Indexes and Algorithms for Measuring the Connectivity of Linked Data

Michalis Mountantonakis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, Heraklion, GR-70013, Greece

Thesis Advisor: Assistant Prof. *Yannis Tzitzikas*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

Indexes and Algorithms for Measuring the Connectivity of Linked Data

Thesis submitted by
Michalis Mountantonakis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author:

Michalis Mountantonakis

Committee approvals:

Yannis Tzitzikas
Assistant Professor, University of Crete
Thesis Supervisor

Dimitris Plexousakis
Professor, University of Crete
Committee Member

Kostas Magoutis
Assistant Professor, University of Ioannina
Committee Member

Departmental approval:

Antonis Argyros
Professor, University of Crete
Director of Graduate Studies

Heraklion, June 2016

Indexes and algorithms for measuring the connectivity of Linked Data

Abstract

Linked Data is a method for publishing structured data that allows them to be interlinked (by using URIs instead of simple values) for assisting their integration. A big number of such datasets, hereafter sources, has already been published according to the principles of Linked data and their number and size keeps increasing. However, currently it is not evident how connected these datasets are. In particular, it is difficult (a) to obtain complete information about one particular URI (or a set of URIs), (b) to discover a dataset which is relevant to another one, (c) to compute and visualize the degree of connectivity between two or more datasets. All the aforementioned tasks are important for the integration process in an open and involving environment.

To alleviate this problem in this thesis, we introduce metrics, indexes and algorithms which allow the computation and quantification of connectivity among several datasets. For achieving scalability, we propose (i) a namespace-based prefix index, (ii) a sameAs catalog for computing the symmetric and transitive closure of the sameAs relationships encountered in the datasets, (iii) a semantics-aware element index (that exploits the aforementioned indexes), (iv) a lattice of the common elements of any set of datasets, and (v) two lattice-based incremental algorithms for speeding up the computation of the lattice.

We apply and evaluate the proposed approach in the context of a real and operational semantic warehouse containing information about the marine domain (where the metrics are used for assessing the quality of the semantic warehouse and its underlying sources, and for monitoring the quality of the semantic warehouse after a reconstruction), as well as for three hundred LOD cloud datasets. We report measurements that have not been carried out in the past (like the number of common URIs among three or more datasets, the frequency of prefixes, i.a.), we offer novel services (like finding equivalent URIs, find the most relevant datasets for a specific dataset, i.a.) and finally we discuss the speedup obtained by the proposed indexes and algorithms. Finally, we propose an extension of the VoID ontology for publishing, sharing and exploiting such measurements.

Ευρετήρια και αλγόριθμοι για τη μέτρηση του βαθμού διασύνδεσης των διασυνδεδεμένων δεδομένων

Περίληψη

Τα Διασυνδεδεμένα Δεδομένα (Linked Data) είναι ένας τρόπος δημοσίευσης δεδομένων που επιτρέπει τη διασύνδεσή τους (μέσω της χρήσης URIs αντί απλών τιμών) και διευκολύνει την ολοκλήρωσή τους. Ήδη υπάρχουν χιλιάδες τέτοια σύνολα δεδομένων, στο εξής πηγές, και ο αριθμός και το μέγεθος τους διαρκώς αυξάνεται. Παρά ταύτα, αυτή τη στιγμή είναι δύσκολο να εκτιμήσει κανείς πόσο συνδεδεμένες είναι αυτές οι πηγές, και συγκεκριμένα είναι δύσκολη (α) η εύρεση όλων των δεδομένων που αφορούν ένα συγκεκριμένο URI, (β) η ανακάλυψη μιας πηγής που σχετίζεται με μία άλλη, (γ) ο υπολογισμός και η οπτικοποίηση του βαθμού διασύνδεσης μεταξύ δύο ή περισσότερων πηγών. Τα παραπάνω είναι αναγκαία στη διαδικασία ολοκλήρωσης σε ένα ανοικτό και εξελισσόμενο περιβάλλον.

Για να απαλύνουμε αυτό το πρόβλημα σε αυτήν την εργασία, παρουσιάζουμε μέτρα, ευρετήρια και αλγορίθμους που επιτρέπουν τη μέτρηση και ποσοτικοποίηση του βαθμού διασύνδεσης πολλών πηγών. Για λόγους κλιμακωσιμότητας προτείνουμε i) ένα ευρετήριο για τα προθέματα των URIs ii) έναν κατάλογο για σχέσεις ισοδυναμίας που λαμβάνει υπ' όψιν του το συμμετρικό και μεταβατικό κλείσιμο των σχέσεων ισοδυναμίας που εμφανίζονται στα σύνολα δεδομένων, iii) ένα σημασιολογικό ευρετήριο στοιχείων (που χρησιμοποιεί τα προαναφερθέντα ευρετήρια), iv) ένα πλέγμα (lattice) των κοινών στοιχείων που μετράει όλα τα κοινά στοιχεία ενός συνόλου πηγών, και v) δύο αυξητικούς αλγορίθμους που επιταχύνουν τον υπολογισμό του πλέγματος.

Εφαρμόζουμε και αξιολογούμε την προσέγγιση τόσο στο πλαίσιο μιας συγκεκριμένης σημασιολογικής αποθήκης δεδομένων με πληροφορίες για θαλάσσια είδη (όπου εκεί τα μέτρα αυτά χρησιμοποιούνται για την αξιολόγηση της αποθήκης και των συνιστωσών πηγών της, καθώς και για τον έλεγχο της ποιότητας της αποθήκης μετά από ανακατασκευή), καθώς και σε τρακόσες πηγές του νέφους διασυνδεδεμένων δεδομένων. Αναφέρουμε τα αποτελέσματα μετρήσεων που δεν έχουν γίνει στο παρελθόν (όπως το πλήθος της τομής των κοινών URIs μεταξύ τριών ή παραπάνω πηγών, συχνότητα των prefixes , κ.α.), προσφέρουμε νέες υπηρεσίες (όπως εύρεση ισοδύναμων URIs, εύρεση των κοντινότερων πηγών ως προς μία, κ.α.), και τέλος αξιολογούμε την επιτάχυνση που επιτυγχάνεται με τα προτεινόμενα ευρετήρια και αλγορίθμους. Τέλος, προτείνουμε μία επέκταση της οντολογίας VoID που επιτρέπει τη δημοσίευση, το διαμοιρασμό και την αξιοποίηση τέτοιων μετρήσεων.

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω θερμά τον επόπτη καθηγητή μου κ. Γιάννη Τζίτζικα για την άψογη συνεργασία, ορθή καθοδήγηση και ουσιαστική συμβολή του στην ολοκλήρωση της παρούσας μεταπτυχιακής εργασίας. Επίσης, θέλω να εκφράσω τις ευχαριστίες μου στον κ. Δημήτρη Πλεξουσάκη και στον κ. Κώστα Μαγκούτη για την προθυμία τους να συμμετέχουν στην τριμελή επιτροπή.

Ακόμα να ευχαριστήσω το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας για την πολύτιμη υποστήριξη σε υλικοτεχνική υποδομή και τεχνογνωσία, καθώς και για την υποτροφία που μου προσέφερε καθ΄ όλη τη διάρκεια της μεταπτυχιακής μου εργασίας.

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω την αγαπημένη μου Έφη για την αγάπη και τη στήριξη της και τις όμορφες στιγμές που περνάμε μαζί. Επίσης θέλω να ευχαριστήσω την οικογένεια μου για την αγάπη, για την υποστήριξη και την συμπαράσταση τους σε κάθε βήμα της ζωής μου. Έπειτα να ευχαριστήσω τα παιδιά από το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας που ήταν πάντα πρόθυμοι να με βοηθήσουν σε οποιαδήποτε απορία είχα. Ακόμα, θέλω να ευχαριστήσω τους φίλους μου για την υποστήριξη τους όλα αυτά τα χρόνια. Τέλος, η εργασία αυτή είναι αφιερωμένη στον παππού μου Αντώνη, που αποτελεί τον πιο θερμό υποστηρικτή μου.

στον παππού μου, Αντώνη

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Linked Data is a method for publishing structured data that allows them to be interlinked (by using URIs instead of simple values) for assisting their integration. A big number of such datasets (or sources), has already been published according to the principles of Linked data and their number and size keeps increasing. However, currently it is not evident how connected these datasets are and even some basic tasks are nowadays challenging because of the scale and heterogeneity of the datasets: the LODStats website provides statistics about approximately ten thousand discovered linked datasets until August 2014[1], and it keeps growing. To alleviate this problem in this thesis, we introduce metrics, indexes and algorithms which allow the computation and quantification of connectivity among several datasets. Such indexes and measurements are important in a plethora of tasks: (a) for obtaining *complete information* about one particular URI (or set of URIs) enriched with their provenance, (b) for obtaining measurements that are useful for *dataset discovery* and *dataset selection* [21, 51, 53], (c) for *assessing the connectivity* between any set of datasets for *quality checking* and for *monitoring their evolution* over time [47], (d) for constructing *visualizations* [9] that provide more informative overviews and could also aid dataset discovery.

An example of task (a) follows. Suppose that one user or an application wants to find all the available data associated with "http://www.dbpedia.org/Aristotle", including URIs being `sameAs` with this entity (i.e., object coreference), coming from multiple sources. This is not currently possible. With the proposed approach this is possible, and one can also get the provenance of the returned triples. This task is not trivial since the `sameAs` relationships model an equivalence relation and therefore its transitive closure has to be computed and this presupposes knowledge of all datasets. Dataset discovery and visualization, i.e. tasks (b) and (d), are emerging challenges for the web of data, for instance according to [31] "Being able to discover data from other sources, to rapidly integrate that data with one's own, and to perform simple analyses, often by eye (via visualizations), can lead to insights that can be important assets." Currently the community uses catalogs

[1] http://stats.lod2.eu

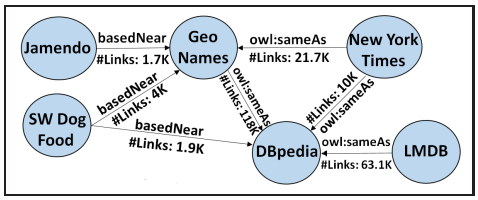


Figure 1.1: LargeRDFBench Cross Domain Datasets

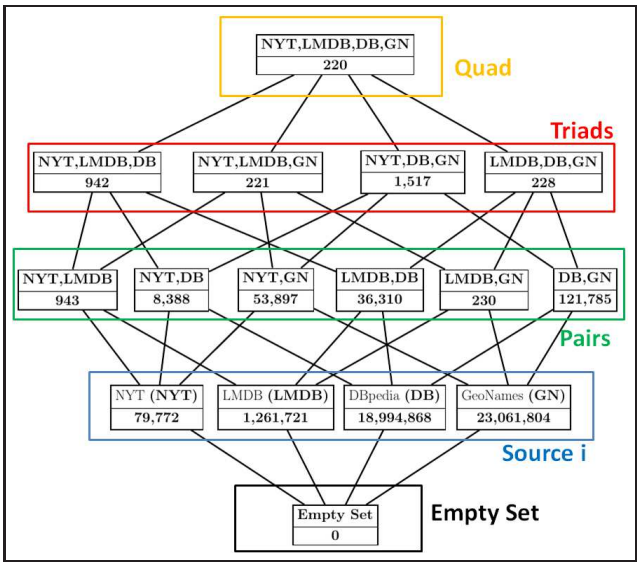


Figure 1.2: Lattice of four datasets showing the common Real world Objects

that contain some very basic metadata, and diagrams like the Linking Open Data cloud diagram[2], as well as LargeRDFBench[3] (an excerpt is shown in Figure 1.1).

These diagrams illustrate how many links exist between *pairs* of datasets, however they do not make evident if *three or more* datasets share any URI or Literal! Consequently, only upper bounds of intersections can be deduced, e.g. in Figure 1.1 which shows the intersection of links of multiple datasets, one can easily understand that since $|DBpedia \cap NewYorkTimes| = 10K$, it is implied that $0 \leq |DBpedia \cap NewYorkTimes \cap GeoNames| \leq 10K$ but this is rather a very coarse approximation of the size of the actual intersection. To fill this gap in this thesis we show how we can make efficiently measurements that involve more than two datasets.

The results can be visualized as Lattices, like that of Figure 1.2 which shows the lattice of the four datasets of Figure 1.1. From this lattice one can see the number of common real world objects in the triads of datasets, e.g. it is evident that the triad of *DBpedia*, *GeoNames* and *NYT* shares 1,517 real world objects,

[2]http://lod-cloud.net/
[3]http://github.com/AKSW/LargeRDFBench

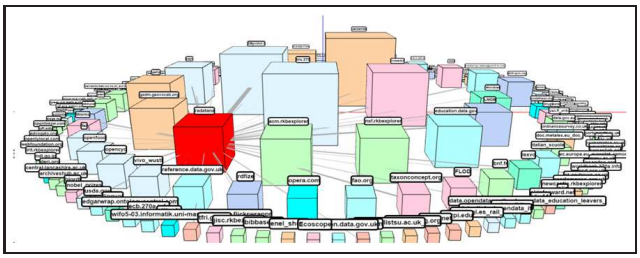


Figure 1.3: A 3D Visualization of 287 LOD datasets

and that there are 220 real world objects shared in all four datasets. Instead, the classical visualizations of the LOD cloud, like that of Figure 1.1, stops at the level of pairs. As another example in Figure 1.3, we show a visualization of 287 RDF datasets where each dataset is illustrated as a building and bridges are used to illustrate `sameAs` relationships between two datasets where the volume of each bridge is equal to the number of such relationships. If the transitive closure is not computed then the visualization will suffer for missing bridges (i.e. missing connections between datasets) and smaller in size bridges.

As regards *dataset discovery*, i.e. task (b), apart from visualization the proposed measurements can be directly used for answering queries like "get the K datasets that are more connected to a particular dataset" (without the proposed method such queries cannot be answered).

Nextly, for aiding the execution of all these important tasks we propose connectivity metrics, special indexes and algorithms. In subsection 1.1 we introduce what a *Semantic Warehouse* is and we show in brief the research questions and our contributions, which include a number of connectivity metrics for assessing the quality of the connectivity of a *Semantic Warehouse*. Moreover, we mention which are the main tasks that can be executed because of the proposed metrics for a warehouse. Then, in subsection 1.2 we mention that we generalize our approach for more and bigger datasets and we explain in brief why we need indexes and algorithms for computing such metrics when the number and size of datasets increase, while we notice the main tasks that are fulfilled because of the proposed approach. Finally, in subsection 1.3 we introduce ways to exploit the measurements for executing the four aforementioned tasks.

1.1 Connectivity of Semantic Warehouses

The spark for this work was the real semantic warehouse for the *marine* domain which harmonizes and connects information from different sources of marine information[4]. We use the term *Semantic Warehouse* (for short warehouse) to refer to a read-only set of RDF triples fetched (and transformed) from different sources that

[4]Used in the context of the projects iMarine (FP7 Research Infrastructures, 2011-2014), `http://www.i-marine.eu` and BlueBRIDGE (H2020 Research Infrastructures, 2015-2018), `http://www.bluebridge-vres.eu/`

aims at serving a particular set of query requirements. We we apply the proposed metrics Most past works have focused on the notion of conflicts (e.g., [44]), and have not paid attention to *connectivity*. In this thesis we introduce and evaluate upon real and synthetic datasets several metrics for quantifying the connectivity of a warehouse and we focus on the following questions:

- How to measure the value and quality (since this is important for e-science) of the warehouse?
- How to monitor its quality after each reconstruction or refreshing (as the underlying sources change)?
- How to understand the evolution of the warehouse?
- How to measure the contribution of each source to the warehouse, and hence deciding which sources to keep or exclude?

Regarding our contributions for assessing the connectivity of semantic warehouses:

- We propose "source-to-source" metrics
- We introduce metrics for evaluating the whole warehouse.
- We introduce various *single-valued* metrics for quantifying the contribution of a source or the value of the warehouse. To make easier and faster the identification and inspection of pathological cases (redundant sources or sources that do not contribute new information) we propose a single-valued metric that characterizes the overall contribution of a source. This value considers the contribution (in terms of triples and connectivity) of a source and its size, and it indicates the degree up to which the source contributes to the warehouse.
- We propose methods that exploit these metrics for understanding and monitoring the *evolution* of the warehouse.

The measurements above aid predominantly the execution of task (c): the assessment of connectivity between any set of datasets and secondarily the execution of task (d): the construction of visualizations, since we visualize the connectivity between the sources of a *Semantic Warehouse*.

1.2 Connectivity among several RDF Datasets

Afterwards, we generalize our approach for larger number of datasets that have been published in the LOD Cloud. However, it is very expensive to perform all these measurements straightforwardly. There are many datasets and some of them are very big comparing to a domain-specific semantic warehouse that contains usually a few number of datasets. Moreover, the possible combinations of datasets is exponential in number. To tackle this challenge and for achieving scalability, we introduce a set of indexes (and their construction algorithms) for speeding up such measurements. We show that with the proposed method it is feasible to perform such measurements even with one machine! More than one machines could be included for further speedup but this is not necessary since the task of measurement is not a daily activity. Nevertheless, the introduced method and its

experimental analysis paves the way for effectively parallelizing the task. In brief in this thesis:

- we introduce a *namespace-based prefix index* for speeding up the computation of the metrics,
- we introduce a *sameAs catalog* for computing the symmetric and transitive closure of the `sameAs` relationships encountered in the datasets s (the algorithm is based on incremental signatures allowing each pair of URIs to be read only once),
- we introduce a *semantics-aware element index* (that exploits the previous two indexes), and *two lattice-based incremental algorithms* for speeding up the computation of the intersection URIs of *any set* of datasets (the lattice can be used also for the visualization of commonalities if the number of datasets is low, and as a navigation mechanism if the number of datasets is high),
- we measure the speedup obtained by the proposed indexes and algorithms (just indicatively they enable computing the sameAs closure of 300 datasets in 45 seconds and the lattice of measurements of all possible sets of datasets that share (i) more than 30 URIs in 3.5 minutes and (ii) more than 20 URIs in 35 minutes), and we report connectivity measurements for a subset of the current LOD that comprises 300 datasets.

The measurements above aid the execution of all four tasks, since the indexes help us in task (a) (i.e., for obtaining information for a URI), the measurements and algorithms in task (b) (i.e., for discovering datasets and for aiding dataset selection) and in task (c) (i.e., for the assessment of connectivity between any set of sources) while the results of the measurements can be used as an input for constructing more informative visualizations (i.e. task (d)).

1.3 Exploitation of Metrics

Finally, we show ways to exploit, publish and visualize such measurements in order to be able to execute all the four tasks. More specifically, (a) we propose an extension of VoID that models all the proposed metrics and we use it for publishing exchanging an sharing such measurements, (b) we propose an alternative way of modeling the values of the proposed metrics using 3D models and (c) we show queries (which are accessible in a running SPARQL endpoint) that can aid dataset discovery and object coreference. The 3D visualization allows quickly understanding the situation of a warehouse and how the underlying sources contribute and are connected while it can also be used for modeling the evolution of a warehouse over time (using animations).

More specifically, we have published the measurements in *datahub.io*[5] using VoID [37] and VoIDWH [47] vocabularies, a set of queries for dataset discovery and object coreference are available in `www.ics.forth.gr/isl/LODsyndesis/`,

[5]http://datahub.io/dataset/connectivity-of-lod-datasets

while a prototype that exploits these measurements and provides an interactive 3D visualization is already accessible at `www.ics.forth.gr/isl/3DLod/` (by clicking on a dataset the user can see the connected datasets).

1.4 Outline of Thesis

The rest of this thesis is organized as follows:

Chapter 2 discusses the context and describes related work and what distinguishes the current one.

Chapter 3 introduces the quality metrics and demonstrates their use for semantic warehouses while it reports measurements over a real and operational semantic warehouse containing information about the marine domain.

Chapter 4 states the problem for LOD Cloud datasets, introduces the indexes and their construction algorithms while it shows how to exploit properties from lattice theory for computing the metrics for any set of datasets. Then, it reports measurements and experimental results over 300 datasets of the entire LOD cloud, and discuss the speedup obtained with the introduced indexes and algorithms.

Chapter 5 shows how one can publish, exchange and visualize the metrics.

Finally, Chapter 6 concludes and identifies directions for future research.

Publications Derived by this Thesis

Parts of this work were published predominantly in *Proceedings of the Very Large Databases Endowment (VLDB)* [48] and *International Journal on Semantic Web and Information Systems (ISJWIS)* [47], and secondarily (a) in the *International Workshop on Dataset Profiling & Federated Search for Linked Data* [46], (b) in the *Extended Semantic Web Conference (ESWC)* [65] and (c) in the *International Workshop on Linked Web Data Management (LWDM)* [64].

Chapter 2

Context and Related Work

2.1 Background

The Resource Description Framework (`RDF`) [38] is a graph-based data model for linked data interchanging on the web. RDF uses `Triples` in order to relate Uniform Resource Identifiers (`URIs`) or anonymous resources (`blank nodes`) where both of them denote a `Resource`, with other `URIs`, `blank nodes` or constants (`Literals`). Let $\mathcal{U}$ be the set of all URIs, $\mathcal{B}$ the set of all `blank nodes`, and $\mathcal{L}$ the set of all Literals. A *triple* is any element of $\mathcal{T} = (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$, while an *RDF graph* (or dataset) is any finite subset of $\mathcal{T}$. Linked Data [11] refers to a method of publishing structured data, so that it can be interlinked and become more useful through semantic queries, founded on HTTP, RDF and URIs . The linking of datasets is essentially signified by the existence of common URIs, referring to schema elements (defined through RDF Schema [1] and OWL [2]), or data elements. Since we focus on the second, hereafter we will consider only URIs that are either *subjects* or *objects* of triples whereas we ignore properties, since they are schema elements. However, we also consider `sameAs`-triples, where `sameAs` is a built-in OWL property for linking an individual to an individual meaning that both are equivalent i.e. they refer to the same real world object.

2.2 Context

The spark for this work was the recently completed *iMarine* project (and the ongoing *BlueBRIDGE* project) that offers an operational distributed infrastructure that serves hundreds of scientists from the marine domain. As regards semantically structured information, the objective was to integrate information from various marine sources, specifically from:

- **WoRMS** [5]: it is a marine registry containing taxonomic information and

[1] http://www.w3.org/TR/rdf-schema/
[2] http://www.w3.org/TR/owl2-overview/

lists of common names and synonyms for more than 200 thousand species in various languages.

- **Ecoscope** [1]: it is knowledge base containing geographical data, pictures and information about marine ecosystems.
- **FishBase** [2]: is a global database of fish species, containing information about the taxonomy, geographical distribution, biometrics, population, genetic data and many more.
- **FLOD** [3]: is a network of marine linked data containing identification information using different code lists.
- **DBpedia** [12]: is a knowledge base containing content that has been converted from Wikipedia, that by the time of writing this thesis, the English version contained more than 20 million resources.

The integrated warehouse[3] is operational and it is exploited in various applications, including the gCube infrastructure [13], or for enabling exploratory search services (e.g., `X-ENS` [22] that offers semantic post-processing of search results). For the needs of the *iMarine* and *BlueBRIDGE* projects, the materialized (warehouse) integration approach was more suited because it offers (a) flexibility in transformation logic (including ability to curate and fix problems), (b) decoupling of the release management of the integrated resource from the management cycles of the underlying sources, (c) decoupling of access load from the underlying sources, and (d) faster responses (in query answering but also in other tasks, e.g., in entity matching). Below we list the main functional and non functional requirements for constructing such warehouses.

Functional Requirements

- *Multiplicity of Sources.* Ability to query SPARQL endpoints (and other sources), get the results, and ingest them to the warehouse.
- *Mappings, Transformations and Equivalences.* Ability to accommodate schema mappings, perform transformations and create `sameAs` relationships between the fetched content for connecting the corresponding schema elements and entities.
- *Reconstructibility.* Ability to reconstruct the warehouse periodically (from scratch or incrementally) for keeping it fresh.

