information

MDPI

Article

High Performance Methods for Linked Open Data Connectivity Analytics

Michalis Mountantonakis [1,2,*] **and Yannis Tzitzikas** [1,2,*]

[1] Institute of Computer Science, FORTH-ICS, Heraklion 70013, Greece
[2] Department of Computer Science, University of Crete, Heraklion 70013, Greece
* Correspondence: mountant@ics.forth.gr (M.M.); tzitzik@ics.forth.gr (Y.T.)

Received: 9 May 2018; Accepted: 29 May 2018; Published: 3 June 2018

Abstract: The main objective of Linked Data is linking and integration, and a major step for evaluating whether this target has been reached, is to find all the connections among the Linked Open Data (LOD) Cloud datasets. *Connectivity* among two or more datasets can be achieved through *common Entities*, *Triples*, *Literals*, and *Schema Elements*, while more connections can occur due to equivalence relationships between URIs, such as `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass`, since many publishers use such equivalence relationships, for declaring that their URIs are equivalent with URIs of other datasets. However, there are not available connectivity measurements (and indexes) involving more than two datasets, that cover the whole content (e.g., entities, schema, triples) or "slices" (e.g., triples for a specific entity) of datasets, although they can be of primary importance for several real world tasks, such as *Information Enrichment*, *Dataset Discovery* and others. Generally, it is not an easy task to find the connections among the datasets, since there exists a big number of LOD datasets and the transitive and symmetric closure of equivalence relationships should be computed for not missing connections. For this reason, we introduce scalable methods and algorithms, (a) for performing the computation of transitive and symmetric closure for equivalence relationships (since they can produce more connections between the datasets); (b) for constructing dedicated global semantics-aware indexes that cover the whole content of datasets; and (c) for measuring the connectivity among two or more datasets. Finally, we evaluate the speedup of the proposed approach, while we report comparative results for over two billion triples.

Keywords: content-based connectivity measurements; semantic web; linked data; dataset discovery; information enrichment; LOD scale analytics; lattice of measurements; MapReduce; big data

1. Introduction

The main objective of Linked Data is linking and integration, and a major step for evaluating whether this target has been reached, is to find all the connections among the Linked Open Data (LOD) Cloud datasets. *Connectivity* among two or more datasets can be achieved through *common Entities*, *Triples*, *Literals*, and *Schema Elements*, while more connections can occur due to equivalence relationships between URIs, such as `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass`, since many publishers use such equivalence relationships, for declaring that their URIs are equivalent with URIs of other datasets. However, there are not available connectivity measurements (and indexes) involving more than two datasets, that cover the whole content (e.g., entities, schema, triples) or "slices" (e.g., triples for a specific entity) of datasets, although it is important to obtain information from many datasets for several real world tasks, i.e., (a) for enabling *Object Coreference* and *Information Enrichment*, i.e., finding all the available URIs, triples, properties for an entity, e.g., "Aristotle", w.r.t the provenance of data; (b) for assessing and improving the *Veracity* of data or/and for *Fact Checking* [1]; (c) for creating features for a set of entities for being used in a *Machine Learning* problem [2] (e.g., in word2vec algorithms [3]); (d) for offering advanced *Dataset Discovery* and *Selection* services; [4,5] and (e) for

improving *Schema* and *Instance Matching* techniques [6,7]. However, it is not an easy task to find the connections among the datasets, since there exists a big number of LOD datasets [8], the transitive and symmetric closure of equivalence relationships should be computed for not missing connections, and the possible combinations of datasets is exponential in number.

For tackling this challenge, in this paper we focus on answering the following questions: (a) "why plain SPARQL [9] is not enough for performing such measurements?"; (b) "how to compute the transitive and symmetric closure for producing the classes of equivalence for URIs, in instance and schema level?"; (c) "how to use the produced catalogs containing the classes of equivalence for constructing semantics-aware indexes?"; and (d) "how to use any inverted index containing a posting list of dataset identifiers, for performing connectivity metrics for any subset of datasets, by focusing also on specific dataset "slices" (e.g., connectivity of triples, literals, triples referring to "Aristotle", etc.)?". Regarding (d), it is possible that different users desire to measure the connectivity of different "slices" of datasets. For instance, a publisher would like to find the common entities of their dataset with other datasets, in order to enrich the information of their entities, a user would desire to find datasets containing common triples, but only for a specific entity, e.g., "Aristotle", for building a more accurate dataset for this entity. Concerning our contribution, we extend the set of indexes and algorithms that we have proposed in the past [4,5]. In particular, we introduced indexes and their construction algorithms (for a single machine and for a cluster of machines by using MapReduce [10] techniques), for the URIs (but not for schema elements) and literals, and we measured the commonalities of them for 1.8 billion triples, by using "lattice"-based algorithms that can compute efficiently the intersection (e.g., of URIs and literals) among any set of datasets. With respect to that work, in this paper:

- we show why plain SPARQL is not enough for performing such measurements,
- we show how to produce catalogs that contain the computation of transitive and symmetric closure for `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass` relationships (in [4,5] we covered only `owl:sameAs`),
- we exploit the aforementioned catalogs, for constructing in parallel (by using MapReduce [10] techniques) semantically enriched indexes for entities, classes, properties and literals, and an entity-based triples index, for enabling the assessment of connectivity of a specific entity, and immediate access to the available information for that entity,
- we show how the measurements, proposed in [4,5], can be used for performing fast connectivity measurements, by using any inverted index containing a posting list of dataset identifiers,
- we measure the speedup and scalability obtained by the proposed indexes and algorithms, and we introduce indicative statistics and measurements, by using 400 datasets containing over 2 billion triples (200 million more triples comparing to [4,5]) and a cluster of machines.

A research prototype, called *LODsyndesis* (http://www.ics.forth.gr/isl/LODsyndesis/), exploits the indexes and measurements, that are introduced in this paper, for offering services for several real world tasks, such as *Object Coreference*, *Finding all the available Information about an Entity*, *Dataset Discovery* and others. Moreover, as a product, to the best of our knowledge, the indexes of *LODsyndesis* constitute the biggest knowledge graph of LOD that is complete with respect to the inferable equivalence relations.

The rest of this paper is organized as follows: Section 2 introduces the background and discusses related work, Section 3 shows the requirements and states the problem. Section 4 introduces the limitations of SPARQL for the proposed problem, Section 5 describes ways for computing the transitive and symmetric closure of equivalence relationships, and introduces the semantically enriched indexes (and their construction algorithms). Section 6 shows how to measure the connectivity of any measurement type and any subset of datasets, while Section 7 reports the results of the evaluation and discusses the efficiency of the proposed approach. Finally, Section 8 concludes the paper and outlines directions for future work.

2. Background and Related Work

2.1. Background

First, we introduce *RDF* and *Linked Data*, and secondly we describe the *MapReduce* Framework, since we shall use *MapReduce* [10] techniques to parallelize (and thus speedup) the construction of our indexes.

2.1.1. RDF and Linked Data

Resource Description Framework (RDF) [11] is a graph-based data model. It uses Internationalized Resource Identifiers (IRIs), or anonymous resources (blank nodes) for denoting resources, and constants (Literals), while triples are used for relating a resource with other resources or constants. Hereafter, we shall use the term URIs (Uniform Resource Identifiers) to refer to IRIs (since the term URI is more commonly used). A triple is a statement of the form subject-predicate-object ⟨s,p,o⟩, and it is any element of $T = (U \cup B_n) \times (U) \times (U \cup B_n \cup L)$, where U, B_n and L denote the sets of URIs, blank nodes and literals, respectively, whereas an RDF graph (or dataset) is any finite subset of T. For instance, the triple ⟨d1:Aristotle, d1:birthPlace, d1:Stagira⟩, contains three URIs, where the first one (i.e., d1:Aristotle) is the subject, the second one (i.e., d1:birthPlace) is the predicate (or property) and the last one (i.e., d1:Stagira) is the object. Moreover, we can divide the URIs in three categories, i.e., $U = E \cup P \cup C$, where E refers to the entities, P to the properties and C to the classes, where these sets of URIs are pairwise disjoint. In particular, a property describes a relation between a subject and an object and the set of properties P is defined as $P = \{p \mid \langle s, p, o\rangle \in T\}$. Concerning the classes, they are used for grouping entities into classes, e.g., Humans, Philosophers, Actors, and so forth. They can be found through triples of the form ⟨s,rdf:type,c⟩, where s is an entity and c is a class, i.e., $C = \{c \mid \langle s, rdf{:}type, c\rangle \in T\}$. Finally, the remaining URIs are defined as entities, i.e., $E = U \setminus (P \cup C)$. By using Linked Data, the linking of datasets can be achieved by the existence of common URIs or Literals, or by defining equivalence relationships, e.g., `owl:sameAs`, among different URIs (entities, properties and classes).

2.2. MapReduce Framework

MapReduce [10] is a distributed computation framework, that can be used for processing big data in parallel and it is widely applied to large scale data-intensive computing. The *MapReduce* program (or job) is carried out in two different tasks (or phases), called Map and Reduce, which are two different functions that are user-defined, and they can become tasks, which can be executed in parallel. The Map function receives as input a set of data, it performs a user-defined task over an arbitrary part of the input, it partitions the data and converts them into intermediate key-value pairs. Afterwards, the key-value pairs of the map phase are grouped by their key and are passed to the reducers. Concerning the reduce phase, a reduce function (a user defined function) is called for each unique key. The reducer processes the list of values for a specific key and produces a new set of output key-value pairs.

2.3. Related Work

Here, we first describe approaches that perform measurements at LOD scale (in Section 2.3.1), secondly we introduce services for hundreds (or even thousands) of LOD datasets (in Section 2.3.2), and finally, (in Section 2.3.3) we discuss approaches for indexing RDF datasets in parallel.

2.3.1. Measurements at LOD Scale

Recently, there have been proposed approaches for offering measurements for hundreds or thousands of datasets. The authors of [12] have collected and cleaned over 650,000 documents and they focus on providing measurements about the validity of documents, their triples number,

their format and so forth, through the *LODLaundromat* service. LOD-a-Lot [13] has integrated the previous set of documents into a single file, and the authors introduced some measurements, such as the degree distribution of specific elements (subjects, properties, etc.). In [14] one can find the cardinality of mappings between pairs of 476 datasets, while in [15] the authors introduced measurements such as the degree distribution of datasets, the datasets with the most in-coming and out-coming links, and others. *LODStats* [16] offers several metadata and statistics for a big number of datasets, such as links between pairs of datasets, the number of triples of each dataset and so forth. In another approach [17], the authors computed the PageRank for 319 datasets for showing the popularity of specific datasets and the degree up to which other datasets trust a specific dataset. In [18], the authors selected 27 quality metrics for assessing the quality of 130 datasets, by using Luzzu framework [19]. Regarding connectivity, the authors introduced a metric called "Links to External Linked Data Providers", which is used for identifying the number of external links which can be found in a specific dataset. Comparing to our work, the aforementioned approaches do not take into account the closure of equivalence relationships for producing measurements. On the contrary, we compute the closure of such relationships and we offer connectivity measurements for any set of datasets (not only for pairs of datasets) and for the whole content of datasets (e.g., entities, triples, literals, etc.).

2.3.2. Global RDF Cross-Dataset Services

Here, we introduce 16 RDF services/tools for large number of datasets, and we categorize them according to the services that they offer (see Table 1), while we compare all these services with *LODsyndesis*, which is the web page that exploits the measurements which are introduced in this paper. Table 1 is an extended version of a Table introduced in [5], i.e., here, we cover 6 more RDF cross-dataset services.

Table 1. Global RDF cross-dataset services (last accessed date: 3 May 2018).

Tool	Number of RDF Datasets	URI Lookup	Keyword Search	Connectivity	Dataset Discovery	Dataset Visualization	Dataset Querying	Dataset Evolution
LODsyndesis [4,5,20]	400	✓		✓	✓	✓		
LODLaundromat [12]	>650,000 (documents)	✓	✓		✓			
LOD-a-Lot [13]	>650,000 (documents)						✓	
LODStats [8,16]	9960	✓(Schema)		✓	✓			
LODCache	346	✓		✓(via SPARQL)			✓	
LOV [21]	637 (vocabularies)	✓(Schema)	✓				✓	
WIMU [22]	>650,000 (documents)	✓						
Loupe [23]	35	✓						
sameAs.org [24]	>100	✓						
Datahub.io	1270		✓		✓			
LinkLion [14]	476			✓	✓			
DyLDO [25,26]	86,696 (documents)							✓
LODCloud [15]	1184			✓	✓	✓		
Linghub [27]	272		✓		✓		✓	
SPARQLES [28]	557				✓			✓
SpEnD [29]	1487				✓			✓

LODsyndesis [4,5,20] (http://www.ics.forth.gr/isl/LODsyndesis) is a web page that provides query services and measurements for 400 real RDF datasets. It offers connectivity measurements that can be exploited for several tasks. In particular, *LODsyndesis* offers several services, such as an *Object Coreference* and *Finding all the available Information about an Entity* service, where one can find all the datasets, URIs, and triples of a specific entity, a *Dataset Discovery* service, where one can discover all the connected datasets for a given dataset, a *Fact Checking* service (i.e., for checking which datasets agree that a fact holds for a specific entity) and others. Moreover, the proposed measurements are exploited for producing informative 3D visualizations (http://www.ics.forth.gr/isl/3DLod) [30].

LODLaundromat [12] (http://lodlaundromat.org) is a service that fetches several datasets and transforms them to a common format. By using this service, one can search for specific URIs, while it provides a keyword search engine (called Lotus [31]), where one can find all the triples and URIs where a specific keyword occurs. Comparing to our approach, we take into consideration the equivalence relationships between the datasets, therefore, one can find all the equivalent URIs for a given one, while we offer connectivity measurements for any set of datasets.

LOD-a-Lot [13] (http://lod-a-lot.lod.labs.vu.nl/) has integrated the set of *LODLaundromat* documents in a single self-indexed file in HDT format [32]. This integrated dataset can be used for query resolution at Web scale. The key difference with our approach is that we index the datasets mainly for performing connectivity measurements, while we take into account equivalence relationships (and their closure) among the different datasets.

LODStats [8,16] (http://stats.lod2.eu/) is a service including some basic metadata for a large number of datasets. For example, one can find the number of datasets' triples, or the languages that each dataset uses, while one can find all the datasets for specific schema elements (i.e., properties and classes). Comparing to *LODsyndesis*, we take into consideration the closure for finding equivalent URIs for entities and schema elements, while we provide measurements concerning the connectivity among two or more datasets.

LODCache (http://lod.openlinksw.com) is a SPARQL endpoint service, based on Virtuoso database engine [33], that includes several datasets and billions of triples. By using this service, one can send queries that can be answered from two or more datasets, while one can type queries for measuring the connectivity among several datasets. However, in that case the closure of equivalence relations is performed on query time which can be time-consuming (http://docs.openlinksw.com/virtuoso/rdfsameas/), while plain SPARQL is not efficient enough for such measurements (more details are given in Section 4). On the contrary, *LODsyndesis* computes the transitive and symmetric closure once, while we use "lattice"-based algorithms for measuring fast the connectivity among several datasets.

