currently in our corpus). The number of usable tables is further reduced due to import errors (essentially empty, "headers only", CSV files), to 67,974 tables with overall 685,276 columns.

As mentioned above we restrict our comparison to (tables with) columns with textual content only, which further significantly reduces the number of columns to be analysed: overall, the reduced corpus for the experiment considers 61,110 tables and a total of 294,485 textual columns (i.e., an avg. of 4.8 textual columns per table) to be annotated. Looking at the individual values within the remaining data set we find that only around 19% (and only 2% among the unique values) of those values can be associated with at least one DBPedia or Wikidata type:[8]

	Total	With type	Fraction
Values	28,442,981	5,278,327	0.186
Unique	5,299,125	104,985	0.02

Obvious additional measures to be taking into consideration for the type annotations of table columns are the total number of values, the number of distinct values, and the selectivity (the number of distinct values divided by the number of total values): it proved useful to only look at columns with a minimum number of distinct values: For instance, in order to rule out "essentially boolean" attributes,[9] we only considered columns with at least three values. The listing shows the selectivity of columns with at least three values.

Columns	Avg number of values	Avg distinct values	Selectivity
233,416	121.5	46.0	0.28

[8] While this needs further investigation, and obviously more sophisticated matching techniques (substrings- or similarity-based), we note that this low percentage seems to hint at the specific textual information in OD tables not necessarily being covered by the more general, encyclopedic knowledge typical in public KGs.

[9] E.g., "Ja" and "Nein" (German for "yes" and "no"), are labels for entities in Wikidata.

Another measure for annotating columns with KG types is the fraction of types covered in the value labels of a given column. For our column-type annotation we consider the following threshold: type coverage for columns with at least one common type with a fraction of 0.8 or greater.

Among the finally remaining **74,467** columns, we collect intersecting types per colums and add those as column annotations (using the CSVWeb vocabulary) to the corpus, cf. the example in Fig. 4. Other column characteristics, such as selectivity, are also added as annotations; via our API one could for instance only consider a specific subcorpus based on these annotations.

The most often identified types are associated with organizations, locations, and various types of media. Note that types from various knowledge graphs are overlapping to varying degrees.

Type	Number
wd/group	10,383
dbpedia.org/ontology/Location	8,601
schema.org/Place	8,601
wd/entity	8,405
wd/intellectual work	7,285
wd/series	7,057
dbpedia.org/ontology/Place	6,799
wd/creative work	6,749
wd/information	5,991
wd/communication medium	5,989

5.2 Finding Reference Tables

Apart from class annotations per column, which serve to link OD datasets with KGs, we also analyzed potential interlinkage between OD tables in our corpus by looking for potential references between tables; whereas e.g. [7] used semantic and machine learning methods to find relations among web tables; here we apply a basic but scalable approach (by again modularly restricting the number of columns to be compared): a reference table contains one or more key columns whose values are referenced i.e. are identical with values in other tables. Our basic approach to identifying reference tables simply compares values in candidate key columns with the complete set of columns from other tables by going through all columns in all tables. We limit this brute force approach as described in the following.

Overall, we compare two approaches for determining a possible reference:

- *Strict* i.e. all values of column A must be present in column B
- *Lax* i.e. at least a fraction of 0.9 of the values in A must be present in B

Both computations are done on the *sets* of column values rather than the original *list* of values.

To further limit the amount of processing we put the following restrictions on reference candidates: (i) the number of distinct values in B must be at least 10, (ii) the selectivity must be 1 in the referenced column B (i.e., we only consider single attribute candidate keys as references).

The "brute force" approach consists in checking every column of every table with each column of every other table satisfying the restrictions (28,524 candidate reference tables), where the Python library ray [10] used for parallelisation allowed us to scale this pairwise comparison between candidate tables and every same-typed column of every other table. Applying this reference search by doing a lax or strict check on each column with each reference candidate results in the following number of actually (at least once) referenced tables, where we see that applying a strict check does not decrease the number dramatically:

Reference tables	
lax	15,977
strict	15,052

That is, more than half of the candidate reference tables are actually referenced from other tables, according to these heuristics.