Non Functional Requirements

- *Scope control.* Make concrete and testable the scope of the information that should be stored in the warehouse. Since we live in the same universe, everything is directly or indirectly connected, therefore without stating concrete objectives there is the risk of continuous expansion without concrete objectives regarding its contents, quality and purpose.

[3]The warehouse can be accessed from `https://i-marine.d4science.org/`.

- *Connectivity assessment.* Ability to check and assess the connectivity of the information in the warehouse. Putting triples together does not guarantee that they will be connected. In general, connectivity concerns both schema and instances and it is achieved through common URIs, common literals and `sameAs` relationships. Poor connectivity affects negatively the query capabilities of the warehouse. Moreover, the contribution of each source to the warehouse should be measurable, for deciding which sources to keep or exclude (there are already hundreds of SPARQL endpoints).
- *Provenance.* More than one level of provenance can be identified and are usually required, e.g., warehouse provenance (from what source that triple was fetched), information provenance (how the fact that the x species is found in y water area was produced), and query provenance (which sources and how contributed to the answer of this query).
- *Consistency and Conflicts.* Ability to specify the desired consistency level of the warehouse, e.g., do we want to tolerate an association between a fish commercial code and more than one scientific names? Do we want to consider this as inconsistency (that makes the entire warehouse, or parts of it, unusable), or as resolvable (through a rule) conflict, or as a normal case (and allow it as long as the provenance is available).

2.2.1 Context: The Integration Process

For making clear the context, here we describe in brief the steps of the process that we follow for creating the warehouse. Figure 2.1 shows an overview of the warehouse contents, while Figure 2.3 sketches the construction process. For the construction (and the update) of the warehouse, we have used the tool *MatWare* [65], that automates the entire process.

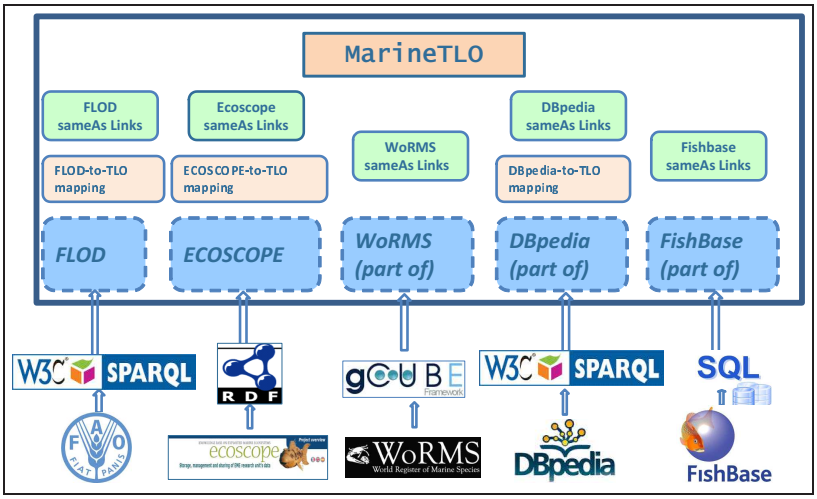


Figure 2.1: Overview of the warehouse.

(1) Define the requirements. The first step is to *define requirements* in terms of *competency queries*. It is a set of queries (provided by the community) indicating the queries that the warehouse is intended to serve. Figure 2.2 displays the textual description for some indicative competency queries as they were supplied by the communities. The full list of the competency queries is web accessible[4].

#Query	For a **scientific name** of a **species** (e.g. **Thunnus Albacares** or **Poromitra Crassiceps**), find/give me
Q_1	the biological environments (e.g. **ecosystems**) in which the **species** has been **introduced** and more general descriptive information of it (such as the **country**)
Q_2	its **common names** and their complementary info (e.g. **languages** and **countries** where they are used)
Q_3	the **water areas** and their **FAO codes** in which the **species** is **native**
Q_4	the **countries** in which the **species lives**
Q_5	the **water areas** and the **FAO** portioning **code** associated with a country
Q_6	the presentation w.r.t **Country**, **Ecosystem**, **Water Area** and **Exclusive Economical Zone** (of the water area)
Q_7	the projection w.r.t. **Ecosystem** and **Competitor**, providing for each competitor the **identification information** (e.g. several codes provided by different organizations)
Q_8	a map w.r.t. **Country** and **Predator**, providing for each predator both the **identification information** and the **biological classification**
Q_9	**who** discovered it, in which **year**, the **biological classification**, the **identification information**, the **common names** - providing for each common name the **language**, the **countries** where it is used in.

Figure 2.2: Some indicative competency queries.

It is always a good practice to have (select or design) a *top-level schema/ontology* as it alleviates the schema mapping effort (avoiding the combinatorial explosion of pair-wise mappings) and allows formulation of the competency queries using that ontology (instead of using elements coming from the underlying sources, which change over time). For our case in *iMarine*, we used the *MarineTLO* [63] ontology.

(2) Fetch. The next step is to *fetch the data* from each source and this requires using various access methods (including SPARQL endpoints, HTTP accessible files, JDBC) and specifying what exactly to get from each source (all contents or a specific part). For instance, and for the case of the *iMarine* warehouse, we fetch all triples from FLOD through its SPARQL endpoint, all triples from Ecoscope obtained by fetching OWL/RDF files from its Web page, information about species (ranks, scientific and common names) from WoRMS by accessing a specific-purpose service, called Species Data Discovery Service (SDDS) (provided by gCube infrastructure [13]), information about species from DBpedia's SPARQL endpoint, and finally information about species, water areas, ecosystems and countries from the relational tables of FishBase.

(3) Transform and (4) Ingest. The next step is to *transform and ingest* the

[4] `http://www.ics.forth.gr/isl/MarineTLO/competency_queries/MarineTLO_Competency_Queries_Version_v4.pdf`

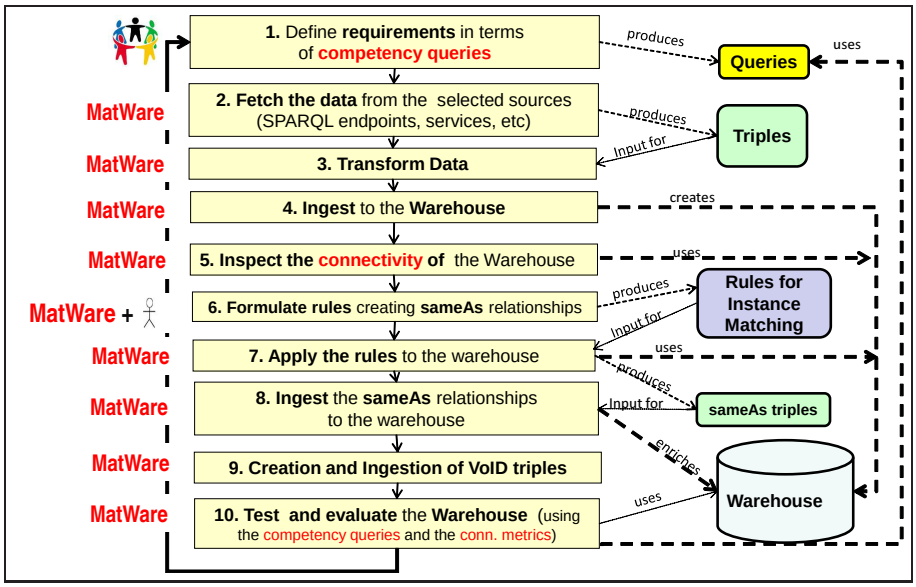


Figure 2.3: The process for constructing and evolving the warehouse.

fetched data. Some data can be stored as they are fetched, while others have to be transformed, i.e., a *format* transformation and/or a *logical* transformation has to be applied for being compatible with the top-level ontology. For example, a format transformation may be required to transform information expressed in DwC-A [69] (a format for sharing biodiversity data) to RDF. A logical transformation may be required for transforming a string literal to a URI, or for splitting a literal for using its constituents, or for creating intermediate nodes (e.g., instead of `(x,hasName,y)` to have `(x,hasNameAssignment,z), (z,name,y), (z,date,d)`, etc.). This step also includes the definition of the required *schema mappings* that are required for associating the fetched data with the schema of the top level ontology. Another important aspect for domain specific warehouses is the management of provenance [65]. In our case we support what we call "warehouse"-provenance, i.e., we store the fetched (or fetched and transformed) triples from each source in a separate *graphspace* (a graphspace is a named set of triples which can be used for restricting queries and updates in a RDF triple store). In this way we know which source has provided what facts and this is exploitable also in the queries.

As regards conflicts (e.g., different values for the same properties), the adopted policy in our case is to make evident the different values and their provenance, instead of making decisions, enabling thereby the users to select the desired values, and the content providers to spot their differences. The adoption of separate graphspaces also allows refreshing parts of the warehouse, i.e., the part that corresponds to one source. Furthermore, it makes feasible the computation of the metrics that are introduced in the next section.

(5) Inspect/Test the Connectivity. The next step is to *inspect and test the connectivity* of the "draft" warehouse. This is done through the competency queries

as well as through the metrics that we will introduce. The former (competency queries) require manual inspection, but automated tests are also supported. In brief, let q be a query in the set of competency queries. Although we may not know the "ideal" answer of q, we may know that it should certainly contain a particular set of resources, say Pos, and should not contain a particular set of resources, say Neg. Such information allows automated testing. If $ans(q)$ is the answer of q as produced by the warehouse, we would like to hold $Pos \subseteq ans(q)$ and $Neg \cap ans(q) = \emptyset$. Since these conditions may not hold in a real dataset or warehouse (due to errors, omissions, etc.), it is beneficial to adopt an IR-inspired evaluation, i.e., compute the *precision* and *recall* defined as: $precision = 1 - \frac{|Neg \cap ans(q)|}{|ans(q)|}$, $recall = \frac{|Pos \cap ans(q)|}{|Pos|}$. The bigger the values we get the better (ideally 1). The better we know the desired query behaviour, the bigger the sets Pos and Neg are, and consequently the more safe the results of such evaluation are.

(6) Formulate rules for Instance Matching. Based also on the results of the previous step, the next step is to *formulate rules for instance matching*, i.e., rules that can produce `sameAs` relationships for obtaining the desired connections. For this task we employ the tool SILK[5] [67].

(7) Apply the rules and (8) Ingest the derived `SameAs` relationships. Then, we *apply the instance matching rules* (SILK rules in our case) for producing, and then ingesting to the warehouse, `sameAs` relationships.

(9) Create and Ingest VoID triples.

For publishing and exchanging the characteristics of the warehouse (e.g., number of triples, endpoint, publisher, etc.), as well as the several metrics that we will introduce, we create and ingest to the warehouse triples based on an extension of `VoID` [37] that is described in [46] (`VoID` is an RDF-based schema that allows metadata about RDF datasets to be expressed).

(10) Test/Evaluate the Warehouse. Finally, we have to test the produced repository and evaluate it after ingesting the produced `sameAs` relationships. This is done through the competency queries and through the metrics that we will introduce.

Above we have described the steps required for the first time. After that, the warehouse is reconstructed periodically for getting refreshed content using *MatWare*. The metrics that we will introduce are very important for *monitoring* the warehouse after reconstructing it. For example by comparing the metrics in the past and new warehouse, one can understand whether a change in the underlying sources affected positively or negatively the quality (connectivity) of the warehouse.

[5] `http://wifo5-03.informatik.uni-mannheim.de/bizer/silk/`

2.3 Related Work

Data quality is commonly conceived as *fitness of use* for a certain application or use case [41,68]. The issue of data quality, especially for the case of a *data warehouse*, is older than the RDF world, e.g., the database community has studied it in the relational world [8, 58]. *Connectivity*, as defined in the Introduction, can be considered as a dimension of data quality in the context of a Semantic Warehouse. Recall that a Semantic Warehouse refers to a read-only set of RDF triples fetched and transformed from different sources that aims at serving a particular set of query requirements. Thereby, in this context *connectivity* is an important aspect of data quality aiming to measure the degree up to which the contents of the warehouse are connected (and thus satisfy its query requirements).

Fürber and Hepp [23] investigated data quality problems for RDF data originating from relational databases, while a systematic review of approaches for assessing the data quality of Linked Data is presented by Zaveri et al. [72]. In that work, the authors surveyed 21 approaches and extracted 26 data quality dimensions (such as *completeness*, *provenance*, *interlinking*, *reputation*, *accessibility*, and others) along with the corresponding metrics.

Below, we first (in §2.3.1) discuss some quality aspects that are especially useful for the case of a Semantic Warehouse, we report approaches that have tried to address them and we place our work in the literature. In §2.3.2 we compare (based on different perspectives) several frameworks and systems that automate quality assessment for the RDF world. Then, in §2.3.3 we show state-of-the-art approaches for performing *experiments in LOD scale*, and finally, in §2.3.4 we discuss engines that use *indexes* for searching or query answering.

2.3.1 Quality Aspects

Completeness. *Completeness* refers to the degree up to which all required information is presented in a particular dataset [72]. In the RDF world, completeness can be classified (according to Zaveri et al. [72]) as: *schema completeness* (degree to which the classes and properties of an ontology are represented), *property completeness* (measure of the missing values for a specific property), *population completeness* (percentage of all real-world objects of a particular type that are represented in the datasets), and *interlinking completeness* (degree to which instances in the dataset are interlinked).

The problem of assessing completeness of Linked Data sources was discussed by Harth and Speiser [26]. Darari et al. [16] introduce a formal framework for the declarative specification of *completeness statements* about RDF data sources and underline how the framework can complement existing initiatives like VoID. They also show how to assess completeness of query answering over plain and RDF/S data sources augmented with completeness statements, and they present an extension of the completeness framework for federated data sources.

Provenance. *Provenance* focuses on how to represent, manage and use information about the origin of the source to enable trust, assess authenticity and allow reproducibility [72]. Hartig [28] presents a provenance model for Web data which handles both data creation and data access. The author also describes options to obtain provenance information and analyzes vocabularies to express such information. Hartig and Zhao [29] propose an approach of using provenance information about the data on the Web to assess their quality and trustworthiness. Specifically, the authors use the provenance model described in [28] and propose an assessment method that can be adapted for specific quality criteria (such as accuracy and timeliness). This work also deals with missing provenance information by associating certainty values with calculated quality values. In [30], the same authors introduce a vocabulary to describe provenance of Web data as metadata and discuss possibilities to make such provenance metadata accessible as part of the Web of Data. Furthermore, they describe how this metadata can be queried and consumed to identify outdated information. Given the need to address provenance, the W3C community has standardised the PROV Model[6], a core provenance data model for building representations of the entities, people and processes involved in producing a piece of data or thing in the world. The PROV Family of Documents[7] [45] defines the model, corresponding serializations and other supporting definitions to enable the inter-operable interchange of provenance information in heterogeneous environments such as the Web.

Amount-of-data. *Amount-of-data* is defined as the extent to which the volume of data is appropriate for the task at hand [10, 72]. This dimension can be measured in terms of general dataset statistics like number of triples, instances per class, internal and external links, but also coverage (scope and level of detail) and metadata "richness". Tsiflidou and Manouselis [61] carried out an analysis of tools that can be used for the valid assessment of metadata records in a repository. More specifically, three different tools are studied and used for the assessment of metadata quality in terms of statistical analysis. However, such works do not consider the characteristics of RDF and Linked Data. Auer et al. [7] describe `LODStats`, a statement-stream-based approach for gathering comprehensive statistics (like classes/properties usage, distinct entities and literals, class hierarchy depth, etc.) about RDF datasets. To represent the statistics, they use VoID and the RDF Data Cube Vocabulary. The RDF Data Cube Vocabulary[8] [14] provides a means to publish multi-dimensional data (such as statistics of a repository) on the Web in such a way that it can be linked to related datasets and concepts. Hogan et al. [35] performed analysis in order to quantify the conformance of Linked Data with respect to Linked Data guidelines (e.g., use external URIs, keep URIs stable). They found that in most datasets, publishers followed some specific guidelines, such as using HTTP URIs, whereas in other cases, such as providing human readable metadata,

[6] `http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/`
[7] `http://www.w3.org/TR/prov-overview/`
[8] `http://www.w3.org/TR/2013/PR-vocab-data-cube-20131217/`

the result were disappointing since only a few publishers created metadata for their datasets.

Accuracy. *Accuracy* is defined as the extent to which data is correct, that is, the degree up to which it correctly represents the real world facts and is also free of errors [72]. Accuracy can be measured by detecting outliers, conflicts, semantically incorrect values or poor attributes that do not contain useful values for the data entries. Fürber and Hepp [24] categorize accuracy into semantic and syntactic accuracy. Semantic accuracy checks whether the data value represents the correct state of an object, whereas syntactic accuracy checks if a specific value violates syntactical rules. For measuring accuracy, the authors used three rules and four formulas, whereas the results were evaluated by using precision and recall measures. They managed to detect syntactic and semantic errors such as invalid country combinations, rules for phone numbers and so forth. *ODCleanStore* [40,44] names *conflicts* the cases where two different quads (e.g., triples from different sources) have different object values for a certain subject and predicate. To such cases conflict resolution rules are offered that either select one or more of these conflicting values (e.g., ANY, MAX, ALL), or compute a new value (e.g., AVG). Finally, Knap and Michelfeit [39] describe various quality metrics for scoring each source based on conflicts, as well for assessing the overall outcome.

Relevancy. *Relevancy* refers to the provision of information which is accordant with the task at hand and suitable to the users' query [72]. The existence of irrelevant data can have negative consequences for the query performance, while it will be difficult for the user to explore this data, since the user expects to receive the correct information. Zaveri et al. [71] divide relevancy (for DBpedia) into the following sub-categories: (i) extraction of attributes containing layout information, (ii) image related information, (iii) redundant attribute values, and finally (iv) irrelevant information. The existence of a number of different properties for a specific subject-object pair is an example of redundant information.

Dynamics / Evolution. *Dynamics* quantifies the evolution of a dataset over a specific period of time and takes into consideration the changes occurring in this period. Dividino et. al [20] lists probably all works related to the dynamics of LOD datasets. A related quality perspective, identified by Tzitzikas et al. [66], is that of the *specificity* of the ontology-based descriptions under ontology evolution, an issue that is raised when ontologies and vocabularies evolve over time.

Interlinking. *Interlinking* refers to the degree to which entities that represent the same concept are linked to each other [72]. This can be evaluated by measuring the existence of `sameAs` links and chains, the interlinking degree, etc. Zaveri et al. [71] classify interlinking into two different categories: (i) external websites (checking whether there are links among sources which are not available), and (ii) interlinks with other datasets (trying to detect incorrect mappings and links which do not provide useful information).

Our Placement: Connectivity

We use the term *connectivity* to express the degree up to which the contents of the semantic warehouse form a connected graph that can serve, ideally in a correct and complete way, the query requirements of the semantic warehouse, while making evident how each source contributes to that degree. The proposed connectivity metrics reflect the query capabilities of a warehouse as a whole (so they are important for evaluating its value), but also quantify the contribution of the underlying sources allowing evaluating the importance of each source for the warehouse at hand. Connectivity is important in warehouses whose schema is not small and consequently the queries contain paths. The longer such paths are, the more the query capabilities of the warehouse are determined by the connectivity.

In the related literature, the aspect of *connectivity* is not covered sufficiently and regards mainly the existence of `sameAs` links and chains (e.g., [71]). If we would like to associate *connectivity* with the existing quality dimensions, we could say that it is predominantly *interlinking* and secondly *relevancy* and *amount-of-data*. Of course, it can be exploited together with approaches that focus on *completeness* (e.g., [16]), *provenance* (e.g., [29]), *accuracy*, (e.g., [39]), etc. Regarding *relevancy*, we should stress that, by construction, a Semantic Warehouse as created by the proposed process (see Figure 2.3) does not contain irrelevant data since the data has been fetched based on the requirements defined in terms of competency queries. Furthermore, the proposed metrics can even detect redundant sources and sources containing data which are not connected with data found in the other sources. Compared to existing approaches on *amount-of-data* (like `LODStats` [7]), the proposed connectivity metrics can be used to gather statistics that regard more than one source (like common URIs, common literals, etc.) Finally, as regards *dynamics/evolution*, existing works (e.g., [20]) concern atomic datasets, not warehouses comprising parts of many datasets.

2.3.2 Frameworks/Systems for Quality Assessment

Here, we discuss frameworks/systems that automate quality assessment. At first we give a brief description of each framework/system and what quality aspects it can handle, and then we compare them regarding several aspects.

ODCleanStore [39,40,44] is a tool that can download content (RDF graphs) and offers various transformations for cleaning it (deduplication, conflict resolution), and linking it to existing resources, plus assessing the quality of the outcome in terms of *accuracy*, *consistency*, *conciseness* and *completeness*.

Sieve [43] is part of the Linked Data Integration Framework (LDIF) [4] and proposes metrics for assessing the dimensions in terms of *schema completeness*, *conciseness* and *consistency*. The role of this tool is to assess the quality by deciding which values to keep, discard or transform according to a number of metrics and functions which are configurable via a declarative specification language.

RDFUnit [42] measures the *accuracy* and the *consistency* of a dataset containing Linked Data. More specifically, it checks the correct usage of vocabularies according to a number of constraints (e.g., cardinality restriction on a property). One can use some custom SPARQL queries to quantify the quality of a specific dataset for the aforementioned aspects.

LinkQA [25] uses a number of metrics to assess the quality of Linked Data mappings regarding the dimensions of *interlinking* and *completeness*. This tool can be used for detected pathological cases, such as bad quality links, before they are published.

Luzzu [18] is a framework for assessing the quality of Linked data for 10 different dimensions, such as *availability*, *provenance*, *consistency* and so forth. In particular, by using this tool one can perform quality evaluation either by using some of the 25 available metrics or by defining his own metrics.

SWIQA [24] is a framework for the validation of the values of semantic resources based on a set of rules. This framework allows the calculation of quality scores for various dimensions, such as *completeness*, *timeliness* and *accuracy*, in order to identify possible problems with the data values. Moreover, it is also applicable on top of relational databases with the support of wrapping technologies (i.e., D2RQ).

Finally, *MatWare* [65] is a tool that automates the process of constructing semantic warehouses by fetching and transforming RDF triples from different sources. *MatWare* computes and visualizes the connectivity metrics that are proposed in this thesis.

Figure 2.4 illustrates the dimensions that the aforementioned approaches and *MatWare* measure.

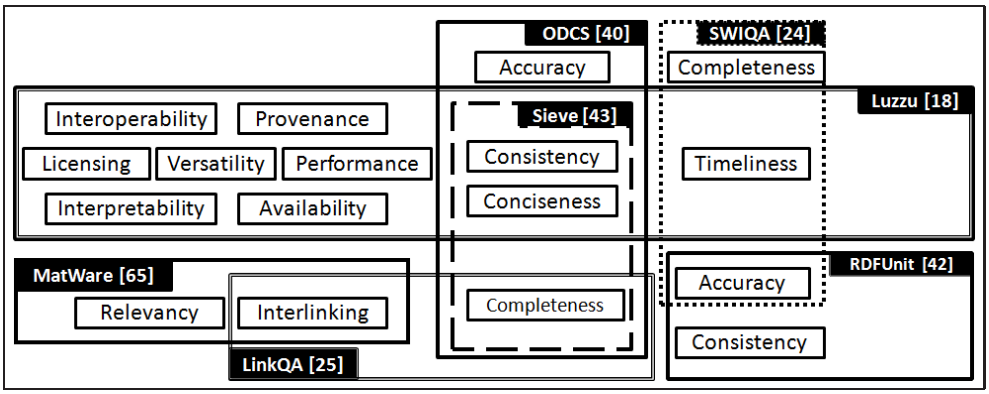


Figure 2.4: Existing frameworks and the dimensions they measure

Table 2.1 categorizes the aforementioned frameworks according to various aspects like data format, number of sources, output kind, output format, computability and extensibility. Firstly, *LinkQA* [25] takes as an input a set of Linked data Mappings and produces an HTML page containing the results of the measures which were computed through a Java Application. On the contrary, the input of *Luzzu* [18] is a specific dataset and the output is produced in

RDF format by using the Dataset Quality Ontology [17] and the Quality Problem Report Ontology (QPRO)[9]. In particular, they use `qpro:QualityReport` and `qpro:QualityProblem` in order to represent problems that emerged during the assessment of quality on a specific dataset. It is a common point with our approach, since *MatWare* can produce the results in RDF format by using an extension of VoID [46]. Concerning *ODCleanStore* [39, 40, 44], a common point with *MatWare* is that it computes the metrics for aggregated RDF data (not only for a specific source) and it produces the results in HTML and RDF. Regarding *Sieve* [43], it relies on a set of configurable metrics in order to assess the quality of an integrated dataset while the results are produced in the form of Quads. On the contrary, *SWIQA* [24] and *RDFUnit* [42] offer generalized SPARQL query templates for assessing the quality of the data, therefore, these metrics are domain-independent. Finally, compared to other tools, *MatWare* [65] offers a variety of output formats, including 3D visualization, whereas it can also be used for any domain.

