Identifying, Querying, and Relating Large Heterogeneous RDF Sources

Andre Valdestilhas, Tommaso Soru, Muhammad Saleem, Edgard Marx, Wouter Beek, Claus Stadler, Bernardo Pereira Nunes, Konrad Höffner, Thomas Riechert

Abstract. Although we have witnessed a growing adoption of Linked Open Data principles for publishing data on the Web, connecting data to third parties remains a difficult and time-consuming task. One question that often raises during the publication process is: “Is there any data set available on the Web we can connect with?”. This simple question unfolds a set of others that hinders data publishers to connect to other data sources. For instance, if there are related data sets, where are they? How many? Do they share concepts and properties? How similar are they? Is there any duplicated data set? How to identify and query a huge amount of heterogeneous datasets. To answer these questions, this paper introduces: (i) a new class of data repositories; (ii) a method to identify datasets containing a given URI; (iii) a query engine and source selection in a large RDF dataset collection; (iv) a novel method to detect and store data set similarities including duplicated data set and data set chunk detection; (v) an index to store data set relatedness; and, (vi) a search engine to find related data sets. To create the index, we harvested more than 668k data sets from LOD Stats and LOD Laundromat, along with 559 active SPARQL endpoints corresponding to 221.7 billion triples or 5 terabytes of data. Our evaluation on state-of-the-art real-data shows that more than 90% of data sets in the LOD Laundromat do not use owl:equivalentProperty or owl:equivalentClass to relate to one another data, which reaffirms and emphasizes the importance of our work.

Keywords: Indexing, Querying the Web of Data, Federated Query, Source selection, Big Data techniques, Integrating different sources, Identifying datasets

1. Introduction

Due to the heterogeneity of the datasets, this uniqueness is reflected in the datasets structure, making it a hard task to find relations among those datasets, i.e. to identify how similar they are. In this way, we can say that Linked Data involves datasets and linksets and those linksets needs to be maintained. There are many ways to maintain linksets, one of those is to create a semantic web link repository, e.g. LinkLion [1].

The concept of Linked Data relates to a collection of best practices to publish and connect structured web-based data. However, the number of available datasets has been growing significantly over the last decades [2]. Those datasets represent now the well known as Web of Data, which represents a large collection of concise and detailed interlinked data sets from multiple domains with large datasets [3]. Thus, linking entries across heterogeneous data sources, such as databases or knowledge bases, becomes an increasingly difficult issue [4–6]. However, connections between datasets play a leading role in significant activities such as cross-ontology question answering [7], large-scale inferences [8] and data integration [9]. In Linked Data, the linksets are well known for executing the task of generating links between datasets [5].

The current challenges addressed in this paper are identifying and querying datasets from a huge hetero-
One of the Semantic Web foundations is the possibility to dereference URIs to let applications negotiate their semantic content. However, this exploitation is often infeasible as the availability of such information depends on the reliability of networks, services, and human factors. Moreover, it has been shown that around 90% of the information published as Linked Open Data is available as data dumps and more than 60% of endpoints are offline [12]. To this end, there is a need for a service to identify, in which dataset a URI is defined, in order to let Linked Data consumers find the respective RDF data source, in case such information cannot be retrieved from the URI alone.

In order to query such an amount of LOD datasets, various interfaces, such as LOD Stats [13], LOD Laundromat [14], and SPARQL endpoints provide access to the hundreds of thousands of RDF datasets, representing billions of facts. These datasets are available in different formats such as raw data dumps, HDT files, or directly accessible via SPARQL endpoints. Querying such a large amount of distributed data is particularly challenging and many of these datasets cannot be directly queried using the SPARQL query language.

1.2. The need for integration and relation of LOD datasets

People will always describe knowledge in different ways, due to the individuality of the being, different points of view, time and space, in which we assume as a natural phenomenon and standard. Aware of the state of the art from Ontology/Schema/Instance Matching from Linked Data from venues such as VLDB⁵, OEAF², ISWC³, ESWC⁴, among others, we observe that the LOD datasets are following this standard.

We harvested more than 668k data sets from LOD Stats and LOD Laundromat, along with 559 active SPARQL endpoints corresponding to 221.7 billion triples or 5 terabytes of data. We observed several duplicates and chunks among the datasets and the fact that more than 90% of data sets in the LOD Laundromat do not use owl:equivalentProperty or owl:equivalentClass⁴ to relate to one another data, which reaffirms and emphasizes the importance of our work.

We notice that there are several similar data sets. Yet, there is no place that stores or provides such kind of information about how similar the data sets are and which attributes the data sets share. This work aims to build an index to store the relation among the data sets to have a place to see how similar the data sets are. We provide a method to index the relations among LOD data sets⁶ based on the properties and classes that they share.

As a motivation of our work, we describe two distinct scenarios where the proposed index may be useful:

Scenario 1 (finding datasets): Given the SPARQL query at Listing 1, the user wants to know if a given data set contains three properties, in which the user already know the query performs well at the SPARQL endpoint from CKAN⁷:

```
1  SELECT * WHERE { ?s ?p ?o.
2  FILTER(?s=<http://purl.org/dc/terms/date> ||
3      ?p=<http://example.com/property1> ||
4      ?o=<http://example.com/resource1> )
```

⁠³https://www.vldb.org/
⁠²http://oeai.ontologymatching.org/
⁠⁴https://iswc2019.semanticweb.org/
⁠⁵https://2019.eswc-conferences.org/
⁠⁶Some concepts, such as skos:closeMatch, skos:exactMatch, and owl:sameAs were excluded from this study because they are not relevant enough to identify the equivalence of properties.
⁠⁷CKAN SPARQL endpoint https://linked.opendata.cz/sparql
The following questions are raised:

- How to know which data sets are able to execute the query?
- Which of the data sets contains the most valuable results?
- Could the results from different data sets complement each other?

In this case an approach is needed to enable the user to know more datasets to extract the required information. An index of dataset relations and similarities could be used to know that the query can also be executed with results at https://eu.dbpedia.org/sparql but not at http://dbpedia.org/sparql, and we can execute at least five other data sets to complement the results.

**Scenario 2 (finding properties):** The user needs information about cities from all datasets in your repository, more specifically datasets that are compatible with the class http://dbpedia.org/ontology/City. Using our previous approach WIMU [10], this URI was found in 11 datasets, but how many URIs in other datasets that also represents a city that were not listed, for instance a city at WikiData is represented by https://www.wikidata.org/wiki/Property:P131. We consider here that to execute such task manually on more than 600 000 datasets is not feasible. Thus, having an index of dataset relation will help in this case.

### 1.3. Compliance with Linked Data rules and LOD Cloud

Aware of the four rules on Linked Data [15], we justify our work concerning these rules. (1) **Use URIs as names for things**, in which all the work presented here uses URIs as names for things, and WIMU [10] is the largest index to locate datasets containing a given URI. (2) **Use HTTP URIs so that people can look up those names**, where WIMU [10] is an index allowing people lookup datasets for a given URI. Facilitating people to look up those names. (3) **When someone looks up a URI, provide useful information**, using the standards (RDF*, SPARQL). Our approach, WimuQ[11], allows obtaining information from URIs non-dereferenceable anymore, providing more source of data, making an Intelligent Source Selection to achieve more results from a given SPARQL query. (4) **Include links to other URIs, so that they can discover more things**, ReLOD, which is the extension of WIMU and WimuQ, which has the aim of identifying similar datasets that share properties and classes, by using the WIMU lookup index, the user can lookup datasets from a given URI and by using WimuQ to dereference URIs non-dereferenceable anymore. ReLOD allows for discovering more information by the similarity of the datasets, for a given input that can be a Dataset, Property, Class, or SPARQL query.