WIMU [22] (http://w3id.org/where-is-my-uri/) is a service that uses the datasets of *LODLaundromat* and *LODstats*, and one can search for a specific URI. This service returns a ranked list of documents, where a particular URI occurs. Moreover, one can download the documents containing that URI, or/and the triples for a URI from a specific document. Comparing to our approach, *WIMU* does not take into account the equivalence relationships, therefore, it is not easy to find all the information for a specific real world entity.

LOV (Linked Open Vocabularies) [21] (http://lov.okfn.org) is a service containing over 600 vocabularies from different datasets. It offers a keyword search and a SPARQL endpoint, which contains schema triples and it can be used in order to search for schema elements. On the contrary, we provide services for the whole content of datasets (not only for schema elements).

Loupe [23] (http://loupe.linkeddata.es) is a service containing data summaries, and it can be used for understanding which vocabularies each dataset uses. Moreover, one can search for datasets where a specific property or class occurs. Comparing to *LODsyndesis*, it focus on schema elements and it does not take into account equivalence relationships.

SameAs.org [24] (http://sameas.org) is a URI lookup service containing over 203 million URIs, where one can find all the equivalent URIs for a given URI. Comparing to us, except for URIs, we find also the equivalent schema elements and triples, while we provide connectivity measurements and services (based on that measurements).

Datahub.io (http://datahub.io) is a portal that provides several datasets in different formats, and one can find some major statistics for each dataset (e.g., number of triples, number of links to other datasets). Moreover, one can fetch datasets provided in dumps and get updates from datasets. We have fetched hundreds of datasets from this portal, while we exploit *datahub.io* for uploading the results of the connectivity measurements.

LinkLion [14] (http://linklion.org/portal) contains dumps of mappings between pairs of datasets and statistics about them. One can find all the mappings between pairs of datasets, however, it is not possible to find mappings between three or more datasets.

DyLDO [25] (http://km.aifb.kit.edu/projects/dyldo/) is a Dynamic Linked Data Observatory whose target is to monitor the evolution of Linked Datasets over time. On the contrary, we mainly provide connectivity measurements about hundreds of LOD Datasets.

LOD Cloud [15] provides statistics about the datasets and the domains where they belong, while one can see all the connections between pairs of datasets through the popular *LOD Cloud Diagram* (http://lod-cloud.net). However, one cannot find connections among three or more datasets (which are provided through *LODsyndesis*).

Linghub [27] (http://linghub.lider-project.eu) is a portal that collects thousands of datasets from linguistic domain (272 of them in RDF format), and uses the Linked Data Principles for displaying the metadata of datasets. This portal offers a faceted browsing service, a keyword search service and a SPARQL endpoint, for easing the process of discovering datasets according to different users' requirements. Comparing to *LODsyndesis*, it mainly focus on providing services for linguistics domain, while we offer services for datasets from nine different domains.

SPARQLES [28] (http://sparqles.ai.wu.ac.at) and *SpEnD* [29] (http://wis.etu.edu.tr/spend/) are services containing metadata and statistics for hundreds of SPARQL endpoints. Comparing to *LODsyndesis*, we mainly focus on providing measurements and services about hundreds of LOD datasets (and not for SPARQL endpoints).

2.3.3. Indexes for RDF Datasets by using Parallel Frameworks

There have been proposed approaches for indexing RDF datasets by using parallel frameworks (such as *MapReduce*). For instance, tools such as AMADA [34], H2RDF [35], Rya [36] and MAPSIN [37] use three indexes, i.e., for finding fast all the triples for a subject, a predicate, and an object respectively. However, the main focus of such systems (a large list of such systems can be found in [38]), is to create indexes for performing fast parallel SPARQL query answering, while most of them do not take into account the closure of equivalence relationships. On the contrary, we mainly focus on creating parallel indexes, that include the computation of closure of equivalence relationships, for performing fast connectivity measurements among any set of datasets.

3. Requirements and Problem Statement

In this section, we introduce the main requirements, which are essential for constructing semantically enriched indexes (such as the computation of transitive and symmetric closure of equivalence relationships), while we define the problem statement (i.e., we define formally the measurements that we want to perform efficiently). In particular, at first, we show some basic notations (in Section 3.1), secondly, we introduce the major requirements (in Section 3.2), and finally, we state the problem (in Section 3.3).

3.1. Notations

Let $D = \{D_1, D_2, ..., D_n\}$ be a set of datasets. For a subset of datasets $B \subseteq D$, we define as $triples(B) \subseteq T$, all its triples, as L_B all its literals, and as U_B all its URIs. Moreover, for a single dataset D_i, U_{D_i} is the set of its URIs (where E_{D_i}, P_{D_i} and C_{D_i} are the sets of entities, properties and classes, respectively), $triples(D_i)$ is the set of its triples and L_{D_i} is the set of its literals. It is worth mentioning that we ignore triples containing blank nodes, since we do not apply blank node matching techniques for finding common blank nodes, like those proposed in [39].

3.2. Requirements

Since we do not want to miss connections between the datasets, our major requirement is to take into consideration the equivalence relationships between the datasets. We consider the following

equivalences in schema and instance level in a subset of datasets B. Let $Equiv(B, r)$ be all the triples that contain a property $r \in Eqv$, $Eqv = \{$`owl:sameAs`,`owl:equivalentProperty`,`owl:equivalentClass`$\}$, that defines a specific equivalence between two URIs: $Equiv(B, r) = \{(u, u') | (u, \mathrm{r}, u') \in triples(D_i), D_i \in B, r \in Eqv\}$. The `owl:sameAs` property denotes that two URIs refer to the same entity, while the remaining ones denote equivalences among schema elements (i.e., properties and classes). These types of equivalences are transitive, symmetric and reflexive, and our target is to compute their closure, in order not to miss equivalence relationships, while it is worth mentioning that one could take also into account additional equivalence relationships between URIs, such as `skos:exactMatch`. If R denotes a binary relation, consequently, we use $C(R)$ to denote the transitive, symmetric and reflexive closure of R, while $C(Equiv(B, r))$ stands for this type of closure of a relation r in all datasets in B. We can now define the equivalent URIs (considering all datasets in B) of an entity $u \in E_B$, a property $p \in P_B$ and a class $c \in C_B$ as:

$$\begin{aligned} EqEnt(u, B) &= \{\, u' \mid (u, u') \in C(Equiv(B, \texttt{owl:sameAs})), u, u' \in E_B\} \\ EqProp(p, B) &= \{\, p' \mid (p, p') \in C(Equiv(B, \texttt{owl:equivalentProperty})), p, p' \in P_B\} \\ EqClass(c, B) &= \{\, c' \mid (c, c') \in C(Equiv(B, \texttt{owl:equivalentClass})), c, c' \in C_B\} \end{aligned}$$

From URIs to Real World Objects. For taking into account the equivalences, our target is to replace any URI with its corresponding class of equivalence, where the URIs that are "semantically" the same belong to the same class of equivalence, e.g., all the URIs referring to "Aristotle" belong to the same equivalence class. We denote as $[u]_e$, $[u]_{pr}$ and $[u]_{cl}$ the class of equivalence of a URI u, if it belongs to entities, properties or classes, respectively. In particular, for a dataset B, we define as $rwo(B) = \cup_{u \in E_B}[u]_e$, its real world entities, where $\forall u' \in EqEnt(u, B), [u]_e = [u']_e$, as $rwp(B) = \cup_{u \in P_B}[u]_{pr}$, its real world properties (where $\forall u' \in EqProp(u, B), [u]_{pr} = [u']_{pr}$), and as $rwc(B) = \cup_{u \in C_B}[u]$, its real world classes (where $\forall u' \in EqClass(c, B), [u]_{cl} = [u']_{cl}$).

Literals Conversion. For the literals, we define as $L' = \cup_{l \in L} l_{conv}$, the set of transformed literals, where we convert each literal l to lower case, we remove its language tag (e.g., "Aristotle"@en $\rightarrow$ "aristotle"), while we remove its datatype (e.g., 1^^xsd:integer $\rightarrow$ "1"). We perform these conversions for not missing connections, e.g., for the same literal, one dataset can use capital letters, another one lowercase letters, a third one both capital and lowercase letters (e.g., "ARISTOTLE" vs. "aristotle" vs. "Aristotle"), a literal can be the same in different languages (e.g., "Aristotle"@en, "Aristotle"@ca), while the same literal can be represented from different datasets with different types (e.g., 1^^xsd:integer, 1^^xsd:double).

From Triples to Real World Triples. We define the real world triples for a dataset D_i, by replacing each URI with its corresponding class of equivalence and by converting each literal (in the way that we showed before). A triple $t = \langle s, p, o \rangle$ is replaced with a new triple $\mu(t) = \langle s', p', o' \rangle$, where,

$$s' = \begin{cases} [s]_e, s \in E_{D_i} \end{cases} \quad p' = \begin{cases} [p]_{pr}, p \in P_{D_i} \end{cases} \quad o' = \begin{cases} o_{conv}, o \in L_{D_i} \\ [o]_e, o \in E_{D_i} \\ [o]_{cl}, o \in C_{D_i} \end{cases}$$

As a consequence, the real world triples, for a dataset D_i and for a subset of datasets B, are defined as $rwt(D_i) = \cup_{t \in triples(D_i)}\mu(t)$ and $rwt(B) = \cup_{D_i \in B} rwt(D_i)$, respectively. Moreover, for a specific URI u, $rwt(u, B) = \{\langle s, p, o \rangle \in rwt(B) \mid s = [u]_e \text{ or } o = [u]_e, u \in E_B\}$ is the set of real world triples where u (or an equivalent URI of u) occurs. Finally, we define as $rwt(u, D_i) = \{\langle s, p, o \rangle \in rwt(D_i) \mid s = [u]_e \text{ or } o = [u]_e\}$ the corresponding set of real world triples of a dataset D_i, which contains entity u, (or an equivalent URI of u).

3.3. Problem Statement

Concerning the proposed measurements, let $P(D)$ denote the power set of D, comprising elements each being a subset of D, i.e., a set of datasets, and $B \subseteq D$ be any subset of datasets. Moreover, let F be the measurement types that we focus, i.e., $F = \{RWO, RWP, RWC, RWT, LIT\}$, and let $S \in F$ be a specific measurement type (e.g., RWO). Furthermore, as $f_S(D_i)$ we denote the "corresponding" feature of a dataset D_i (e.g., $f_{RWO}(D_i) = rwo(D_i)$). Our objective is to be able to perform measurements fast for any subset of datasets $B \in P(D)$, for any measurement type $S \in F$. In particular, our target, is for any measurement type $S \in F$ to compute the cardinality of intersection (i.e., commonalities) among any subset of datasets B, i.e., $|cmn(B, S)|$, where

$$cmn(B, S) = \cap_{D_i \in B} f_S(D_i) \tag{1}$$

More specifically, for any subset $B \in P(D)$, we focus on measuring the following ones:

$$\text{\# of Common Real World Entities:} \quad |cmn(B, RWO)| = |\cap_{D_i \in B} \ rwo(D_i)| \tag{2}$$

$$\text{\# of Common Real World Properties:} \quad |cmn(B, RWP)| = |\cap_{D_i \in B} \ rwp(D_i)| \tag{3}$$

$$\text{\# of Common Real World Classes:} \quad |cmn(B, RWC)| = |\cap_{D_i \in B} \ rwc(D_i)| \tag{4}$$

$$\text{\# of Common Real World Triples:} \quad |cmn(B, RWT)| = |\cap_{D_i \in B} \ rwt(D_i)| \tag{5}$$

$$\text{\# of Common Literals:} \quad |cmn(B, LIT)| = |\cap_{D_i \in B} \ L'(D_i)| \tag{6}$$

Finally, since we would like to be able to compute the cardinality of common real world triples for an entity u (e.g., for "Aristotle"), for any subset of datasets B, we measure also the following one:

$$\text{\# of Common Real World Triples of Entity u:} \quad |cmnTriples(B, u)| = |\cap_{D_i \in B} \ rwt(u, D_i)| \tag{7}$$

4. Why Plain SPARQL Is Not Enough

Here, we show how one can exploit SPARQL query language [9] for computing the cardinality of intersection among any subset of datasets for real world entities, properties, classes, triples and literals. Suppose that there exists $|D|$ datasets. At first, one should upload and store in a SPARQL endpoint all the triples of each dataset (a separate graph is needed for each dataset), and upload also all the equivalence relationships. By using a SPARQL query, it is possible for one to write a query for finding the intersection of several subsets of datasets; however, each such a subset can contain exactly k datasets, e.g., $k = 2$ corresponds to pairs of datasets, $k = 3$ to triads of datasets, and so forth. For instance, for computing the measurements for all the pairs and triads of datasets, two different queries are needed. Generally, for $|D|$ datasets, there exists $|D| - 1$ such levels of subsets, where a level k ($2 \leq k \leq |D|$) consists of subsets having exactly k datasets. Alternatively, one can use one SPARQL query for each different subset of datasets, i.e., $2^{|D|}$ queries. Below, we show five SPARQL queries, for computing the cardinality of commonalities among entities, properties, classes, literals, and triples, respectively, which can be executed by using *Openlink Virtuoso* database engine [33].

Common Real World Entities. For computing the number of common real world entities between combinations containing exactly k datasets, one should use the complex query which is introduced in Listing 1, where for D datasets, we need $|D| - 1$ queries, for computing the cardinality of intersection of any possible combination of datasets.

Listing 1: SPARQL query for computing the number of common entities among several datasets.

```
DEFINE input:same-As ''yes''

select ?Di ?Dj ... ?Dn count (distinct ?u) as ?commonEntities
where {
{graph ?Di {{?u ?p ?o} union {?o ?p ?u . filter(?p!=rdf:type)}}
. filter(isURI(?u))}.
{graph ?Dj {{?u ?p2 ?o2} union {?o2 ?p2 ?u .filter(?p2!=rdf:type)}}} .
...
{graph ?Dn {{?u ?pn ?on} union {?on ?pn ?u .filter(?pn!=rdf:type)}}} .
filter(?Di>?Dj && ... && ?Dn-1>?Dn)
}
group by ?Di ?Dj ... ?Dn
```

As we can observe, one should put the command "define input:same-As "yes"" for enabling the computation of closure on query time for `owl:sameAs` relationships. For this query, we need a unique line for each dataset, for finding the union of all its subjects and objects that are URIs (but not classes). Moreover, one should include in the query a filter statement for denoting that each dataset is different with the other ones, otherwise, it will find also the intersection of real world entities for the same dataset D_i, i.e., $rwo(D_i) \cap rwo(D_i)$. We use the $>$ operator instead of the in-equivalence one (i.e., $!=$), for not computing many times the intersection of a specific combination of datasets. For instance, for pairs of datasets, if we put $D_i != D_j$, the query will compute the intersection of all the pairs of datasets twice, i.e., it will compute the intersection for all the possible permutations. For instance, for two datasets D_1 and D_2, it will compute the measurements twice (i.e., in the aforementioned example, there are two possible permutations, D_1, D_2 and D_2, D_1). Finally, a *group by* clause should be used, where each such group corresponds to a subset of datasets.

Common Real World Properties. For computing the cardinality of common properties, one should use the query which is shown in Listing 2, where it is required to compute the closure of `owl:equivalentProperty` relationships on query time. By using *Virtuoso*, one should follow some steps for enabling the computation of closure for schema elements on query time (http://vos.openlinksw.com/owiki/wiki/VOS/VirtSPARQLReasoningTutorial#Step4.:SettingUpInferenceRules). By following the aforementioned steps, for the query of Listing 2, we have defined an inference rule with name "SchemaEquivalence".