Indeed, some tables are referenced very frequently, for instance reference tables with regionally important area codes, such as US state codes, national ISO country codes, or – as a less obvious example – the following table was cited in 1,811 other tables in the corpus; it has 402 rows showing area codes for German car license plates:

Kennzeichen, Juli 2012	Stadt bzw. Landkreis	Bundesland
A	Augsburg	Bayern
AA	Aalen Ostalbkreis	Baden-Württemberg
AB	Aschaffenburg	Bayern
ABI	Anhalt-Bitterfeld	Sachsen-Anhalt
ABG	Altenburger Land	Thüringen
AC	Aachen	Nordrhein-Westfalen

Overall, while we defer a more detailed analysis to future work, these results hint at a large number of possible additional inter-dataset links, apart from only considering links to existing KGs. We explicitly invite usage of our resource to enable further large scale respective experiments by the community.

[10] https://github.com/ray-project/ray, accessed 2020-08-17.

6 Related Work

While OD tables are rarely covered (due to the lack of a readily available corpus as we presented it herein) there are various works on Web tables; we see our resource as a valuable contribution (i) to compare to an alternative evaluation corpus, and (ii) to research the potential of applying existing approaches from these works.

Lehmberg et al. [6] – comparable to our work – presented a large corpus of (typically much smaller) Web tables, consisting of 233 million content tables which they classified as either relational, entity, or matrix tables depending on the orientation and structure of a table, detecting sub-header rows/multi-tables and subject columns in a dataset. In future work, we want to apply this classification to our corpus of tabular resources, in order to highlight and compare the differences of a corpus of Web/HTML and CSV tables. A survey on profiling relational data can be found in [1].

As for related work on entity recognition and semantic interpretation on Web tables [15,17,22] our working hypothesis (partially confirmed by our experiments herein) is that relational data as found on OD repositories are fundamentally different from such Web table corpora, and we will have to leverage additional non-textual/numerical cues in the datasets in order to facilitate linkage to existing KGs. Our archived resources will allow us to test this hypothesis further by applying and reproducing existing works on Web tables as future work. A survey on Web table extraction approaches can be found in [21].

Related to our work on Knowledge Graph types and reference tables, [10] studied the table union problem on a dataset from several OD portals using Locality Sensitive Hashing (LSH) among other approaches, reporting a precision of 0.9005 and a recall of 0.8377 [10, p. 823] on sample queries for unionable tables. Since the subset operation over all columns is essentially a n^2 operation LSH was implemented as an additional alternative approach using minHash LSH Ensemble.[11] Originally designed for queries over large sets of documents the fraction $|Q \cap X|/|Q|$ is the required intersection of query Q with document X which can be specified as threshold when querying the LSH ensemble. This corresponds with the required fraction of elements in column Q present in parent table column X for a referencing relationship as defined in this work.

In our own experiments using this LSH approach we achieved recall and precision values that are somewhat higher than the figures reported in [10] with a speedup of about 5–10 times, but still with a significant number of false positives and negatives, compared to our own brute force set intersection results. A more detailed comparison is on our agenda.

7 Conclusions

ODArchive is set to provide easy access to a large, up-to-date corpus of datasets from OD portals: we archive regularly re-crawled versions of underlying data

[11] http://ekzhu.com/datasketch/lshensemble.html, accessed 2020-08-17.

resources for datasets from these portals, based on an adaptive, heuristically estimated crawl rate and have presented a scalable extensible infrastructure to sustainably run such an archive. Apart from overall characteristics of the crawled corpus, in order to demonstrate its use, we presented two experiments in terms of linking tabular OD datasets to existing KGs as well as interlinking them amongst each other by finding reference tables within the corpus. Our initial results clearly suggest that the characteristics of the structured data found on OD portals and readily provided in our corpus are quite different from other available copora, such as Web Tables. In future work we plan to also analyze and attempt to interlink other structured formats in our corpus; additionally, as our framework keeps on running, it shall also enable temporal analyses over the evolution of OD resources. The infrastructure shall allow detailed analyses overall, but also with a narrower scope, restricting to data from particular portals or regions. Last, but not least, we invite the community to use ODArchive and provide feedback (e.g., in terms of additional API feature requests).