Concerning the LOD-cloud dataset, which currently contains 1239 datasets with 16 147 links (as of March 2019)[16]. There are more than 600 000 datasets from LODLaundromat, LODStats, and public SPARQL endpoints. One of our intentions is to provide ways to build a new version of the LOD-cloud with all those more than 600k datasets, to help this aim we provide an index containing the similarity level of the datasets. As a first step, we present the ReLOD, showing a methodology to store the similarity level of all datasets. The index is queryable, accepting as input the dataset, SPARQL query, properties and classes, and giving as output the respective similar datasets, properties, and classes.

To deal with all needs presented before we create an integrated approach to identify, relate and query a massive amount of heterogeneous RDF sources, in which join the two previous works[10, 11] With this new approach to deal with the similarity of the datasets, we obtained some improvements, such as the identification of more datasets and more results, where the original contributions of this paper are listed below:

- A method to create an incremental index of similarity among LOD datasets. Thus, to the best of our knowledge, ReLOD is the first dataset relation index over a net total of 221.7 billion triples.
- Creation of a mechanism to search this index by Dataset URI, property, class or SPARQL query. For the first time, to the best of our knowledge, we make a relation index that is able to query by dataset URI, property, class or SPARQL query.
- A method to identify duplicated datasets and chunks.
- The number of datasets identified by wimuQ[11] was increased due to the use of the new ReLOD index.

---

8Datasets available here: https://tinyurl.com/5dataset
9Datasets available here: https://tinyurl.com/wimuDbpediaCity
10https://www.wikidata.org/
The new contributions are presented in more details on Section 2.3, Section 3.3, and Section 4.4 and the rest of the paper is organized as follows: Section 2 presents related works, followed by our approach in Section 3. Section 4 shows the evaluation results and discussion. Finally, Section 5 concludes the paper along with future directions.

2. State of the art

In this section we will present the state of the art relate to identifying datasets with WIMU [10], querying and source selection with wimuQ[11] and relating datasets with ReLOD.

2.1. Identifying LOD datasets

The state-of-the-art of identifying LOD datasets is presented in the following and highlighting the differences with our approach.

The work presented in [17] shows how to set up a Linked Data repository called DataTank and publish data as turtle files or through a SPARQL endpoint. The difference with WIMU is that we provide a RESTful service instead of a setup to configure a Linked Data repository.

The work in [18] is based on an approach developed for the 3store RDF triple store and describes a technique for indexing RDF documents allowing the rank and retrieval of their contents. Their index contained 10^8 triples, which was remarkable for the early years of the Semantic Web. Moreover, their system is not available for tests anymore. A similar point here is that the authors claim that for a given URI from an RDF document, the system will retrieve the URLs of documents containing that URI.

In the approach called LOD-A-LOT [19], which is a queryable dump file of the LOD Cloud11, there are some differences with WIMU. The first, it is not possible to know the provenance of the URI in order to know which dataset the URI was defined. They provide a huge dump file12 containing all the data from LOD Laundromat13. LOD Laundromat itself provides an endpoint to an inverted index of their data14. How-ever, finding the original document a URI was defined in is not trivial, as the returned metadata only describe the datasets themselves [14]. Moreover, as the primary aim of LOD Laundromat is to “clean” Linked Data, most dumps are possibly not continuously monitored, once cleaned.

Comparing with all the approaches above, the main advantage of WIMU is that the datasets a URI likely belongs to are ranked using a score. Our index has also a larger coverage, as it includes data from the two largest Linked Data hubs, i.e., LODStats [13] and LOD Laundromat [14], and the most updated SPARQL endpoints. Finally, WIMU is able to process RDF files containing more than one URI at the same time15.

2.2. Querying and source selection of LOD datasets

In this section, we provide a brief overview of the state-of-the-art pertaining to querying a huge amount of heterogeneous LOD datasets, in which we start with approaches for federated query processing.

Query federation over multiple SPARQL endpoints: DARQ [20], ADERIS [21], FedEx [22], Lusail [23], SPLENDID [24], CostFed [25], ANAPSID [26], Sema-Grow [27], Odyssey [28], and MULDER [29] are examples of state-of-the-art SPARQL endpoint federation engines. These approaches can be further divided into 3 categories namely index-only, index-free, and hybrid (index+ASK) approaches [30].

DARQ and ADERIS are examples of the index-only SPARQL query federation approaches over multiple SPARQL endpoints. DARQ implements a cardinality-based query planner with bind join implementation in nested loops. ADERIS is an adaptive query engine that implements a cost-based query planner. It also makes use of the index-based nested loop join.

FedX and Lusail are examples of the index-free query federation approaches over multiple SPARQL endpoints. FedX only makes use of ASK queries for source selection. It implements a heuristic-based query planner. Comparing to DARQ, the number of endpoints requests is greatly reduced by using bind joins in a block nested loop fashion [22]. A query rewriting algorithm is used to push computation to the local endpoints by relying on information about the underlying RDF datasets. It implements a selectivity-aware query execution plan generation.

11http://lod-cloud.net/
12A HDT file with more than 500GB which requires more than 16 GB RAM to process.
13http://lodlaundromat.org/
14http://index.lodlaundromat.org/
SPLENDID, CostFed, ANAPSID, SemaGrow, and Odyssey are examples of the hybrid (index+ASK) SPARQL query federation approaches over multiple SPARQL endpoints. SPLENDID performs cost-based optimization using VOID statistics from datasets. It makes use of bind and hash joins [30]. CostFed also implements a cost-based query planner. The source selection is closely related to HiBISCuS [31]. Both bind and symmetric hash joins are used for data integration. ANAPSID [26] is an adaptive query federation engine that adapts its query execution at runtime according to the data availability and condition of the SPARQL endpoints. ANAPSID implements adaptive group and adaptive dependent joins [30]. SemaGrow adapts source selection approach from SPLENDID. It performs a cost-based query planning based on VOID statistics about datasets. SemaGrow implements bind, hash, and merge joins. Odyssey is also a cost-based federation engine. MULDER describes data sources in terms of RDF molecule templates and utilize these templates for source selection, and query decompositions and optimization.

DAW [32] and Fedra [33] are examples of duplicate-aware query federation approaches over multiple SPARQL endpoints.

SaGe [34] is a stateless preemptable SPARQL query engine for public endpoints. The system makes use of preemptable query plans and time-sharing scheduling. SaGe tackles the problem of RDF data availability for complex queries in public endpoints. Consequently, SaGe provides a convenient alternative to the current practice of copying RDF data dumps.

**Link Traversal based SPARQL federation:** LDQPS [35], SIHJoint [36], WoDQA [37], and SQUIN [38] are examples of traversal-based federated SPARQL query processing approaches. Both LDQPS and SIHJoint make use of the index and online discovery via link-traversal to identify the relevant sources pertaining to the given SPARQL query. They implement a symmetric hash join. WoDQA performs a hybrid (index+ASK) source selection approach. It implements nested loop and bind joins. SQUIN discovers the potentially relevant data during the query execution and thus produce incremental query results. SQUIN uses a heuristic for query execution plan generation, adapted from [39]. As a physical implementation of the logical plans, SQUIN uses a synchronized pipeline of iterators such that the i-th operator is responsible for the i-th triple pattern of the given SPARQL query. More recent studies [40] investigated 14 different approaches to rank traversal steps and achieve a variety of traversal strategies. A more exhaustive survey of the traversal-based SPARQL query federation is provided in [41].

Beside the above query federation strategies, low-cost triple pattern fragments (TPF) interfaces [42] can also be used to execute federated SPARQL queries. Comunica [43] is a highly modular meta engine for federated SPARQL query evaluation over support heterogeneous interfaces types, including self-descriptive Linked Data interfaces such as TPF. The system also enables querying over heterogeneous sources, such as SPARQL endpoints and data dumps in RDF serializations. The main drawback here is that the user should know in advance the dataset where the query will be executed.