Listing 2: SPARQL query for computing the number of common properties among several datasets.

```
DEFINE input:inference ''SchemaEquivalence''

select ?Di ?Dj ... ?Dn count (distinct ?property) as ?commonProperties
where {
{graph ?Di {?s ?property ?o}} .
{graph ?Dj {?s1 ?property ?o1}} .
...
{graph ?Dn {?sn ?property ?on}} .
filter(?Di>?Dj && ... && ?Dn-1>?Dn)
}
group by ?Di ?Dj ... ?Dn
```

Common Real World Classes. For computing the cardinality of common classes, one should use the query which is introduced in Listing 3, where it is required to compute the closure of `owl:equivalentClass` relationships on query time.

Listing 3: SPARQL query for computing the number of common classes among several datasets.

```
DEFINE input:inference ''SchemaEquivalence''

select ?Di ?Dj ... ?Dn count (distinct ?class) as ?commonClasses
where {
{graph ?Di {?s rdf:type ?class}} .
{graph ?Dj {?s1 rdf:type ?class}} .
...
{graph ?Dn {?sn rdf:type ?class}} .
filter(?Di>?Dj && ... && ?Dn-1>?Dn)
}
group by ?Di ?Dj ... ?Dn
```

Common Literals. For computing the cardinality of common Literals between combinations containing exactly k datasets, one should use the query which is shown in Listing 4.

Listing 4: SPARQL query for computing the number of common literals among several datasets.

```
select ?Di ?Dj ... ?Dn count (distinct ?l) as ?commonLiterals where {
{graph ?Di {?s ?p ?l} . filter(isLiteral(?l))} .
{graph ?Dj {?s1 ?p1 ?l}} .
...
{graph ?Dn {?sn ?pn ?l}} .
filter(?Di>?Dj && ... && ?Dn-1>?Dn)
}
group by ?Di ?Dj ... ?Dn
```

Common Real World Triples. For the computation of the number of common real world triples, we need to compute the closure for finding the equivalences for both instance and schema relationships (which is computed on query time). Listing 5 shows the corresponding query for measuring the number of common triples among several datasets.

Listing 5: SPARQL query for computing the number of common triples among several datasets.

```
DEFINE input:inference ''SchemaEquivalence''
DEFINE input:same-As ''yes''

select ?Di ?Dj ... ?Dn count (*) as ?commonTriples where {
{graph ?Di {?s ?p ?o}} .
{graph ?Dj {?s ?p ?o}} .
...
{graph ?Dn {?s ?p ?o}} .
filter(?Di>?Dj && ... && ?Dn-1>?Dn)
}
group by ?Di ?Dj ... ?Dn
```

Comparison of SPARQL approach with the proposed one. The queries above can be exploited for performing such measurements, however, this approach has many limitations, which follow. At first, the computation of closure is performed on query time, which can be time consuming. Therefore, for n queries, the closure will be computed n times. On the contrary, we compute the closure of equivalence relationships once. Second, one (complex) query is needed for each different size of datasets combinations, i.e., in total $|D| - 1$ queries are needed for all the possible combinations of datasets,

while as the number of datasets grow, such a query can be huge. On the other hand, we compute the intersection between any set of datasets by using an incremental algorithm, which requires a single input for computing the intersection of any combination of datasets. Third, several joins (and comparisons) should be performed for finding the intersection for a specific subset of datasets, while the computation of closure (which is computed on query time) increases the number of joins. On the contrary, we compute the closure once, and we exploit the posting lists of indexes (instead of comparing URIs, triples and literals) for reducing the number of comparisons. Finally, since each query computes the intersection for a fixed number of combinations of datasets, i.e., pairs, triads, etc., it is not an easy task to exploit set theory properties, which hold between two or more subsets of datasets, B and B', where $B \subseteq B'$ (more details are given in Section 6). On the contrary, we take into consideration set theory properties for making the measurements in an incremental way. In Section 7, we introduce some indicative experiments containing the execution time for measuring the number of commonalities among datasets by using SPARQL queries and a "lattice"-based approach.

5. Global Semantics-Aware Indexing for the LOD Cloud Datasets

The process of global indexing comprises of four different steps, which can be seen in the running example of Figure 1. The first step is the collection of input datasets (which is performed manually at the time-being), which is a set of datasets' triples and a set of RDF relationships (e.g., `owl:sameAs` relationships). In our running example, the input contains 4 datasets, each one having six triples, and several instance and schema relationships. The next step includes the computation of closure in schema and instance level, where catalogs containing for each URI an identifier are produced. In the third step, we use the initial datasets and the aforementioned catalogs for creating "semantically" enriched triples (i.e., real world triples). In the fourth step, we create semantically enriched inverted indexes for different sets of elements (e.g., triples, entities), where we collect all the information of each different entity, and we store the dataset IDs (i.e., a posting list) where real world entities, properties, classes, triples and literals occur. In this section, in Section 5.1 we show how to partition the different sets of elements, in order to construct the indexes in parallel, in Section 5.2 we describe ways to compute the transitive and symmetric closure of equivalence relationships and we introduce the produced catalogs. Afterwards, in Section 5.3 we show how to use them for creating semantics-aware triples, which are finally exploited for creating semantics-aware indexes, i.e., in Section 5.4, we show how to construct these indexes.

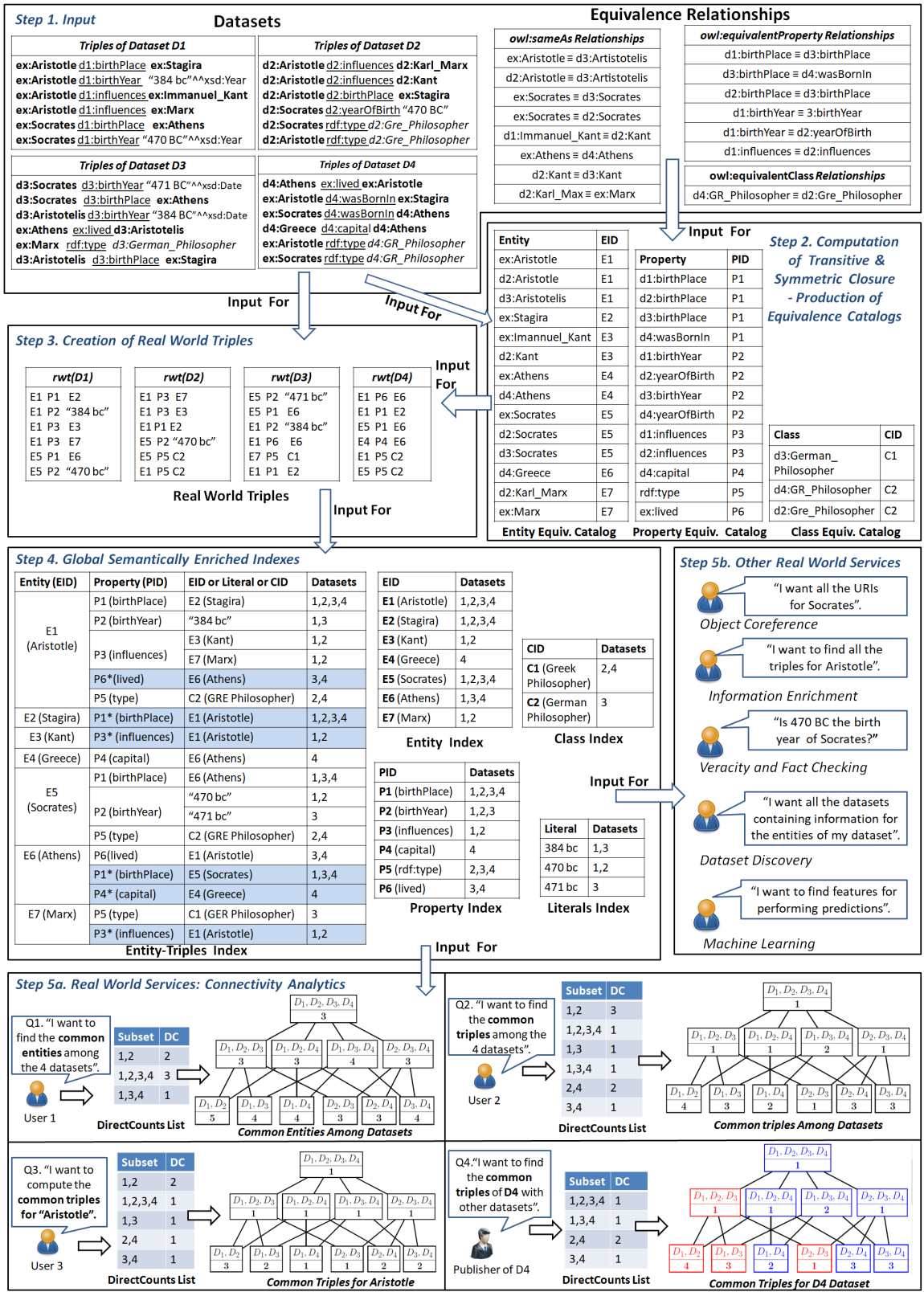


Figure 1. Running example containing four datasets.

5.1. Partitioning the Different Sets of Elements

Here, we define how one can split each set of the described elements (e.g., URIs, triples, equivalence relationships), for running the algorithms in parallel. Let m be the set of available machines. Since the size of each dataset varies [5], we did not select each machine to read all the URIs, triples and literals of a specific dataset. On the contrary, each of the partitions described below can contain URIs, triples or equivalence relationships from one or more datasets. For the computation of closure of equivalence relationships, let $EQ = \{EQ_1, ..., EQ_m\}$ be a partition of the equivalence relationships, i.e., `owl:sameAs`, `owl:equivalentClass` and `owl:equivalentProperty`, while we need also partitions of URIs, where $U = \{UR_i, UR_j, .., UR_m\}$ and $(UR_i \cup UR_j \cup ... \cup UR_m = U)$. For the construction of indexes, we will use a partition of triples $TR = \{tr_1, ..., tr_m\}$, where $tr_1 \cup ... \cup tr_m = triples(D)$, and a partition of real world triples $RWTR = \{rwt_1, ..., rwt_m\}$, where the set of real world triples can be derived as $rwt(D) = rwt_1 \cup ... \cup rwt_m$.

5.2. Equivalence Relationships

Here, we define the equivalence catalogs for each URIs' category, where all the URIs referring to the same thing are getting the same unique identifier (or ID). We create one such catalog for each different URI's category, i.e., properties, classes and entities.

- Entity Equivalence Catalog (`EntEqCat`): For each of the entities $u \in E_B$ we assign a unique ID, where EID denotes this set of identifiers (i.e., a binary relation $\subseteq EID$). This catalog will be exploited for replacing each URI that occur in a triple with an identifier. For constructing this catalog, we read the URIs of each dataset (marked in bold in Figure 1) and the `owl:sameAs` relationships (see the upper right side of Figure 1), and we compute the transitive, symmetric and reflexive closure of `owl:sameAs` relationships, for finding the classes of equivalence. Finally, all the entities belonging to the same class of equivalence will be assigned the same identifier, e.g., see the `EntEqCat` in the running example of Figure 1.
- Property Equivalence Catalog (`PropEqCat`): For each of the properties $p \in P_B$, we store a unique ID, where PID denotes this set of identifiers (i.e., a binary relation $\subseteq PID$). As we shall see, this catalog is used for replacing the property of each triple with an identifier. For constructing it, one should read the properties of each dataset (underlined in Figure 1), the `owl:equivalentProperty` relationships (see the upper right side of Figure 1), and compute the closure of that relationships for producing the classes of equivalence for properties. Afterwards, all the properties belonging to the same class of equivalence are assigned the same identifier, e.g., in Figure 1, we can observe the `PropEqCat` of our running example.
- Class Equivalence Catalog (`ClEqCat`): For any class $c \in C_B$, we store a unique ID, where CID denotes this set of identifiers (i.e., a binary relation $\subseteq CID$). We will exploit this catalog for replacing each class occurring in triples with an identifier. For constructing it, one should read the classes (marked in italics in Figure 1), the `owl:equivalentClass` relationships, and compute their closure for finding the classes of equivalence. Finally, all the classes that refer to the same thing will take the same identifier. The resulted `ClEqCat` for our running example can be seen in Figure 1.

Parallelization Overview. Each machine m_i is assigned the responsibility to read a subset of equivalence pairs EQ_i, and a subset of URIs UR_i, and to compute a partial function eq_i: $U \rightarrow ID$. In the reducer, any equivalence catalog ($EqCat$) can be derived as $EqCat = eq_1 \cup ... \cup eq_m$. Since the sets of entities, properties and classes are pairwise disjoint, a single algorithm can be used for computing the closure of all the equivalence relationships.

Construction method. We exploit a variation of the parallel connected components algorithm, called Hash-to-Min [40], for computing the equivalences between entities, classes and properties, which was proposed in [5]. The first step is to find the direct equivalences (or neighbors) of each URI. For this reason, in each case we first find $nbrs(u, r) = \{u' | u, r, u'\}, r \in Eqv\} \cup \{u\}$.

Consequently, we need a single job that reads each URI and all the relationships in the mapper and produces the set $nbrs(u,r)$ for each u. It is worth mentioning that when $|nbrs(u,r)| = 1$, u is equivalent only to itself, e.g., in the running example of Figure 1, d4:Greece does not belong to an `owl:sameAs` relationship. Therefore, we can immediately assign to this URI a unique identifier. For all the URIs having at least one neighbor (apart from themselves), we use the variation of Hash-to-Min algorithm, which takes as input all the URIs having $|nbrs(u,r)| > 1$ and creates an undirected graph $G = (V, E)$, where V is a set of vertices, i.e., $V = \{u | \langle s,r,o\rangle \in triples(D)|\ u = s$ or $u = o, r \in Eqv\}$, and E a set of edges, which connects nodes (or URIs) that are neighbors (they are connected through an equivalence relationship), i.e., $E = \{s,o|\langle s,r,o\rangle \in triples(D)$, $r \in Eqv\}$. The algorithm computes in parallel (in logarithmic rounds of *MapReduce* jobs) all the connected components (or clusters) of graph G, and the result of this algorithm is a set of such clusters, where each cluster C_u contains all the URIs of the same entity. For instance, in the running example of Figure 1, the cluster of "Aristotle" entity is the following one: $C_{Aristotle} =$ {ex:Aristotle,d2:Aristotle,d3:Aristotle}, while the cluster for "birthPlace" property contains four URIs, i.e., $C_{birthPlace} =$ {d1:birthPlace,d2:birthPlace,d3:birthPlace,d4:wasBornIn}. For all the URIs of the same class of equivalence, it assigns to each of them the same unique identifier, e.g., the identifier of all URIs referring to Aristotle is E1, while P1 is the identifier of all the URIs referring to the property "birthPlace". In [5], one can find all the details about Hash-to-Min algorithm and the variation of the algorithm that we have proposed.