	LinkQA [25]	Luzzu [18]	ODClean-Store [39]	Sieve [43]	SWIQA [24]	RDFUnit [42]	MatWare [65]
Input	RDF/XML	RDF/XML	RDF/XML	RDF/XML	RDF/XML	RDF/XML	RDF/XML
Num of Sources	Set of Mappings	One Source	Collection of Quads	One (integrated) Source	One Source	One Source	Set of Sources
Output Kind	Numeric values	Numeric values	Numeric values	Numeric values	Numeric values	Numeric values	Numeric values, 3D
Output Format	HTML	RDF	RDF, HTML	Quads	HTML	RDF, HTML	RDF, HTML, 3D
Computability	JAVA	JAVA	JAVA	JAVA	SPARQL	SPARQL	JAVA, SPARQL
Extensible	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 2.1: Categorizing Existing Tools

Moreover, we could mention systems for keyword searching over RDF datasets. Such systems can be very helpful for creating domain independent warehouses and offering user-friendly search and browsing capabilities. However, they are not closely related with semantic warehouses, since they aim at collecting everything, and do not have strict requirements as regards quality of data and query answering. Additionally, they do not measure quality aspects. For instance, the *Semantic Web Search Engine* (SWSE) [32] adapts the architecture of the common Web search engines for the case of structured RDF data (it offers crawling, data enhancing, indexing and a user interface for search, browsing and retrieval of RDF data). *Swoogle* [19] is a crawler-based indexing and retrieval system for the Semantic Web that uses multiple crawlers to discover Semantic Web Documents (SWDs) through meta-search and link-following, analyzes SWDs and produces metadata, computes ranks of SWDs using a rational random surfing model, and indexes SWDs using an information retrieval system. *Sindice* [52] is an indexing infrastructure with a Web front-end and a public API to locate Semantic Web data sources such as

[9] `http://butterbur04.iai.uni-bonn.de/ontologies/qpro/qpro`

RDF files and SPARQL endpoints. It offers an automatic way to locate documents including information about a given resource, which can be either a URI or full-text search. Finally, *Watson* [15] is a Semantic Web search engine providing various functionalities not only to find and locate ontologies and semantic data online, but also to explore the content of these semantic documents.

2.3.3 Measurements in LOD Scale.

Regarding approaches for performing such measurements in LOD Scale, the authors of [56] indexed thirty eight billion triples from over six hundred thousand documents, for cleaning them and providing statistics about the cleaned data. They index the resources and the namespaces, allowing one to find the documents in which a resource or a namespace belongs. A key difference with our approach is that even though we index the resources and the namespaces, we take into account the semantics, specifically the `sameAs` relationships. As regards the measurements we focus on connectivity, while [56] focuses on other aspects (like validity of documents). Moreover, they do not compute common URIs between three or more datasets. The authors of [49] created a portal for link discovery, called LinkLion, which contains mappings between pairs of 462 datasets. In that repository, one can find relationships between any pair of datasets. Similarly to our work, one can find relationships (e.g. `sameAs`) between two datasets and each dataset is represented as a whole. Comparing to our approach, they take into account only pairs of datasets and they do not index the URIs.

The authors in [57] focused on crawling a large number of datasets and on providing statistics for them. They categorize the datasets into eight different domains such as publications, geographical etc. Concerning connectivity, they provide measurements like the degree distribution of each dataset (how many datasets links to a specific dataset). Another work that analyzes a big number of linked datasets is LODStats [7]. In particular, they retrieve a number of documents and provide useful statistics about each document like the number of triples, number of `sameAs` links and so forth. Comparing our work with the previous two approaches, we focus on connectivity of datasets URIs (meaning that we provide more refined measurements) and we measure the connectivity among two or more datasets. Finally, the work [60] focuses only on features of the semantic web schemas, not on datasets.

2.3.4 Indexes for search and queries.

The work described in [33] proposes an object consolidation algorithm which analyses inverse functional properties in order to identify and merge equivalent instances in an RDF dataset. They build also an index for the `sameAs` relationships in order to find all the URIs for a real world object. This index is used in *YARS2* [27], a federated repository that queries linked data coming from different datasets. The system uses a number of indexes, such as a keyword index and a complete index on

quads in order to allow direct lookups on multiple dimensions without requiring join. *YARS2* is part of the Semantic Web Search Engine (*SWSE*) [34] which aims at providing an end-to-end entity-centric system for collecting, indexing, querying, navigating, and mining graph-structured Web data. Another popular system is *Swoogle* [19] which is a crawler-based indexing and retrieval system for the semantic web. It analyzes a number of documents, and provides an index for URIs and character N-Grams for answering user's queries or compute the similarity among a set of documents. A lookup index over resources crawled on the Semantic Web is presented in [62]. In particular, they provide an index containing for each URI the documents it occurs, a keyword index and an index that allows lookup of resources with different URIs identifying the same real world object. The work [50] describes an engine for scalable management of RDF data, called *RDF-3X* which is an implementation of SPARQL [55]. This system maintain indexes for all possible permutations of an RDF triple members (s p o) in six separate indexes while they store only the changes between triples instead of full triples. Comparing to our work, we mainly focus on finding how connected are the different subsets of the LOD Cloud, how to perform fast such measurements and how these measurements can be visualized while the focus of aforementioned work is not on speeding up measurements but on answering fast different types of user queries.

Chapter 3

Assessing the Connectivity of Semantic Warehouses

Def. 1 We use the term *connectivity metric* (or *connectivity measure*) to refer to a measurable quantity that expresses the degree up to which the contents of the semantic warehouse form a connected graph that can serve, ideally in a correct and complete way, the query requirements of the semantic warehouse, while making evident how each constituent source contributes to that degree. ⋄

They include measures of similarity between two sources in the form of percentages (e.g. regarding common URIs), natural numbers (e.g. cardinalities of intersections), matrices of measures, means of other measures, as well as relative measures (e.g. increase of average degrees, unique contribution and others). We use the term *metric* as it is used in software engineering (i.e. software metrics) and computer performance (i.e. computer performance metrics), and for being consistent with past works [46, 64]. What we call *metrics* could be also called *measures* and they should not be confused with distance functions.

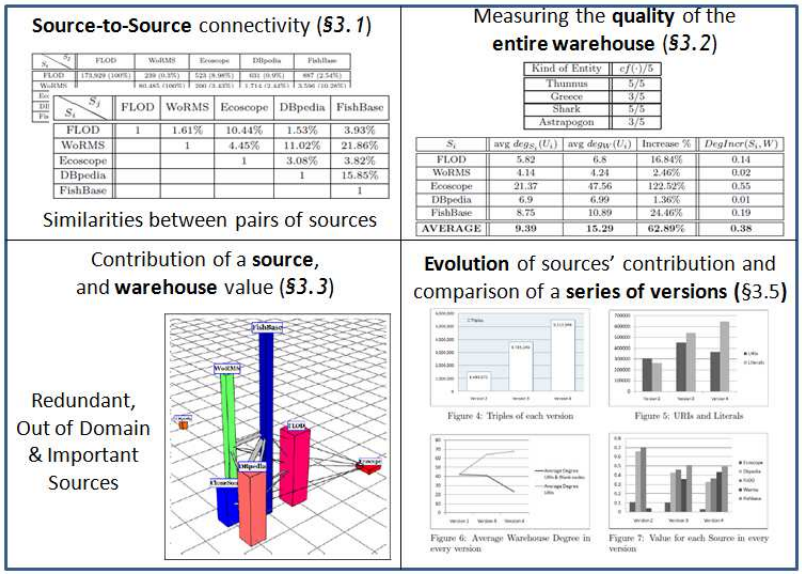
Such measures can assist humans on assessing in concrete terms the quality and the value offered by the warehouse. In addition they provide a summary of the contents of the warehouse which can be exploited by external applications in the context of distributed query answering.

To aid understanding in the sections that follow, after defining each metric we show the values of these metrics as computed over the `MarineTLO`-based warehouse which is built using data from five marine-related sources: FLOD, WoRMS, Ecoscope, DBpedia, and FishBase (cf. Figure 2.1). Since the warehouse is real and operational[1], this way of presentation also allows the reader to see how the metrics behave in a real setting.

This section is organized as follows: §3 introduces notations and discusses ways for comparing URIs, §3.1 introduces metrics for comparing *pairs of* sources, §§3.2

[1]In the evaluation of related tools, like *Sieve* [43] and *ODCleanStore* [44], real datasets have been used but not "real" operational needs. In our evaluation we use an operational warehouse with concrete (query) requirements which are described by the competency queries.

introduces metrics for quantifying the value of the entire warehouse, §3.3 introduces metrics for quantifying the value of one source (in the context of one warehouse). An overview of the categories of the various metrics and measurements is given in Figure 3.1.



S_i \ S_j	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	1	1.61%	10.44%	1.53%	3.93%
WoRMS		1	4.45%	11.02%	21.86%
Ecoscope			1	3.08%	3.82%
DBpedia				1	15.85%
FishBase					1

Kind of Entity	$cf(\cdot)/5$
Thunnus	5/5
Greece	3/5
Shark	5/5
Astrapogon	3/5

S_i	avg $deg_{S_i}(U_i)$	avg $deg_W(U_i)$	Increase %	$DegIncr(S_i, W)$
FLOD	5.82	6.8	16.84%	0.14
WoRMS	4.14	4.24	2.46%	0.02
Ecoscope	21.37	47.56	122.52%	0.55
DBpedia	6.9	6.99	1.36%	0.01
FishBase	8.75	10.89	24.46%	0.19
AVERAGE	**9.39**	**15.29**	**62.89%**	**0.38**

Figure 3.1: An overview of the categories of the proposed metrics.

Notations and Ways to Compare URIs

At first we introduce some required notations. Let $S = S_1, \ldots S_k$ be the set of underlying sources. Each contributes to the warehouse a set of triples (i.e., a set of `subject-predicate-object` statements), denoted by $triples(S_i)$. This is not the set of all triples of the source. It is the subset that is contributed to the warehouse (fetched mainly by running SPARQL queries). We shall use U_i to denote the URIs that appear in $triples(S_i)$. Hereafter, we consider only those URIs that appear as *subjects* or *objects* in a triple. We do not include the URIs of the properties because they concern the schema and this integration aspect is already tackled by the top level schema. Let W denote triples of all sources of the warehouse. In general, the set of all triples of the warehouse, say W_{All}, is superset of W (i.e., $W_{All} \supset W = \cup_{i=1}^{k} triples(S_i)$) because the warehouse apart from the triples from the sources, contains also the triples representing the top-level ontology, the schema mappings, the `sameAs` relationships, etc.

On Comparing URIs. For computing the metrics that are defined next, we need methods to compare URIs coming from different sources. There are more than one method, or policy, for doing so. Below we distinguish three main policies:

i *Exact String Equality.* We treat two URIs u_1 and u_2 as equal, denoted by $u_1 \equiv u_2$, if $u_1 = u_2$ (i.e., strings equality).

ii *Suffix Canonicalization.* Here we consider that $u_1 \equiv u_2$ if $last(u_1) = last(u_2)$ where $last(u)$ is the string obtained by a) getting the substring after the last "/" or "#", and b) turning the letters of the picked substring to lowercase and deleting the underscore letters as well as space and special characters that might exist. According to this policy:
`http://www.dbpedia.com/Thunnus_Albacares` ≡
`http://www.ecoscope.com/thunnus_albacares`
since their canonical suffix is the same, i.e., `thunnusalbacares`. Another example of equivalent URIs:
`http://www.s1.com/entity#thunnus_albacares` ≡
`http://www.s2.org/entity/thunnusAlbacares`.

iii *Entity Matching.* Here consider $u_1 \equiv u_2$ if u_1 `sameAs` u_2 according to the entity matching rules that are (or will be eventually) used for the warehouse. In general such rules create `sameAs` relationships between URIs. In our case we use SILK for formulating and applying such rules.

Note that if two URIs are equivalent according to policy [i], then they are equivalent according to [ii] too. Policy [i] is very strict (probably too strict for matching entities coming from different sources), however it does not produce any false-positive. Policy [ii] achieves treating as equal entities across different namespaces, however false-positives may occur. For instance, `Argentina` is a *country* (`http://dbpedia.org/resource/Argentina`) but also a *fish genus* (`http://dbpedia.org/resource/Argentina_(fish)`). Policy [iii] is fully aligned with the intended query behavior of the warehouse (the formulated rules are expected to be better as regards false-negatives and false-positives), however for formulating and applying these entity matching rules, one has to know the contents of the sources while it requires human effort. Consequently one cannot apply policy [iii] the first time, instead policies [i] and [ii] can be applied automatically (without any human effort). Policy [ii] should be actually used for providing hints regarding what entity matching rules to formulate. Below we define and compute the metrics assuming policy [ii], i.e., whenever we have a set operation we assume equivalence according to [ii] (e.g., $A \cap B$ means $\{ a \in A \mid \exists\ b \in B \text{ s.t. } a \equiv_{[ii]} b\}$). Then (after applying the entity matching rules), we compute the metrics according to Policy [iii], which is actually the policy that characterizes the query behavior of the final and operational warehouse.

3.1 Metrics for Comparing two Sources

3.1.1 Matrix of Percentages of Common URIs

The number of *common URIs* between two sources S_i and S_j, is given by $|U_i \cap U_j|$. We can define the *percentage of common URIs* (a value ranging $[0, 1]$), as follows:

$$curi_{i,j} = \frac{|U_i \cap U_j|}{\min(|U_i|, |U_j|)} \tag{3.1}$$

In the denominator we use $\min(|U_i|, |U_j|)$ although one could use $|U_i \cup U_j|$ that is used in the Jaccard similarity. With Jaccard similarity the integration of a small triple set with a big one would always give small values, even if the small set contains many URIs that exist in the big set, while the Jaccard similarity reveals the overall contribution of a source. We now extend the above metric and consider *all pairs of* sources aiming at giving an overview of the warehouse. Specifically, we compute a $k \times k$ matrix where $c_{i,j} = curi_{i,j}$. The higher values this matrix contains, the more glued its "components" are.

For the warehouse at hand, Table 3.1 shows the matrix of the common URIs (together with the corresponding percentages). We notice that the percentages range from 0.3% to 27.39%, while in some cases we have a significant percentage of common URIs between the different sources. The biggest intersection is between FishBase and DBpedia.

S_i \ S_j	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	173,929 (100%)	239 (0.3%)	523 (8.98%)	631 (0.9%)	887 (2.54%)
WoRMS		80,485 (100%)	200 (3.43%)	1,714 (2.44%)	3,596 (10.28%)
Ecoscope			5,824 (100%)	192 (3.3%)	225 (3.86%)
DBpedia				70,246 (100%)	9,578 (27.39%)
FishBase					34,974 (100%)

Table 3.1: Matrix of common URIs (with their percentages) using Policy [ii].

However, note that we may have 3 sources, such that each pair of them has a high *curi* value, but the intersection of the URIs of all 3 sources is empty. This is not necessarily bad, for example, consider a source contributing triples of the form `person-lives-placeName`, a second source contributing `placeName-has-postalCode`, and a third one contributing `postCode-addressOf-cinema`. Although these three sources may not contain even one common URI, their hosting in a warehouse allows answering queries: *"give me the cinemas in the area where the x person lives"*. On the other hand, in a case where the three sources contribute triples of the form `person-lives-placeName`, `person-worksAt-Organization` and `person-owns-car`, then it would be desired to have common URIs in all sources, since that would allow having more complete information for many persons. Finally, one might wonder why we do not introduce a kind of average path length or diameter for the warehouse. Instead of doing that, we inspect the paths that are useful for answering the queries of the users, and this is done through the competency queries.

Measurements after Adding the Rule-derived 'sameAs' Relationships and Applying the Transformation Rules

So far in the computation of the above metrics we have used policy [ii] (suffix canonicalized URIs) when comparing URIs. Here we show the results from computing again these metrics using policy [iii] and after adding the triples as derived from the transformation rules (described in §2.2.1). Moreover, extra URIs have been produced due to transformation rules (e.g., in order to assign a URI to a species name). As a result, now when comparing URIs, we consider the `sameAs` relationships that have been produced by the entity matching rules of the warehouse. In the current warehouse we have used 11 SILK rules. An indicative SILK rule is the following: *"If the value of the attribute "preflabel" of an Ecoscope individual (e.g., Thunnus albacares) in lower case is the same with the attribute "label" in latin of a FLOD individual (e.g., 'thunnus albacares'@la), then these two individuals are the same (create a* `sameAs` *link between them)"*. We should also note that in policy [ii], we have considered the triples as they are fetched from the sources. Computing the metrics using policy [iii], not only allows evaluating the gain achieved by these relationships, but it also reflects better the value of the warehouse since query answering considers the `sameAs` relationships.

Table 3.2 shows the matrix of the common URIs after the rule-derived relationships and the execution of the transformation rules (together with the corresponding percentages). We can see that, compared to the results of Table 3.1, after considering the `sameAs` relationships the number of common URIs between the different sources is significantly increased (more than 7 times in some cases).

Furthermore, Table 3.3 shows the Jaccard similarity between the pairs of sources. If we compare the results between these two tables, we can see that the percentages when using Jaccard similarity table have been reduced remarkably.

S_i \ S_j	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	190,749 (100%)	1,738 (2.64%)	869 (11.2%)	4,127 (5.46%)	6,053 (17.31%)
WoRMS		65,789 (100%)	809 (10.43%)	1,807 (2.75%)	4,373 (12.5%)
Ecoscope			7,759 (100%)	1,117 (14.4%)	2,171 (27.98%)
DBpedia				75,518 (100%)	10,388 (29.7%)
FishBase					34,973 (100%)

Table 3.2: Matrix of common URIs (and their percentages) using Policy [iii].

3.1.2 Matrix of Percentages of Common Literals between two Sources

The *percentage of common literals*, between two sources S_i and S_j can be computed by:

$$colit_{i,j} = \frac{|Lit_i \cap Lit_j|}{\min(|Lit_i|, |Lit_j|)} \tag{3.2}$$

S_i \ S_j	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	1	0.68%	0.44%	1.56%	2.69%
WoRMS		1	1.11%	1.29%	4.5%
Ecoscope			1	1.36%	5.35%
DBpedia				1	10.31%
FishBase					1

Table 3.3: Matrix of percentages of common URIs using Policy [iii] and Jaccard Similarity.

To compare 2 literals coming from different sources, we convert them to lower case, to avoid cases like comparing "Thunnus" from one source and "thunnus" from another. Additionally, we ignore the language tags (e.g., "*salmon*"@*en* ≡ "*salmon*"@*de*). Table 3.4 shows the matrix of the common literals (together with the corresponding percentages). We can see that, as regards the literals, the percentages of similarity are even smaller than the ones regarding common URIs. The percentages range from 2.71% to 12.37%.

S_i \ S_j	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	111,164 (100%)	3,624 (7.1%)	1,745 (12.37%)	5,668 (5.1%)	9,505 (8.55%)
WoRMS		51,076 (100%)	382 (2.71%)	2,429 (4.76%)	4,773 (9.34%)
Ecoscope			14,102 (100%)	389 (2.76%)	422 (2.99%)
DBpedia				123,887 (100%)	14,038 (11.33%)
FishBase					138,275 (100%)

Table 3.4: Matrix of common Literals (and their percentages).

3.1.3 Matrix of the Harmonic Mean of Common URIs and Literals

We can define a single metric by combining the previous two metrics. More specifically, we can define the harmonic mean of the above two metrics.

$$cUrisLit_{i,j} = \frac{2 * curi_{i,j} * colit_{i,j}}{curi_{i,j} + colit_{i,j}} \tag{3.3}$$

Table 3.5 presents the results of this metric.

3.2 Metrics for Evaluating the Entire Warehouse

Here we introduce metrics that measure the quality of the entire warehouse.

S_i \ S_j	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	1	1.61%	10.44%	1.53%	3.93%
WoRMS		1	4.45%	11.02%	21.86%
Ecoscope			1	3.08%	3.82%
DBpedia				1	15.85%
FishBase					1

Table 3.5: Harmonic Mean of Common URIs and Literals.

3.2.1 Increase in the Average Degree

Now we introduce another metric for expressing the degree of a set of nodes, where a node can be either a URI or a blank node[2]. Let E be the entities of interest (or the union of all URIs and blank nodes).

If T is a set of triples, then we can define the *degree* of an entity e in T as: $deg_T(e) = |\{(s,p,o) \in T \mid s = e \text{ or } o = e\}|$, while for a set of entities E we can define their average degree in T as $deg_T(E) = avg_{e \in E}(deg_T(e))$. Now for each source S_i we can compute the average degree of the elements in E considering $triples(S_i)$. If the sources of the warehouse contain common elements of E, then if we compute the degrees in the graph of W (i.e., $deg_W(e)$ and $deg_W(E)$), we will get higher values. So the increase in the degree is a way to quantify the gain, in terms of connectivity, that the warehouse offers. Furthermore, we can define a normalized metric for average degree increment i.e., a metric whose value approaches 1 in the best case, and 0 in the worst. To this end, we define

$$DegIncr(S_i, W) = \frac{deg_W(U_i) - deg_{S_i}(U_i)}{deg_W(U_i)} \tag{3.4}$$

For each source S_i, Table 3.6 shows the average degree of its URIs and blank nodes, and the average degree of the same URIs and blank nodes in the warehouse graph. It also reports the increment percentage, and the normalized metric for the average degree increment. The last row of the table shows the average values of each column. We observe that the average degree is increased from 9.39 to 15.29.

S_i	avg $deg_{S_i}(U_i)$	avg $deg_W(U_i)$	Increase %	$DegIncr(S_i, W)$
FLOD	5.82	6.8	16.84%	0.14
WoRMS	4.14	4.24	2.46%	0.02
Ecoscope	21.37	47.56	122.52%	0.55
DBpedia	6.9	6.99	1.36%	0.01
FishBase	8.75	10.89	24.46%	0.19
AVERAGE	**9.39**	**15.29**	**62.89%**	**0.38**

Table 3.6: Average degrees in sources and in the warehouse using policy [ii].

[2]In our case the size of blank nodes in the warehouse is much bigger than the size of unique triples (about twice).

Measurements after Adding the Rule-derived 'sameAs' Relationships and Applying the Transformation Rules

Table 3.7 shows the average degree of the URIs and blank nodes of each source S_i, and the average degree of the same URIs and blank nodes in the warehouse graph, when policy [iii] is used. It also reports the increment percentage, and the normalized metric for the average degree increment. The last row of the table shows the average values of each column. We can see that the average degree, of all sources, after the inclusion of the `sameAs` relationships is significantly bigger than before. In comparison to Table 3.6, the increase is from 2 to 9 times bigger. This means that we achieve a great increase in terms of the connectivity of the information in the warehouse.