One of the targets and motivation to build WimuQ was to discover, which dataset the SPARQL query can be executed. Unfortunately, with the exception of SQUIN [38], none of the related works provides this feature. Despite the fact that some of them incorporate Nquad support that allows the possibility to know the graph of the triple. The provenance system of WimuQ also includes dump files and endpoints with a rank provided by WIMU [10].

### 2.3. Relating and Integrating LOD datasets

**LOD Dataset Relation:** The works from Asprino et al. [44–46] present important statistics about LOD datasets, including a data structure called Equivalence Set Graph (ESG) that allows specifying compact views of large RDF graphs thus easing the accomplishment of statistical observations such as the number of concepts defined in a graph. Their work helps to show that LOD datasets are not really linked and the main difference here is that they do not have a goal to compute similarity level among the datasets based on the properties and class similarities.

Loupe [47] has an index of property and classes from some datasets, but still fewer datasets and it is not incremental.

Schultz, in his work [48] presents LDIF, a framework for Large-Scale Linked Data Integration, which has the intention to translate heterogeneous Linked Data from the Web into a clean, local target representation while keeping track of data provenance. The difference here is that there is no study on identifying the similarity among the datasets, in which the aim is to integrate data.

The work [49] presents integration rules between arbitrary datasets and a mediated schema as a method for
linking datasets. It is a very important work establishing complex mappings between linked open datasets, but still but still fewer datasets, the evaluation only on DBpedia, and not incremental.

The approach from [50] is built on the assumption that similar datasets should have a similar structure and include semantically similar resources and relationships based on the combination of Frequent Subgraph Mining (FSM) techniques, used to synthesize the datasets and find similarities among them.

The works from [51] and [52] also provide an approach to identify duplicated data in huge datasets from LODLaundromat and DBpedia. A good point to highlight in those works is the use of bloom filters, which helps to identify duplicated data. The identified different aims compared to our approach are an index of similarity of the datasets based on properties and classes shared among them and the identification of datasets to execute a given SPARQL query.

In [53], they present an approach to determine degrees of equivalences between relations (properties) defined by different LOD vocabularies. They process numbers of occurrences of matching pairs of RDF triples to assess intervals representing lower and upper levels of property equivalences. As a result, they obtain a graph of equivalent properties where interval-based strength of edges represent degrees of similarity between properties. The idea is similar to ReLOD, getting the similarity according to the number of properties the dataset share, they have an experiment with getting the similarity between DBpedia and Wikidata, where ReLOD experiments are with more than 600 datasets and ReLOD also consider classes to obtain the similarity score.

An quantitative empirical survey, where 130 datasets from the LOD Cloud were evaluated, gives insights into current quality conformance of the LOD Cloud datasets[54] and shows that quality issues in datasets have severe implications on consumers who rely on information from the Web of Data. Similar to this study, they also take in consideration a large scale amount of data. The difference is that the study does not cover how similar the datasets are.

**LOD Dataset Integration:** In [9] the author discusses the development of more holistic, clustering-based data integration approaches that scale to many data sources. The author outlines different use cases and provides an overview of initial approaches for holistic schema/ontology integration and entity clustering, highlighting problems about the non-trivial task of mapping and data integration, including a discussions about how such tasks are not performed fully automatically with a high level of manual interaction, in which usual problems are low data quality, unstructured, different data formats (relational, Json, etc.), a high degree of semantic heterogeneity. The paper highlights that the majority of approaches are for a small collection of data sources. Unfortunately, the paper does not discuss enough approaches to clustering LOD datasets on a large scale in reasonable time and scalable.

In [55], the authors describe systematic blocking methods, having a comfortable impact on the number of detected duplicates. They provide an approach for attribute clustering and comparison scheduling, identifying pairs of matching entities among two large, heterogeneous, and duplicate free but overlapping collections of entities. They have a technique called Data Blocking, which cluster similar entities into blocks and performs comparisons only among entities in the same block. It is an excellent blocking approach, but the method assumes that there is no dirty data among the relations, that means, for instance, that they assume all owl:sameAs links as true and there is no study about the similarity level among the datasets.

The approach[56] replaces the basic schema matching, showing that even attributes with different meaning can sometimes be useful in aligning data. They provide three algorithms for search linkage points of similarity, find overlaps using a sample and filtering. The output is a ranked list of pairs of paths \((p_s, p_t)\) in which the rank do not bring matches, but possibilities in which are not similar and not applied to a large amount of LOD datasets.

[57] provides the ability to incorporate new sources. The approach avoids the comparison of new entities with all members of a cluster, thereby limiting the matching overhead. They integrate equivalent entities in a more holistic manner such that equivalent entities from arbitrary many sources are clustered together e.g. within knowledge graphs. Such a clustering facilitates the combination of the property values of all clustered entities, e.g., persons, products, cities, etc, keeping the provenance from the sources of the entities. The approach was build to cluster entities, not LOD datasets, which we cannot use the same concepts here; also, the approach assumes that there are no duplicates within data sources. The evaluation uses a subset of 5 datasets, in which the focus was to evaluate the method, not apply to huge data, which is understandable because once they use huge data, they also need a huge gold standard to evaluate.
The work presented in[58] provides a new data source management system that automatically assesses the quality of data sources based on a collection of rigorous data quality metrics and enables the automated and interactive discovery of valuable sources for user applications. The system explores the trade-off between accuracy and coverage and allows users to discover the most relevant data sources for their application. They argue that the proposed method can dramatically simplify the Discover-Appraise-Evaluate interaction loop that many users follow today to identify sources for their applications. The difference here is that there is no approach for dataset similarity, and it is not applied for RDF datasets.

The best practice to query interlinked datasets is provided by WoDQA (Web of Data Query Analyzer)[59], which discovers datasets relevant to a query in an automated manner using VOID documents as metadata. Unfortunately, people are not following and we still have a huge amount of dirty datasets, which also reinforce the need for our approach.

The paper[60] introduces the rationale of each strategy shortly. Then, they evaluate the most commonly used methods on a number of mapping tasks and try to find the most robust strategy that behaves well across all given tasks. The method is quite clear and possible to follow. They show a study about approaches to finding mappings between complex schemas where the rationale of each strategy is fairly clear, and find the most robust strategy. Unfortunately, the study is not applied to LOD RDF datasets, neither to a huge amount of datasets and there is no study about the similarity among the datasets.

The method introduced in[61] is quite clear and possible to follow, in which they use owl:sameAs links to help in property matching. The method is organized into two steps: (1) Extract sameAs and transform it into tables. (2) Apply table matching techniques and propose a matching criterion to find relationships between properties. They use similarity functions, including string similarity and numeric similarity. A difference here is that they assume existing a owl:sameAs relation, which cannot be applied to thousands of HDT files, and the approach only relies on owl:sameAs links.

We also consider the Mapping Languages, which are used to create mappings among datasets. This position paper[62] starts an essential discussion about the relation of the datasets on Semantic Web. Another point to consider here is that this study reinforces the need for an approach to relate a large amount of heterogeneous RDF datasets.

The ENTITY RECONCILIATION COMMUNITY GROUP[16], are doing valuable work, in which aim to document an existing API, share the experiences and lessons learned from it[63].

We cannot finish this section without taking into consideration the valuable thoughts over the current state of Linked Data, described on the paper[64]. In which discuss the identity crisis on the Semantic Web community, how things are being related, and highlight the considerable problems that arise when we try to deploy semantics at Web-scale, reinforcing the need for a study about the relations over an extensive collection of datasets.

Those works cover an important relationship but none of them look into our main drawback here, which is to obtain the similarity level among a huge amount of datasets.

3. The proposed approach

3.1. Identifying LOD datasets

WIMU [10] uses the number of literals as a heuristic to identify the dataset which most likely defines a URI. The intuition behind this can be explained in two points: (1) Literal values are the raw data that can disambiguate a URI node in the most direct way and (2) The Semantic Web architecture expects that datasets reusing a URI only refer to it without defining more literal values. One more reason for point (1) is that: it is straightforward to understand whether two literal values are different, whereas disambiguating URIs usually requires more effort.