5.3. Creation of Semantics-Aware RDF Triples

The objective of this step is to create semantics-aware triples, which will be exploited for constructing semantically enriched indexes. This step includes the replacement of each URI with its corresponding identifier by exploiting the equivalence catalogs, i.e., `EntEqCat`, `PropEqCat` and `ClEqCat`, and the conversion of each literal (each literal is converted to lowercase, while we remove its language tag and its datatype). The resulted real world triples of each dataset, for our running example, is shown in the left side of Figure 1. There exists three different types of triples according to the type of their object, i.e., (a) triples with a literal as object; (b) triples with a class as object; and (c) triples with an entity as object. As we will see, for the third type of objects, we need an additional *MapReduce* job.

Parallelization Overview. Each machine m_i reads a subset of triples tr_i, and a subset of `EntEqCat`, and in the reducer each triple is replaced with its corresponding semantically-enriched triple. Since some triples need two steps to be transformed (i.e., it depends on object type), two *MapReduce* jobs are needed. After the execution of the aforementioned jobs, each machine will have a specific part of real world triples (rwt_i), i.e., the real world triples can be derived as $rwt(D) = rwt_1 \cup ... \cup rwt_m$.

Construction Method. Algorithm 1 shows the steps for producing the set of real world triples. First, we read in parallel a set of triples and the `EntEqCat`, since their size is huge. On the contrary, since the number of properties and classes is low, we keep in memory the other catalogs, i.e., `PropEqCat` and `ClEqCat`, which contain the identifiers of each property and class, respectively. We check whether the input is a triple or an entry of the `EntEqCat` (see line 3), and then we check the object type. In the first case (see lines 4–5), i.e., the object is a literal, we replace the property with its identifier (by using `PropEqCat`), we convert each literal to lower case, and we remove its language type and its datatype (when such information is included). Afterwards, we put as a key the subject of the triple and as value the rest part of the triple along with the dataset ID (for preserving the provenance). For example, for the triple ⟨ex:Aristotle,d1:birthYear,384 BC$^\wedge\wedge$xsd:Year⟩ of dataset D_1 (see Figure 1), we replace d1:birthYear with $P1$, we convert the literal into lower case, we remove its datatype (e.g., "384 BC"$^\wedge\wedge$xsd:Year → "384 bc"), and finally we emit a tuple (ex:Aristotle, {P2,384 bc,D_1}). In the second case (see lines 6–7), i.e., the object is an RDF class, we use `PropEqCat` and `ClEqCat` for replacing the property and the object with their corresponding identifiers, while in the third case (see lines 8–9), we replace only the property, and we emit a key-value pair, having as a key the subject of the triple, and as a value the rest part of the triple. For instance, for the triple ⟨d2:Aristotle,rdf:type,d2:Gre_Philosopher⟩ we will emit the following

key-value pair: (d2:Aristotle, {P5,C2,D_2}), while for the triple ⟨ex:Aristotle,d1:influences,ex:Marx⟩, the following key-value pair will be sent to the reducers: (ex:Aristotle, {P3,ex:Marx,D_1}). On the contrary, when the input is an entry of `EntEqCat`, we put as a key the URI, and as a value its corresponding class of equivalence, e.g., we create a tuple (ex:Aristotle, E1).

Algorithm 1: Creation of Real World Triples.

Input: All triples and equivalence catalogs, `EntEqCat`, `PropEqCat`, and `ClEqCat`
Output: Real World Triples

```
1  function Mapper (input = tr_i ∪ EntEqCat )
2      forall inp ∈ input do
3          if inp = ⟨s, p, o⟩, D_i ∈ tr_i then
4              if o ∈ L then
5                  emit (s, {[p]_pr, o_conv, D_i}) ;          // PropEqCat used and literal converted
6              else if o ∈ C then
7                  emit (s, {[p]_pr, [o]_cl, D_i}) ;                  // PropEqCat and ClEqCat used
8              else if o ∈ E then
9                  emit (s, {[p]_pr, o, D_i}) ;                                  // PropEqCat used
10         else if inp = (u, [u]_e) ∈ EntEqCat then
11             emit (u, [u]_e)
12
13 function SubjectReducer (URI key, values = list({[p]_pr, o, D_i}), [key]_e)
14     forall v ∈ values do
15         if (o ∉ E) then
16             t' ← ⟨[key]_e, [p]_pr, o⟩
17             store (t', D_i) ;                          // All conversions finished.  t' ∈ rwt(D_i)
18         else
19             emit (o, {[key]_e, [p]_pr, D_i}) ;                    // Object Replacement Needed
20     emit(key, [key]_e)
21
22 function ObjectReducer (URI key, values = list({s', p', D_i}), [key]_e)
23     forall v ∈ values do
24         t' ← ⟨s', p', [key]_e⟩
25         store (t', D_i) ;                          // All conversions finished.  t' ∈ rwt(D_i)
```

In a reduce function (see SubjectReducer in Algorithm 1), we just replace each URI occurring as a subject with its identifier. For instance, the tuple (ex:Aristotle, E1) and all the tuples having as subject ex:Aristotle, e.g., (ex:Aristotle, {P2,384 bc,D_1}), will be sent to the same reducer, therefore we can replace ex:Aristotle with E1, e.g., (E1,P2,384 bc,D_1). After replacing the URIs of the subjects with their corresponding identifier, for the triples containing literals or classes as objects, we can store their corresponding real world triple (and its provenance), i.e., $\langle E1, P1, 384\ bc\rangle, D_1$, since we have finished with all the conversions (see lines 15–17). On the contrary, for the triples containing objects that belong to entities, we should also replace these URIs with their class of equivalence (see lines 18–19). For instance, after the execution of the first *MapReduce* job, the triple ⟨ex:Aristotle,d1:influences,ex:Marx⟩ has been transformed into ⟨$E1$, $P3$,ex:Marx⟩; however, we should also replace the URI ex:Marx with its corresponding identifier. In particular, we put as a key the object and as a value the rest part of the triple and the dataset ID, e.g., (ex:Marx, {E1,P3, D_1}), while we create again a tuple containing each URI and its corresponding identifier, e.g., (ex:Marx, E7) (see lines 19–20 of Algorithm 1). By using an other reduce function (see ObjectReducer in Algorithm 1), we replace the URIs occurring as objects with their identifier, and we store the real world triple (and its provenance), i.e., $\langle E1, P3, E7\rangle, D_1$.

In general, two *MapReduce jobs* are needed, where in the first job we transfer all the triples and all the entries of `EntEqCat` from the mappers to the reducers, i.e., communication cost is $\mathcal{O}(|triples(B)| + |\texttt{EntEqCat}|)$, while in the second job, we transfer only the triples containing an entity as an object

(and the EntEqCat), i.e., communication cost is $\mathcal{O}(|triples'(B)| + |\texttt{EntEqCat}|)$, where $triples'(B) = \{\langle s,p,o\rangle \mid \langle s,p,o\rangle \in triples(B), o \in E\}$ (all triples having an entity as an object).

5.4. Constructing Semantics-Aware Indexes

Here, we define the five semantically enriched indexes that we construct. For each index we store the IDs of datasets, where a specific element (i.e., an entity, a class, a property, a literal or a triple) occurs. In Table 2, we define the set of datasets where a specific element (or an equivalent one) occurs, since we will use these notations later in this section. For instance, for the URI d1:Aristotle, $dsetsEnt_{\sim}$(d1:Aristotle, B) returns all the datasets where d1:Aristotle (or any equivalent URI of d1:Aristotle) occurs. For creating all these indexes, we use the desired information from the semantically enriched triples (i.e., real world triples). At first, we introduce the *Entity-Triples Index*, and afterwards we show how to construct indexes for specific sets, i.e., literals, entities, classes and properties. In general, with $Left(r)$ we will denote the set of elements that occur in the left side of a binary relation. Finally, the sets of EID, PID and CID denote the identifiers of real world entities, properties and classes, respectively.

Table 2. Notations for finding the provenance of different elements.

Element	Datasets Where an Element Occurs
Entity u	$dsetsEnt_{\sim}(u,B) = \{D_i \in B \mid EqEnt(u,B) \cap E_{D_i} \neq \emptyset\}$
Property p	$dsetsProp_{\sim}(p,B) = \{D_i \in B \mid EqProp(p,B) \cap P_{D_i} \neq \emptyset\}$
Class c	$dsetsClass_{\sim}(c,B) = \{D_i \in B \mid EqClass(c,B) \cap C_{D_i} \neq \emptyset\}$
Literal l	$dsetsLit(l_{conv},B) = \{D_i \in B \mid l_{conv} \in L'_{D_i}\}$
Triple t	$dsetsTr_{\sim}(t,B) = \{D_i \in B \mid \mu(t) \in rwt(D_i)\}$

5.4.1. Entity-Triples Index

For each real world entity we create a multi-level index. This index stores in the first level an entity identifier (i.e., belonging in EID) and a pointer to a list of its (real world) properties (a set of PIDs), whereas in the second level each property points to a list of values. Finally, in the third level, it stores for a specific entity-property pair, all its values and the dataset IDs (i.e., a posting list) where each different triple (entity-property-value) occurs. In the left side of Figure 1, we can see the *Entity-Triples* index of our running example. We selected to store together all the values of a property for a given entity, for enabling the comparison of the values of each property, e.g., in Figure 1, we can see that two datasets support that the birth date of "Socrates" is "470 bc" and one that is "471 bc". Such a design can be useful for data fusion algorithms, since most of them compare the conflicting values of a given subject and predicate (e.g., the birth date of a person) for deciding which value is the correct one [1].

It is worth noting that for finding fast all the available information for a specific entity, this index stores together all the triples for a specific entity, either if that entity occurs in a triple as a subject, or as an object, which means that some triples will be stored twice in the index, i.e., those having entities as objects. Moreover, we also store the position of an entity in the triple, Specifically, when an entity occurs in a triple as an object (see the entry for "Stagira" in *Entity-Triples* index of Figure 1), we add an $*$ in the property ID of such a real world triple. Consequently, we define this set of property IDs as $PID*$. It is essential to store such triples twice, for being able to answer queries like "Give me the number of common triples for Aristotle" (i.e., for taking also into consideration triples where the entity "Aristotle" occurs as an object). On the contrary, if our target is only to find all the common triples among any set of datasets, there is no need to store such triples twice.

Therefore, it is a function $eti : EID \rightarrow list((PID \cup PID*) \rightarrow list((EID \cup CID \cup L) \rightarrow P(D)))$, i.e., for a specific $t = \langle s,p,o\rangle \in triples(B)$, $eti([s].[p].[o]) = dsetsTr_{\sim}(t,B)$. We use $eti(e)$ for denoting the entry of a specific entity e, e.g., in Figure 1, $eti(E1)$ is the entry for "Aristotle", we use $eti(e.p)$ for denoting the sub-entry containing all the values for a specific entity-property pair, e.g., $eti(E1.P3)$, is the sub-entry for the combination "Aristotle", "influences", while $eti(e.p.o)$ denotes the sub-entry

that contain all the dataset IDs for a specific combination of an entity-property-value, e.g., $eti(E1.P3.E3)$ is the sub-entry for the combination"Aristotle", "influences", "Kant".

Parallelization Overview. Each machine m_i reads a subset of real world triples rwt_i and each reducer collects all the real world triples for a subset of entities E'. After the execution of the aforementioned jobs, each machine will have a specific part of this index (eti_i), i.e., the entity-triples index can be derived as $eti = eti_1 \cup ... \cup eti_m$.

Construction Method. Algorithm 2 describes the steps for producing this index. In the map phase, we read a set of real world triples and we emit a key-value pair consisting of the entity occurring as subject and the rest part of the triple (with its provenance) as value (see lines 3–4). For example, for the real world triple ⟨E1,P2,384 bc⟩, D_1 (which corresponds to "Aristotle", "birth year", "384 bc"), it will emit a key-value pair having as key $E1$ and as value {P2,384 bc, D_1}. On the contrary, for the triples having entities as objects (and not literals or classes), we create two key-value pairs, one having as key the subject and one having as key the object (see lines 3–6). Moreover, for the second key-value pair (lines 5–6), we add also a "*" character in the right side of the property ID, for denoting that the entity occurs in the triple as an object. For instance, for the real world triple ⟨E1,P3,E7⟩, D_1 (which corresponds to "Aristotle", "influences", "Marx") two key-value pairs will be created, i.e., one having as key the subject, which refers to "Aristotle" (E1,{P3,E7,D_1}), and one having as key the object, that refers to "Marx" (E7,{P3*,E1,D_1}).

Algorithm 2: Construction of *Entity-Triples* Index.

Input: Real World Triples
Output: *Entity-Triples* Index

```
1  function Entity-Triples Index-Mapper (input = rwt_i)
2      forall ⟨s, p, o⟩, D_i ∈ rwt_i do
3          if s ∈ EID then
4              emit (s, {p, o, D_i})
5          if o ∈ EID then
6              emit (o, {p*, s, D_i})
7
8  function Entity-Triples Index-Reducer (Entity e, values = list({p, k, D_i})
9      eti(e) → ∅
10     forall v = p, k, D_i ∈ values do
11         if (p ∉ Left(eti(e))) then
12             eti(e) → eti(e) ∪ {p, {{k, {i}}}}
13         else
14             if (k ∈ Left(eti(e.p))) then
15                 eti(e.p.k) → eti(e.p.k) ∪ {i}
16             else
17                 eti(e.p) → eti(e.p) ∪ {k, {i}}
18     store eti(e)
```

In the reduce phase (of Algorithm 2), we collect all the real world triples for a specific entity e, i.e., we construct the whole entry of e in the *Entity-Triples* index. We iterate over all the values, where each value contains a property p, the corresponding value k for that property and a dataset ID where the aforementioned information occurs. We first check (see line 11) whether p exists in the sub-entries of entity e. When it is false (see line 12), we create a sub-entry, i.e., $eti(e.p.k)$, and we add the dataset ID where this combination, i.e., $e.p.k$, occurs. For instance, suppose that we first read the $\{P3, E3, D_1\}$ for entity $E1$. In such a case, we just create an entry where $eti(E1.P3.E3) = \{1\}$. When the property p exists in the sub-entry of e (see lines 13–17), we check whether the value k also exists in the sub-entry of $eti(e.p)$. In such a case (see lines 14–15), we just update the posting list of $eti(e.p.k)$. Suppose that for the entity $E1$, we read the following value: $\{P3, E3, D_2\}$. Since the entry $E1.P3.E3$

already exists, we just update its posting list, i.e., $eti(E1.P3.E3) = \{1, 2\}$. Otherwise, we create a new sub-entry for $eti(e.p)$, where we add the value k and its provenance (see lines 16–17). For instance, if we read the value $\{P3, E7, D_2\}$, we have already created an entry for $E1.P3$; however it does not contain $E7$. For this reason, we create a new entry for storing also this information, i.e., $eti(E1.P3)$={{E3,{1,2}}, {E7,{2}}}. Finally, we can store the entry of a specific entity on disk (see line 18). It is worth noting that for finding the common triples among different datasets, there is no need to store triples that occur only in one dataset. On the contrary, for reusing the index for other real world tasks, e.g., for finding all the triples for an entity, for checking conflicting values, etc., all the triples should be stored.