S_i	avg $deg_{S_i}(U_i)$	avg $deg_W(U_i)$	Increase %	$DegIncr(S_i, W)$
FLOD	5.82	52.01	739.64%	0.89
WoRMS	4.14	8.19	97.94%	0.49
Ecoscope	21.37	90.52	323.51%	0.76
DBpedia	6.9	42.97	523.23%	0.84
FishBase	8.71	18.99	117.19%	0.54
AVERAGE	**9.39**	**42.53**	**353%**	**0.78**

Table 3.7: Average degrees in sources and in the warehouse using Policy [iii].

3.2.2 Unique Triples Contribution

We now define metrics for quantifying the complementarity of the sources. The "contribution" of each source S_i can be quantified by counting the triples it has provided to the warehouse, i.e., by $|triples(S_i)|$. We can also define its "unique contribution" by excluding from $triples(S_i)$ those belonging to the triples returned by the other sources. Formally, for the k sources of the warehouse, we can define:

$$triplesUnique(S_i) = triples(S_i) \setminus (\cup_{1 \leq j \leq k, j \neq i} triples(S_j)) \tag{3.5}$$

It follows that if a source S_i provides triples which are also provided by other sources, then we have $triplesUnique(S_i) = \emptyset$. Consequently, and for quantifying the contribution of each source to the warehouse, we can compute and report the number of its triples $|triples(S_i)|$, the number of the unique triples $|triplesUnique(S_i)|$, the unique contribution of each source as:

$$UniqueTContrib(S_i) = \frac{|triplesUnique(S_i)|}{|triples(S_i)|} \tag{3.6}$$

Obviously, it becomes 0 in the worst value and 1 in the best value. To count the unique triples of each source, for each triple of that source we perform suffix canonicalization on its URIs, convert its literals to lower case, and then we check if the resulting (canonical) triple exists in the canonical triples of a different source. If not, we count this triple as unique. Let $triplesUniques$ be the union of the

unique triples of all sources, i.e., $triplesUniques = \cup_i triplesUnique(S_i)$. This set can be proper subset of W (i.e., $triplesUniques \subset W$), since it does not contain triples which have been contributed by two or more sources.

Table 3.8 shows for each source the number of its triples ($|triples(S_i)|$), the number of unique triples ($|triplesUnique(S_i)|$), and the unique triples contribution of that source ($UniqueTContrib(S_i)$). We can see that every source contains a very high ($> 99\%$) percentage of unique triples, so we can conclude that all sources are important.

S_i	$\|triples(S_i)\|$	$\|triplesUnique(S_i)\|$	$UniqueTContrib(S_i)$
FLOD	665,456	664,703	99.89%
WoRMS	461,230	460,741	99.89%
Ecoscope	54,027	53,641	99.29%
DBpedia	450,429	449,851	99.87%
FishBase	1,425,283	1,424,713	99.96%

Table 3.8: (Unique) triple contributions of the sources using policy [ii].

Measurements after Adding the Rule-derived 'sameAs' Relationships and Applying the Transformation Rules

As regards the unique contribution of each source using Policy [iii], Table 3.9 shows the number of the triples of each source ($|triples(S_i)|$), the number of unique triples ($|triplesUnique(S_i)|$), and the unique triples contribution ($UniqueTContrib(S_i)$). We observe that the values in the first column are increased in comparison to Table 3.8. This is because of the execution of the transformation rules after the ingestion of the data to the warehouse, which results to the creation of new triples for the majority of sources. Finally we observe that, in general, the unique triples contribution of each source is decreased. This happens because the transformation rules and the same-as relationships have turned previously different triples, the same.

S_i	$\|triples(S_i)\|$	$\|triplesUnique(S_i)\|$	$UniqueTContrib(S_i)$
FLOD	810,301	798,048	98.49%
WoRMS	528,009	527,358	99.88%
Ecoscope	138,324	52,936	38.27%
DBpedia	526,016	517,242	98.33%
FishBase	1,425,283	1,340,968	94.08%

Table 3.9: (Unique) triple contributions of the sources using Policy [iii].

3.2.3 Complementarity of Sources

We now define another metric for quantifying the value of the warehouse for the entities of interest. With the term "entity" we mean any literal or URI that contains a specific string representing a named entity, like the name of a fish or

country. The set of triples containing information about the entity of interest can be defined as $triples_W(e) = |\{< s, p, o >\in W \mid s = e \text{ or } o = e\}|$. Specifically we define the *complementarity factor* for an entity e, denoted by $cf(e)$, as the percentage of sources that provide *unique* material about e. It can be defined declaratively as:

$$cf(e) = \frac{|\{\ i \mid triples_W(e) \cap triplesUnique(S_i) \neq \emptyset\}|}{|S|} \tag{3.7}$$

where S is the set of underlying sources.

Note that if $|S| = 1$ (i.e., we have only one source), then for every entity e we will have $cf(e) = 1.0$. If $|S| = 2$, i.e., if we have two sources, then we can have the following cases:
$-$ $cf(e) = 0.0$, if both sources have provided the same triple (or triples) about e or no source has provided any triple about e,
$-$ $cf(e) = 0.5$, if the triples provided by the one source (for e) are subset of the triples provided by the other, or if only one source provide triple(s) about e,
$-$ $cf(e) = 1.0$, if each source has provided at least one different triple for e (of course they can also have contributed common triples). Consequently for the entities of interest we can compute and report the average *complementarity factor* as a way to quantify the value of the warehouse for these entities.

Table 3.10 shows (indicatively) the *complementarity factors* for a few entities which are important for the problem at hand. We can see that for the entities "Thunnus" and "Shark" each source provides unique information ($cf = 1.0$). For the entity "Greece" and "Astrapogon" we obtain unique information from three sources ($cf = 3/5 = 0.6$). The fact that the complementarity factor is big means that the warehouse provides unique information about each entity from many/all sources. Moreover, Table 3.11 shows the average complementarity factor of the species that are native to Greece. One can observe that there are no species with very small complementarity factor, which means that at least 2 sources provide unique information for each species $cf(e) \geq 0.4$. Indeed, exactly 2 sources provide unique information for 116 species, while for 35 species we get unique data from all the sources. In general the average complementarity factor for all species that are native in Greece is approximately 0.63 (3.15/5) (meaning that at least 3 sources contain unique information for such species).

Kind of Entity	$cf(\cdot)$
Thunnus	1.0 (5/5)
Greece	0.6 (3/5)
Shark	1.0 (5/5)
Astrapogon	0.6 (3/5)

Table 3.10: Complementarity factor (cf) of some entities.

$cf(\cdot)$	No. of Species
0.2 (1/5)	0
0.4 (2/5)	116
0.6 (3/5)	180
0.8 (4/5))	113
1.0 (5/5)	35
Average: 0.63 (3.15/5)	Sum: 444

Table 3.11: cf of species that are native to Greece.

3.3 Metrics for Evaluating a Single Source

In this section we focus on metrics for quantifying the value that a source brings to the warehouse. Such metrics should also allow identifying pathological cases (e.g., redundant or irrelevant sources). In particular, §3.3.1 provides examples of such cases and introduces rules for identifying them, while §3.3.2 introduces a single-valued metric based on these rules for aiding their identification by a human.

3.3.1 Detecting Redundancies or other Pathological Cases

The metrics can be used also for detecting various pathological cases, e.g., sources that do not have any common URI or literal, or "redundant sources". To test this we created three artificial sources, let us call them *Airports*, *CloneSource* and *AstrapogonSource*. The *Airports* source contains triples about airports which were fetched from the DBpedia public SPARQL endpoint, the *CloneSource* is a subset of Ecoscope's and DBpedia's triples as they are stored in the warehouse, and the *AstrapogonSource* contains only 1 URI and 4 triples for the entity *Astrapogon*. In the sequel, we computed the metrics for 8 sources.

Table 3.12 shows the unique triples and Table 3.13 shows the average degrees. The metrics were calculated according to policy [iii]. As regards *Airports*, we can see that its unique contribution is 100% (all the contents of that source are unique). As regards *CloneSource* we got 0 unique contributions (as expected, since it was composed from triples of existing sources). Finally, concerning the *AstrapogonSource*, we noticed that although the number of triples contribution is very low, all its triples are unique.

S_i	$\|triples(S_i)\|$	$\|triplesUnique(S_i)\|$	$UniqueTContrib(S_i)$
FLOD	810,301	798,048	98.49%
WoRMS	528,009	527,358	99.88%
Ecoscope	138,324	17,951	12.9%
DBpedia	526,016	505,013	96%
FishBase	1,425,283	1,340,968	94.08%
AstrapogonSource	4	4	**100.00%**
CloneSource	56,195	0	**0%**
Airports	31,628	31,628	**100%**

Table 3.12: (Unique) triple contributions of the sources.

Rules for Detecting Pathological Cases. It follows that we can detect pathological cases using two rules: (a) if the average increase of the degree of the entities of a source is low, then this means that its contents are not connected with the contents of the rest of the sources (this is the case of *Airports* where we had only 0.1% increase), (b) if the unique contribution of a source is very low (resp. zero), then this means that it does not contribute significantly (resp. at all) to the warehouse (this is the case of *CloneSource* where the unique contribution was zero).

S_i	avg $deg_{S_i}(U_i)$	avg $deg_W(U_i)$	Increase %	$DegIncr(S_i, W)$
FLOD	5.82	52.01	739.64%	0.89
WoRMS	4.14	8.19	97.94%	0.49
Ecoscope	21.37	90.52	323.51%	0.76
DBpedia	6.9	42.97	523.23%	0.84
FishBase	8.71	18.99	117.19%	0.54
AstrapogonSource	4.0	57	1325%	0.93
CloneSource	6.47	60.39	833.08%	0.89
Airports	7.55	7.56	**0.1%**	0.001
AVERAGE	8.12	42.23	420.07%	0.8

Table 3.13: Average degrees in sources and in the warehouse.

3.3.2 A Single Metric for Quantifying the Value of a Source

To further ease the inspection of pathological cases (and the quantification of the contribution of each source), we can define a single (and single-valued) measure. One method is to use the *harmonic mean* of the unique contribution, and the increment in the average degree (the harmonic mean takes a high value if both values are high). Therefore, we can measure the harmonic mean of the above two metrics and define the value of a source S_i, denoted by $value_0(S_i, W)$, as:

$$value_0(S_i, W) = \frac{2 * UniqueTContrib(S_i) * DegIncr(S_i, W)}{UniqueTContrib(S_i) + DegIncr(S_i, W)} \tag{3.8}$$

Table 3.14 shows these values for all sources of the warehouse, including the artificial ones, in decreasing order. We can see that the problematic sources have a value less than 0.04 while the good ones receive a value greater than 0.2. However, *AstrapogonSource* has the highest score although it contains only 4 triples. The two reasons why this source seems the best according to this metric are that all the triples are unique and the only instance that it contains has a lot of properties in other sources. Therefore, the degree increment of this source is almost 1. Consequently this metric makes evident the contribution of each source to the warehouse.

S_i	$UniqueTContrib(S_i)$	$DegIncr(S_i, W)$	$value_0(S_i, W)$
AstrapogonSource	1	0.93	**0.9637**
FLOD	0.9849	0.89	**0.935**
DBpedia	0.96	0.84	**0.896**
FishBase	0.9408	0.54	**0.686**
WoRMS	0.9988	0.49	**0.6575**
Ecoscope	0.129	0.76	**0.2206**
Airports	1	0.001	**0.02**
CloneSource	0	0.89	**0**

Table 3.14: The value of a source in the Warehouse (using $value_0(S_i, W)$).

Although the above metric is good for discriminating the good from the not as good (or useless) sources, it ignores the number of triples that each source contributes. This is evident from Table 3.14 where AstrapogonSource gets the

highest score. In general, a source with a small number of triples can have big values in the above two metrics.

For tackling this issue, we need an analogous metric for the size of a specific source in the warehouse, specifically we can define $S_iSizeInW(S_i, W) = \frac{|triples(S_i)|}{|triples(W)|}$. We can now compute the harmonic mean of these three metrics and define the value of a source S_i, denoted by $value_1(S_i, W)$, as

$$value_1(S_i, W) = \frac{3}{\frac{1}{UniqueTContrib(S_i)} + \frac{1}{DegIncr(S_i,W)} + \frac{1}{S_iSizeInW(S_i,W)}} \quad (3.9)$$

Table 3.15 shows these values for all sources of the warehouse, including the artificial ones, in decreasing order. Now we can see that FishBase is the most useful source, and the score of AstrapogonSource is very low (almost 0).

Consequently, the first metric can be used for deciding whether to include or not a source in the warehouse, while second for inspecting the importance of source for the warehouse. In case of adding a huge out-of-domain source in our warehouse while there exist a lot of useful sources which are much smaller, the values of the useful sources will remain almost stable for the first metric. On the contrary, their values will be decreased for the second metric. Regarding, the value of the out-of-domain source, it will be low in both metrics, since the increase of the average degree for this source will be almost 0. Therefore, both metrics will show that the new source should be removed from the warehouse, however, the second metric will not show the real value for each of the remaining sources in this case.

S_i	$UniqueTContrib(S_i)$	$DegIncr(S_i, W)$	$S_iSizeInW(S_i, W)$	$value_1(S_i, W)$
FishBase	0.9408	0.54	0.405	**0.5572**
FLOD	0.9849	0.89	0.2304	**0.463**
DBpedia	0.96	0.84	0.1496	**0.3364**
WoRMS	0.9988	0.49	0.1501	**0.3091**
Ecoscope	0.129	0.76	0.0393	**0.0869**
Airports	1	0.001	0.0089	**0.0027**
AstrapogonSource	1	0.93	0.000001	**0.000001**
CloneSource	0	0.89	0.0089	**0**

Table 3.15: The value of a source in the warehouse (using $value_1(S_i, W)$).

3.4 Summary of the Metrics

Table 3.16 shows all the definitions of the metrics that were described in this section.

3.4.1 Computing the Connectivity Metrics using SPARQL queries

Here we show how the values of the connectivity metrics can be computed using SPARQL queries (this corresponds to the process "2. Compute Metrics" of Figure 5.1). Some of the SPARQL queries contain some specific rules (e.g., `define`

Metric Name	Metric Definition	Kind
Common URIs between two sources S_i and S_j	$= \|U_i \cap U_j\|$	natural number
Percentage of common URIs between S_i and S_j	$curi_{i,j} = \frac{\|U_i \cap U_j\|}{\min(\|U_i\|,\|U_j\|)}$	degree of similarity
Common literals between S_i and S_j	$= \|Lit_i \cap Lit_j\|$	natural number
Percentage of common literals between S_i and S_j	$colit_{i,j} = \frac{\|Lit_i \cap Lit_j\|}{\min(\|Lit_i\|,\|Lit_j\|)}$	degree of similarity
Harmonic mean of common URIs and literals	$cUrisLit_{i,j} = \frac{2 * curi_{i,j} * colit_{i,j}}{curi_{i,j} + colit_{i,j}}$	mean
Increase in the average degree	$DegIncr(S_i, W) = \frac{deg_W(E) - deg_S(E)}{deg_S(E)}$	relative measure
Unique triples of S_i	$triplesUnique(S_i)$ $= triples(S_i) \setminus (\cup_{1 \le j \le k, j \neq i} triples(S_j))$	relative measure
Unique triples contribution of a source	$UniqueTContrib(S_i) = \frac{\|triplesUnique(S_i)\|}{\|triples(S_i)\|}$	relative measure
Value of a source	$value_1(S_i, W) = \frac{3}{\frac{1}{UT} + \frac{1}{DI} + \frac{1}{SW}}$ where $UT = UniqueTContrib(S_i)$, $DI = DegIncr(S_i, W)$, $SW = S_i SizeInW(S_i, W)$	mean
Complementarity factor of an entity e	$cf(e) = \frac{\|\{ i \mid triples_W(e) \cap triplesUnique(S_i) \neq \emptyset \}\|}{\|S\|}$	percentage

Table 3.16: Connectivity Metrics.

`input:same-as "yes"`) these rules are used by Virtuoso[3] for enabling inference for triples containing `owl:sameAs` relations.

Common URIs. The metric *Common URIs* over two sources S_i and S_j, can be computed with the following query:

```
SELECT COUNT (DISTINCT ?o)
WHERE { GRAPH :Si {{?s1 ?p1a ?o} UNION {?o ?p1b ?o1}} . FILTER(isURI(?o))
        GRAPH :Sj {{?s2 ?p2a ?o} UNION {?o ?p2b ?o2}}  }
```

In the context of the warehouse, this metric should be computed over all pairs of sources, i.e., all (S_i, S_j) such that $S_i, S_j \in S$ and $i \neq j$. Note that this metric is symmetric, i.e., the value of the pair (S_i, S_j) is equal to the value of (S_j, S_i).

Regarding policy [iii], one can use the following query in order to collect all the matching pairs between 2 sources.

```
DEFINE input:same-as "yes"
SELECT DISTINCT (bif:concat(?o,'\t',?s)) as ?matchingPair
FROM :Si
FROM :Sj
FROM :SameAs
WHERE {
    GRAPH :SameAs {{?s ?p ?o}  UNION {?o ?p ?s} }
    .GRAPH :Si
    {{?s ?p1 ?o1}  UNION {?o1 ?p1 ?s} .filter(isURI(?s))}
    .GRAPH :Sj
    {{?o ?p2 ?o2}  UNION {?o2 ?p2 ?o} } . filter(isURI(?o)) }
```

Common Literals. The *Common Literals* between two sources S_i and S_j can be computed in a similar manner, i.e.:

```
SELECT COUNT DISTINCT ?o
WHERE { graph :Si { ?s ?p ?o} . FILTER(isLiteral(?o))
        graph :Sj { ?a ?b ?o} }
```

Again, this metric should also be computed over all pairs (S_i, S_j) of the warehouse.

Unique Triples Contribution. To compute the unique triple contribution of a source, say S_1, to the warehouse $S = S_1, \ldots, S_k$, we have to count the number of triples of S_1 that do not intersect with the triples of any of the other sources of S (i.e., with none of the sources in S_2 ... S_n). This can be done using the following query:

```
SELECT COUNT(*)
WHERE { graph :S1 { ?s ?p1 ?o} .
    FILTER NOT EXISTS { graph :S2 { ?s ?p2 ?o} } .
                    ...
                    ...
    FILTER NOT EXISTS { graph :Sn { ?s ?pn ?o} } }
```

[3] http://virtuoso.openlinksw.com/

On the contrary, concerning policy[ii] one can use the following SPARQL query in order to get the suffixes of the subject and object of each triple for every source. After that, one can use the produced sets in order to calculate the unique triples for each source.

```
SELECT bif:lower(bif:regexp_substr('[^#|/]+$',?s,0)) as ?s
       bif:lower(bif:regexp_substr('[^#|/]+$',?o,0)) as ?o
FROM :Si
WHERE {?s ?p ?o}
```

Complementarity Factor. This metric is computed for a specific entity over all sources of the warehouse. In particular, the complementarity factor of an entity e is increased for each source $S_i \subseteq S$ that contains at least one unique triple having the entity e. This means that if all sources in S contain unique triples for e, then its complementarity factor will be 1.0. The query below gives the complementarity factor of an entity e over S. Notice that the WHERE clause contains n graph patterns. In the SELECT statement n denotes the number of sources and it is provided by the user. Each graph pattern i returns 1, if S_i contains unique triples for the entity e, otherwise it returns 0.

```
SELECT  (?CF1+ .. + ?CFn)/n AS ?CF
WHERE { { SELECT xsd:integer(COUNT(*)>0) as ?CF1
          WHERE { { graph :S1 { ?s ?p1 ?o } }
               FILTER NOT EXISTS { graph :S2 { ?s ?p2 ?o} } .
                 ...
               FILTER NOT EXISTS { graph :Sn { ?s ?pn ?o} }
               FILTER (regex(?s, e,'i') || (regex(?o, e,'i'))) } }
                 ...
                 ...
        { SELECT xsd:integer(COUNT(*)>0) as ?CFn
          WHERE { { graph :Sn { ?s ?pn ?o } }
               FILTER NOT EXISTS { graph :S1 { ?s ?p1 ?o } } .
                 ...
               FILTER NOT EXISTS { graph :Sn-1 { ?s ?pn-1 ?o } }
               FILTER (regex(?s, e,'i') || (regex(?o, e,'i'))) } }
     }
```

Increase in the Average Degree. Let E be a set of entities coming from a source S_i. To compute the increase in the average degree of these entities when they are added into the warehouse, the following query computes both average values (before and after the addition to the warehouse) and reports the increase. Note that that above query considers the "entity matching" policy [iii].

```
DEFINE input:same-as "yes"
SELECT ((?avgDW-?avgDS)/?avgDS) as ?IavgD
WHERE { { SELECT xsd:double((count(?in)+count(?out)))
                /xsd:double(count (distinct ?e)) as ?avgDS
          FROM :Si
          WHERE{ ?e rdf:type :E.
               {?e ?in ?o} UNION {?o1 ?out ?e} } }
```

```
      { SELECT xsd:double((count(?in)+count(?out)))
               /xsd:double(count (distinct ?e)) as ?avgDW
         FROM :W
         WHERE { ?e rdf:type :E .
                 { ?e ?in ?o} UNION {?o1 ?out ?e} } }
     }
```

Regarding the average degree of all the URIs and blank nodes of a source, one can use the following query.

```
SELECT xsd:double(count(?incomingProperties)+count(?outgoingProperties))
       /xsd:double(count (distinct ?node)) as ?sourceDegree
FROM :Si
WHERE{
    {?node ?outgoingProperties ?o .
    FILTER (?outgoingProperties!=<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)}
    UNION
    {?o1 ?incomingProperties ?node .
    FILTER (?incomingProperties!=<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)}
    FILTER(isURI(?node) || isBlank(?node))}
```

Finally, for the average degree of the URIs and blank nodes of a source in the warehouse, one can use the following two queries for URIs and blank nodes, respectively. We use a temporary graph to store all the URIs of each graph, since this way was proved more efficient in order to measure the degree of each source in the warehouse. Moreover, one can find the average degree of the union of its URIs and blank nodes by using the following formula:

$$avgdeg_W(U_W \cup BN_W) = \frac{avgdeg_W(U_W) * |U_W| + avgdeg_W(BN_W) * |BN_W|}{|U_W| + |BN_W|} \tag{3.10}$$

The following SPARQL queries can be used for computing the average degree for entities that are URIs and blank nodes respectively.

```
DEFINE input:same-as "yes"
SELECT xsd:double(count(?incomingProperties)+count(?outgoingProperties))
      /xsd:double(count (distinct ?node))) as ?avgURIsDegree
      count (distinct ?node) as ?URIs
FROM :Wi
WHERE { GRAPH :temp
    {?node a :SiURI} .
    {{?node ?outgoingProperties ?o .
    FILTER(?outgoingProperties!=<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)}
    UNION
    {?o1 ?outgoingProperties ?node .
    FILTER(?incomingProperties!=<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)} }}
```

```
DEFINE input:same-as "yes"
SELECT xsd:double(count(?incomingProperties)+count(?outgoingProperties))
      /xsd:double(count (distinct ?node))) as ?avgBnodesDegree
```

```
      count (distinct ?node) as ?Bnodes
FROM :Wi
WHERE{
    {?node ?outgoingProperties ?o .
    FILTER(?outgoingProperties!=<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)}
    UNION {?o1 ?incomingProperties ?node .
    FILTER(?incomingProperties!=<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)}
    {SELECT distinct ?node
    WHERE { graph :Si
    {{?node ?p ?o } UNION {?o ?p ?node}}
    .FILTER(isBlank(?node))}}}
```

Time efficiency

Table 3.17 shows the query execution times (in minutes) for computing the metric *Common URIs* for each of the three policies, i.e., (i) *Exact String Equality*, (ii) *Suffix Canonicalization* and (iii) *Entity Matching*. The first row corresponds to the *pure SPARQL* approach that was presented earlier. The second row corresponds to a *hybrid* approach, where more simple queries are used for getting the resources of interest (i.e., the two sets of URIs, one for each source S_i, S_j), and Java code is used for computing their intersection. We observe that the *hybrid* approach is faster than the *pure SPARQL*, as the comparisons are implemented faster in Java. In general, we have observed that the *hybrid* approach loses in time efficiency when the implemented queries return a big amount of data (as in the case of *Unique Triples Contribution*), while it is faster (than *pure SPARQL*) in comparisons.