We store the collected data in an Apache Lucene index. Due to runtime performance and complexity reasons, we found that storing the information into Lucene was more convenient than a traditional triple store such as Virtuoso[18]. The rationale behind this choice is that a tuple such as (URI, Dataset, Score) would be expressed using at least three triples; for instance:

```
:R001 :hasURI :URI001
:R001 :hasDataset :Dataset001
```

References:

[16] https://www.w3.org/community/reconciliation/
[18] https://virtuoso.openlinksw.com/
where :R001 is an index record URI. Therefore, materializing all records would have substantially increased the space complexity of our index.

3.2. Querying and source selection of LOD datasets

In this section, we explain our approach to the SPARQL query processing. We assume that the reader is familiar with the concepts of RDF and SPARQL, including the notions of an RDF triple, a triple pattern, a basic graph pattern (BGP), and subject, predicate, object of the triple pattern.

Figure 1 shows the workflow of the query processing in our approach, which comprises of four main steps: (1) the user issue a SPARQL query to the WimuQ interface, which (2) extracts all the URIs used in the given user query. Note the URIs can be used in subject, predicate, or objects of the SPARQL triple patterns. (3) The extracted URIs are then searched in the WIMU index, which gives all the relevant datasets where the extracted URIs can be found. (4) Our Relational index(ReL0D)\textsuperscript{19} will provide the most similar basic graph pattern (BGP), and subject, predicate, object of the triple pattern.

3.2.1. The source selection

One of the important optimization steps in federated SPARQL query processing is the source selection [31, 65]. The goal of the source selection is to identify the potentially relevant datasets (also called sources) to the given query. We make use of the WIMU service [10] to select the potentially relevant sources. The WIMU service\textsuperscript{20} provides URIs lookup facility and identify those datasets which contain the given URI. Currently, this service processed more than 58 billion unique triples and indexed 4.2 billion URIs from more than 660k RDF datasets obtained from the most used Linked Data hubs including LODStats [13] and LOD Laundromat [14]. The identified WIMU datasets can be of four types: (1) SPARQL endpoint, (2) HDT file, (3) dataset with dereferenceable URIs, (4) data dump with non-dereferenceable URIs. The service is both available from a web interface as well as can be queried from a client application using the standard HTTP protocol.

3.2.2. The query execution

We make use of the four – SPARQL endpoint, Link Traversal-based, SPARQL-a-alot, Data dumps – query processor to execute federated queries over the aforementioned four types of WIMU datasets. The relevant data sources for each of these query processors are returned by the WimuQ source selection discussed in previous section.

We used FedX [22] query processor for SPARQL endpoints query federation and SQUIN for traversal-based query federation. The reason for choosing FedX for SPARQL endpoints federation and SQUIN for traversal-based federation is due the fact that they do not require any pre-computation of dataset statistics and hence are able to retrieve up-to-date results. Thus both are able to run federated queries with zero initial knowledge. In addition, both produce reasonably query runtime performances comparing to state-of-the-art approaches [25, 30, 38].

The list of required endpoints URLs for FedX are returned from the previously discussed source selection Algorithm (ref. \textbf{E} of Algorithm 2). The potentially relevant dereferenceable URIs data sources (ref. \textbf{T} of Algorithm 2) are already identified by the WimuQ sources selection algorithm. Thus, we reduced the search space by only considering the dereferenceable URIs data sources.

As mentioned before, FedX can only works with public SPARQL endpoints. SQUIN needs dereferenceable URIs. Both of these engines are unable to execute SPARQL queries over non-dereferenceable URIs datadumps: SPARQL endpoint federation approaches cannot execute queries over such datadumps as they are not exposed as SPARQL endpoints, link traversal-based approaches fail to retrieve results as the URIs are non-dereferenceable. We need to download the dumps first, load it locally, and run some query processing API (e.g., JENA or Sesame) on the loaded datasets.
However, we cannot simply download the complete dumps and process it locally due to their large amount of data. To solve this problem, we make use of the Wimu index to only select those data dumps which are potentially capable to execute the given query. These datadumps are further sliced by using the RDF-Slice technique, to only select the required chunks of the datadumps which will provide results to the given SPARQL query. The identified chunks are finally loaded in a local Apache Jena model. The model is then used to execute federated queries.

The results generated by each of WimuQ’s processors are finally integrated and sent back to the user.

### 3.3. RELOD—The Incremental LOD Dataset relation index

Our observations guide us to a method to create an index of LOD datasets, in which involves a compilation of many other works, starting collecting data from more than 650,000 LOD datasets from LODStats, LODLaundromat and more than 500 endpoints.

### 3.4. The index creation

Given a set of Datasets $\mathbf{D}\{d_1, \ldots, d_n\}$, assuming $d_n$ represents a RDF dataset, in which each $d$ contains a set of properties and classes represented by $P$. The next step consists in creating a set $\mathbf{E}$ containing the identified duplicated datasets. Thus, we can exclude from $\mathbf{D}$ all datasets contained in $\mathbf{E}$.

The goal of algorithm 1 is to create an index to store the relations among datasets by comparing the occurrences of classes and properties of each dataset, by String similarity and Instance Matching. The Input is a set of Datasets and the Output is the index of dataset relations.

Fig. 2 shows a workflow about how the index is created and the algorithm 1 describe with more details the process of creation of the dataset relation index.

The function $\text{getExactMatches}(d, t)$ compares all properties and classes from each dataset $d$ and $t$ and return a set of properties and classes that are exact the same, keeping the provenance of the dataset.

The function $\text{getSimMatches}(d, t, 0.8, M_e)$ compares all properties and classes from each dataset $d$ and $t$ using Jaccard and MFKC String Similarity function with a threshold of 0.8, excluding the exact matches identified previously by the function $\text{getEx-}$...
**Algorithm 1** Creation of the LOD dataset relation index

**Input:** $D, E$  
$\rightarrow$ Datasets and datasets duplicated

**Output:** $C, X, Y, Z$  
$\rightarrow$ Four tsv files containing the matches(datasets, classes and properties).

1: $M_e \{\}$  
$\rightarrow$ HashMap containing dataSetTarget and another hash containing properties and classes exact matched

2: $M_s \{\}$  
$\rightarrow$ HashMap containing dataSetTarget and another hash containing properties and classes matched using String similarity

3: $M_a \{\}$  
$\rightarrow$ HashMap containing datasets already compared

4: $D - E$  
$\rightarrow$ Remove duplicates contained in $E$

5: $T = D$

6: for all $d \in D$ do

7: \hspace{1em} for all $t \in T$ do

8: \hspace{2em} if $(d <> t) \wedge \text{alreadyCompared}(d, t)$ then

9: \hspace{3em} continue  
$\rightarrow$ Skip the current $d$ and $t$

10: \hspace{2em} end if

11: \hspace{2em} $M_e.add(getExactMatches(d, t))$  
$\rightarrow$ Add a set containing the properties/classes that are exact the same in both datasets

12: \hspace{2em} $M_s.add(getSimMatches(d, t, 0.8, M_e))$  
$\rightarrow$ Add a set containing the properties that are similar in both datasets, excluding the matches from $M_e$

13: \hspace{2em} $M_a.add(d, t)$  
$\rightarrow$ Add the dataset pair already processed

14: \hspace{2em} end for

15: \hspace{2em} end for

16: \hspace{2em} printMaps($M_e, M_a$)  
$\rightarrow$ Print the content of $M_e$ and $M_a$ inside the files $C, X, Y, Z$

\textbf{actMatches} returning a set of properties and classes that has the similarity greater than the threshold 0.8, keeping the provenance of the dataset, the Fig. 4 shows our string similarity process.