References

1. Abedjan, Z., Golab, L., Naumann, F.: Profiling relational data: a survey. VLDB J. **24**(4), 557–581 (2015). https://doi.org/10.1007/s00778-015-0389-y
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a web of open data. In: Aberer, K., et al. (eds.) ASWC/ISWC -2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_52
3. Brickley, D., Burgess, M., Noy, N.F.: Google dataset search: building a search engine for datasets in an open web ecosystem. In: The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, 13–17 May 2019, pp. 1365–1375. ACM (2019). https://doi.org/10.1145/3308558.3313685
4. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). In: Web Semantics: Science, Services and Agents on the World Wide Web 2019, pp. 22–41 (2013). http://www.websemanticsjournal.org/index.php/ps/article/view/328
5. Guha, R.V., Brickley, D., Macbeth, S.: Schema.org: evolution of structured data on the web. Commun. ACM **59**(2), 44–51 (2016). https://doi.org/10.1145/2844544
6. Lehmberg, O., Ritze, D., Meusel, R., Bizer, C.: A large public corpus of web tables containing time and context metadata. In: Proceedings of the 25th International Conference Companion on World Wide Web, pp. 75–76 (2016). https://doi.org/10.1145/2872518.2889386
7. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. Proc. VLDB Endow. **3**(1–2), 1338–1347 (2010). https://doi.org/10.14778/1920841.1921005
8. Maali, F., Erickson, J.: Data Catalog Vocabulary (DCAT). W3C Recommendation, January 2014. http://www.w3.org/TR/vocab-dcat/
9. Mitloehner, J., Neumaier, S., Umbrich, J., Polleres, A.: Characteristics of open data CSV files. In: Proceedings - 2016 2nd International Conference on Open and Big Data, OBD 2016 (2016). https://doi.org/10.1109/OBD.2016.18
10. Nargesian, F., Zhu, E., Pu, K.Q., Miller, R.J.: Table union search on open data. Proc. VLDB Endow. **11**(7), 813–825 (2018). https://doi.org/10.14778/3192965.3192973, http://www.vldb.org/pvldb/vol11/p813-nargesian.pdf

11. Neumaier, S.: Semantic enrichment of open data on the Web - or: how to build an open data knowledge graph. Ph.D. thesis, Technische Universität Wien, Vienna, Austria (2019). https://permalink.catalogplus.tuwien.at/AC15550378
12. Neumaier, S., Umbrich, J.: Measures for assessing the data freshness in open data portals. In: 2nd International Conference on Open and Big Data, OBD 2016, Vienna, Austria, 22–24 August 2016, pp. 17–24. IEEE Computer Society (2016). https://doi.org/10.1109/OBD.2016.10
13. Neumaier, S., Umbrich, J., Parreira, J.X., Polleres, A.: Multi-level semantic labelling of numerical values. In: Groth, P., et al. (eds.) ISWC 2016. LNCS, vol. 9981, pp. 428–445. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46523-4_26
14. Neumaier, S., Umbrich, J., Polleres, A.: Automated quality assessment of metadata across open data portals. J. Data Inf. Qual. **8**(1), 21–229 (2016). https://doi.org/10.1145/2964909
15. Oulabi, Y., Bizer, C.: Extending cross-domain knowledge bases with long tail entities using web table data. In: Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019 (2019). https://doi.org/10.5441/002/edbt.2019.34
16. Pollock, R., Tennison, J., Kellogg, G., Herman, I.: Metadata Vocabulary for Tabular Data. W3C Recommendation, December 2015. https://www.w3.org/TR/2015/REC-tabular-metadata-20151217/
17. Sarma, A.D., et al.: Finding related tables. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, 20–24 May 2012, pp. 817–828. ACM (2012). https://doi.org/10.1145/2213836.2213962
18. Umbrich, J., Mrzelj, N., Polleres, A.: Towards capturing and preserving changes on the Web of data. In: CEUR Workshop Proceedings (2015). https://pdfs.semanticscholar.org/971b/178200a0bc14735116ace49a0b164e68a926.pdf
19. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014). https://doi.org/10.1145/2629489
20. Weik, M.H.: Nyquist Theorem, p. 1127. Springer, Boston (2001). https://doi.org/10.1007/1-4020-0613-6_12654
21. Zhang, S., Balog, K.: Web table extraction, retrieval, and augmentation: a survey. ACM Trans. Intell. Syst. Technol. **11**(2), 13:1–13:35 (2020). https://doi.org/10.1145/3372117
22. Zhang, Z.: Effective and efficient semantic table interpretation using tableminer+. Semantic Web **8**(6), 921–957 (2017). https://doi.org/10.3233/SW-160242