Common URIs			
Computation Method	Policy [i]	Policy [ii]	Policy [iii]
pure SPARQL	7	20	8
hybrid	3	4	4

Table 3.17: Times (in min) needed to compute metrics on various approaches and policies.

Regarding policy [ii], the *pure SPARQL* approach becomes less efficient, as the string comparisons cost more when implemented over the endpoint. When adopting policy [iii], both approaches are increased by 1 minute. This uniform increase is reasonable as an additional graph that contains the triples with the *sameAs* properties is taken into account.

3.5 Experimental Evaluation

3.5.1 MarineTLO-Warehouse Evolution

The objective here is to investigate how we can understand the evolution of the warehouse and how we can detect problematic cases (due to changes in the remote

sources, mistakes in the equivalence rules, addition of a redundant or a "useless" source etc). Let v denote a version of the warehouse and v' denote a new version of the warehouse. A number of questions arise:

- Is the new version of the warehouse better than the previous one? From what aspects, the new warehouse is better than the previous one, and from what aspects it is worse?
- Can the comparison of the metrics of v and v', aid us in detecting problems in the new warehouse, e.g., a change in an underlying source that affected negatively the new warehouse?

It is also useful to compare a series of versions for:

- understanding the evolution of the entire warehouse over time
- understanding the evolution of the contribution of a source in the warehouse over time

To tackle these questions, we first (in §3.5.2) describe the datasets (real and synthetic) that were used, and then (in §3.5.3) we focus on how to inspect a *sequence* of versions. Finally §3.5.4 summarizes the drawn conclusions.

3.5.2 Datasets Used

To understand the evolution we need several series of warehouse versions. To this end, we used both real and synthetically derived datasets.

3.5.2.1 Real Datasets

We used 3 real versions of the `MarineTLO`-based warehouse. Specifically we considered the following versions:

- `MarineTLO`-based warehouse version 2 (July 2013): 1,483,972 triples
- `MarineTLO`-based warehouse version 3 (December 2013): 3,785,249 triples
- `MarineTLO`-based warehouse version 4 (June 2014): 5,513,348 triples

3.5.2.2 Synthetic Datasets

We created a series of synthetic datasets in order to test various aspects, e.g., source enlargements, increased or reduced number of `sameAs` relationships, addition of new sources (either relevant or irrelevant to the domain), addition of erroneous data, etc.

Tables 3.18 and 3.19 show the 9 different versions and their size in triples, while Table 3.20 shows the changes from version to version. For each version from 1 to 4, we add new "useful" data, allowing in this way to check not only how the integration of the warehouse is affected by the new useful data, but also to observe the extent up to which the value of a source is changed every time. Versions 5, 6, contain "not so good" sources, hence we would expect to see low values for these two sources. Specifically, version 5 includes a source from a different domain, while version 6 contains a redundant source (this source includes triples from Fishbase and WoRMS). Concerning version 7, we have created a possible error that one

could face. In particular, we have replaced in the prefix of DBpedia's URIs $'/'$ with $'\#'$, whereas in version 8, we have used an invalid SILK Rule in order to produce some wrong `sameAs` relationships. Taking all these into account, we expect that the following experiments will allow us to detect all the above errors and will show the real importance of each source in every version.

	V1	V2	V3	V4	V5
FLOD	904,238	904,238	904,238	904,238	904,238
WoRMS	62,439	62,439	1,383,168	1,383,168	1,383,168
Ecoscope	61,597	61,597	61,597	61,597	61,597
DBpedia	394,373	394,373	394,373	394,373	394,373
FishBase	-	-	-	2,332,893	2,332,893
Clone Source	-	-	-	-	-
Airports	-	-	-	-	121,113
All	1,422,637	1,422,637	2,743,376	5,076,269	5,197,382

Table 3.18: Triples of the synthetically derived versions (versions 1-5).

	V6	V7	V8	V9
FLOD	904,238	904,238	904,238	904,238
WoRMS	1,383,168	1,383,168	1,383,168	1,383,168
Ecoscope	61,597	61,597	61,597	61,597
DBpedia	394,373	488,989	645,228	782,469
FishBase	2,332,893	2,332,893	2,332,893	2,332,893
Clone Source	1,036,194	-	-	-
Airports	-	-	-	-
All	6,112,463	5,170,885	5,327,124	5,464,365

Table 3.19: Triples of the synthetically derived versions (versions 6-9).

3.5.3 Inspecting a Sequence of Versions

Here we discuss how one can inspect a sequence comprising more than two versions. Suppose that we have n versions, $v1, v2, \ldots, vn$. We can get the big picture by various plots each having in the X axis one point for each warehouse version. Below we describe several useful plots.

i For each vi we plot $|triples(W_{vi})|$. Figure 3.3 shows the resulting plot for the real datasets.

ii For each vi we plot $|U_{Wvi}|$ and $|Lit_{Wvi}|$ where U_{Wvi} is the set of all URIs and Lit_{Wvi} is the set of all Literals in the warehouse of that version. Figure 3.4 shows the resulting plot for the real datasets.

iii For each vi we plot the average degree of the URIs of the warehouse $deg_W(U_{Wvi})$ and the average degree of the blank nodes and URIs of the warehouse

Version	Description
1	Low number of `sameAs` relationships
2	Addition of more `sameAs` relationships
3	Increase of WoRMS triples
4	Addition of a new source (FishBase)
5	Addition of an out-of-domain source (Airports)
6	Deletion of Airports source and addition of a redundant source (CloneSource)
7	Deletion of CloneSource and update of DBpedia where the policy of its URIs changed.
8	Update of DBpedia and production of 4000 wrong `sameAs` relationships due to an invalid SILK rule
9	The final version without errors

Table 3.20: Description of how each synthetic warehouse version was derived.

$deg_W(U_{Wvi} \cup BN_{Wvi})$. Figure 3.5 shows the resulting plot for the real datasets.

iv For each vi and for each source S_j we plot $value_1(S_i, W)$ as defined in §3.3.2 (one diagram with k plots one for each of the k sources). Figure 3.6 shows how the contribution of the sources in the warehouse evolves, for the real datasets.

v For each source S_i and version j we plot:

(a) The Jaccard similarity between the set of triples in $(S_i)_{j-1}$ and $(S_i)_j$ (e.g., see the 1st column of Figure 3.2)

(b) The normalized number of the triples in every version, defined as the division of the number of a source's triples with the number of the biggest source's triples in the warehouse (e.g., see the 2nd column of Figure 3.2):

$$NormalizedTriples(S_i)_j = \frac{|triples(S_i)_j|}{\max_{m \in 1..k} |triples(S_m)_j|} \tag{3.11}$$

(c) The normalized average degree defined as the division of the average degree of a source by the maximum average degree among all the sources in the warehouse (e.g., see the 3rd column of Figure 3.2):

$$NormalizedAverageDegree(S_i)_j = \frac{avgdeg_{S_i}(U_i)}{\max_{m \in 1..k} avgdeg_{S_m}(U_m)} \tag{3.12}$$

(d) The value in each version, as it has defined in §3.3.2 (e.g., see the 4th column of Figure 3.2):

The first three (i-iii) concern the warehouse per se, while (iv) and (v) show how the contribution of the source in the warehouse evolves. Regarding the results, for

Figure 3.2: Measurements per source for the real datasets.

the Jaccard distance one can observe that the source WoRMS completely changed from version to version. On the other hand, DBpedia had the smallest change among all sources. Concerning the value of the normalized triples, the addition of Fishbase and the increase in the number of triples of WoRMS had a huge impact in the normalized triple value of the other sources. Moreover, DBpedia, has the biggest average degree in all versions while Fishbase and WoRMS have a consecutive increase in their values from version to version. Finally, Table 3.21 shows the average degree increment of the URIs and blank nodes of each source for the 3 different versions of the real datasets.

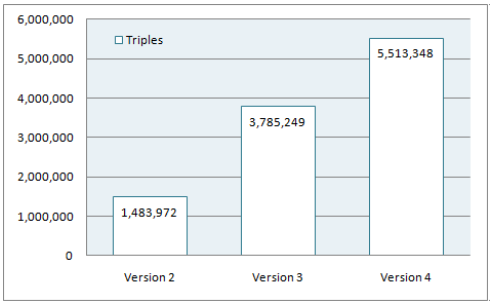


Figure 3.3: Triples of each version.

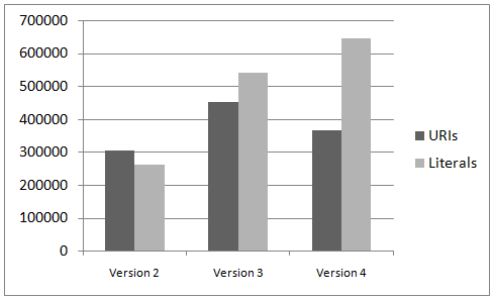


Figure 3.4: URIs and Literals.

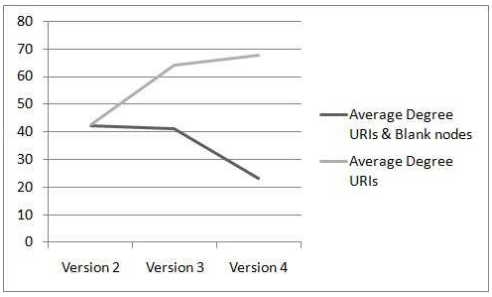


Figure 3.5: Average degree of the warehouse in every version.

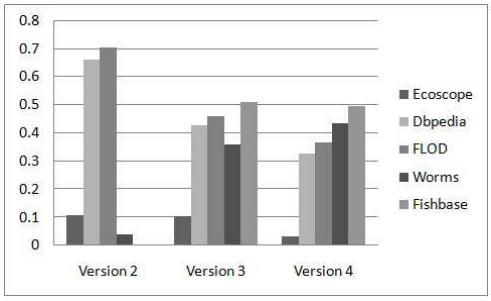


Figure 3.6: Value for each Source in every version.

Source \ V_i	Version2	Version 3	Version 4
FLOD	465.59%	793.64%	797.61%
WoRMS	548.67%	97.82%	103.61%
Ecoscope	108.97%	325.58%	396.84%
DBpedia	271.84%	522.75%	505.65%
FishBase	—-	117.02%	58.43%

Table 3.21: Average degree increment percentages for the URIs and blanks nodes of each source in every version.

Results over the Synthetic Datasets

Common URIs/Literals. Concerning common URIs, Figure 3.7 shows how the percentages of common URIs (according to policy [iii]) changed from version to version among DBpedia and all the other sources. As we can see, the percentages increased in version 2 comparing to version 1, because of the production of the `sameAs` relationships. On the contrary, in version 7 the percentages of common URIs were heavily reduced, since the change in DBpedia's URIs affected negatively the production of the `sameAs` relationships. Moreover, the percentages increased in version 8, predominantly because of the wrong `sameAs` relationships while in the last version, the increase in the common URIs percentage between DBpedia and FishBase is remarkable.

Regarding common Literals, as we can see in Figure 3.8 the percentages didn't change so much from version to version. However, it is clear that the highest percentage exists between DBpedia and FishBase, exactly as it happened with common URIs percentage. Therefore, it is obvious that DBpedia shares more common information with FishBase than with all the other sources.

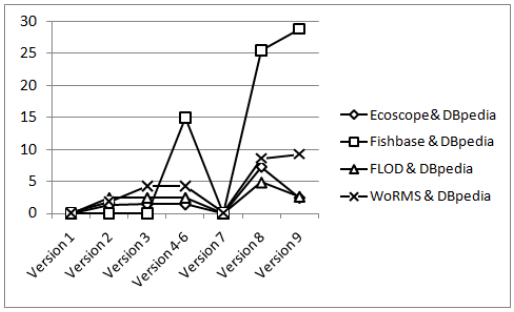


Figure 3.7: Common URIs %

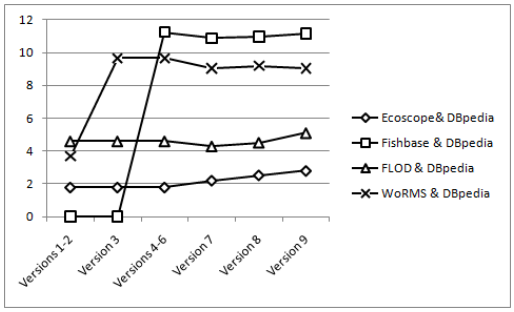


Figure 3.8: Common Literals %

Average Degree Increment. Figure 3.9 shows the normalized increment degree percentage of each source. As we can see, it is evident that a source containing useful data has a positive effect on the percentages of all the other sources. For instance, when we added the source FishBase (in version 4), the percentages of all the other sources were notably increased, while when we added the Airports source (version 5), the percentages remained almost the same.

In version 7, the change of the URI policy in DBpedia reduced the `sameAs` relationships in the warehouse. As a result, it affected negatively the percentages of all the sources (predominantly those from DBpedia). Despite the fact that there are considerable benefits to see big values in this measure, sometimes there is also an important drawback that cannot be ignored. For instance, in version 8, as we can see better in Figure 3.10, the percentages were highly increased because of an invalid SILK rule which produced $4,000$ wrong `sameAs` relationship, leading to significant negative consequences for the warehouse. For instance, the queries will return a lot of imprecise results. Therefore, having big values for integration has no meaning if the precision is low.

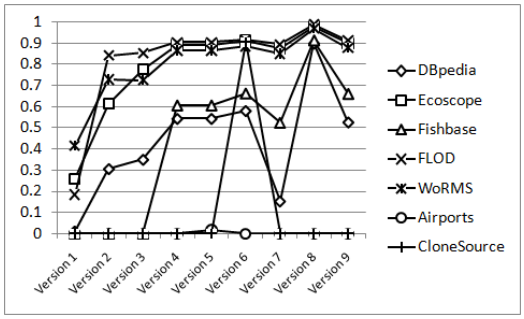


Figure 3.9: Normalized Average Degree Increment of each source.

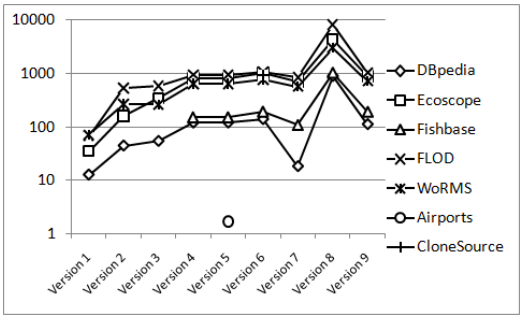


Figure 3.10: Average Degree Increment of each source.

Unique Triples Percentage. As regards unique triples, Figure 3.11 shows the unique triples percentage of each source. The addition of CloneSource in version 6 had as a result the reduction of WoRMS and FishBase percentages. In fact, CloneSource shared a lot of triples with these two sources. Therefore, through this diagram one can easily observe that CloneSource is redundant, while one can see the remarkable change in the unique triples percentage of WoRMS because of the addition of FishBase in version 4.

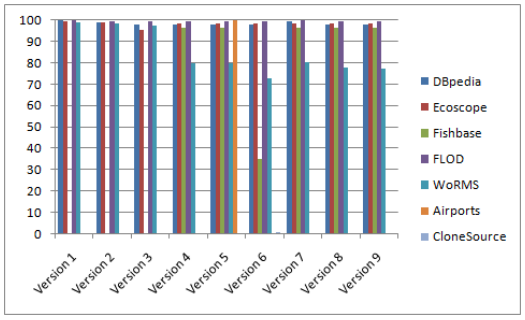


Figure 3.11: Unique Triples Percentage of each source.

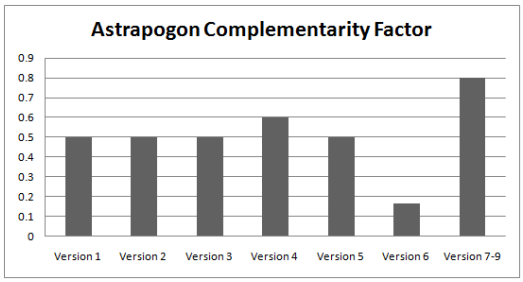


Figure 3.12: Complementarity Factor of Astrapogon.

Complementarity Factor. Regarding complementarity factor, Figure 3.12 shows how the updates affected the complementarity factor of a specific entity. In versions 1-3, only 2 (out of 4) sources provided unique triples about "Astrapogon". In version 4, the complementarity factor of this entity increased because FishBase contained unique data about it. On the other hand, the out-of-domain source that we added in version 5 had negative impact for this measure. It is worth-mentioning that the CloneSource, which included all the triples about Astrapogon from WoRMS and FishBase, heavily reduced the complementarity factor in version 6. Finally, the addition of the new triples of DBpedia and the deletion of CloneSource had considerable advantages for the value of this measure. Therefore, in the last 3 versions, 4 sources (out of 5) provided unique data about "Astrapogon".

Value of Sources. Figure 3.13 depicts the values of each source in all different versions. Initially, there is no doubt that in the first two versions the most important source was FLOD, since it had high values for the average degree increment and the unique triples percentage, and was consisted of a large number of triples. Furthermore, we can see that in version 2 the value of each source was increased because of the production of the `sameAs` relationships. Regarding version 3, the increment of the size of WoRMS had negative effects on the values of the other sources. Therefore, the value and possibly the ranking of the sources can be affected when we add to the warehouse a bigger source with useful triples. In this way, FishBase value surpassed all the other sources values in version 4. Concerning version 5, the new source (Airports) that we added seemed to be useless because of the low average degree increment percentage. This is rational, because it provided information which is useless for our warehouse. In version 6, we added a source (CloneSource) containing approximately $1,500$ new triples and a lot of triples which were already existed in the warehouse and specifically, in FishBase and WoRMS. Indeed, despite the fact that this source was consisted of a lot of triples, its value was the lowest. Moreover, by adding these triples, the values of the sources sharing common triples with CloneSource were affected, too. In fact, the reduction of the value of FishBase is remarkable although the contents of this source were exactly the same as in version 5. As regards version 7, DBpedia was affected negatively because of the change in the policy of it's URIs. In version 8, DBpedia and FishBase were mostly benefited by the production of the wrong `sameAs` relationships, since their normalized average degree increased to a great extent comparing to the other sources. Finally, it is worth mentioning that the value of DBpedia increased in version 9, since there was added a large number of unique triples for this source.

Measurements per Source. Finally, Figure 3.14 shows the values of several metrics for each source (regarding the synthetic datasets). Specifically, the 1st column shows the Jaccard distance between the versions, the 2nd column shows the normalized value of the triples in each version, the 3rd column shows the normalized average degree in each version, and the 4th column shows the value in each version. Firstly, one can easily see the change in the Jaccard distance of DBpedia from version 6 to version 7. In this case, the change of the policy of DBpedia's URI had a negative consequence for the value of DBpedia. By observing this figure, one can realize that the aforementioned problem possibly occurred because of this sudden change in DBpedia's Jaccard distance. As regards the values of the normalized triples, FLOD had the biggest decrease because of the insertion or the increase of other sources. The most remarkable figures concerns the value of each source. We can observe the variations in each source's value due to the pathological case that we saw previously.

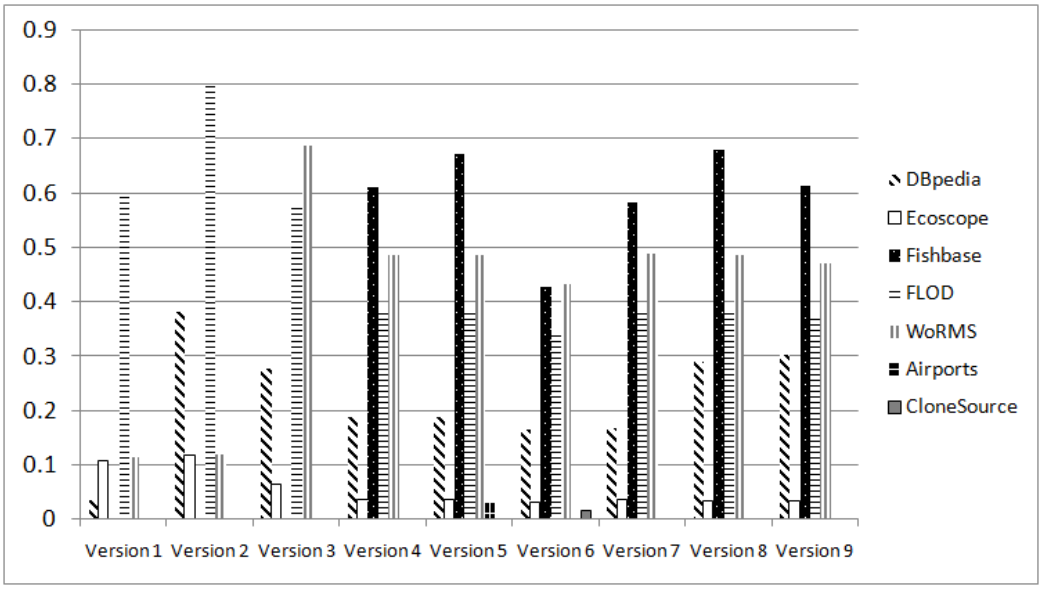


Figure 3.13: Value of each source per version (using $value_1(S_i, W)$).

3.5.4 Executive Summary Regarding Evolution

Taking all the aforementioned into consideration, we managed through the above experiments to meet the main objectives. In particular, through the real datasets we saw the evolution of a real warehouse. Indeed, by observing the changes in the *Average Degree Increment percentage*, we concluded that the average Degree percentage of the warehouse URIs increased from version to version. However, by taking into account not only the URIs but also the blank nodes, we observed that the percentage decreased from version to version. Additionally, through the single-valued metric *Value*, we saw the most important sources in every version and the contribution of each source in the warehouse over time.

As regards the synthetic datasets, we managed to detect a number of problematic cases. Particularly, we found the problematic cases in versions 5,6,7,8 predominantly through *Value*, *Average Degree Increment percentage* and *Unique Triples percentage*. Specifically, we identified that *Airports* source was out-of-domain, because both *Value* and *Average Degree Increment percentage* of this source were very low. On the contrary, in version 6, it was easy to understand that *CloneSource* was redundant, since, its *Value* and *Unique Triples percentage* were very low. In addition to this, it was remarkable that the values of the sources sharing common triples with this source were affected to a great extent. In particular, the *Unique Triples percentage* of these sources decreased greatly.

Concerning version 7, the large decrease in the *Average Degree Increment percentage* of *DBpedia* and in the *common URIs percentage* between DBpedia and each of the remaining sources indicated that probably either the schema or the policy of this source's URIs changed. However, such a large decrease can be also

Figure 3.14: Measurements per source for the synthetic datasets.

arisen when the content of a source changed while neither the schema nor the policy of URIs changed (e.g., information about some fishes has been removed). For being sure that this decrease arose due to schema or URIs policy, one can measure the *common URIs percentage* between the pairs of sources with policy [ii]. Of course, there is also the case where two different instances refer to the same real-world object but their suffixes are completely different. In such cases, policy [ii] cannot be used for understanding their `sameAs` relationship.

Finally, in version 8, the huge increase in the *Average Degree Increment percentage* was unexpected. Indeed, the production of more and more `sameAs` relationships always improves the *Average Degree Increment percentages*. However, if the percentages are far higher than the previous versions, it is highly possible that there have been produced a lot of wrong `sameAs` relationships, probably due to an invalid `SILK` rule.