The function \textbf{printMaps($M_e, M_a$)} print the content from the matching functions into the files according the structure described in section 3.6.
3.5. Identifying duplicated and chunk datasets

With the current hardware available (HD 1Tb, Memory 16GB, processor intel core i7). To compare all LOD datasets, in this case more than 600 thousand datasets, implies in more than 360 billion of comparisons\(n \cdot n\), assuming that each comparison takes on average 1 millisecond, the whole operation will take more than 10 years.

Thus, we decided to have a blocking method to avoid unnecessary comparisons. To this aim, we create a way to detect duplicated and chunk-dump-files datasets, described on Algorithm 2. Our blocking method avoid datasets with less than 1 triple, duplicates, less than 5 properties, less than 5 classes.

The function `eliminateDuplicates()`, identify and eliminate duplicated HDT files by comparing the property occurrences of each dataset and the header metadata.

We formalize the problem of Clustering datasets identifying duplicates and chunks as follows:

We consider a set of \( K \) data sources \( S_1, \ldots, S_k \) containing property occurrences \( O_1, \ldots, O_m \). Each of property \( P \) is referenced by an URI, e.g., dbo:City. Each \( S \) contains a header \( H \), with meta-data about the data in \( S \), such that \( H \subset S \).

The goal is to create clusters of datasets \( C \) in two groups of elements from \( K \), in which are duplicates \( D \) and chunks \( E \), such that \( D \subset K : E \subset K : K \subset C \).

We assume that all datasets are not empty and are RDF compatible with the HDT format.

Firstly we create a dense matrix of property occurrence and dataset. Then it was observed a standard in the dense matrix, that for some datasets the occurrence of the properties was exactly the same, as Fig. 5 try to show a sample with 11 datasets, with 7 chunks identified. Then, we identify that we can put together those datasets to observe more characteristics. In this second phase, we have a sub-collection of our initial collection of datasets and looking into the header metadata of those files we observe that for some files the metadata is different. Then we have another sub-collection of files with different headers, and manually reading file by file we realize that they are chunks, in other words, part of a bigger dataset.

Thus we have two phases for clustering dataset (1) Put together datasets that present the same occurrence of properties. (2) Read the header metadata and separate files with a different header. The files with different header are chunks and the rest are duplicates.

The function `getPropertyOccurrence(ds)` returns a hash map containing the property and the number of occurrences of each property from a given dataset with the SPARQL query at Listing 2

25Those 11 datasets and the source are available here: https://tinyurl.com/11chunkLaundromat

26More details, please see the implementation and the documentation on https://github.com/firmao/wimuT and the specific implementation java class for this task: https://github.com/firmao/wimuT/blob/master/src/org/wimu/datasetselection/parallelv1/ClusterKmeans.java
Algorithm 2 Identifying duplicates and chunk

```
Input: D  \rightarrow Datasets
Output: C, X \rightarrow two files chunks.txt and duplicates.txt.
1: sep = t  \rightarrow Separator for the file, in this case, a tab, but could be another such as ",", etc..
2: setHeader = {}
4: for all d \in D do
5:   H = getPropertyOccurrence(ds)
6:   for all key, value \in H do  \rightarrow key represent the property and value represents the number of occurrences.
7:      setHeader.add(key)
8:      line = line + sep + value
9:   end for
10:  printLineDenseMatrix(d + line)
11:  if setLines.contains(line) then  \rightarrow Here we identify and put chunks and duplicates together
12:     D'.add(d)
13:  else
14:     setLines.add(line)
15:  end if
16: end for
17: printHeaderDenseMatrix(setHeader)
18: C.add(diff(D'))  \rightarrow Here we separate chunks and duplicates
19: X.add(diff(D'))
20: return C, X  \rightarrow two files chunks.txt and duplicates.txt.
```

Listing 2: Property occurrence query.

The function printLineDenseMatrix(line) print a line in a file and the function printHeaderDenseMatrix(setHeader) print the first line of the dense matrix file containing the header of the matrix, in which refers to the identification of the properties. The function diff(D') separates the chunks from duplicates, for this task we look into the header of the files, in this case, only HDT files, where duplicates present exactly the same header metadata information and chunks are different respect to the header.

Theorem 1: Let P be a collection of property occurrences of datasets D, such that each D has one P, in which D' represents the cluster candidates, in which are the datasets identified as chunks and duplicates together, and the function diff() return the dump-files that are not duplicated. Thus, the output of Algorithm 1 when applied to D, the following hold:

- \(|\text{diff}(D')| = 0\)
- All elements from D' are duplicated datasets.
- To identify the chunks:
  - \(|\text{diff}(D')| > 0\)

Fig. 5. The standard observed in the dense matrix, showing that for some datasets the occurrence of the properties was exactly the same, in this case a sample with 11 datasets with 7 dataset chunks identified.
The result from $\text{diff}(D')$ are the dump-files identified as chunks.

### 3.6. The file structure

Now we describe the file structure that we created to use our index, which consists of 3 TSV\(^{27}\) files.

- **tableMatches\_Exact.tsv**: Contains the exact match of the properties, with the following fields:
  - **Property**: containing the property URI itself.
  - **Source**: Contains the dataset source where the property was found.
  - **Target**: Contains the dataset matched containing the property.

- **tableMatches\_Sim.tsv**: Represents the similarity of properties from datasets Source and Target, with the following fields:
  - **PropertyS**: The property from the dataset Source.
  - **PropertyT**: The property from the dataset Target.
  - **Source**: The dataset Source.
  - **Target**: The dataset Target.

- **tableMatches.tsv**: Represents the datasets Matched and his respectives number of properties exact matched and number of properties matched with String similarity, with the following fields:
  - **Source**: The dataset source.
  - **Target**: The dataset Target.
  - **#ExactMatch**: Number of properties with exact match.
  - **#sim>0.9**: Number of properties with similarity threshold greater than 0.9.

### 3.7. Querying the index

The index provides the following information based on three types of input:

- **Dataset**: The index will provide a list of datasets, number of exact matched\(^{28}\) properties and number of similar\(^{29}\) properties.
- **Set of properties (URIs) separated by comma**:
  - Json file containing the property and the list of datasets where this property were found by exact match.
  - A table containing the matches by similarity with the following fields:
    - **Property Source**: Representing the property found on the dataset source.
    - **Property Target**: Representing the property found on the dataset target.
    - **Dataset Source**: The name of the dataset source.
    - **Dataset Target**: The name of the dataset target.
    - **SPARQL query**: This type of input extracts the set of properties from the SPARQL query and performs the same operation as **Set of properties (URIs) separated by comma**.

The index is also incremental, allowing to add more datasets to be processed once a month\(^{30}\). The prototype is available online for proof of concept online\(^{31}\) and the Fig. 3 shows a workflow about how the index is used.

### 4. Evaluation

#### 4.1. Evaluation on Identifying LOD datasets

To the best of our knowledge, LODStats\(^{13}\) is the only project oriented to monitoring dump files; however, its last update dates back to 2016. Observing Table 2, we are able to say that from LODStats, not all datasets are ready to use. Especially, more than 58% are off-line, 14% are empty datasets, 8% of the triples that have literals as objects are blank nodes and 35% of the online datasets present some error using the Apache Jena parser\(^{32}\). A large part of those data was processed and cleaned by LOD Laundromat\(^{14}\).

### Table 2

<table>
<thead>
<tr>
<th>LOD Laundromat</th>
<th>LODStats</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4185,133,445</td>
<td>31,121,342</td>
<td>4,216,254,787</td>
</tr>
<tr>
<td>658,206</td>
<td>9960</td>
<td>668,166</td>
</tr>
<tr>
<td>19,891,702,020</td>
<td>38,606,408,854</td>
<td>58,498,111,056</td>
</tr>
</tbody>
</table>

\(^{27}\)Tab Separated Value(TSV)

\(^{28}\)Exact Match where the URI is exact the same following the principle of uniqueness of the URI

\(^{29}\)Similarity match occurs when the similarity of the URI is greater than 0.9 if less we perform Instance matching.