Generally, one *MapReduce* job is needed, while all the real world triples are passed from the mappers to the reducers at least once, whereas the set of triples $rwt'(B) = \{\langle s, p, o\rangle | \langle s, p, o\rangle \in rwt(B)$ and $o \in EID\}$, which contains an entity in the object position, is transferred twice. Consequently, the communication cost is $\mathcal{O}(|rwt(B)| + |rwt'(B)|)$.

5.4.2. Semantically Enriched Indexes for Specific Sets of Elements

Here, we introduce the indexes for entities, properties, classes and literals.

- *Entity Index*: it is a function $ei : EID \rightarrow P(D)$, where for a $[u] \in EID$, $ei([u]) = dsetsEnt_{\sim}(u, D)$, i.e., for each different real world entity, this index stores all the datasets where it occurs (see the *Entity Index* in Figure 1).
- *Property Index*: it is a function $pi : PID \rightarrow P(D)$, where for a $[p] \in PID$, $pi([p]) = dsetsProp_{\sim}(p, D)$, i.e., it stores all the datasets where each different real world property occurs (see the *Property Index* in Figure 1).
- *Class Index*: it is a function $ci : CID \rightarrow P(D)$, where for a $[c] \in CID$, $ci([c]) = dsetsClass_{\sim}(c, D)$, i.e., it stores the datasets where a real world class occurs (see the *Class Index* in Figure 1).
- *Literals Index*: it is a function $li : L' \rightarrow P(D)$, where for a $l_{conv} \in L'$, $li(l_{conv}) = dsetsLit(l_{conv}, D)$, i.e., it stores all the datasets where a converted literal occurs (see the *Literals Index* in Figure 1).

Parallelization Overview. Each machine m_i reads a subset of real world triples rwt_i and each reducer creates the index of a subset of elements. In the end, each machine will have a specific part of an inverted index (ind_i), i.e., any inverted index ind can be derived as $ind = ind_1 \cup ... \cup ind_m$.

Construction Method. For constructing the index of each different set of elements (entities, properties, etc.), one can use the algorithm described in Algorithm 3. In particular, we read all the real world triples and we select the part(s) of the triple that we are interested in (see lines 3–16), e.g., for constructing the class index, we need only the objects referring to a real world class. In any case, we emit a key-value pair, having as key an element (e.g., a literal, a real world entity) and as value the dataset where the triple, containing this element, occurs. In the reducer, for any (inverted) index, we just create a posting list with dataset IDs for a specific element. In particular, we read a set of dataset IDs, and we just update the posting list of each element (see line 21). Finally, we store in the corresponding index the element and its posting list (see line 22). For finding the common elements (e.g., entities, literals, etc.), we can just store only elements occurring in at least two datasets. On the other hand, by storing all the elements, one can use each index for different tasks, e.g., for building a global URI lookup service. For each index, one *MapReduce* job is needed; however, it is worth mentioning that all the five indexes (or any subset of them) can be constructed simultaneously by using one *MapReduce* job, since the input in any case is the real world triples. Regarding the communication cost, it depends on the inverted index that we construct each time.

Algorithm 3: Creation of a Semantically-Enriched Inverted Index for any set of elements.

Input: Real World Triples
Output: An inverted index for a set of specific elements

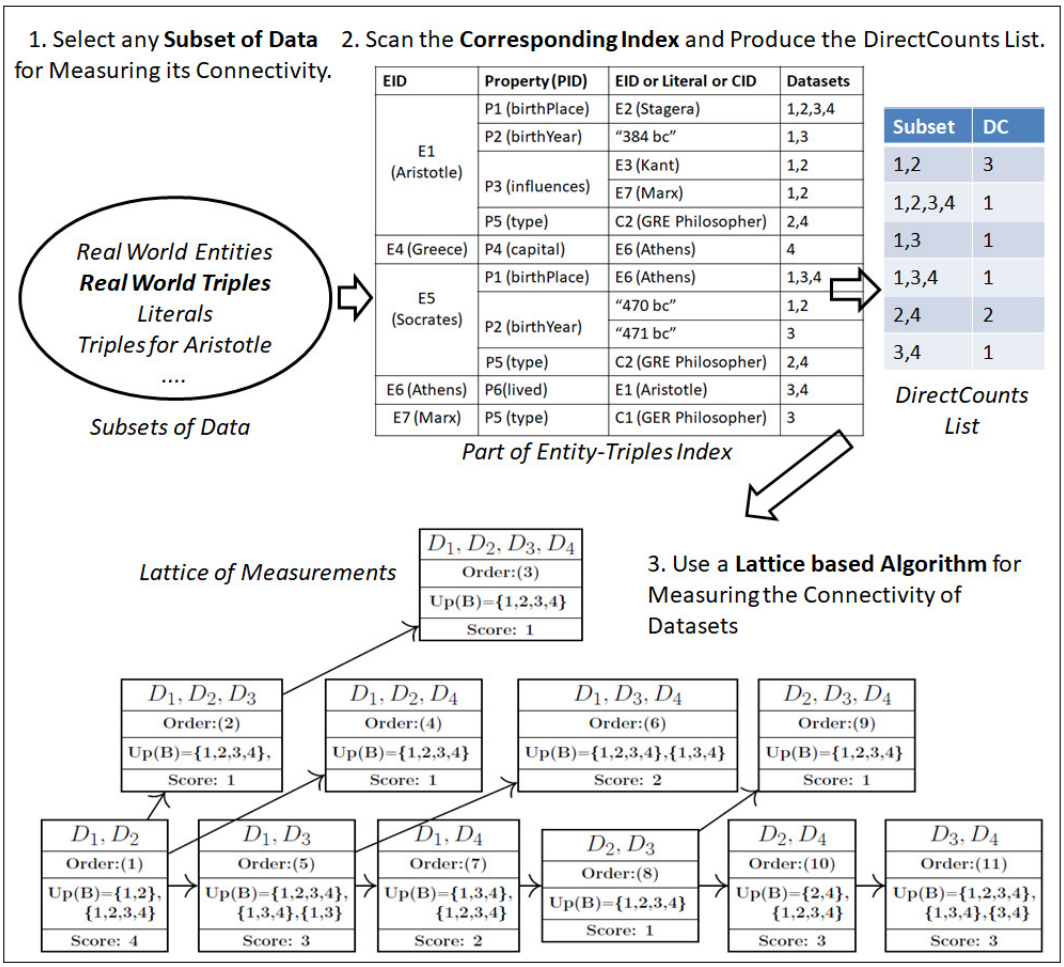
```
1  function Inverted Index-Mapper (input = rwt_i)
2      forall ⟨s, p, o⟩, D_i ∈ rwt_i do
3          /*For constructing the Entity Index, include lines 4-7 */
4          if s ∈ EID then
5              emit (s, D_i)
6          if o ∈ EID then
7              emit (o, D_i)
8          /*For constructing the Property Index, include lines 9-10*/
9          if p ∈ PID then
10             emit (p, D_i)
11         /*For constructing the Class Index, include lines 12-13*/
12         if o ∈ CID then
13             emit (o, D_i)
14         /*For constructing the Literals Index, include lines 15-16*/
15         if o ∈ L' then
16             emit (o, D_i)
17
18 function Inverted Index-Reducer (Element t, values = {D_i, D_j, ..., D_n})
19     invIndex(t) ← ∅
20     forall D_i ∈ values do
21         invIndex(t) ← invIndex(t) ∪ {i}
22     store invIndex(t)
```

6. Lattice-Based Connectivity Measurements for any Measurement Type

The "lattice"-based measurements were first introduced in [4,5]; however, we covered only measurements about the entities and the literals. A lattice (examples are shown in Figure 1) is a partially ordered set, $(P(D), \subseteq)$ and is represented as a Directed Acyclic Graph $G = (V, E)$. It contains $2^{|D|}$ nodes, where each node corresponds to a subset of datasets, and $|D|$+1 levels, where each level k ($0 \leq k \leq |D|$) consists of subsets having exactly k datasets, e.g., level 3 contains only triads of dataset. Each edge connects two subsets of datasets B and B', where $B \subset B'$ and $B' = B \cup D_k, D_k \notin B$, i.e., B' contains the same datasets as B plus a new dataset D_k. Below, we describe the different steps that should be followed for performing such measurements. As we can see in Figure 2, the first step is to select any measurement type S (e.g., real world triples), that one is interested in. Afterwards, we should scan the desired part of the corresponding index (or the whole index) and compute how many times a subset B (containing at least two datasets) occurs in all the posting lists (containing dataset IDs), and we define it as $directCount(B, S)$, i.e., it denotes its frequency. As we will see in Section 7, the number of entries of such a list (for any measurement type S) is very small comparing to the number of the entries of each index. In Table 3, we can observe how to find $directCount(B, S)$ for any measurement type S, by scanning one or more indexes, which are essential for measuring the connectivity of the desired measurement types (which were introduced in Section 3).

Table 3. DirectCounts of different measurement types.

Formula	Which $directCount(B,S)$ to Use.
$\|cmn(B,RWO)\|$	$directCount(B,RWO) = \|\{[u] \in Left(ei) \mid ei([u]) = B, \|B\| \geq 2\}\|$
$\|cmn(B,RWC)\|$	$directCount(B,RWC) = \|\{[u] \in Left(ci) \mid ci([u]) = B, \|B\| \geq 2\}\|$
$\|cmn(B,RWP)\|$	$directCount(B,RWP) = \|\{[u] \in Left(pi) \mid pi([u]) = B, \|B\| \geq 2\}\|$
$\|cmn(B,LIT)\|$	$directCount(B,LIT)) = \|\{l \in Left(li) \mid li(l) = B, \|B\| \geq 2\}\|$
$\|cmn(B,RWT)\|$	$directCount(B,RWT) = \|\{s.p.o \in Left(eti) \mid eti(s.p.o) = B, p \in PID, \|B\| \geq 2\}\|$
$\|cmnTriples(B,u)\|$	$directCount(B,u)) = \|\{p.o \in Left(eti([u])) \mid eti([u].p.o) = B, p \in (PID \cup PID*), \|B\| \geq 2\}\|$



EID	Property (PID)	EID or Literal or CID	Datasets
E1 (Aristotle)	P1 (birthPlace)	E2 (Stagera)	1,2,3,4
	P2 (birthYear)	"384 bc"	1,3
	P3 (influences)	E3 (Kant)	1,2
		E7 (Marx)	1,2
	P5 (type)	C2 (GRE Philosopher)	2,4
E4 (Greece)	P4 (capital)	E6 (Athens)	4
E5 (Socrates)	P1 (birthPlace)	E6 (Athens)	1,3,4
	P2 (birthYear)	"470 bc"	1,2
		"471 bc"	3
	P5 (type)	C2 (GRE Philosopher)	2,4
E6 (Athens)	P6(lived)	E1 (Aristotle)	3,4
E7 (Marx)	P5 (type)	C1 (GER Philosopher)	3

Subset	DC
1,2	3
1,2,3,4	1
1,3	1
1,3,4	1
2,4	2
3,4	1

Figure 2. The steps for performing "lattice"-based connectivity measurements.

Figure 1 shows the *directCount* list for different queries (for reasons of simplicity, we put in the *directCount* list only the IDs of the datasets, e.g., 1,2, instead of D_1, D_2). For finding the *directCount* list for the first query, one should scan the *Entity* Index, while for the second query, one should scan the *Entity-Triples* index, by ignoring the entries containing a "*" character after a specific real world property (these entries are shown in blue), since these triples occur twice in that index (the whole process for the second query is shown in Figure 2). On the contrary, for the third query, one should scan only the entry of Aristotle in the *Entity-Triples* index, by taking also into account triples where this entity occurs as object (i.e., containing a "*" character after the property). Finally, for the last query, one should take into account entries that contain only D_4 dataset. Now, let $Up(B,S) = \{B' \in \mathrm{P}(D) | B \subseteq B', directCount(B',S) > 0\}$, i.e., all the supersets of a specific subset B, that occur in *directCount* list for a

specific measurement type S. The key point is that the sum of the *directCount* of $Up(B,S)$ gives the intersection value of each subset B for a measurement type S.

$$|cmn(B,S)| = \sum_{B' \in Up(B,S)} directCount(B',S) \quad (8)$$

The aforementioned equation relies on the following proposition:

Proposition 1. *Let F and F' be two families of sets. If $F \subseteq F'$ then $\bigcap_{S \in F'} S \subseteq \bigcap_{S \in F} S$ (The proof can be found in [41]).*

Commonalities among any set of Datasets. In [4,5], we proposed two different lattice traversals, a top-down Breadth First Search (BFS) and a bottom-up Depth First Search (DFS) traversal. In [5], we showed that the bottom-up Depth-First Search traversal is faster for large number of datasets, it needs less memory comparing to the BFS approach while it can be easily parallelized. As a consequence, we introduced an algorithm for splitting the lattice in smaller parts for performing such measurements in parallel [5]. Here, we describe how one can exploit the bottom-up Depth-First Search Traversal for computing the measurements for any measurement type S.

Preprocessing steps. The algorithm takes as input the corresponding *directCount* list for a measurement type S. In particular, we scan the corresponding index for the desired measurement type S, for producing its *directCount* list. For instance, in Figure 2, for finding the *directCount* list for real world triples, we scan a part of *Entity-Triples* index, i.e., we do not scan triples having a "*" character after the property ID, since these triples occur twice in that index (see Figure 1). Moreover, by traversing this list, we find the $Up(B,S)$ of each subset B of level two, i.e., the level that contains pairs of datasets. In Figure 2, one can see the $Up(B,RWO)$ for all the pairs of datasets for the corresponding *directCount* list (which was created by scanning the *Entity-Triples* index of that example).

Lattice Traversal and Computation of Commonalities. The algorithm starts from a pair of datasets, e.g., $B = \langle D_1, D_2 \rangle$, and it computes $|cmn(B,S)|$ by adding the *directCount* score of $Up(B,S)$. In the example of Figure 2, the $Up(B,RWO)$ of D_1, D_2 contains the subsets $1,2$ and $1,2,3,4$ (which correspond to $\langle D_1, D_2 \rangle$ and $\langle D_1, D_2, D_3, D_4 \rangle$). If we take the sum of their score in the *directCount* list, (i.e., the score of $1,2$ is 3 and the score of $1,2,3,4$ is 1), we will find that these two datasets have four triples in common. Afterwards, it continues upwards, by exploring the supersets of a specific node that have not been explored yet. The exact order that we use for visiting the nodes can be observed in Figure 2. Specifically, it first visits a triad of datasets B', where $B' = B \cup D_k, D_k \notin B$, i.e., it contains all the datasets of B, plus a new dataset D_k. Each time that it visits a superset, it should check which of the $Up(B,S)$ of B can be transferred to B', since $Up(B,S) \supseteq Up(B',S)$. For example, $\langle D_1, D_2 \rangle \supseteq \langle D_1, D_2 \rangle$, however, $\langle D_1, D_2 \rangle \not\supseteq \langle D_1, D_2, D_3 \rangle$, therefore, we cannot "transfer" $1,2$ to $\langle D_1, D_2, D_3 \rangle$ in the example of Figure 2. Afterwards, it computes $|cmn(B',S)|$ and continues with a quad of datasets by following the same process (i.e., checking $Up(B',S)$). It continues upwards, until there is not another superset to visit (e.g., $\langle D_1, D_2, D_3, D_4 \rangle$ in the example of Figure 2). In such a case, it returns to the previous node and checks if there are other supersets to explore, e.g., in Figure 2, when it returns to $\langle D_1, D_2 \rangle$, it should also visit the node $\langle D_1, D_2, D_4 \rangle$. It is worth mentioning, that for not visiting a superset B' many times, we follow a numerical ascending order, i.e., we visit a superset B' of a subset B containing a new dataset D_k only if $\forall\, D_i \in B, k > i$. For example, when we visit $B = \langle D_1, D_2 \rangle$ we continue with $B' = \langle D_1, D_2, D_3 \rangle$, since the ID of the new dataset, i.e., D_3, is bigger than the IDs of D_1 and D_2. On the contrary, when we visit $B = D_1, D_3$, we do not continue with $B' = \langle D_1, D_2, D_3 \rangle$, since the ID of the new dataset, i.e., D_2, is smaller than the ID of D_3.