Chapter 4

Measuring the Connectivity of Several LOD Datasets

In chapter 3, we described connectivity metrics for evaluating semantic Warehouses. In this section, we show how to compute the connectivity metrics for any combinations of LOD Cloud datasets. Comparing to a semantic warehouse which usually contains a small number of datasets, it is prohibitively expensive to compute the metrics either in a straightforward way or by using SPARQL queries. For this reason we propose indexes and algorithms for computing fast such measurements. At first, we formulate the problem and we show the proposed indexes and algorithms. Then, we report interesting measurements for 300 LOD cloud datasets and we discuss the speedup obtained by the proposed techniques.

4.1 Indexes for Measuring the Connectivity of RDF Datasets

At first we formulate the problem and afterwards we describe the semantics-aware element index and detail the steps for creating it.

4.1.1 Problem Statement

Let $\mathcal{D} = \{D_1, \ldots, D_n\}$ be a set of RDF datasets (or sources). For each D_i we shall use $triples(D_i)$ to denote its triples ($triples(D_i) \subseteq \mathcal{T}$), and U_i to denote the URIs that occur as subjects or objects in these triples. As running example we will use four datasets each containing six URIs as shown in Figure 4.1 (upper-left corner).
Common URIs in Datasets.
Let $\mathcal{P}(\mathcal{D})$ denote the powerset of $\mathcal{D}$, comprising elements each being a subset of $\mathcal{D}$, i.e. a set of datasets. Our objective is to find the *common URIs* in every element of $\mathcal{P}(\mathcal{D})$, meaning that for each set of datasets $B \in \mathcal{P}(\mathcal{D})$ we want to compute $cu(B)$ defined as:

$$cu(B) = \cap_{D_i \in B}\ U_i \tag{4.1}$$

Datasets containing a particular URI.
Another objective is to find all datasets that contain information about one particular URI u, i.e. to compute

$$dsets(u) = \{D_i \in \mathcal{D} \mid u \in U_i\} \tag{4.2}$$

Considering Equivalence Relationships.
So far we have ignored the `sameAs` relationships. Below we shall see how to semantically complete the previous definitions. Let $sm(D_i)$ be the `sameAs` relationships of a dataset D_i, i.e.:

$$sm(D_i) = \{(u, u') \mid (u, \texttt{sameAs}, u') \in triples(D_i)\} \tag{4.3}$$

If B is a set of datasets, i.e. $B \in \mathcal{P}(\mathcal{D})$, we will denote with $SM(B)$ the union of the `sameAs` relationships in the datasets of B, i.e. $SM(B) = \cup_{D_i \in B}\ sm(D_i)$. Notice that our running example includes five `sameAs` relationships, shown in Figure 4.1 (upper-right). If R denotes a binary relation, we shall use $\mathcal{C}(R)$ to denote the transitive and symmetric closure of R. Consequently, $\mathcal{C}(sm(D_i))$ stands for the transitive and symmetric closure of $sm(D_i)$, while $\mathcal{C}(SM(B))$ is the transitive and symmetric closure of the `sameAs` relationships in all datasets in B. The number of real world objects (in abbreviation *rwo*) in a dataset D_i is the number of *classes of equivalence* of $\mathcal{C}(sm(D_i))$, plus those URIs of U_i that do not occur in $\mathcal{C}(sm(D_i)$ (for not counting some URIs more than once), e.g. if $U_{temp} = \{u_1, u_2, u_3, u_4, u_5\}$ and we have two `sameAs` relationships, $u_1 \sim u_3$ and $u_1 \sim u_4$, then their closure derives the following classes of equivalence $U_{temp}/\sim\ = \big\{\{u_1, u_3, u_4\}, \{u_2\},\ \{u_5\}\big\}$. The number of real world objects in a set of datasets B is defined analogously.

We can now define the equivalent URIs (considering all datasets in B) of a URI u (and of a set of URIs U) as:

$$Equiv(u, B) = \{\ u' \mid (u, u') \in \mathcal{C}(SM(B))\} \tag{4.4}$$

$$Equiv(U, B) = \cup_{u \in U} Equiv(u, B) \tag{4.5}$$

We are now ready to "semantically complete" the definitions (4.1) and (4.2):
Datasets containing a particular (or equivalent) URI.
The set of datasets that contain information about u (or a URI equivalent to u) is defined as:

$$dsets_{\sim}(u) = \{D_i \in \mathcal{D} \mid (\{u\} \cup Equiv(u, \mathcal{D})) \cap U_i \neq \emptyset\} \tag{4.6}$$

Obviously, it holds $dsets(u) \subseteq dsets_{\sim}(u)$.
Common (or equivalent) URIs in Datasets.
The common or equivalent URIs in the datasets in B are defined as:

$$cu_{\sim}(B) = \{\ u \in U \mid dsets_{\sim}(u) \supseteq B\} \tag{4.7}$$

Obviously it holds $cu(B) \subseteq cu_{\sim}(B)$. Now the real world objects that are in common in the datasets in B are the classes of equivalence of $cu_{\sim}(B)$, i.e. the set

$cu_{\sim}(B)/\sim$. Therefore we define the number of common real world objects in the datasets of B as:

$$co_{\sim}(B) = |cu_{\sim}(B)/\sim| \tag{4.8}$$

In a nutshell, in this section we focus on how to compute efficiently formulas 4.6 and 4.8, for any $u \in U$ and $B \subseteq \mathcal{D}$ respectively.

4.1.2 The Proposed Indexes

Figure 4.1 illustrates the proposed indexes over our running example. Let $U = U_1 \cup \ldots \cup U_n$ ($n = 4$ in our example). We propose three indexes:
• `PrefixIndex`: It is a function $pi : Pre(\mathcal{D}) \rightarrow \mathcal{P}(\mathcal{D})$ where $Pre(\mathcal{D})$ is the set of prefixes of the datasets in $\mathcal{D}$, i.e. $Pre(\mathcal{D}) = \{\ pre(u) \mid u \in U_i, D_i \in \mathcal{D}\}$, e.g. see step 1 of Fig. 4.1.
• `SameAsCat`: For each u that participates to $SM(\mathcal{D})$ this catalog stores a unique id. All URIs in the class of equivalence of u are getting the same id. Let SID denotes this set of identifiers. The `SameAsCat` is essentially a binary relation $\subseteq U \times SID$, e.g. see step 2 of Fig. 4.1.
• `ElementIndex`: For each element of $U \cup SID$ this index stores the datasets where it appears, i.e. it is a function $ei : U \cup SID \rightarrow \mathcal{P}(\mathcal{D})$ where $ei(u) = dsets_{\sim}(u)$, e.g. see step 3 of Fig. 4.1.
The rationale and the construction method for each one is given below.

4.1.3 Prefix Index

Rationale: Most data providers publish their data by using prefixes indicating their company or university (e.g., *DBpedia* URIs starts with prefix *http://dbpedia.org*). A `PrefixIndex` can greatly reduce the cost of finding common URIs. First, there is no need to compare URIs containing different namespaces. Second, if a prefix p exists in only one dataset, it is not possible for the URIs starting with p to be found in another dataset.
Construction Method: For getting the prefixes of a dataset stored in a triple-store, one can either submit a SPARQL query or scan each U_i once. We also count the frequency of each prefix in the dataset. If nm is a namespace, $pi(nm)$ is the set of ids of the datasets that contain it. This set is stored in ascending order with respect to the frequency, e.g. in our running example we can see that for the prefix *dbp* the dataset *DBpedia* contains the most URIs. Consequently, the ID of this source (i.e. 2) is in the last position of $pi(dbp)$. This ordering is beneficial for reducing the ASK queries as we shall see later in §4.1.5.

Moreover, `PrefixIndex` enables a fast method for finding the upper bound of $|dsets(u)|$ for a particular u (since $dsets(u) \subseteq pi(pre(u))$), e.g. in our running example a URI starting with prefix *en_wiki* can be possibly found in 4 datasets, because all the datasets contain this prefix. However, a URI with prefix *yg* can appear only in one dataset since this prefix appears only in *Yago*.

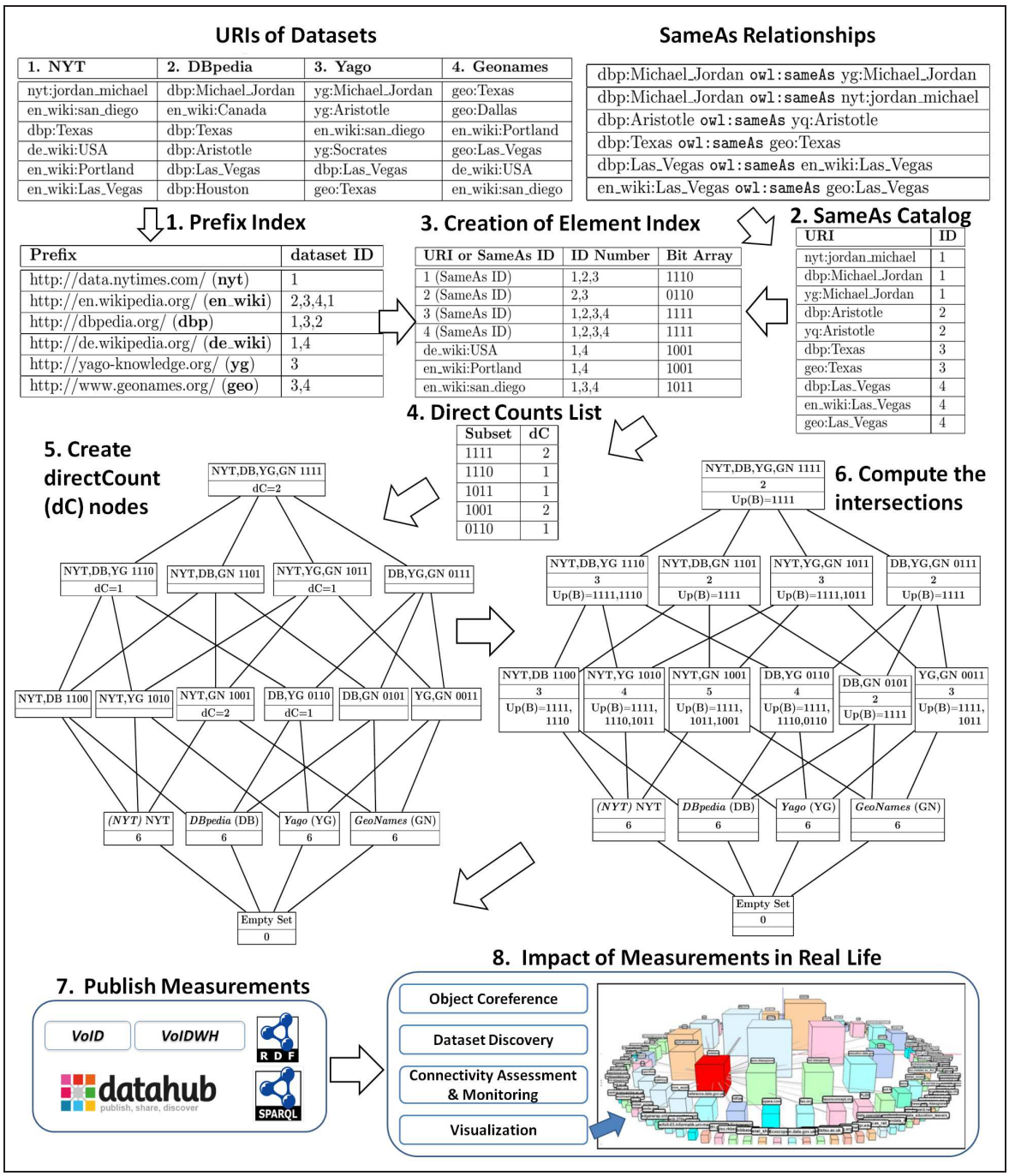


Figure 4.1: Running Example

The following query returns the distinct prefixes for the URIs starting with *http://*.

```
select distinct ?prefix from <GraphSpace>
where { {?s ?p ?o} union {?o ?p ?s}
. BIND(bif:regexp_substr('^http://(.*?)(/)',?s,0)
  as ?prefix)
. filter(isURI(?s)) }
```

For getting the count of each distinct prefix for the URIs starting with *http://*, one can use the query below.

```
select ?prefix count( distinct ?s)
from <GraphSpace>
where { {?s ?p ?o} union {?o ?p ?s}
. BIND(bif:regexp_substr('^http://(.*?)(/)',?s,0)
  as ?prefix)
. filter(isURI(?s))
} group by(?prefix)
```

4.1.4 SameAs Catalog

Rationale: It is required for formulas 4.6, 4.7 and 4.8 as explained in §4.1.1.
Construction Method: We introduce a signature-based algorithm where each class of equivalence will get a signature (id) and the signature is constructed incrementally during the computation. After the completion of the algorithm, all URIs that belong to the same class of equivalence will have the same identifier. The algorithm assigns to each pair $(u, u') \in SM(B)$ an identifier according to the following rules:

1. If both URIs have not an identifier, a new identifier is assigned to both of them. E.g. Table 4.1 contains two classes of equivalence and four URIs. In the next step, a new `sameAs` pair containing two URIs without identifier is inserted resulting to a new class of equivalence which is shown in Table 4.2.
2. If u has an identifier while u' has not, u' gets the same identifier as u. Table 4.3 shows such an example where a new URI (u_7) takes the identifier of an existing one (u_3).
3. If u' has an identifier while u has not, u gets the same identifier as u'
4. If both URIs have the same identifier, the algorithm continues
5. If the URIs have a different identifier, these identifiers are concatenated to the lowest identifier. In case of Table 4.4, both URIs exist in the `SameAsCat` and have a different identifier. For this reason, the classes of equivalence of these identifiers are merged.

We can say that the algorithm constructs incrementally chains of `sameAs` URIs where each URI becomes a member of a chain if and only if there is a `sameAs` relationship with a URI which is already member of this chain. Its correctness is based on the following proposition.

Proposition 1 If $(a, b) \in SM(B)$ and $(a, c) \in SM(B)$ then $(b, c) \in C(SM(B))$.

Proof. 1 *Firstly,* $(b, a) \in C(SM(B))$ *because of symmetry. By taking the transitive closure of* $(b, a), (a, c)$*, we get that* $(b, c) \in C(SM(B))$*.*

Its benefit is that it needs only one pass for each `sameAs` pair in order to compute the transitive and symmetric closure, and its time complexity is $\mathcal{O}(n)$, where n is the number of `sameAs` pairs. However, it is needed to keep in memory chains of `sameAs` in order to connect such chains with new `sameAs` relationships. Indeed the space complexity is $\mathcal{O}(m)$, where m is the number of unique URIs that occur in `sameAs` pairs, since in the worst case each URI is saved both in `SameAsCat` and in classes of equivalence until the end of the algorithm. Regarding the size of the catalog we store for each URI a distinct arbitrary number. Since each real world object is represented by exactly one identifier, the number of unique identifiers in the `SameAsCat` is equal to the number of unique real world objects of $SM(B)$. In our running example, the `sameAs` catalog is shown in Figure 4.1 (step 2).

ID	URIs
1	u_1, u_2
2	u_3, u_4

Table 4.1: Classes of Equivalence

ID	URIs
1	u_1, u_2
2	u_3, u_4
3	$\mathbf{u_5}, \mathbf{u_6}$

Table 4.2: Insert u_5 `sameAs` u_6

ID	URIs
1	u_1, u_2
2	$u_3, u_4, \mathbf{u_7}$
3	u_5, u_6

Table 4.3: Insert u_3 `sameAs` u_7

ID	URIs
1	$u_1, u_2, \mathbf{u_3}, \mathbf{u_4}, \mathbf{u_7}$
~~2~~	~~u_3, u_4, u_7~~
3	u_5, u_6

Table 4.4: Insert u_1 `sameAs` u_3

Algorithm 1 shows in details how to find the classes of equivalence of each URI and how to produce the `sameAs` catalog. With $Left(r)$ we denote the set of elements that occur in the left side of a binary relation or function r, with $Right(r)$ the set of elements that occur in the right side of r and with $[u]$ the class of equivalence of u. Firstly, it reads a number of pair of URIs $\{u, u'\}$. When both URIs have not an identifier, a new identifier is assigned to both of them (lines 4-5) and a new class of equivalence is created (line 6). If u has an identifier while u' has not, u' gets the same identifier as u (lines 9-10) while u' is added in $[u]$. Analogously, for the case when u' has an identifier while u has not (lines 12-13). When the URIs have a different identifier (i.e., they are members of different classes of equivalence), their classes of equivalence are merged (line 15) while each URI of the resulted class of equivalence gets the lowest identifier (lines 16-18). On the contrary, if both URIs have the same identifier (i.e., they are members of the same class of equivalence), the algorithm continues with the next pair. In our implementation we use two HashMaps: the first for each URI (key) it keeps its signature (value), while the second for each signature (key) it keeps the set of URIs

that have this signature (value). In case of executing rule 5, the URIs of the two signatures of the second HashMap are merged in the lowest signature, while the entry of the highest signature is removed. The signature of URIs having previously the highest signature (of the two) should also be updated in the first HashMap.

Algorithm 1 Same As Catalog Creation

Input: SM(B)
Output: A catalog containing for each class of equivalence a signature.

1: $ID \leftarrow 1$
2: **for all** $(u, u') \in SM(B)$ **do**
3: **if** $u \notin Left(\texttt{SameAsCat})$ and $u' \notin Left(\texttt{SameAsCat})$ **then**
4: $\texttt{SameAsCat}(u) \leftarrow ID$
5: $\texttt{SameAsCat}(u') \leftarrow ID$
6: $[u] \leftarrow \{u, u'\}$
7: $ID \leftarrow ID + 1$
8: **else if** $u \in Left(\texttt{SameAsCat})$ and $u' \notin Left(\texttt{SameAsCat})$ **then**
9: $\texttt{SameAsCat}(u') \leftarrow \texttt{SameAsCat}(u)$
10: $[u] \leftarrow [u] \cup \{u'\}$
11: **else if** $u \notin Left(\texttt{SameAsCat})$ and $u' \in Left(\texttt{SameAsCat})$ **then**
12: $\texttt{SameAsCat}(u) \leftarrow \texttt{SameAsCat}(u')$
13: $[u'] \leftarrow [u'] \cup \{u\}$
14: **else if** $Right(\texttt{SameAsCat}(u)) \neq Right(\texttt{SameAsCat}(u'))$ **then**
15: $[u] \leftarrow [u] \cup [u']$
16: $min_{ID} \leftarrow min(Right(\texttt{SameAsCat}(u)), Right(\texttt{SameAsCat}(u')))$
17: **for all** $u_j \in [u]$ **do**
18: $\texttt{SameAsCat}(u_j) \leftarrow min_{ID}$
19: **return** SameAsCat

Alternatively, one can use Tarjan's connected components (CC) algorithm [59] that uses Depth-First Search (DFS). The input of CC algorithm is a graph which should be created before running the algorithm. For this reason, we read all the `sameAs` pairs once $\mathcal{O}(n)$ (i.e., each `sameAs` represents an edge) in order to construct the graph. The time complexity of CC algorithm is $\mathcal{O}(m+n)$. Regarding the space, the creation of graph requires space $n + 2m$ because the graph is undirected and we should create bidirectional edges while the CC algorithm needs space $\mathcal{O}(m)$, since in the worst case it needs to keep in DFS stack all the nodes (i.e., unique URIs). However, the graph should be loaded in memory in order to run the CC algorithm, thereby the total space needed is $\mathcal{O}(n + m)$. In §4.3.1 we compare the execution time of the two aforementioned approaches.

4.1.5 Element Index

Rationale: `ElementIndex` is essentially a function $ei : U \cup SID \rightarrow \mathcal{P}(\mathcal{D})$ where $ei(u) = dsets_{\sim}(u)$ which is needed for finding fast the datasets to which a URI

appears. To reduce its space we can avoid storing URIs that occur in only one dataset (this information can be obtained by the `PrefixIndex`).

We can identify two basic candidate data structures for this index: (1) a bit array of length $|\mathcal{D}|$ that indicates the datasets to which each element belongs (each position in the bit string corresponds to a specific dataset), or (2) an IR-like inverted index [73], in which for each URI store the ID of the datasets (a distinct arbitrary number). A bitmap index for each element of $U \cup SID$ keeps a bit array of length $|\mathcal{D}|$, therefore its total size is $|U \cup SID| * |\mathcal{D}|$ bits. An inverted index for each element of $U \cup SID$ keeps a posting list of dataset identifiers. If ap is the average size of the posting lists, then the total size is $|U \cup SID| * ap * \log |\mathcal{D}|$ bits (where $\log |D|$ corresponds to the required bits for encoding $|\mathcal{D}|$ distinct identifiers). If we solve the inequality $Bitmap \leq InvertedIndex$, we get that the size of bitmap is smaller than the size of inverted index when $ap > \frac{|\mathcal{D}|}{\log |\mathcal{D}|}$.

Construction Method: Algorithm 2 creates the element index while considering the aforementioned indexes (this algorithm can be used for both types of data structures). With $Left(r)$ we denote the set of elements that occur in the left side of a binary relation or function r, and with $[u]$ the class of equivalence of u. Figure 4.1 (middle-right) shows the resulted index for our running example. The combination of the first and the second column of the element index represents the inverted index of the running example, while the combination of the first and the third represents the element index with a bit array of length n (instead of datasets IDs).

Returning to Algorithm 2, at first, if a URI u (of a dataset D_i) belongs to the `SameAsCat` (assuming that each class of equivalence involves URIs that occur in different datasets) it is added to the element index an entry comprising the identifier of u (taken by the `SameAsCat`) and the dataset ID of D_i (lines 3-5). For instance, URI *yq:Aristotle* exists in `SameAsCat`(see Figure 4.1), thereby, its' identifier and dataset ID is added to the element index. When the identifier already exists in the element index, the corresponding entry is updated by adding only the dataset ID (line 7). When u does not exist in the `SameAsCat`, the next step is to check if u already belongs to the element index (because it was encountered previously). Then the index entry of u is updated by adding the dataset ID (lines 8-9).

The last step (if the two previous failed) corresponds to URIs that neither belongs to $Left$(`SameAsCat`) nor to $Left(ei)$. In this step, we exploit `PrefixIndex` by using the function pi for taking the datasets containing the namespace nm of u (line 12). One approach is to add $ei([u_j]) \leftarrow \{i\}$ if the nm of u_j exists in two or more datasets. In this way, at the end the ei could contain *URIs* that occur in one D_i. For this reason, such URIs should be deleted at the end (an extra step is required). Alternatively, one can perform an extra check for ensuring that a URI exists in at least two datasets and this is what is described in lines 12-17.

Let $ask(u, D_k) =$ "$ASK\ D_k\ \{\ u\ ?p\ ?o\ \}\ \cup\ \{\ ?s\ ?p\ u\ \}$" be an ASK query for a URI u and $answer(ask(u, D_k))$ a function which returns either true or

false. At first, we find which datasets contain URIs containing the namespace nm. In particular, we read the datasets IDs that $pi(nm)$ returned in reverse order and we send ASK queries only to a dataset D_k that contain more URIs containing nm (starting from the D_k with the most URIs for nm). In case of $answer(ask(u, D_k)) = true$, a new entry is added to ei which is composed of u and the IDs i and k. For instance, the prefix *en_wiki* can be found in all datasets. However, for the URI *en_wiki:san_diego* of dataset NYT (with ID 1), we do not send a query since NYT dataset contains the most URIs for *en_wiki* prefix. For the same URI *en_wiki:san_diego* of dataset *Yago* (with ID 3) the first step is to send an ASK query to NYT source. In this case $answer(ask(en\_wiki{:}san\_diego, NYT)) = true$, therefore we create a new entry to ei for this URI and we add both the IDs of NYT and *Yago*. On the contrary, for URIs starting with prefixes that exist only in one dataset (e.g., *yg:Socrates*), the algorithm continues without sending any ASK query (see §4.1.3). The approach with the ASK queries is the only one that can be used if the data cannot fit in memory. On the contrary, when the required memory space is available it is faster to keep the URIs until the end (and then remove them that do not occur in ≥ 2 datasets).