\(^{30}\)Once a month due to the the time to generate that with our hardware takes at least 88 hours

\(^{31}\)http://w3id.org/relod/

\(^{32}\)See https://github.com/dice-group/wimu/blob/master/ ErrorJenaParser.tsv.
Table 1
Sample datasets from [66] to evaluate our string similarity approach.

| Data set   | Source (S)     | Target (T)     | |S| × |T| | Source Property | Target Property |
|------------|----------------|----------------|----------------------|----------------|------------------|
| (P3) Abt-Buy | Abt | Buy | 1.20 × 10^{6} | product name, description manufacturer, price | product name, description manufacturer, price |
| (P4) Amazon-GP | Amazon | Google Products | 4.40 × 10^{6} | product name, description manufacturer, price | product name, description manufacturer, price |
| (P5) DBLP-ACM | ACM | DBLP | 6.00 × 10^{6} | title, authors venue, year | title, authors venue, year |
| (P6) DBLP-Scholar | DBLP Scholar | Google Scholar | 0.17 × 10^{9} | title, authors venue, year | title, authors venue, year |
| (P7) MOVIES | DBpedia | LinkedMDB | 0.17 × 10^{9} | dbp:name | dc2:title |

The algorithm took three days and seven hours to complete the task. Thus, we will create a scheduled job to update our database index once a month. With respect to the information present in the Fig. 6, we can observe that the majority of files from LODStats are in RDF/XML format. Moreover, the endpoints are represented in greater numbers (78.6%), the dominant file format is RDF with 84.1% of the cases, and 56.2% of errors occurred because Apache Jena was not able to perform SPARQL queries. Among the HDT files from LOD Laundromat, 2.3% of them could not be processed due to parsing errors. Another relevant point is that 99.2% of the URIs indexed with WIMU come from LOD Laundromat, due to 69.8% of datasets from LODStats contain parser errors in which WIMU was not able to process the data.

Finally, we validated our heuristic assessing if the URI really belongs to the dataset with more literals. To this end, we took a sample of 100 URIs that belong to at least two datasets, where we assess manually the data in order to check if the results are really correct. As a result, the dataset containing the correct information was found as first result in 90% of the URIs and among the top three in 95% of the URIs.

To see how and where people are using WIMU the Fig. 7 shows the usage of WIMU by country, where we have 63526 number of queries, using 35984 URIs, in which 35918 are valid URIs. The data was collected from December 2018 until February 2020.

4.2. Evaluation on Querying and source selection LOD datasets

In this section, we present the evaluation setup and the corresponding results that validate our hypothesis that we can improve the resultset retrieval if we automatically identify potentially relevant sources from heterogeneous RDF data even if the URIs are not dereferenceable anymore. The goal of this evaluation is to show that by combining different SPARQL query processing approaches, we are able to retrieve more complete results as compared to the results retrieved by the individual approaches.

4.3. Experimental setup

4.3.1. Benchmarks used:

Since WimuQ aims to execute SPARQL queries over real-world RDF datasets, we chose three – FedBench [67], LargeRDFBench [68], Feasible [69] – real-world RDF datasets benchmarks in our evaluation:

- **FedBench** is federated SPARQL querying benchmarks. It comprises of a total of 25 queries and 9 real-world interconnected datasets. FedBench queries are further divided into three main categories: (1) 7 queries from Life Sciences (LS) domain, (2) 7 queries from Cross Domain (CD), and (3) 11 queries named Linked Data (LD) for link traversal-based approaches. The detailed statistics of the benchmark’s datasets and queries are given in FedBench[67].

https://github.com/dice-group/wimu/blob/master/result100.csv
– **LargeRDFBench** is also a federated SPARQL querying benchmark. It comprises a total of 40 queries and 13 real-world interconnected datasets. FedBench queries are further divided into four main categories: (1) 14 *Simple* queries, (2) 10 *Complex* queries, (3) 8 *Large Data* queries, and (4) 8 *Complex+High Data Sources* queries. The detailed statistics of the benchmark’s datasets and queries are given in [68].

– **FEASIBLE** is a benchmark generation framework which generates customized benchmarks for the queries logs. In our evaluation, we chose exactly the same benchmarks used in [69]: (1) 175 queries benchmark generated from DBpedia queries log and (2) 175 queries benchmark generated from Semantic Web Dog Food (SWDF) queries log. Further advanced statistics of the used datasets and queries can be found in [69].

To the best of our knowledge, these are the state-of-the-art from the real-data SPARQL benchmarks. All of the 415 queries used in our evaluation is publicly available.\(^{34}\)

### 4.3.2. Hardware:

All the experiments were done on a modest machine with 200 GB of Hard Disk, 8 GB of RAM and a 2.70GHz single core processor. Each of the queries was run 5 times and the average of the results are presented.

### 4.3.3. SPARQL endpoints:

As previously stated, the query federation over multiple SPARQL endpoints approaches requires the set of endpoint URLs to be provided as input to the federation engine. We chose a total of 539 active SPARQL endpoints available from LOD cloud.\(^{35}\) We filtered the endpoints URLs\(^{36}\) and the total number of triples hosted by each of these endpoints.

\(^{34}\)Queries available from https://github.com/firmao/wimuT/blob/master/queriesLocation.txt

\(^{35}\)List of SPARQL endpoints: https://lod-cloud.net/lod-data.json

\(^{36}\)Endpoints URLs with size: https://goo.gl/H2t5ko
4.3.4. Metrics:
Since WimuQ aims to retrieve more complete results within the reasonable amount of time, we choose two metrics: (1) coverage in terms of the number of results retrieved from the query executions and (2) the time taken to execute the benchmark queries.

4.3.5. Approaches:
As mentioned in Section 2, different federation engines available to federate SPARQL queries over endpoints and traversal-based federation. We chose FedX [22] for SPARQL endpoint federation and SQUIN [38] for traversal-based query federation. The reason for choosing these two engines is due the fact they do not require any pre-computation of dataset statistics and hence able to retrieve up-to-date results and able to run federated queries with zero initial knowledge. In addition, both these engines perform reasonably well in terms of query runtime performances w.r.t state-of-the-art approaches [25, 30, 38]. For the sake of completeness, we also compared WimuQ with SPARQL-a-lot and WimuDumps.

4.3.6. Results
Coverage of the results: The main purpose of WimuQ is to devise a federation engine which is able to retrieve more complete results for the given SPARQL queries. Figure 8 shows a comparison of the selected approaches in terms of the average of the number of results retrieved for the different queries categories of the selected benchmarks. The complete results for individual queries can be found on our aforementioned project website. By using different query processing engines, our approach is able to retrieve more results as compared to the results retrieved by only using SPARQL endpoints federation engine (i.e., FedX) link traversal engine (i.e., SQUIN), data dumps, or HDT files. In our evaluation, the average resultset size of WimuQ is 8481 across the three benchmarks. Out of these, WimuQ collects about 91% of the results from wimuDumps (avg. resultset size 7651), 7% from SPARQL endpoints (avg. resultset size 556), and 1% from SPARQL-a-lot (avg. resultset size 74).

For FedBench, the WimuQ avg. resultset size is 2253. Out of these results, about 55% are collected from SPARQL endpoints by using FedX query processing engines (avg. resultset size 1262). The LinkTraversal (SQUIN) contributed about 25% of the total results (avg. resultset size 549), wimuDumps contributed about 10% of the total results (avg. resultset size 226), and SPARQL-a-lot also contributed about 10% of the results (avg. resultset size 215).

For LargeRDFBench, the WimuQ avg. resultset size 123. Out of these results, about 81% are collected from wimuDumps (avg. resultset size 100). The SPARQL endpoints contributed about 14% of the results (avg. resultset size 17). The LinkTraversal(SQUIN) contributed 6% of the total results (avg. resultset size 6), and SPARQL-a-lot did not provide results (avg. resultset size 0).