Moreover, by using this algorithm, we can exploit Proposition 1 for avoiding creating nodes for subsets which do not have commonalities (nodes with zero score). In particular, from Proposition 1 we know that for two subsets of datasets $B \subset B'$, $|cmn(B',S)| \leq |cmn(B,S)|$, thereby, if the cardinality of intersection of B is zero, it implies that the cardinality of intersection of all the supersets of B will be also zero. For this reason, there is no need to visit the supersets of a subset B in such cases. Moreover, it is

worth mentioning that one can exploit Proposition 1, for computing also nodes for a given threshold (e.g., all the subsets having at least 10 common elements). The time complexity of this algorithm is $\mathcal{O}(|V|)$, where $|V|$ is the number of vertices (for the whole lattice, $V = 2^{|D|}$), since it passes once from each node and it creates one edge per node ($|V| + |V|$). The space complexity is $\mathcal{O}(d)$, where d is the maximum depth of the lattice (in our case d is the maximum level having at least one subset of datasets B where $|cmn(B, S)| > 0$). In [5] one can find more details about this algorithm, i.e., how to split the lattice in smaller parts for performing the measurements in parallel. Below, we show how to compute the measurements for a specific part of a lattice, i.e., for a specific subset B and for a dataset D_i.

Commonalities of a specific subset B. For a specific subset B and a measurement type S, we scan the corresponding *directCount* list once, i.e., all B_i having $directCount(B_i, S) > 0$, and we sum all the $directCount(B_i,S)$ if $B_i \in Up(B, S)$.

Commonalities of a specific Dataset D_i. For a specific dataset D_i and a measurement type S, we take into account the *directCount* of a subset $B_i \in D$ in the corresponding *directCount* list, when $directCount(B_i, S) > 0, D_i \in B_i$ (see the fourth query in Figure 1), and we use the aforementioned algorithm, where we visit only the nodes containing D_i. For example, in the fourth query of Figure 1, there is no need to visit the nodes in red color.

7. Experimental Evaluation

Here, we report the results concerning measurements that quantify the speedup obtained by the introduced *MapReduce* techniques and interesting connectivity measurements for over 2 billion triples and 400 LOD Cloud datasets. We used a cluster in okeanos cloud computing service [42], containing 96 cores, 96 GB main memory and 720 GB disk space. We created 96 different virtual machines, each one having 1 core and 1 GB memory. In Table 4, we can see some basic metadata for 400 real datasets which are used in our experiments (belonging in 9 domains), i.e., it shows the number of datasets, triples, URIs, literals, unique subjects and unique objects for each domain (in descending order with respect to their size in triples), which were manually derived through the *datahub.io* online portal data (i.e., for this set of 400 datasets, an RDF dump was provided). In particular, we have indexed over 2 billion triples, 412 million URIs and 429 million literals, while we used 45 million equivalence relationships. Moreover, the number of unique subjects (by taking into account all the triples) is 308 million, while the corresponding number for the unique objects is 691 million. Most datasets belong to the social network domain; however, most triples occur in cross-domain datasets (i.e., 48% of triples), while a large percentage of triples belong to datasets from publication domain (i.e., 33% of triples). Concerning entities and literals, again most of them occur in datasets from cross-domain and publications (i.e., 79.2% of entities and 86.5% of literals). Finally, the size of all the triples on disk is 251 GB, while the size of equivalence relationships is 3.45 GB. In *LODsyndesis* website (http://www.ics.forth.gr/isl/LODsyndesis) one can find the data which were used for performing the experiments, the code and guidelines for reproducing the results, and metadata for each of the 400 datasets that were used in the experiments.

Table 4. Metadata of datasets which are used in the experiments.

Domain	$\|D\|$	|Triples|	|Entities|	|Literals|	|Unique Sub.|	|Unique Obj.|
Cross-Domain **(CD)**	24	971,725,722	199,359,729	216,057,389	125,753,736	308,124,541
Publications **(PUB)**	94	666,580,552	127,624,700	155,052,015	120,234,530	271,847,700
Geographical **(GEO)**	15	134,972,105	40,185,923	25,572,791	20,087,371	47,182,434
Media **(MED)**	8	74,382,633	16,480,681	9,447,048	14,635,734	20,268,515
Life Sciences **(LF)**	18	74,304,529	10,050,139	10,844,398	9,464,532	18,059,307
Government **(GOV)**	45	59,659,817	6,657,014	7,467,560	10,978,458	14,848,668
Linguistics **(LIN)**	85	20,211,506	3,825,012	2,808,717	2,946,076	6,381,618
User Content **(UC)**	14	16,617,837	7,829,599	901,847	3,904,463	8,708,650
Social Networks **(SN)**	97	3,317,666	762,323	853,416	506,525	1,512,842
All	400	2,021,772,367	412,775,120	429,005,181	308,419,818	691,140,591

7.1. Comparative Results

Here, we report measurements that quantify the speedup obtained by the proposed methods and techniques.

Construction of Catalogs and Indexes. First, we report the execution time for computing the closure and for constructing the real world (RW) triples and the semantics-aware indexes. In Table 5, we can see the size of each index and the execution time for constructing the equivalence catalogs, the real world triples and the indexes by using 96 machines. The most time-consuming job is the creation of real world triples, where we replace all the URIs with an identifier, and we transform the literals. In particular, 33.5 min are needed for this job by using 96 machines. For constructing the equivalence catalogs, the real world triples and the *Entity-Triples* Index, one hour is needed by using 96 machines, while we need additionally 21 min for constructing the other indexes (for entities, literals, classes and properties), i.e., the execution time for performing all the jobs (in total 10 *MapReduce* jobs) is 81.55 min. For storing the real world triples and the equivalence catalogs, 106.4 GB are needed. However, the size of `PropEqCat` and `ClEqCat` is only 13.6 MB and 40.9 MB, correspondingly. The size of all the indexes on disk are 92.3 GB, where the size of *Entity-Triples* index, which contains all the triples (occurring either as a subject or as an object) for each unique entity on disk is 70.3 GB. It is worth mentioning that approximately 672 million triples contain an entity as object, therefore these triples can be found twice in the Entity-Triples index. Moreover, we managed to achieve scalability as it can be seen in Figure 3. Specifically, we created the catalogs and indexes by using 12, 24, 48 and 96 machines. As we can observe, each time that we double the number of machines, the execution time is almost reduced in half in many cases, especially in the construction of real world triples.

Table 5. Construction time and size of catalogs and indexes.

Index/Catalog	Execution Time (96 Machines)	Size on Disk
Equivalence Catalogs	9.35 min	24 GB
Real World Triples	33.5 min	82.4 GB
Entity-Triples Index	17 min	70.3 GB
URI Indexes	13.2 min	6 GB
Literals Index	8.5 min	16 GB
All	81.55 min	198.7 GB

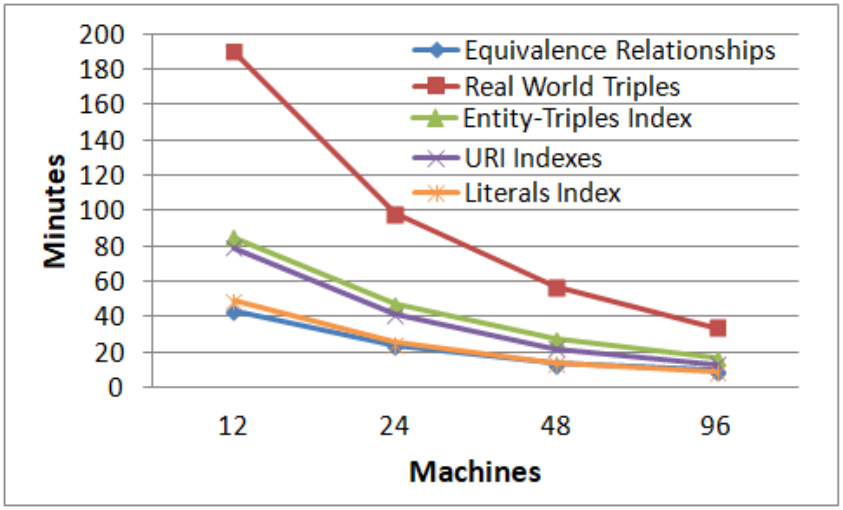


Figure 3. Creation time of indexes and catalogs for different number of machines.

Plain SPARQL versus a Lattice-Based Approach. Here, we compare the lattice-based approach with a SPARQL approach by using a single machine, having an i7 core, 8 GB main memory and 1 TB

disk space. For testing the SPARQL approach, we used a Virtuoso triplestore i.e., *Openlink Virtuoso Version 06.01.3127* (http://virtuoso.openlinksw.com/), and we selected and uploaded a small subset of 10 datasets, having 58 million triples and 1 million equivalence relationships. Afterwards, we sent queries for finding the number of common entities, triples and literals of these 10 datasets. In particular, we report the execution time for computing the commonalities between pairs (45 different pairs) and triads (120 different pairs) for this set of datasets. As we can see in Table 6, by using a SPARQL approach, for finding the common real world entities between all the pairs of datasets, we needed approximately 45 min, (on average 1 min per pair). The computation of closure is the main reason for this delay, i.e., if we do not take into account the `owl:sameAs` relationships and their transitive and symmetric closure, 15 min are needed for computing the common URIs between the pairs of datasets (however most connections are missing in such a case). Concerning the triples, we need even more minutes for finding the common triples, i.e., 50 min, since we should compute the closure for both instance and schema elements. On the contrary, without considering the equivalence relationships, the execution time is only 1.5 min (however, it finds very few common triples). Regarding the common literals, the execution time is 7 min (approximately 10 s per pair on average). The difference in the execution time of measuring common literals, comparing to entities and triples, is rational, since for the literals there is no need to compute a closure. Concerning triads of sources, we can see that the execution time increases for each measurement, from 1.84 to 4.82 times.

Table 6. Execution time for SPARQL queries for measuring the connectivity of 10 datasets.

Measurement	Time for 45 Pairs	Time for 120 Triads
Common Entities	44.9 min	87.55 min
Common URIs (without closure)	15 min	29.1 min
Common Triples	50 min	92 min
Common Triples (without closure)	1.45 min	7 min
Common Literals	6.8 min	15 min

On the contrary, by using even a single machine and the lattice-based algorithm, one can find all the commonalities between millions of subsets in seconds [4,5]. In Table 7, we introduce the execution time for reading the *directCount* list, assigning $Up(B, S)$ to pairs of datasets and computing (by using the lattice-based algorithm) the number of common entities, triples and literals for pairs, triads, quads, and quintets i.e., $2 \leq |B| \leq 5$, of 400 datasets, that share at least one common element (we ignore subsets that do not share elements). At first, we can see that the *directCount* list size is very small comparing to each index size. For measuring the cardinality of intersection of common entities for 18 million subsets (belonging to pairs, triads and quads and quintets), we needed 51 s, i.e., approximately 363,000 subsets per second. Regarding the computation of common real world triples, the execution time was 3 s for computing the commonalities between 1.7 million subsets, i.e., 592,000 subsets per second, since only a small number of subsets of datasets share common triples. On the contrary, a lot of subsets of datasets share common literals, and for this reason we measured the common literals of datasets only for pairs and triads of sources. The execution time for measuring the common literals for 4.9 million subsets was 328 s, i.e., 15,000 subsets per second. Moreover, the size of the *directCount* list affects the execution time. In particular, the size of the *directCount* list of common literals is 14 times bigger than the *directCount* list of common entities, and 67 times bigger than the corresponding list of triples. As a consequence, in one second we can measure 39 times more subsets containing common triples, and 24 times more subsets having common entities, in comparison to the number of subsets containing common literals. More experiments concerning the efficiency of the aforementioned lattice-based approach can be found in [4,5].

Table 7. Execution time for finding the commonalities for subsets of 400 RDF Datasets by using a "lattice"-based algorithm.

Connectivity Measurement	Direct Counts List Size (% of Index Size)	Number of Subsets Measured	Execution Time
Common RW Entities	21,781 (0.006%)	18,531,752	51 s
Common RW Triples	4700 (0.0002%)	1,776,136	3 s
Common Literals	318,978 (0.09%)	4,979,482 (pairs and triads)	328 s

Generally, the difference comparing to a plain SPARQL approach is obvious, e.g., by using a SPARQL approach, for the common entities one minute is needed on average for computing the commonalities of a specific pair of datasets, while the lattice-based approach can compute at the same time the measurements of over 18 million subsets of datasets.

7.2. Connectivity Measurements for LOD Cloud Datasets.

Here, we show some indicative measurements for 400 LOD Cloud datasets. At first, we show the impact of transitive and symmetric closure. Second, we introduce some general connectivity measurements that indicate the power-law distribution for any category of elements, and afterwards we show subsets of datasets that are highly connected. Finally, we describe some conclusions derived by the experiments.

Inference Results. In Table 8, we report the results of the transitive and symmetric closure for `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass` relationships. By computing the closure, we inferred more than 73 million new `owl:sameAs` relationships, i.e., the increase percentage of `owl:sameAs` triples was 163%. Moreover, for 26 million entities there exists at least two URIs that refer to them (on average 2.6 URIs for the aforementioned entities). On the contrary, we inferred only 935 `owl:equivalentProperty` relationships (i.e., increase percentage was 11.46%) and 1164 `owl:equivalentClass` triples (i.e., increase percentage was 29%), while there exists 4121 properties and 2041 classes containing having at least two URIs that describe them (on average 2.05 for such properties and 2.11 for such classes).

Table 8. Statistics for equivalence relationships.

Category	Value
`owl:sameAs` Triples	44,853,520
`owl:sameAs` Triples Inferred	73,146,062
RW Entities having at least two URIs	26,124,701
`owl:equivalentProperty` Triples	8157
`owl:equivalentProperty` Triples Inferred	935
RW Properties having at least two URIs	4121
`owl:equivalentClass` Triples	4006
`owl:equivalentClass` Triples Inferred	1164
RW Classes having at least two URIs	2041

Power-Law Distribution of Elements. In Table 9, we can see that only a small percentage of each set of elements exists in two or more datasets. In particular, only 0.8% of triples occur in ≥ 2 datasets and 0.24% in ≥ 3 datasets. The corresponding percentages of entities (i.e., 7.73%) and literals (i.e., 11.88%) occurring in ≥ 2 datasets are far higher comparing to triples. However, again most entities and literals occur in 1 dataset. Regarding classes and properties, only a small percentage of them (i.e., less than 1%) occur in ≥ 2 datasets, which means that possibly there is a lack of equivalence relationships between schema elements. For investigating such a case, we created also a different index of triples, where we ignore the property of each triple i.e., we find the common subject-object pairs. For constructing such an index, one can use Algorithm 2; however, one should replace in the mapper

(in lines 4 and 6) all the property IDs with a fixed value. As we can see in Table 9, if we ignore the property of each triple, 2.82% of subject-object pairs occur in two or more datasets, whereas, by taking into account the properties, the corresponding percentage was 0.8%. However, we should mention that it is possible that two or more common subject-object pairs use different properties for describing different facts. For instance, suppose that two different datasets contain the following triples in two datasets D_i and D_j, ⟨di:Aristotle, di:wasBornIn, di:Stagira⟩ and ⟨dj:Aristotle, dj:livedIn, dj:Stagira⟩. These two triples describe different things, however, their subject and objects refer to the same entities.