Algorithm 2 Element Index Creation

Input: A set of URIs U_i for each dataset, the `SameAsCat` and the `PrefixIndex`
Output: An element index ei of real world objects that exist in ≥ 2 datasets

1: **for all** $D_i \in D$ **do**
2: **for all** $u_j \in U_i$ **do**
3: **if** $u_j \in Left(\texttt{SameAsCat})$ **then**
4: **if** $[u_j] \notin Left(ei)$ **then** ▷ signature of u_j
5: $e_i([u_j]) \leftarrow \{i\}$
6: **else**
7: $e_i([u_j]) \leftarrow e_i([u_j]) \cup \{i\}$
8: **else if** $u_j \in Left(ei)$ **then**
9: $e_i(u_j) \leftarrow e_i(u_j) \cup \{i\}$
10: **else if** $u_j \notin Left(\texttt{SameAsCat}) \cup Left(ei)$ **then**
11: $nm \leftarrow namespace(u_j)$
12: **for all** $D_k \in pi(nm)$ in reverse order **do**
13: **if** $D_k = D_i$ **then**
14: **break**
15: **if** $answer(ask(u_j, D_k)) = true$ **then**
16: $e_i(u_j) \leftarrow \{i, k\}$
17: **break**
18: **return** ei

Algorithm 2 reads each $u \in U_i$ once for all $D_i \in D$, thereby the complexity is $\mathcal{O}(y)$ where y is the sum of $|U_i|$. Alternatively, one can use a straightforward method (*sf*) that finds the intersections of all subsets in $\mathcal{P}(\mathcal{D})$. In a straightforward

method, the URIs are sorted lexicographically in order to perform binary searches for finding the common URIs of any subset. In particular, for each subset $B \in \mathcal{P}(\mathcal{D})$, for all the URIs (e.g., U_1) of the smallest dataset (regarding the size of URIs) one or more binary searches are performed, starting from the second smallest dataset (e.g., U_2) and so forth. Regarding the complexity, in case of having n datasets inside the subset B, in the worst case we should perform for each URI in U_1 a binary search for each of the $n-1$ remaining datasets. For all subsets of $\mathcal{D}$ the complexity becomes exponential, since there exists $2^{|\mathcal{D}|}$ possible subsets, thereby the cost is $\mathcal{O}(2^{|\mathcal{D}|} n \log n)$. In §4.3.1, we show experiments for comparing the execution time of the index approaches and the straightforward method.

Ordering the Prefix Index for Reducing the ASK Queries

In Algorithm 2 we send ASK queries only to the datasets containing more URIs than D_i for prefix p. Here, we describe a) how different combinations of the dataset IDs sequence in prefix index produce different numbers of ASK queries and b) why the proposed ordering reduces the number of ASK queries. Let $U_p = \{u \in U_i \mid namespace(u) = p\}$ and $U'_i = U_i \cap U_p$. Let pos return the dataset ID of specific position for a prefix p list, i.e. it is a function $pos : \mathbb{Z} \rightarrow \mathbb{Z}_{>0}$ and let n be the number of the different datasets having $|U'_i| > 0$.

Suppose that we store the dataset IDs for each prefix randomly. It means that we do not take into account the frequency of the URIs of a prefix in each dataset. In the example of Table 4.5 we can see the size of each U'_i for a prefix p. Table 4.6 shows the number of ASK queries for the worst case in which for each pair $\langle D_i, D_j \rangle$ we should check $\forall\ u_i\ s.t.\ u_i \in U'_i$ the result of $answer(ask(u_i, D_j))$. The formula that we use in Table 4.6 follows: $Asks = |U'_{pos(0)}| * (n-1) + |U'_{pos(1)}| * (n-2) + ... + |U'_{pos(n-1)}| * 0$.

Concerning the sequence $\langle D_1, D_2, D_3 \rangle$, in the worst case the number of ASK queries is: $Asks = |U'_1| * 2 + |U'_2| = 2,005,000$ ASK queries. In particular, we send $2 * |U'_1|$ Ask queries in order to check for each URI u_1, where $u_1 \in U'_1$, if $answer(ask(u_1, D_2)) = true$ or $answer(ask(u_1, D_3)) = true$. Then, for each u_2 we send $|U'_2|$ Ask queries (e.g., for each $u_2 \in U'_2$ we check if $answer(ask(u_2, D_3)) = true$). Concerning the sequence $\langle D_2, D_3, D_1 \rangle$, for the worst case the number of ASK queries is: $Asks = |U'_2| * 2 + |U'_3| = 20,000$ ASK queries. In the aforementioned sequence the datasets are in ascending order w.r.t. the frequency of URIs starting with p in D_i (the order that we follow in `PrefixIndex`).

Additionally, the fact that we start to send ASK queries for a URI containing a prefix p from the dataset with the most URIs starting with p makes it more possible the answer of first ASK query to be true. In fact, the dataset containing the most URIs for a prefix is usually the dataset of the publisher of this prefix' URIs.

U'_i	**Freq. of** p
U'_1	1,000,000
U'_2	5,000
U'_3	10,000

Table 4.5: Frequency for a prefix p

Position 0,1,2	**ASKs**
D_1,D_2,D_3	2,005,000
D_1,D_3,D_2	2,010,000
D_2,D_1,D_3	1,010,000
D_2,D_3,D_1	20,000
D_3,D_1,D_2	1,020,000
D_3,D_2,D_1	25,000

Table 4.6: ASKs per combination for the worst case

4.2 The Lattice of Measurements

After the creation of the element index, we can compute the commonalities between any subset of datasets in $\mathcal{D}$. For (a) speeding up the computation of the intersections of URIs and (b) visualizing these measurements (for aiding understanding), we propose a method that is based on a lattice (specifically on a meet-semilattice). If the number of datasets is not high, the lattice can be shown entirely, otherwise (i.e. if the number of datasets is high) it can be used a navigation mechanism, e.g. the user could navigate from the desired dataset (at the bottom layer) upwards, as a means for dataset discovery. Specifically, we propose constructing and showing the measurements in a way that resembles the *Hasse Diagram* of the *poset*, partially ordered set, $(\mathcal{P}(\mathcal{D}), \subseteq)$. The lattice can be represented as a Directed Acyclic Graph $G = (V, E)$ where the edges point towards the direct supersets, i.e. each directed edge starts from a set B and points to a superset B', where $B \subset B'$ and $|B| = |B'|-1$ (i.e. there exists exactly one element of B' which is not an element of B). The empty set is the unique `source` node of G (i.e., node with zero in-degree) and the set containing all the datasets (i.e. $\mathcal{D}$) is the unique `sink` node of G (i.e., node with zero out-degree). A lattice of $\mathcal{D}$ datasets contains $|\mathcal{D}|+1$ levels while the value of each level k ($0 \leq k \leq |\mathcal{D}|$) indicates the number of datasets that each subset of level k contains (e.g., level 2 corresponds to pairs). For computing the measurements that correspond to each node (of the $2^{|\mathcal{D}|}$ nodes), one could follow a straightforward approach, i.e. scan the entire element index once per each node, but that would require exponential in number scans, hence prohibitively expensive for high number of datasets. To tackle this problem, below we introduce (i) a *top-down* and (ii) a *bottom-up* lattice-based incremental algorithm that both require *only one scan* of the element index for computing the commonalities between any set of datasets. We should stress that it is not required to compute the entire lattice, one can use the proposed approach for computing only the desired part of the lattice and its incremental nature can offer significant speedups (as we shall see in §4.3.2).

Level	Subsets
0	0000:{}
1	1000:{D_1},0100:{D_2} 0010:{D_3},0001:{D_4}
2	1100:{D_1,D_2},1010:{D_1,D_3},1001:{D_1,D_4} 0110:{D_2,D_4},0101:{D_2,D_4},0011:{D_3,D_4}
3	1110:{D_1,D_2,D_3},1101:{D_1,D_2,D_4} 1011:{D_1,D_3,D_4},0111:{$D_1$2,D_3,D_4}
4	1111:{D_1,D_2,D_3,D_4}

Table 4.7: Subsets of four datasets in normal and binary representation

4.2.1 Lattice Construction

Here we describe how to build the nodes and the edges of the lattice of $|\mathcal{D}|$ datasets. In the following algorithms, each subset is represented as a binary number b. Each dataset is associated with a position in these numbers, and a $'1'$ in that position signifies its presence, e.g. the set of datasets $\{D_3, D_4\}$ is represented by $b = 0011$. Table 4.7 shows how each subset of a power set of four datasets can be represented as a binary number.

Algorithm 3 finds all the subsets of $|\mathcal{D}|$ datasets recursively. The algorithm starts by adding the empty set (lines 6-7) and continues with the next level by using an iteration. In each iteration a "0" is replaced by a "1" (line 10) which means that a new dataset is added (line 11). Moreover, when a new subset (e.g., subset with $b = 1100$) is found, the algorithm keeps stable the binary number until the position of the last "1" (e.g., b=**11**00) and recursively finds all the supersets of this subset (lines 12-13). In particular, the main notion is that each recursion cycle ends only if all the supersets of a specific subset have been produced until that time. The complexity of this algorithm is $\mathcal{O}(2^{|\mathcal{D}|})$ since each subset is found once. Figure 4.2 shows the exact sequence of the creation of the power set by using Algorithm 3. A directed edge from a lower level to a higher level shows a recursive call, while a dashed directed edge indicates the end of a recursive call. On the contrary, a directed edge connecting two nodes of the same level represents an iteration.

Afterwards, by using Algorithm 4, we can find the incoming edges for all nodes of the lattice by replacing each position having a '1' with a '0' without changing the values of the other positions. The input is all the subsets of the power set which were computed by Algorithm 3. The algorithm reads all the nodes (lines 3-4) and for each node replaces each position having a '1' with a '0' without changing the values of the other positions (lines 8-10). For example, the incoming edges of the subset with $b = 0111$ are 0011 0101 and 0110. The complexity of this algorithm is $\mathcal{O}(2^{|\mathcal{D}|})$ since each subset is read once.

Finally, we can optimize Algorithm 3 based on the following observation. Let k be a level where $k = 0, ..., (|\mathcal{D}| + 1)/2$ where $|\mathcal{D}|$ corresponds to the number of datasets. Moreover, let k' be a level where $k' = |\mathcal{D}| - k$. The number of subsets for the level k' is $binomialCoefficient_{k'} = \frac{|\mathcal{D}|!}{k'!(|\mathcal{D}|-k')!} = \frac{|\mathcal{D}|!}{(|\mathcal{D}|-k)!(|\mathcal{D}|-(|\mathcal{D}|-k))!} =$

Algorithm 3 Create Lattice Nodes

Input: $|\mathcal{D}|$: the number of datasets
Output: All the subsets for each level of the lattice

```
1: char bitArray[𝒟]
2: for i ← 0 to |𝒟 − 1| do
3:     bitArray[i] ← 0
   createLatticeNodes(bitArray,0,0)
4:
5: function createLatticeNodes(char[] bitArray, int stableDigits, int level)
6:     if (level = 0) then
7:         subsets(level) ← subsets(level) ∪ bitArray
8:     level ← level + 1
9:     for i ← stableDigits to |bitArray.size − 1| do
10:        bitArray[i] ← 1
11:        subsets(level) ← subsets(level) ∪ bitArray
12:        if (i + 1 < |bitArray.size|) then
13:            createLatticeNodes(bitArray,i+1,level)
14:        bitArray[i] ← 0
```

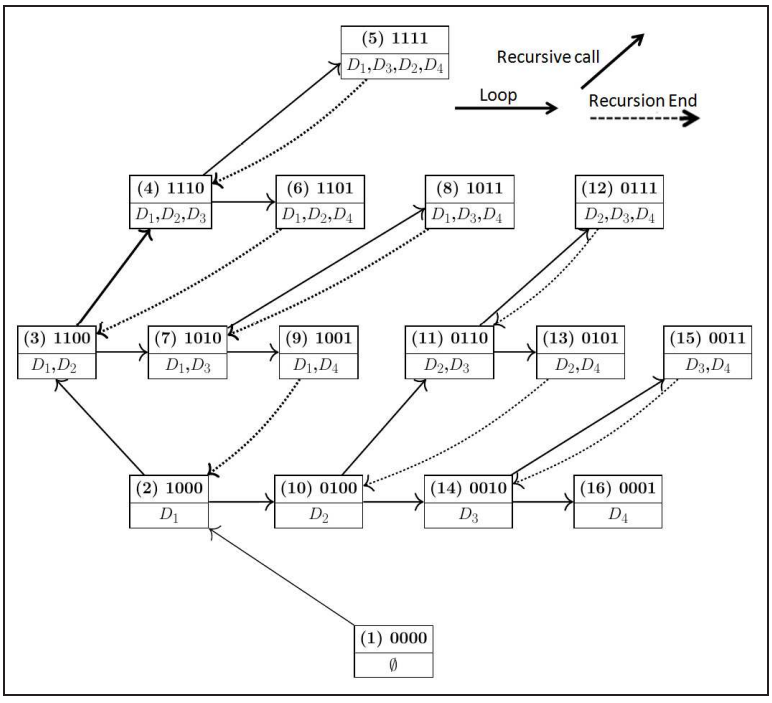


Figure 4.2: Lattice Nodes Creation Sequence

Algorithm 4 Create Lattice Edges

Input: All the subsets of the lattice
Output: Finds all incoming edges for each subset of the lattice

```
1: function CREATE LATTICE EDGES(subsets)
2:     level ← 1
3:     while level ≤ subsets.levels do
4:         for all B_i ∈ subsets(level) do
5:             char[] bitArray ← B_i
6:             for i ← 0 to |bitArray.size − 1| do
7:                 if bitArray[i] = 1 then
8:                     bitArray[i] ← 0
9:                     edges(B_i) ← edges(B_i) ∪ bitArray
10:                    bitArray[i] ← 1
11:        level ← level + 1
       return edges
```

$\frac{|\mathcal{D}|!}{(|\mathcal{D}|-k)!k!} = binomialCoefficient_k$. This means that k and k' always contain the same number of subsets. For instance, in Figure 4.3, for $k = 1$ and $k' = |\mathcal{D}| - k = 4 - 1 = 3$, levels k and k' contain four nodes. Moreover, the complement of each node of k produces a node of k'. For example, in Figure 4.3, all the nodes of the third level {0111, 1011, 1101, 1110} can be produced from the complement of first level's nodes {1000, 0100, 0010, 0001} and the node of the fourth level {1111} can be produced from the complement of the node of level zero {0000}. Therefore, there is no need to explore the last $(|\mathcal{D}| + 1)/2$ levels, since they can be produced from the first $(|\mathcal{D}| + 1)/2$ levels.

4.2.2 Making the Measurements of the Lattice Incrementally

For a subset B, let $directCount(B)$ denote its frequency in the element index, i.e. $directCount(B) = |\{ u \in Left(ei) \mid ei(u) = B \}|$.
We can compute these counters by scanning the element index *once* (in our running example, the outcome is shown in Step 4 of Figure 4.1). Now let $Up(B) = \{B' \in \mathrm{P}(\mathcal{D}) \mid B \subseteq B', directCount(B') > 0\}$. The key point is that for a subset B, the sum of the $directCount$ of $Up(B)$ gives the intersection of the real world objects of the datasets in B, i.e.

$$co_{\sim}(B) = \sum_{B' \in Up(B)} directCount(B') \tag{4.9}$$

because the URIs belonging to the intersection of a superset certainly belong to the intersection of each of the subsets of this superset, as stated by the following proposition.

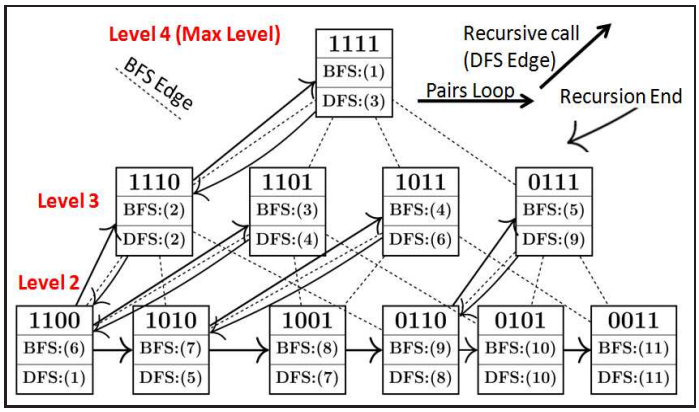


Figure 4.3: Lattice Traversal (BFS and DFS)

Proposition 2 *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 [36].)*

Now we will describe the two different ways for computing the *entire lattice* or a *part of* it.

The **top-down** approach starts from the maximum level having at least one subset B such that $B \in Left(directCount)$. At first, we add B to $Up(B)$ if $B \in Left(directCount)$ and then we compute $co_{\sim}(B)$. Afterwards, the list $Up(B)$ of the current node is "transferred" to each subset B' of the lower level since $B' \subseteq B$ implies $Up(B) \subseteq Up(B')$. For example, if $1110 \in Up(1110)$ then surely $1110 \in Up(1100)$. After finishing with the nodes of the current level, we continue with the nodes of the previous level, and so on. We avoid passing from subsets having $co_{\sim}(B) = 0$ by starting from the maximum level having at least one subset B where $B \in Left(directCount)$ (i.e., all the nodes being in a greater level have $co_{\sim}(B) = 0$) and by creating a node B for the remaining levels only if $|Up(B)| > 0$. As an example, in Figure 4.3 the second (middle) box of each node indicates the order by which it is visited. As we can see, we start from the maximum level (e.g., level 4) and we continue with the triads and finally with the pairs. The dashed edges represent the edges that are created by the algorithm. In our running example (see Figure 4.1), we can observe in Step 6 the final intersection value of each nodes and all the Up(B) for each subset B. For instance, we observe that the $Up(1001)$ are the subsets **1111**, **1011** and **1001**. If we sum their $directCount$, we find $co_{\sim}(1001)$, which is 5.

The time complexity of this algorithm is $\mathcal{O}(|V|+|E|)$, where $|V|$ is the number of vertices ($|V| = 2^{|\mathcal{D}|}$) and $|E|$ the number of edges, and it holds that $|E| = |\mathcal{D}| * 2^{(|\mathcal{D}|-1)}$. As regards memory requirements, this algorithm will create all edges for each node and it has to keep in memory each subset B (and $Up(B)$) of a specific level k (the number of nodes V_k of level k is given by $V_k = \binom{|\mathcal{D}|}{k}$) because the traversal is BFS (breadth-first search).

In the **bottom-up** approach we first assign the $Up(B)$ to each subset B of level two (the level that contains nodes of each pair of datasets) and we continue upwards. However, and in order to reduce the main memory requirements, instead of covering entirely each level before going to its upper level, we follow a kind of Depth-First Search (*DFS*). We will call it *DFS* although we could also call it "Height First Search" since it starts from the leaves and goes towards the root. The visiting order of the nodes of the example of Figure 4.3 is shown in the third (bottom) box of each node. Again, the $co_{\sim}(B)$ of a node is computed by adding the *directCount* of $Up(B)$. However, a part of the list (or the whole list) $Up(B)$ of the current node is "transferred" to each superset B' (where $B' \supset B$) of the next level that has not been visited yet. In this way, we visit once each node and this lead to a total cost of one incoming edge per node, i.e. $|V|$ edges ($|V| = 2^{|\mathcal{D}|}$ for the whole lattice) instead of the $|\mathcal{D}| * 2^{(|\mathcal{D}|-1)}$ edges of the top-down approach. Moreover, we avoid again passing from subsets having $co_{\sim}(B) = 0$ since we know that when $co_{\sim}(B) = 0$ and $B \subset B'$ then $co_{\sim}(B') = 0$ (i.e. see Prop. 2). The time complexity of this algorithm is $\mathcal{O}(|V|)$, where $|V|$ is the number of vertices. Indeed, it passes once from each node and it creates one edge per node ($|V| + |V|$). Moreover, it needs space $\mathcal{O}(d)$ where d is the maximum depth of the lattice (in our case d is the maximum level having at least one B where $co_{\sim}(B) > 0$). However, we should take into account the cost of checking which of the elements of $Up(B)$ belong to $Up(B')$ since we cannot transfer all the $Up(B)$ to B' (because $B' \supseteq B \not\Rightarrow Up(B) \subseteq Up(B')$), e.g. in our running example $1110 \in Up(0110)$ but $1110 \notin Up(0111)$.

Cost analysis. If B_i in $Left(directCount)$, let $bits_1(B_i)$ be the number of 1's in B_i (e.g. $bits_1(1110)$ is 3), and obviously $2 \leq bits_1(B_i) \leq |\mathcal{D}|$. Each such B_i belongs to the $Up(B)$ of $2^{bits_1(B_i)} - 1$ subsets (we subtract 1 for the empty set). Since for each such B_i, we have to perform $2^{bits(B_i)}$ checks (i.e., one check when traversing the list of *directCount* nodes and $|2^{bits_1(B_i)} - 1|$ checks when traversing the lattice), it follows that the total cost of such checks is $checkCost = |\sum_{i=1}^{C} 2^{bits_1(B_i)}|$, where C is the number of nodes occurring in *Left(directCount)*, i.e. $C = |\{ei(u) \mid u \in U \cup SID\}|$. It is essentially the cardinality of the codomain of ei and obviously, $C \leq |U \cup SID|$ (as we shall see in the experiments $C \simeq 0.1\%$ of $|U \cup SID|$).

Comparison. Both algorithms pass from all nodes having $co_{\sim}(B) > 0$. The top-down approach requires creating $|\mathcal{D}|/2$ times more edges, however the bottom-up approach has the additional *checkCost*.

Proposition 3 *If the frequency of URIs to datasets follows a power-law distribution (specifically if we group and order in descending order the directCount nodes according to their number of bits, and assume that the number of nodes of the n-th category ($2 \leq n \leq |\mathcal{D}|$) is given by $f(n) = k * (m/2)^{n-2}$, where k is the number of such nodes having $bits_1(B) = 2$, $m/2$ is the reduction factor ($1 < m \leq 2$), and assume $k = |\mathcal{D}^2|/4$), then the bottom-up approach is more efficient than the top-down if $|\mathcal{D}|^2 * \frac{m^{|\mathcal{D}|-1}-1}{m-1} < (|\mathcal{D}| - 2) * 2^{|\mathcal{D}-1|}$.*

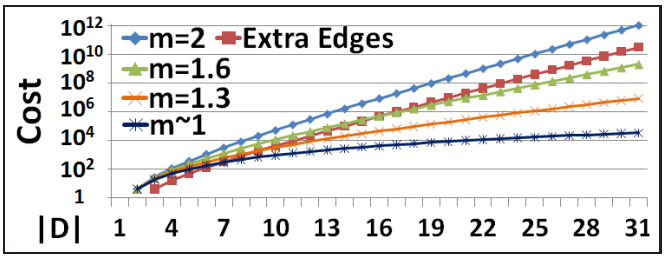


Figure 4.4: CheckCost vs extra edges for various m

Proof sketch: The bottom-up approach is better than the top-down when $checkCost <$ extra edges of the top-down. If we subtract the edges of the bottom-up approach (i.e., $2^{|\mathcal{D}|}$) from the edges of the top-down approach (i.e., $|\mathcal{D}| * 2^{|\mathcal{D}|-1}$) we get $(|\mathcal{D}|-2)*2^{(|\mathcal{D}|-1)}$. Now let calculate $checkCost$ for the assumed power-law distribution. The n-th category (each node B_n of the n-th category has $bits_1(B_n) = n$) of nodes has $k*(m/2)^{|bits_1(B_n)|-2}$ nodes and $k*(m/2)^{|bits_1(B_n)|-2}*2^{|bits_1(B_n)|}$ checks. The corresponding sum (i.e., of checks) leads to $checkCost = 4k * \frac{m^{|\mathcal{D}|-1}-1}{m-1}$. By assuming $k = |\mathcal{D}^2|/4$ (which is quite reasonable based on our experiments) we get the inequality of the proposition.