For FEASIBLE, the WimuQ avg. resultset size 34 537. Out of these results, about 98% are collected from wimuDumps (avg. resultset size 33 893). The SPARQL endpoints contributed about 1.6% (avg. resultset size 577). The LinkTraversal(SQUIN) approach contributed only about 0.15% (avg. resultset size 54). Finally, SPARQL-a-lot query processing engine only retrieved about 0.03% results (avg. resultset size 11).

In summary, WimuQ is able to retrieve at least one resultset for 76% of the overall 415 queries. The results clearly shows that by combining different query processing engines into a single SPARQL query execution framework lead towards more complete resultset retrieval. An important observation is that the selected approaches are mostly not able to retrieve results for the Large Data and Complex+High (Ch) queries categories of LargeRDFBench. The reason is getting zero results for the Large Data queries is that these queries retrieve results from the LinkedTCGA [70] datasets which were not publicly available via SPARQL endpoints, were not indexed by WIMU, also were not reachable via link traversals. While Ch queries often require higher number of distributed datasets in order to compute the final resultset of the queries. Thus, the approaches were able to find all of the relevant datasets, required to compute the final resultset of the queries. The average number of datasets37 queries by WimuQ for the selected benchmarks queries categories in given in Figure 9. We can clearly see the highest number of datasets are selected for Ch queries of LargeRDFBench.

Query runtime performances: Figure 10 shows a comparison of the selected query processing engines in terms of the average query run times for the different queries categories of the selected benchmarks. The average query runtime of WimuQ is 17 minutes across the three benchmarks. The average query execution to collect results from wimuDumps is about 2 minutes, which in turn is followed by query execution over SPARQL endpoints (avg. query runtime

37Here we also point the number of datasets discovered
13 minutes), SPARQL-a-lot (avg. query runtime 58 seconds) and LinkTraversal (SQUIN) (avg. query runtime 36 seconds). Interestingly, WimuQ collects about 91% of the results from wimuDumps yet its average execution time is smaller than query execution over SPARQL endpoints which provide only 7% of the total results. One possible reason for this could be that in SPARQL endpoint federation, the query processing task split among multiple selected SPARQL endpoints and hence network and the number of intermediate results play an important role in the query runtime performances.

For FedBench, the WimuQ average query runtime 20 minutes. Out of this, the average avg. query runtime over SPARQL endpoints is 16 minutes, followed by wimuDumps (avg. query runtime 2 minutes), LinkTraversal (SQUIN) (avg. query runtime 49 seconds), and SPARQL-a-lot (avg. query runtime 46 seconds), respectively. For LargeRDFBench, the average query execution of WimuQ is 11 minutes. Out of this query execution over SPARQL endpoints took 8 minutes on average, followed by wimuDumps (avg. query runtime 2 minutes), SPARQL-a-lot (avg. query runtime 35 seconds), and LinkTraversal (SQUIN) (avg. query runtime 25 seconds), respectively. For FEASIBLE, the WimuQ takes on average of 24 minutes per query execution. Out of this, the query federation over SPARQL endpoints took about 18 minutes on average. Which is followed by wimuDumps (avg. query runtime 3 minutes), SPARQL-a-lot (avg. query runtime 1), and LinkTraversal (SQUIN) (avg. query runtime 36 seconds), respectively.
As an overall query runtime evaluation, we can clearly see there is a trade-off between the recall and query runtimes: the highest the recall the highest the query runtimes. Finding a balance between the recall and runtime would be an interesting research question to be considered in the future.

The results of our evaluation lead us to validate our hypothesis that we can improve the resultset retrieval if we identify potentially relevant sources from heterogeneous RDF data.

### 4.4. Evaluation on Relating LOD datasets

This evaluation aims to answer the following questions: (1) How to identify and quantify similar datasets for a given Dataset? (2) How many datasets are most likely to execute a given SPARQL query? (3) How the detection of duplicated and chunk datasets can help in the process of matching a large amount of datasets? (4) How ReLOD increase the number of datasets identified by wimuQ?

The information contained at Table 4 and Table 5 are used to see how a sample of datasets are similar, we take 8 famous datasets from [rdflib.org](http://www.rdflib.org) and 12 from [71]. To obtain the results from Table 4, which is to see how similar are the datasets, we use the formula Eq. (1) and to obtain the results from Table 5, to see how much a dataset is contained inside each other we use the formula in Eq. (2), where A and B represents datasets source and target. We have an alternative way to visualize this sample data as a graph.

\[
\text{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)
\]

\[
\text{contained}(A, B) = \frac{|A \cap B|}{|A|} \quad (2)
\]

To answer the question (1) we give to our approach as input a version of the dataset DBpedia, according to the properties and classes they share. On the Table 6 we show the number of properties that they share that contains the exact same URI and cases where they share not the exact URI but very similar. Thus, an example of application could be that when the user wants to identify information from other datasets to complement or enrich information for a given dataset, i.e. facilitating federated queries. The Table 7 shows a evaluation with 600 randomly chosen datasets including 3 synthetic manually made datasets.

With our experiments we realize that more than 50% of properties and classes has a match in another dataset. The information can also be observed at Fig. 11, in which highlight the high possibility that one dataset can enrich each other with complementing information from another dataset.

The time consumed can be observed on Fig. 12, which shows that 100 million triples were processed in less than 8 minutes, in which is an acceptable time, in which the starts to be time-consuming only after 1 million triples.

As we are using WimuQ[11] to identify datasets for a given SPARQL query together with our matching algorithm, we can answer the question (2) based on famous Sparql queries from two real-data federated SPARQL querying benchmarks LargeRDFBench [68], FedBench [67] and one non-federated real data SPARQL benchmark selected from FEASIBLE [69] benchmarks generation framework is given in Figure 9. Thus, from the selected datasets we use our approach to select the datasets according to the properties and classes they share among them.

To answer the question (3) we evaluate the dataset duplicated detection algorithm and the detection of dataset chunks. From a 5446 datasets our algorithm detected 2272 duplicates and 1470 chunks in 3 hours.

The Fig. 13 shows a case with 900 datasets chosen randomly, where we identified in 10 cases in which we can see the difference.

The Fig. 14 show how many chunks we were able to identify among our sample data, in which lead us to

---

38To know how many datasets are similar to each other.
40[https://w3id.org/relod/visual.html](https://w3id.org/relod/visual.html)
42We compare with the datasets from [http://www.rdfhdt.org/datasets/](http://www.rdfhdt.org/datasets/)
43With similarity level greater than 0.8 according our similarity algorithm.
45In this case, all datasets identified by wimuQ are sharing properties and classes, that is why we are using the graph from [11].
46A subset from those 600 datasets chosen previously
Fig. 10. Average query runtimes of the selected approaches across different queries categories of the selected benchmarks.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>File</th>
<th>Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>acadonto</td>
<td>acadonto.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>acm.rkbexplorer</td>
<td>acm.rkbexplorer.com,d.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>agrinepaldata</td>
<td>agrinepaldata.com.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>aims</td>
<td>aims.fao.org,os,isu.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>alaska</td>
<td>alaska.eagle-i-net.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>apertium</td>
<td>apertium-rdf-ca-it.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>arrayexpress</td>
<td>arrayexpress-e-afmx-1.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>athelia</td>
<td>athelia.com.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>vivosearch</td>
<td>beta.vivosearch.org_institution_ponce-school-medicine.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>bfs.270a</td>
<td>bfs.270a.info_dataset_.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>biblioteca-nacional-biblioteca-nacional-escolar-bnescolar.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
<td></td>
</tr>
<tr>
<td>brown</td>
<td>brown-corpus-in-rdf-nif.nt.gz.hdt</td>
<td><a href="http://s001.adaptcentre.ie/lod/htdumps/">http://s001.adaptcentre.ie/lod/htdumps/</a></td>
</tr>
<tr>
<td>dblp</td>
<td>dblp-20170124.hdt</td>
<td><a href="http://www.rdfhdt.org/datasets/">http://www.rdfhdt.org/datasets/</a></td>
</tr>
<tr>
<td>DBPedia</td>
<td>DBPedia-3.9.en.hdt</td>
<td><a href="http://www.rdfhdt.org/datasets/">http://www.rdfhdt.org/datasets/</a></td>
</tr>
<tr>
<td>geonames</td>
<td>geonames-11-11-2012.hdt</td>
<td><a href="http://www.rdfhdt.org/datasets/">http://www.rdfhdt.org/datasets/</a></td>
</tr>
<tr>
<td>linkedgeodata</td>
<td>linkedgeodata-2012-09-10.hdt</td>
<td><a href="http://www.rdfhdt.org/datasets/">http://www.rdfhdt.org/datasets/</a></td>
</tr>
<tr>
<td>wordnet</td>
<td>wordnet31.hdt</td>
<td><a href="http://www.rdfhdt.org/datasets/">http://www.rdfhdt.org/datasets/</a></td>
</tr>
<tr>
<td>yago</td>
<td>yago2s-2013-05-08.hdt</td>
<td><a href="http://www.rdfhdt.org/datasets/">http://www.rdfhdt.org/datasets/</a></td>
</tr>
</tbody>
</table>