Table 9. Elements per number of datasets.

Category	Exactly in 1 Dataset	Exactly in 2 Datasets	≥ 3 Datasets
RW Entities	339,818,971 (92.27%)	21,497,165 (5.83%)	6,979,109 (1.9%)
Literals	336,915,057 (88.88%)	29,426,233 (7.77%)	12,701,841 (3.35%)
RW Triples	1,811,576,438 (99.2%)	10,300,047 (0.56%)	4,348,019 (0.24%)
RW Properties	246,147 (99.37%)	569 (0.23%)	997 (0.4%)
RW Classes	542,549 (99.68%)	1096 (0.2%)	605 (0.11%)
RW Subject-Object Pairs	1,622,784,858 (97.18%)	37,962,509 (2.27%)	9,241,676 (0.55%)

Moreover, in Figure 4, we can observe the distribution of different elements (e.g., triples, entities, etc.), i.e., the number of elements that can be found in a specific number of datasets. We can clearly see a power-law distribution for any category of elements, i.e., there exists a large number of elements (e.g., literals, entities) that occur in a small numbers of datasets, while only a small number of elements can be found in a lot of datasets. It is worth mentioning that there exists over 300,000 entities and over 500,000 of literals that occur in 10 or more datasets; however, less than 1600 real world triples can be found in more than 10 datasets.

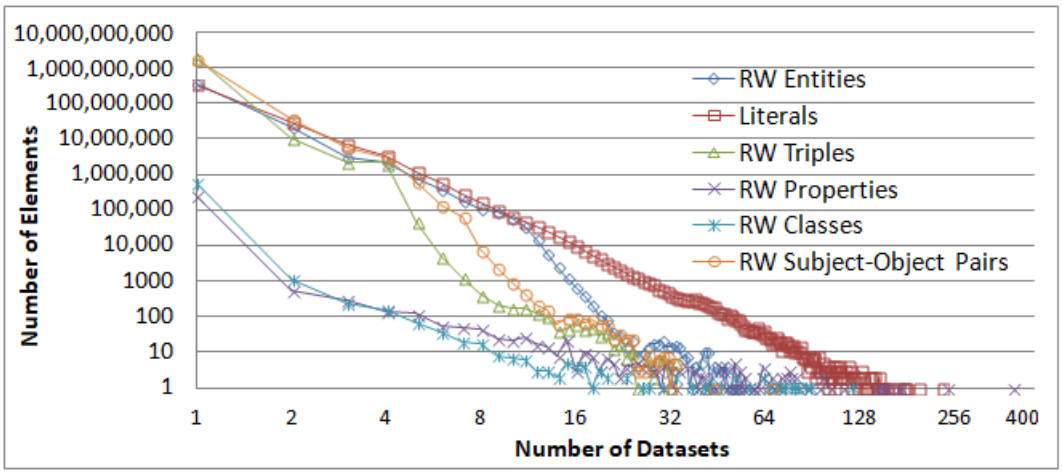


Figure 4. Number of datasets where different sets of elements occur.

Connectivity among subsets of datasets. In Table 10, we show the connectivity among pairs and triads of datasets for different elements, while we also mention the number of datasets that are disconnected, i.e., for a specific measurement type (e.g., number of common literals), these datasets do not have commonalities with other ones. Generally, a big percentage of pairs of datasets share literals, i.e., 78%. It is rational, since each literal can be used for describing several things. For example, the literal "1980" can be the birth year of several people, the year when a movie released, and so forth. Thereby, a lot of datasets can contain this literal, for describing different facts. On the contrary, for the real world entities and triples, only 11.3% of datasets pairs have common entities and 5.59% of pairs of datasets contain same triples. It means that only half of the dataset pairs containing common entities, share also common triples. On the other hand, if we compute the number of common triples

by ignoring the property of each triple (i.e., common subject-object pairs), 10.45% of datasets pairs share common subject-object pairs. It means that almost every pair of datasets that contain at least one common entity, share also common subject-object pairs. Concerning schema elements, almost all pairs of datasets share one property (99%), since most datasets use properties from popular ontologies such as rdf and $rdfs$, $foaf$ and $xmlns$. However, by excluding properties belonging to the aforementioned popular ontologies, 24.5% of datasets pairs share properties. Finally, a lot of datasets pairs (i.e., 30.2%) have at least one class in common; however, if we exclude again classes from popular ontologies, the percentage decreases (i.e., 5.42%).

Table 10. Connected subsets of datasets.

Category	Connected Pairs	Connected Triads	Disconnected Datasets (of 400)
Real World Entities	9075 (11.3%)	132,206 (1.24%)	87 (21.75%)
Literals	62,266 (78%)	4,917,216 (46.44%)	3 (0.75%)
Real World Triples	4468 (5.59%)	35,972 (0.33%)	134 (33.5%)
Real Subject-Object Pairs	7975 (10%)	107,083 (1%)	129 (32.2%)
Real World Properties	19,515 (24.45%)	569,708 (5.38%)	25 (6.25%)
Real World Classes	4326 (5.42%)	53,225 (0.5%)	107 (26.7%)

However, it is also important to check the "degree" of connectivity of the connected pairs of datasets, i.e., how many common elements the aforementioned connected pairs of datasets share. In Figure 5, we show the number of datasets' pairs, whose cardinality of common elements belong to a specific interval of integers, e.g., for the interval $[1, 10)$ we show how many connected pairs have from 1 to 9 common elements (e.g., entities). In particular, most pairs of datasets share less than 10 elements for each set of elements (literals, triples, and entities, properties and classes). In general, we observe a power-law distribution, since many pairs of datasets share a few elements, while only a few pairs of datasets share thousands or millions of elements. Furthermore, we can observe the difference in the level of connectivity among literals and other elements, e.g., over 10,000 pairs of datasets share at least 100 literals, whereas 1525 pairs of datasets have at least 100 common entities. Concerning common triples, the 84% of connected pairs share less than 100 triples, while as we can see in Figure 5, less than 10 pairs share more than one million triples. Regarding entities, most connected pairs (83.2%) have in common 100 or less entities, and only a few pairs share thousands or millions of entities, while for the literals, 77.5% of connected pairs share less than 100 literals. For the remaining categories, most connected pairs sharing properties (96.9% of pairs) and classes (99.4% of pairs) have in common less than 10 properties and classes, respectively.

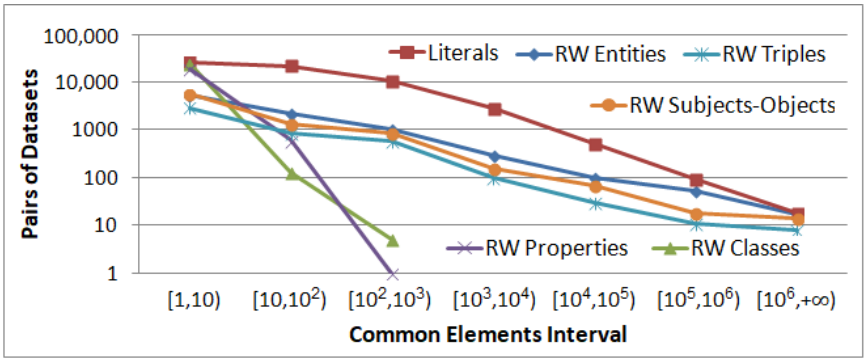


Figure 5. Number of connected pairs of datasets per interval for each measurement type.

For the triads of datasets, the percentages are even smaller. In particular, approximately 1% of triads share common elements, classes and triples. However, for the literals the percentage is quite high, i.e., 46.44% of triads share common literals, while 5.38% of triads share a property. In Figure 6,

we can observe the "degree" of connectivity of triads. Similarly to the case of pairs, we observe a power-law distribution, i.e., most triads of datasets have in common from 1 to 10 elements for each different set of elements. Moreover, it is worth noting that 140 triads of datasets share over 100,000 entities, only 5 triads of datasets contain over 100,000 common triples and 180 triads over 100,000 common literals.

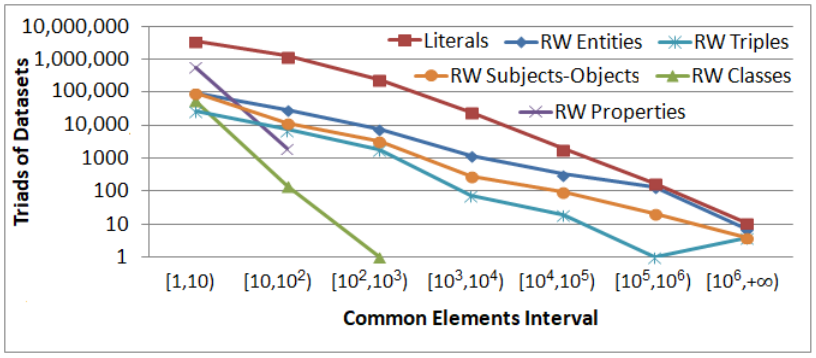


Figure 6. Number of connected triads of datasets per interval for each measurement type.

Finally, the last column of Table 10 shows the number of datasets that do not have commonalities with other ones for each specific elements. It is worth noting that 87 datasets do not have common entities with other ones, while only 3 datasets do not share common literals. On the contrary, the number of disconnected datasets increases in the cases of common triples and common classes.

The most connected subsets of datasets. In Table 11, we introduce measurements for the 10 subsets (containing 3 or more datasets) having the most common triples. As we can see, the combinations of the four popular cross-domain datasets (*DBpedia* [43], *Yago* [44], *Freebase* [45] and *Wikidata* [46]) share a lot of triples. Concerning other subsets of datasets, some datasets from publications domain (i.e., *bl.uk* [47], *BNF* [48] and *VIAF* [49]), and one dataset from government domain (i.e., *JRC-names* [50]) share many triples with the aforementioned cross-domain ones. Regarding the connectivity for other sets of elements, the combinations of the four aforementioned popular cross-domain datasets are again the most connected ones for entities and literals (i.e., the quad of the four popular cross-domain datasets share approximately 3 million entities and 3.5 million literals). The most connected triad of datasets, concerning classes, contains the following set of datasets (*DBpedia*, *Opencyc* [51], *ImageSnippets* [52]) with 188 common classes, while for the properties, the most connected triad includes (*VIVO Wustl* [53], *FAO* [54], *VIVO scripps* [55]) with 68 common properties. All the measurements for pairs, triads, and quads of subsets of datasets (in total 11,689,103 million subsets) for each different measurement type are accessible through *LODsyndesis* and *datahub.io* (http://datahub.io/dataset/connectivity-of-lod-datasets), in CSV and RDF format, by using VoID-WH ontology [56], which is an extension of VoID ontology [57] (in total we have created 99,221,766 million triples).

Table 11. Top-10 subsets with $\geq$ 3 datasets having the most common real world triples.

Datasets of subset B	Common RW Triples
1: {DBpedia,Yago,Wikidata}	2,683,880
2: {Freebase,Yago,Wikidata}	2,653,641
3: {DBpedia,Freebase,Wikidata}	2,509,702
4: {DBpedia,Yago,Freebase}	2,191,471
5: {DBpedia,Yago,Freebase,Wikidata}	2,113,755
6: {DBpedia,Wikidata,VIAF}	396,979
7: {bl.uk,DBpedia,Wikidata}	92,462
8: {BNF,Yago,VIAF}	52,420
9: {bl.uk,DBpedia,VIAF}	24,590
10: {DBpedia,Wikidata,JRC-names}	18,140

The most popular datasets. Table 12 shows the top ten datasets that contain the most entities, literals and triples that can be found in three or more datasets. As we can see, *Wikidata* contains the most entities and triples that can be found in three or more datasets, while *Yago* is the dataset having the most literals that occur at least in three datasets. In all categories, the four datasets from the cross-domain are the most popular, while we can observe that *VIAF* dataset (from publications domain) occurs in the fifth position in all cases. Concerning the remaining positions, most datasets belong to publications domain (e.g., *id.loc.gov* [58], *DNB* [59], *Radatana* [60], and others). Moreover, there exists also a dataset from the geographical domain, i.e., *GeoNames* [61], which contains a lot of entities and literals that occur in three or more datasets, and a dataset from media domain, i.e., *LMDB* [62], which contains several triples that occur in more than two datasets.

Table 12. Top-10 datasets with the most entities, triples and literals existing at least in 3 datasets.

Position	Dataset-RW Entities in $\geq$ 3 Datasets		Dataset-RW Triples in $\geq$ 3 Datasets		Dataset-Literals in $\geq$ 3 Datasets	
1	Wikidata	4,580,412	Wikidata	4,131,042	Yago	9,797,331
2	DBpedia	4,238,209	DBpedia	3,693,754	Freebase	8,653,152
3	Yago	3,643,026	Yago	3,380,427	Wikidata	8,237,376
4	Freebase	3,634,980	Freebase	3,143,086	DBpedia	7,085,587
5	VIAF	3,163,689	VIAF	485,356	VIAF	3,907,251
6	id.loc.gov	2,722,156	bl.uk	125,484	bl.uk	1,819,223
7	d-nb	1,777,553	bnf	55,237	GeoNames	1,501,854
8	bnf	1,056,643	JRC-names	28,687	id.loc.gov	1,272,365
9	bl.uk	1,051,576	Opencyc	26,310	bnf	968,119
10	GeoNames	554,268	LMDB	20,465	radatana	957,734

7.2.1. Conclusions about the Connectivity at LOD Scale

The measurements revealed the sparsity of LOD Cloud. In general, we observed a power-law distribution, i.e., a large percentage of elements (entities, classes, etc.) occur in one dataset, while most connected datasets contain a small number of common elements, which means that a lot of publishers do not connect their entities with other datasets. Most subsets of datasets share literals, while only a few pairs and triads of datasets share triples. Moreover, most datasets share some properties from popular ontologies (such as rdf, rdfs, etc.); however, it seems that there is a lack of connections for schema elements. Consequently, it is hard to find common triples among the datasets, even between datasets sharing a lot of common entities and common literals. Indeed, by ignoring the property of each triple, we identified that there exists 3465 pairs of datasets having common subject-object pairs, but not common triples. Concerning the most connected datasets, they are the four datasets belonging to cross-domain (i.e., *DBpedia*, *Yago*, *Freebase* and *Wikidata*), while there are also combinations containing datasets from cross-domain and publication domain that are highly connected. Moreover, the most popular datasets (containing elements that can be found in three or more datasets) are predominantly the cross-domain ones, while in this list one can find also datasets from publications and geographical domains.

8. Discussion

The main objective of Linked Data is linking and integration, and a major step for evaluating whether this target has been reached, is to find all the connections among the Linked Open Data (LOD) Cloud datasets. In this paper, we proposed connectivity measurements among two or more datasets, which are based on scalable algorithms and semantics-aware indexes. In particular, we introduced algorithms for computing the transitive and symmetric closure of equivalence relationships, while we showed how to create in parallel semantics-aware indexes for different sets of elements (entities, triples, classes, etc.). Moreover, we described how to exploit such indexes for finding fast the commonalities between any subset of datasets by using a "lattice"-based algorithm. Generally, this is the only work

that measures the connectivity among two or more datasets, by taking also into consideration the transitive and symmetric closure of equivalence relationships (e.g., `owl:sameAs`). Moreover, as a product, to the best of our knowledge, the indexes of *LODsyndesis* (which is the suite of services that exploits the aforementioned indexes and measurements) constitute the biggest knowledge graph of LOD that is complete with respect to the inferable equivalence relations.

As regards the experiments, by using 96 machines, we were able to construct all the indexes for 2 billions of RDF triples (from 400 datasets) in 81 min, while the experiments showed that the construction algorithms are highly scalable. Furthermore, we compared the execution time of computing the measurements by using plain SPARQL and by using a lattice-based algorithm. In particular, by using a SPARQL approach, even one minute is needed for finding the commonalities between a pair of datasets, while with a lattice-based algorithm we computed the commonalities for thousands of datasets' pairs in one second. Concerning the connectivity of LOD cloud datasets, the measurements showed the "sparsity" of the current LOD cloud, since most elements (entities, triples, etc.) occur in one dataset, while only a small percentage of them exists in two or more datasets. Moreover, a high percentage of pairs (i.e., 78%) and triads (i.e., 46.44%) of datasets share literals, while only a small percentage of pairs (i.e., 5.59%) and triads (i.e., 0.33%) have triples in common. Concerning the most popular datasets, i.e., they are highly connected with other datasets, they belong predominantly to cross-domain and publications domain.

For future work, we plan to exploit the semantics-aware indexes for performing lattice-based measurements for any set operation (e.g., union, complement, etc.), while we desire to exploit the indexes as input for machine learning-based tasks. Finally, we plan to exploit such measurements for offering more advanced services that concern several real world tasks, such as *Dataset Discovery* and *Data Veracity*.

Author Contributions: Conceptualization, M.M. and Y.T.; Data Curation, M.M.; Funding acquisition, Y.T.; Investigation, M.M.; Methodology, M.M. and Y.T.; Project administration, Y.T.; Software, M.M.; Supervision, Y.T.; Writing-original draft, M.M. and Y.T.

Funding: This work has received funding from the General Secretariat for Research and Technology (GSRT) and the Hellenic Foundation for Research and Innovation (HFRI) (Code No 166).

Conflicts of Interest: The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

References

1. Dong, X.L.; Berti-Equille, L.; Srivastava, D. Data fusion: Resolving conflicts from multiple sources. In *Handbook of Data Quality*; Springer: Berlin, Germany, 2013; pp. 293–318.
2. Mountantonakis, M.; Tzitzikas, Y. How Linked Data can Aid Machine Learning-Based Tasks. In Proceedings of the International Conference on Theory and Practice of Digital Libraries, Thessaloniki, Greece, 18–21 September 2017; Springer: Berlin, Germany, 2017; pp. 155–168.
3. Ristoski, P.; Paulheim, H. RDF2VEC: RDF graph embeddings for data mining. In Proceedings of the International Semantic Web Conference, Kobe, Japan, 17–21 October 2016; Springer: Berlin, Germany, 2016; pp. 498–514.
4. Mountantonakis, M.; Tzitzikas, Y. On Measuring the Lattice of Commonalities Among Several Linked Datasets. *Proc. VLDB Endow.* **2016**, *9*, 1101–1112. [CrossRef]
5. Mountantonakis, M.; Tzitzikas, Y. Scalable Methods for Measuring the Connectivity and Quality of Large Numbers of Linked Datasets. *J. Data Inf. Qual.* **2018**, *9*. [CrossRef]
6. Paton, N.W.; Christodoulou, K.; Fernandes, A.A.; Parsia, B.; Hedeler, C. Pay-as-you-go data integration for linked data: opportunities, challenges and architectures. In Proceedings of the 4th International Workshop on Semantic Web Information Management, Scottsdale, AZ, USA, 20–24 May 2012; p. 3.
7. Christophides, V.; Efthymiou, V.; Stefanidis, K. Entity Resolution in the Web of Data. *Synth. Lect. Semant. Web* **2015**, *5*, 1–122. [CrossRef]

8. Ermilov, I.; Lehmann, J.; Martin, M.; Auer, S. LODStats: The data web census dataset. In Proceedings of the International Semantic Web Conference, Kobe, Japan, 17–21 October 2016; Springer: Berlin, Germany, 2016; pp. 38–46.
9. Prud' Hommeaux, E.; Seaborne, A. SPARQL Query Language for RDF. *W3C Recommendation*, 15 January 2008.
10. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
11. Antoniou, G.; Van Harmelen, F. *A Semantic Web Primer*; MIT Press: Cambridge, MA, USA, 2004.
12. Rietveld, L.; Beek, W.; Schlobach, S. LOD lab: Experiments at LOD scale. In Proceedings of the International Semantic Web Conference, Bethlehem, PA, USA, 11–15 October 2015; Springer: Berlin, Germany, 2015; pp. 339–355.
13. Fernández, J.D.; Beek, W.; Martínez-Prieto, M.A.; Arias, M. LOD-a-lot. In Proceedings of the International Semantic Web Conference, Vienna, Austria, 21–25 October 2017; pp. 75–83.
14. Nentwig, M.; Soru, T.; Ngomo, A.C.N.; Rahm, E. LinkLion: A Link Repository for the Web of Data. In *The Semantic Web: ESWC 2014 Satellite Events*; Springer: Berlin, Germany, 2014; pp. 439–443.
15. Schmachtenberg, M.; Bizer, C.; Paulheim, H. Adoption of the linked data best practices in different topical domains. In *The Semantic Web–ISWC 2014*; Springer: Berlin, Germany, 2014; pp. 245–260.
16. Auer, S.; Demter, J.; Martin, M.; Lehmann, J. LODStats-an Extensible Framework for High-Performance Dataset Analytics. In *Knowledge Engineering and Knowledge Management*; Springer: Berlin, Germany, 2012; pp. 353–362.
17. Giménez-Garcıa, J.M.; Thakkar, H.; Zimmermann, A. Assessing Trust with PageRank in the Web of Data. In Proceedings of the 3rd International Workshop on Dataset PROFIling and fEderated Search for Linked Data, Anissaras, Greece, 30 May 2016.
18. Debattista, J.; Lange, C.; Auer, S.; Cortis, D. Evaluating the Quality of the LOD Cloud: An Empirical Investigation. Accepted for publication in *Semant. Web J.* **2017**.
19. Debattista, J.; Auer, S.; Lange, C. Luzzu—A Methodology and Framework for Linked Data Quality Assessment. *J. Data Inf. Qual. (JDIQ)* **2016**, *8*, 4. [CrossRef]
20. Mountantonakis, M.; Tzitzikas, Y. Services for Large Scale Semantic Integration of Data. *ERCIM NEWS*, 25 September 2017, pp. 57–58.
21. Vandenbussche, P.Y.; Atemezing, G.A.; Poveda-Villalón, M.; Vatant, B. Linked Open Vocabularies (LOV): A gateway to reusable semantic vocabularies on the Web. *Semant. Web* **2017**, *8*, 437–452. [CrossRef]
22. Valdestilhas, A.; Soru, T.; Nentwig, M.; Marx, E.; Saleem, M.; Ngomo, A.C.N. Where is my URI? In Proceedings of the 15th Extended Semantic Web Conference (ESWC 2018), Crete, Greece, 3–7 June 2018.
23. Mihindukulasooriya, N.; Poveda-Villalón, M.; García-Castro, R.; Gómez-Pérez, A. Loupe-An Online Tool for Inspecting Datasets in the Linked Data Cloud. In Proceedings of the International Semantic Web Conference (Posters & Demos), Bethlehem, PA, USA, 11–15 October 2015.
24. Glaser, H.; Jaffri, A.; Millard, I. Managing Co-Reference on the Semantic Web; Web & Internet Science: Southampton, UK, 2009.
25. Käfer, T.; Abdelrahman, A.; Umbrich, J.; O' Byrne, P.; Hogan, A. Observing linked data dynamics. In Proceedings of the Extended Semantic Web Conference, Montpellier, France, 26–30 May 2013; Springer: Berlin, Germany, 2013; pp. 213–227.
26. Käfer, T.; Umbrich, J.; Hogan, A.; Polleres, A. Towards a dynamic linked data observatory. In Proceedings of the LDOW at WWW, Lyon, France, 16 April 2012.
27. McCrae, J.P.; Cimiano, P. Linghub: A Linked Data based portal supporting the discovery of language resources. In Proceedings of the SEMANTiCS (Posters & Demos), Vienna, Austria, 15–17 September 2015; Volume 1481, pp. 88–91.
28. Vandenbussche, P.Y.; Umbrich, J.; Matteis, L.; Hogan, A.; Buil-Aranda, C. SPARQLES: Monitoring public SPARQL endpoints. *Semant. Web* **2016**, *8*, 1049–1065. [CrossRef]
29. Yumusak, S.; Dogdu, E.; Kodaz, H.; Kamilaris, A.; Vandenbussche, P.Y. SpEnD: Linked Data SPARQL Endpoints Discovery Using Search Engines. *IEICER Trans. Inf. Syst.* **2017**, *100*, 758–767. [CrossRef]
30. Papadaki, M.E.; Papadakos, P.; Mountantonakis, M.; Tzitzikas, Y. An Interactive 3D Visualization for the LOD Cloud. In Proceedings of the International Workshop on Big Data Visual Exploration and Analytics (BigVis'2018 at EDBT/ICDT 2018), Vienna, Austria, 26–29 March 2018.

31. Ilievski, F.; Beek, W.; van Erp, M.; Rietveld, L.; Schlobach, S. LOTUS: Adaptive Text Search for Big Linked Data. In Proceedings of the International Semantic Web Conference, Kobe, Japan, 17–21 October 2016; pp. 470–485.
32. Fernández, J.D.; Martínez-Prieto, M.A.; Gutiérrez, C.; Polleres, A.; Arias, M. Binary RDF representation for publication and exchange (HDT). *Web Semant. Sci. Serv. Agents World Wide Web* **2013**, *19*, 22–41. [CrossRef]
33. Erling, O.; Mikhailov, I. Virtuoso: RDF support in a native RDBMS. In *Semantic Web Information Management*; Springer: Berlin, Germany, 2010; pp. 501–519.
34. Aranda-Andújar, A.; Bugiotti, F.; Camacho-Rodríguez, J.; Colazzo, D.; Goasdoué, F.; Kaoudi, Z.; Manolescu, I. AMADA: Web data repositories in the amazon cloud. In Proceedings of the 21st ACM International Conference on Information and knowledge management, Maui, HI, USA, 29 October–2 November 2012; pp. 2749–2751.
35. Papailiou, N.; Konstantinou, I.; Tsoumakos, D.; Koziris, N. H2RDF: adaptive query processing on RDF data in the cloud. In Proceedings of the 21st International Conference on World Wide Web, Lyon, France, 16–20 April 2012; pp. 397–400.
36. Punnoose, R.; Crainiceanu, A.; Rapp, D. Rya: A scalable RDF triple store for the clouds. In Proceedings of the 1st International Workshop on Cloud Intelligence, Istanbul, Turkey, 31 August 2012; p. 4.
37. Schätzle, A.; Przyjaciel-Zablocki, M.; Dorner, C.; Hornung, T.D.; Lausen, G. Cascading map-side joins over HBase for scalable join processing. *arXiv* **2012**, arXiv:1206.6293v1.
38. Kaoudi, Z.; Manolescu, I. RDF in the clouds: A survey. *VLDB J.* **2015**, *24*, 67–91. [CrossRef]
39. Tzitzikas, Y.; Lantzaki, C.; Zeginis, D. Blank node matching and RDF/S comparison functions. In Proceedings of the International Semantic Web Conference, Crete Greece, 31 May 2012; Springer: Berlin, Germany, 2012; pp. 591–607.
40. Rastogi, V.; Machanavajjhala, A.; Chitnis, L.; Sarma, A.D. Finding connected components in map-reduce in logarithmic rounds. In Proceedings of the 2013 IEEE 29th International Conference on Data Engineering (ICDE), Brisbane, Australia, 8–11 April 2013; pp. 50–61.
41. Jech, T. *Set Theory*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2013.
42. Okeanos Cloud Computing Service. Available online: http://okeanos.grnet.gr (accessed on 29 May 2018).
43. DBpedia. Available online: http://dbpedia.org (accessed on 29 May 2018).
44. Yago. Available online: http://yago-knowledge.org (accessed on 29 May 2018).
45. Freebase. Available online: http://developers.google.com/freebase/ (accessed on 29 May 2018).
46. Wikidata. Available online: http://www.wikidata.org (accessed on 29 May 2018).
47. The British Library. Available online: http://bl.uk (accessed on 29 May 2018).
48. Bibliothèque Nationale de France. Available online: http://www.bnf.fr (accessed on 29 May 2018).
49. The Virtual International Authority File. Available online: http://viaf.org (accessed on 29 May 2018).
50. JRC-Names. Available online: http://ec.europa.eu/jrc/en/language-technologies/jrc-names (accessed on 29 May 2018).
51. OpenCyc. Available online: http://www.cyc.com/opencyc/ (accessed on 29 May 2018).
52. ImageSnippets. Available online: http://www.imagesnippets.com/ (accessed on 29 May 2018).
53. VIVO Wustl. Available online: http://old.datahub.io/dataset/vivo-wustl (accessed on 29 May 2018).
54. Food and Agriculture Organization of the United Nations. Available online: http://www.fao.org/ (accessed on 29 May 2018).
55. VIVO Scripps. Available online: http://vivo.scripps.edu/ (accessed on 29 May 2018).
56. Mountantonakis, M.; Minadakis, N.; Marketakis, Y.; Fafalios, P.; Tzitzikas, Y. Quantifying the connectivity of a semantic warehouse and understanding its evolution over time. *Int. J. Semant. Web Inf. Syst. (IJSWIS)* **2016**, *12*, 27–78. [CrossRef]
57. Alexander, K.; Cyganiak, R.; Hausenblas, M.; Zhao, J. *Describing Linked Datasets with the VoID Vocabulary*; W3C Interest Group Note; W3C: Cambridge, MA, USA, 2011.
58. Library of Congress Linked Data Service. Available online: http://id.loc.gov/ (accessed on 29 May 2018).
59. Deutschen National Bibliothek. Available online: http://www.dnb.de (accessed on 29 May 2018).
60. Radatana. Available online: http://data.bibsys.no/ (accessed on 29 May 2018).

61. GeoNames Geographical Database. Available online: http://www.geonames.org/ (accessed on 29 May 2018).
62. Linked Movie Data Base (LMDB). Available online: http://linkedmdb.org/ (accessed on 29 May 2018).