Dataset description used on Table 4, Table 5 and Table 6

know how segregated is data analysed, giving also the chance to know the complete dataset after the union of all chunks.

The current version of the index prototype has information about 539 SPARQL endpoints from LOD cloud and 915 HDT files from LOD Laundromat. We perform more than 1 800 000 comparisons in 88 hours.

47The list of is available here: https://github.com/firmao/wimuT/blob/master/Endpoints_numtriples_lodcloud.csv
48fdsa
analysed. For instance, from 600 datasets, 291 matches
for datasets. We can observe the quantity of properties/classes
with a small sample, where we can see at Fig. 15, Ta-
number of matches in this case cannot be directly re-
lated to the number of triples. Due to this fact, the qual-
ity of the datasets should be considerate a important
phase.
We evaluate the accuracy of our matching algorithm
with a small sample, where we can see at Fig. 15, Ta-
ble 1 and Fig. 16 the F-Measure and run-time on six
A sample of how much a dataset in contained in each other, in which the level of containment is also represented by the intensity of the color, as more intense color, as more contained are the datasets.

Top 10 datasets containing exact the same URI and containing the most similar URIs according to our similarity approach, in which #PropClass represents the total number of properties and classes from the dataset.

The Fig. 6 shows that the majority of files from LODStats are in RDF/XML format. Moreover, the endpoints are represented in greater numbers (78.6%),
Fig. 14. Chunks identified in 900 datasets. Y axis represents the number of datasets.

<table>
<thead>
<tr>
<th>#Datasets</th>
<th>#DropMatch</th>
<th>#triples</th>
<th>Synthetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>2</td>
<td>1198508</td>
</tr>
<tr>
<td>100</td>
<td>39</td>
<td>12</td>
<td>7996408</td>
</tr>
<tr>
<td>200</td>
<td>92</td>
<td>40</td>
<td>22982984</td>
</tr>
<tr>
<td>300</td>
<td>135</td>
<td>57</td>
<td>34792121</td>
</tr>
<tr>
<td>400</td>
<td>186</td>
<td>73</td>
<td>43864522</td>
</tr>
<tr>
<td>500</td>
<td>229</td>
<td>87</td>
<td>54227780</td>
</tr>
<tr>
<td>600</td>
<td>291</td>
<td>425</td>
<td>85041239</td>
</tr>
</tbody>
</table>

Table 7: Evaluation on the Match algorithm, where #PropMatch refers to the number of properties/classes the datasets share among each other.

To answer the question (4) we selected 10 queries from FedBench[67], where we can observe on the Fig. 17 that thanks to the ReLOD approach we could increase the number of datasets identified by wimuQ, i.e., in query 5, using ReLOD allowed us to find 2 more datasets containing complementary information.

On the other hand, the Fig. 18 reinforces that more datasets do not always imply in more results, i.e., in query 2 and 4, more datasets were identified, but the number of results did not change, in this case, the reason was that the datasets identified by ReLOD were practically the same with different property and class names. The results from queries 6 to 10 were only found thanks to the ReLOD approach.

The dominant file format is RDF with 84.1% of the cases, and 56.2% of errors occurred because Apache Jena was not able to perform SPARQL queries. Among the HDT files from LOD Laundromat, 2.3% of them could not be processed due to parsing errors. Another relevant point is that 99.2% of the URIs indexed with WIMU come from LOD Laundromat, due to 69.8% of datasets from LODstats contain parser errors in which WIMU was not able to process the data.

The queries are available here: https://github.com/firmao/LDatasetGenerator/blob/master/10_Queries_fedbench.txt

The query number 9 obtained only one result with the new approach.
5. Conclusion

With WIMU [10], we provide a database index of URIs and their respective datasets built upon large Linked Data hubs such as LODStats and LOD Laundromat. In order to make this data available and easy to use, we developed a semantic web service. For a given URI, it is possible to know the dataset the URI likely was defined in using a heuristic based on the number of literals.

With wimuQ[11], we presented an approach to execute SPARQL queries over a large amount of heterogeneous RDF data sources available from different interfaces and in different formats. We made use of the Wimu service to identify the potentially relevant sources to the given SPARQL query. We discussed two main types of federated SPARQL query processing approaches namely the endpoints federation and traversal-based federation. The former type of federation only able to execute federated queries over the data available from SPARQL endpoint. While the later, faces problem of URI’s dereferenceability. To overcome these issues we proposed a hybrid (endpoints+link-traversal-based) federation engines which integrates four different types of SPARQL query processing engines. Currently, WimuQ[52] able to execute both federated and non-federated SPARQL queries over a total of 668k datasets available from LOD Stats, LOD Laundromat, and LOD cloud active SPARQL endpoints. We evaluated WimuQ by using three state-of-the-art real-data SPARQL benchmarks. We showed that WimuQ is able to successful execute (with some results) majority of the benchmark queries without any prior knowledge of the data sources. In addition, the WimuQ resultset recall is higher with reasonable query execution times.

With ReLOD[53], we present a method to create a repository to store the similarity between a large number of datasets involving the detection of duplicated datasets and dataset chunks.

The method involves the creation of an index over a total of 668 166 datasets from LOD Stats and LOD Laundromat as well as 559 active SPARQL endpoints, representing a total of 221.7 billion triples from more than 5 terabytes of information from datasets partially retrieved using the service “Where is My URI” (WIMU)[10], in which the query engine wimuQ[11] uses the ReLOD to identify the most similar datasets increasing the number of results.

For the first time, to the best of our knowledge, we make a relation index that can query by dataset URI, property, class, or SPARQL query.

Our experiments show that more than 90% of datasets from LODLaundromat datasets are not using owl:equivalentProperty and owl:equivalentClass or another way to relate the data, reinforcing the need for an index of relations among LOD datasets.

We realize that the datasets still not sharing the expected properties, and the ontologies are not aligned, making hard the task to query multiple heterogeneous datasets. We believe that this work can help people to identify similar datasets among a large number of datasets. As future work, the next step is to improve the GUI drawing a weighted graph that shows the similarity level of each dataset.

We cannot finish this paper without reminding the reader of an aporia in which says - Different people describe things differently, and this anguish will always lie with us - That means, at least, this study needs a continuation.

The source-code is available online[54].

References


[52]https://github.com/firmao/LODDatasetRelationsWeb

53https://w3id.org/relod

54https://w3id.org/wimuq/


18. J. Rouches, G. de Melo and K. Hose, Complex schema mapping and linking data: Beyond binary predicates (2016).


1. Valdestilhas et al. / Identifying, Querying, and Relating Large Heterogeneous RDF Sources