As it is shown in Figure 4.4, it follows that (a) for $m \approx 1$ (i.e., nodes are reduced by half as categories grow), the bottom-up approach is better that the top-down when $|\mathcal{D}| > 6$, (b) for $m = 2$ (i.e., each category has the same number of nodes) top-down is always better, whereas (c) for $m = 1.6$ the bottom-up traversal is more efficient than the top-down for $|\mathcal{D}| \geq 17$. The experiments in §4.3.2 are explained by this analysis, and we identify the same trade-off for $m = 1.6$. As regards memory requirements, the top-down approach needs significantly more space since it keeps in the worst case in memory the nodes of a specific level k (i.e., $\binom{|\mathcal{D}|}{k}$ nodes) whose number can be huge for big lattices while the bottom-up at most $|\mathcal{D}|$ nodes (i.e., maximum depth is $|\mathcal{D}|$). Finally, an alternative straightforward, but less efficient, way to compute the lattice is to use a $directCount$ scan (dcs) approach for each subset. The complexity of dcs is $\mathcal{O}(|V| * C)$ (exponential in number $directCount$ scans).

Computing a Part of the Lattice. Here we describe how one can compute only parts of the lattice.

• *Single Node.* For computing a specific node B, we can scan all $B_i \in Left(directCount)$ and sum all the $directCount(B_i)$ if $B_i \in Up(B)$. Its complexity is $\mathcal{O}(C)$.

• *Threshold-based Nodes.* For computing only the $co_\sim(B)$ of subsets that satisfy a specific threshold (e.g., $co_\sim(B) \geq 20$), it is beneficial to use the bottom-up approach. Indeed, fewer nodes will be created since we can exploit Prop. 2 to avoid creating nodes that are impossible to satisfy the threshold.

• *Lattice of a subset of Datasets.* For computing only the nodes of the lattice that contain datasets only from a particular subset D' ($D' \subset \mathcal{D}$), the previous analysis as regards the two types of traversal holds also in this case. The only difference

is that here instead of using the original *directCount* we use $directCount(B, D')$ defined as $directCount(B, D') = |\ \{\ u\ \in\ Left(ei)\ |\ B = ei(u) \cap D'\}|$ which can be produced by scanning once the element index.

• *All Nodes containing a particular dataset.* For computing only the nodes of the lattice that contain a particular subset D_i ($D_i \in \mathcal{D}$), we follow a similar approach, i.e. instead of using the original *directCount* we use $directCount(B, D_i)$ defined as $directCount(B, D_i) = |\ \{\ u\ \in\ Left(ei)\ |\ ei(u) = B\ and\ D_i \in B\ \}\ |$.

Finally, Figure 4.5 presents the lattice concerning the common URIs for the *MarineTLO* warehouse. One can see that the number of common URIs of DBpedia, FishBase, and Ecoscope are more than the number of common URIs among the subsets of the same level, while 74 common URIs are included in all datasets.

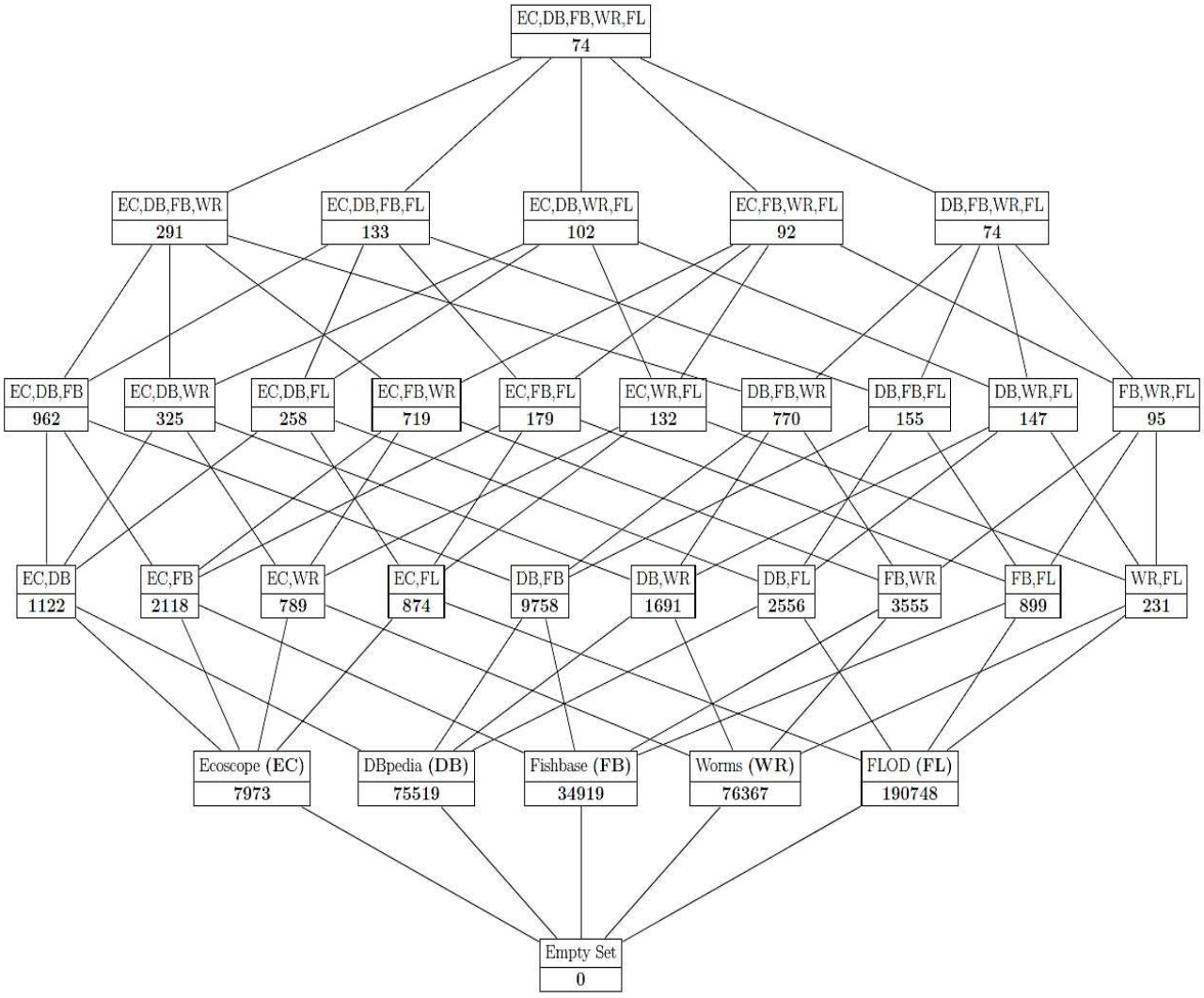


Figure 4.5: Common URIs Lattice for MarineTLO Warehouse

Domain	$\|\mathcal{D}\|$	|Triples|	|URIs|
Cross Domain **(CD)**	19	293,129,862	103,281,343
Geographical **(GEO)**	14	155,591,494	34,169,442
Life Sciences **(LF)**	17	66,684,349	9,725,521
Government **(GOV)**	45	61,189,128	6,896,850
Publications **(PUB)**	76	53,930,138	10,932,689
Media **(MED)**	9	15,267,271	4,434,038
Linguistics **(LIN)**	8	9,128,072	2,059,465
Social Networking **(SN)**	96	2,451,093	561,686
User Content **(UC)**	16	1,059,255	308,193
All	300	658,430,662	172,369,227

Table 4.8: Datasets Statistics

4.3 Experimental evaluation

Here we report the results of two kinds of experiments: a) interesting measurements over the entire LOD, and b) measurements that quantify the speedup obtained by the introduced techniques. We used a single machine having an i7 core, 8 GB main memory and 1TB disk space, and the triplestore *Openlink Virtuoso*[1] *Version 06.01.3127* for uploading the datasets and for sending SPARQL ASK queries.

Datasets. The set of datasets that was used in the experiments contains 300 datasets which were collected from the following resources: (a) the dump of the data which were used in [57], (b) online datasets from `datahub.io` website, and (c) subsets of (i)*DBpedia* version 3.9, (ii) *Wikidata*, iii) *Yago* and iv) *Freebase*. Table 4.8 shows the number of datasets, triples and URIs for each domain (in descending order w.r.t. their size in triples). Most datasets are from the social networking domain, however, most of them contain a small number of triples and URIs. On the contrary, 68% of triples and 79.7% of URIs are part of cross domain and geographical datasets although their union contains 33 (of 300) datasets. The selected set of datasets is quite representative and adequately large for the needs of this thesis (658 million triples, powerset of $|\mathcal{D}| = 300$ containing 2^{300} elements) given that the parallelization of the process is beyond the scope of this thesis.

4.3.1 Measurements over the Datasets

Statistics derived by the Indexes. Table 4.9 synopsizes some interesting statistics for the datasets and the creation of the element index. Firstly, according to the prefix index, each dataset's URIs contains on average 212 different prefixes. Element index contains 6.2 million real world objects (*rwo*). As one can see, there are 3,293,248 *rwo* (2.3% of all the *rwo*) which are part of three or more datasets. This percentage corresponds to 12,296,650 URIs (8% of unique URIs). The number of unique B having $directCount(B) > 0$ are 5,399, i.e. 0.1% of *ei* size. We

[1] *http://virtuoso.openlinksw.com/*

Category	Value
Prefix Index Size	63,803
Unique Real World Objects	141,269,960
Element Index Size (*rwo*)	6,242,344
Element Index Size (URIs)	17,840,499
Asks Number	6,684,242
rwo in 3 or more D_i	3,293,248
URIs corresponding to *rwo* in 3 or more D_i	12,296,650
Num. of Lattice Nodes (threshold ≥ 30)	130,525,631
Num. of Lattice Nodes (threshold ≥ 20)	1,541,968,012

Table 4.9: Index Creation Statistics

Category	Value
SameAs Triples	13,158,621
SameAs Catalog Size	18,789,593
SameAs Triples Inferred	19,450,107
Pairs sharing at least 1 real world object	6,708
New Pairs discovered due to SameAs Alg.	2,393
Triads sharing at least 1 real world object	74,432
New Triads discovered due to SameAs Alg.	48,658
SameAs Unique IDs	6,218,958

Table 4.10: SameAs Catalog Statistics

used the aforementioned *directCount*s for computing the $co_{\sim}(B)$ of 130 million lattice nodes (all pairs, triads, quads, quintets and each subset B where $|B| \geq 6$ and $co_{\sim}(B) \geq 30$) in 3.5 minutes and the $co_{\sim}(B)$ of 1.5 billion nodes (having $co_{\sim}(B) \geq 20$) in 35 minutes by using the bottom-up traversal that described in §4.2.2. Finally, we excluded from this computation URIs belonging to rdf, $rdfs$, owl and popular ontologies such as $foaf$.

Gain from transitive and symmetric closure computation. Regarding the `sameAs` catalog, Table 4.10 shows some statistics derived by the computation of transitive and symmetric closure. The computation of closure had as a result 2,393 new pairs (35.6% of the number of all connected pairs) and 48,658 new triads (65.3%) that have at least one common real world object (without taking into account URIs belonging in popular ontologies). Moreover, the algorithm found more than 19 million new `sameAs` relationships. Indeed, the increase percentage of `sameAs` triples was 147.8%. The unique URIs of the `sameAs` catalog are 18,789,593 while the different *rwo* are 6,218,958. It means that on average there are 3 URIs for a specific real world object. Finally, Figure 4.6 shows how many pairs exist for a number of *rwo* threshold (e.g., threshold 10 means that two datasets shares at least 10 *rwo*) and the analogous measurement for the triads.

Common real world objects among three or more datasets. Table 4.11 shows the ten subsets of size three or more having the most common real world objects (e.g., in descending order according to the number of common *rwo*). The

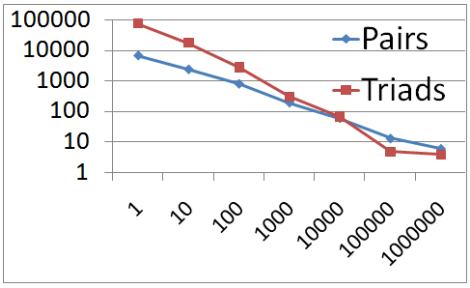


Figure 4.6: # of Pairs, Triads per Threshold

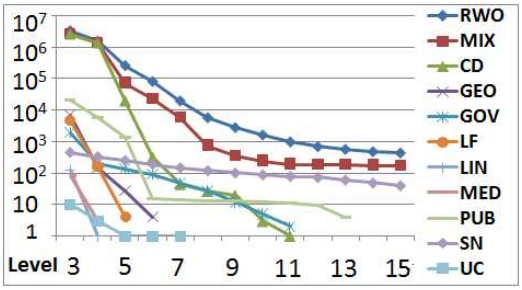


Figure 4.7: Unique(RWO) - Max Subset per Level

Datasets of subset B	$co_{\sim}(B)$
1: {DBpedia,Freebase,Yago}	2,709,171
2: {DBpedia,Freebase,Wikidata}	1,950,319
3: {DBpedia,Yago,Wikidata}	1,435,713
4: {Yago,Freebase,Wikidata}	1,434,407
5: {DBpedia,Yago,Freebase,Wikidata}	1,434,404
6: {DBpedia,GADM,Freebase}	107,968
7: {DBpedia,GeoNames,Freebase}	98,985
8: {DBpedia,GADM,Wikidata}	96,968
9: {GADM,Freebase,Wikidata}	96,968
10: {DBpedia,GADM,Freebase,Wikidata}	96,968

Table 4.11: Top-10 Subsets ≥ 3 with the most common *rwo*

most connected triad contains three cross domain datasets. Particularly, the subset comprising of the datasets *DBpedia*, *Freebase* and *Yago* shares 2.7 million of *rwo* while the quad that contains also *Wikidata* (apart from these datasets) contains 1.43 million of *rwo*. Afterwards, combinations of cross-domain and geographical datasets follows. The first triad that does not contain one of the aforementioned datasets includes three datasets from the publication domain (*d-nb.info*, *bnf.fr*, *id.loc.gov*) which share approximately 21 thousand *rwo*.

Figure 4.7 shows the unique real world objects and the maximum subset (e.g., subset with the most common *rwo*) per lattice level for each domain. The mix corresponds to subsets that possibly contain datasets from more than one domain. The most connected domain from level 3 to 6 is the cross domain whereas in the remaining levels (from 7 to 15) the domain with the most common *rwo* is the social networking domain. Moreover, regarding combinations with datasets from different domains, there exist 8 datasets that shares approximately hundreds of *rwo* and 15 datasets sharing over a hundred of *rwo*. Most of these *rwo* predominantly refer to geographical places and to popular persons. Generally, cross domain datasets take part in the most combinations with datasets from different domains.

Top datasets containing frequent real world objects. Regarding the datasets

Dataset D_i	***rwo* in** ≥ 3 D_i	**(% of** D_i ***rwo*)**
DBpedia	3,246,415	17.3%
Freebase	3,237,604	11.3%
Yago	2,712,930	48.0%
Wikidata	1,952,222	7.3%
GADM Geovocab	108,503	9.4%
GeoNames	102,747	0.4%
d-nb.info	65,076	4.2%
LinkedGeoData (LGD)	43,265	0.6%
Opencyc	34,313	26%
LMDB	30,225	2.3%

Table 4.12: Top-10 datasets with the most *rwo* existing at least in 3 datasets

URI	D_i **Deg.**	**Overall Deg.**	**Incr. %**
nyt:jordan_michael	11	108	881%
yago:Socrates	12	102	702%
dbpedia:Aristotle	42	145	245%
dbtropes:JamesBond	41	85	107%

Table 4.13: Increase degree percentage for 4 Entities

having the most real world objects in a subset of three or more datasets, *DBpedia* is the biggest dataset as we can observe in Table 4.12 (in ascending order with respect to the number of *rwo* in three or more datasets). It is rational that *DBpedia* is first in this category since it is the biggest hub of the LOD Cloud while the three other popular cross domain datasets follow. The other datasets are predominantly from the geographical domain while there exist datasets from the publications, media and lifescience domain containing more than ten thousand of such *rwo* objects. Additionally, 129 of 300 datasets (43%) contains at least hundreds of *rwo* that can be found in three or more datasets. Moreover, we can see the percentage of the *rwo* of each dataset that exist in three or more datasets. It is worth mentioning that *GeoNames* and *LinkedGeoData* have the lowest percentages regarding the datasets of Table 4.12 while the cross domain datasets the bigger ones.

Degree increase of entities due to connectivity. Table 4.13 shows for each of the four URIs (a) the dataset degree (incoming and outgoing properties of the URI in publisher's dataset), (b) the overall degree (incoming and outgoing properties of the URI in all the datasets) and (c) the increase percentage of the degree. In particular, for the URI "dbpedia:Aristotle" in *DBpedia* there are 42 distinct triples containing this entity. After the integration of information, there is a 245% increase in the degree of this URI, since there exists 145 distinct triples containing data about the entity *Aristotle*.

Average degree of real world objects per level. Finally, we compute the average degree of a number of real world objects according to the exact number of datasets that they exist. In particular, we collect 90 random *rwo* for each level

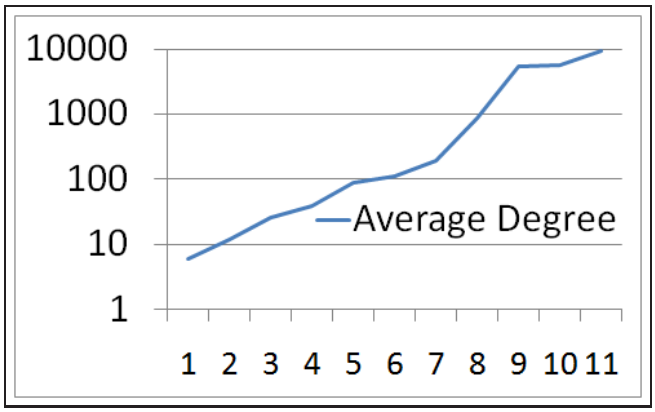


Figure 4.8: Average Degree for Real World Objects per Level

(e.g., 90 *rwo* existing exactly in one source, in two datasets and so on). In Figure 4.8 we can observe that as the number of datasets (in which *rwo* occurs) grows, the average degree increases to a great extent. The average degree of *rwo* of level two is two times higher than the degree of *rwo* of level one. It is worth noting that the average degree of *rwo* existing in eight datasets is 889 (148 times higher than degree of *rwo* of level one) while for *rwo* that are part of eleven datasets the average degree is 9,483. It means that these *rwo* have 10 times higher degree comparing to *rwo* of level eight and 1,580 times higher degree comparing to the *rwo* of level one. Therefore, most *rwo* existing in many datasets are very popular and are part of a big number of triples. Most of these *rwo* predominantly refers to geographical places and to popular persons.

4.3.2 Efficiency of Measurements

Here we focus on measuring the speedup obtained by the introduced indexes and their construction.

Savings by Prefix Index. Regarding the prefix index, the 89% of prefixes exist only in one dataset. However, this percentage corresponds only to the 10.8% of distinct URIs while the 11% of the prefixes to the 89.2% of URIs. In any implementation (e.g., index approach with or without ASK queries), the URIs starting with a prefix existing in one dataset can be ignored, therefore there is no need to compare these URIs with others (e.g., by sending an ASK query or by keeping them until the end). In our case, we ignored 16,689,866 URIs and we send ASK queries for a URI of a specific dataset only to the other datasets having more URIs for each prefix (see subsection 4.1.5). For this reason (e.g., the optimized sequence of datasets in prefix index) 6.68 million ASK queries (1 ASK query per 19 URIs having a prefix that can be found in two or more datasets and does not belong to a sameAs relationship) sent where in any random case the number could be much bigger.

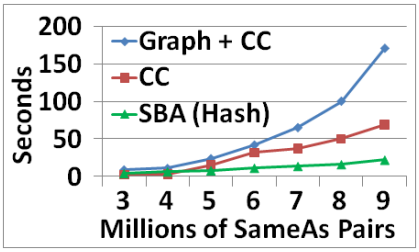


Figure 4.9: SameAs Catalog Construction time

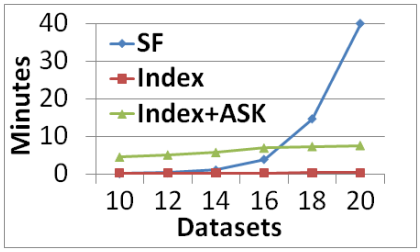


Figure 4.10: Comparison of different approaches

SameAs Signature-Based vs Connected Components Algorithm. Here we compare the signature-based algorithm (*SBA*) versus Tarjan's connected components (*CC*) algorithm [59] that uses Depth-First Search (*DFS*) and was described in §4.1.4. We performed experiments for 3 to 9 million randomly selected `sameAs` relationships and the results are shown in Figure 4.9. As one can see, the experiments confirmed our expectations since the signature-based algorithm is much faster than the combination of the creation of graph and *CC* algorithm while it is even faster than the *CC* algorithm as the number of `sameAs` pairs increases. Regarding the space, for 10 million or more pairs it was infeasible to create and load the graph due to memory limitations. For this reason we failed to run the *CC* algorithm, however, one can use techniques like those presented in [6] to overcome this limitation. The signature-based algorithm needed only 45 seconds to compute the closure of more than 13 million of `sameAs` pairs (which were used in the experiments).

Index Approach vs Straightforward Method. Here we compare the execution time of the following three methods that described in §4.1.5: (a) an index approach without ASK queries (*Index*) (b) an index approach with ASK queries (*Index+ASK*) and (c) a straightforward method (*sf*). For the index approaches we include in the execution time the creation of the prefix index and the calculation of lattice nodes by using the bottom-up approach that described in §4.2. Figure 4.10 shows the execution time in minutes for the aforementioned approaches for varying number of $|\mathcal{D}|$. The datasets of this experiment belong to the publications domain and the number of URIs is approximately nine millions. As the number of datasets grows, the execution time of the *sf* method increases exponentially. On the contrary, the execution time of the *Index* approach ranges from 17 to 23 seconds. The *Index+ASK* approach needs more time comparing to the other two approaches for 10-16 datasets. However, it is faster than the *sf* method for $|\mathcal{D}| > 16$ since its execution time does not increase so much when new datasets are added. In another experiment that is shown in Figure 4.11, we report the efficiency for various values of $|URIs|$ but with stable number of $|\mathcal{D}| = 17$. The number of URIs for these 17 datasets varies from 8,000 to 1,000,000. As one can see, the execution time of both *sf* and *Index+ASK* methods increases linearly as the number of URIs grows. In this experiment, the number of ASK queries increased linearly when

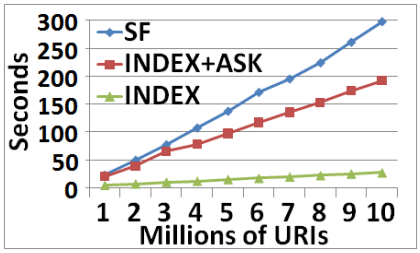


Figure 4.11: Comparison with stable $|\mathcal{D}| = 17$

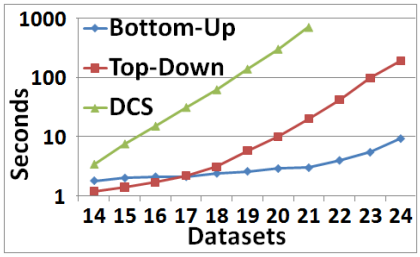


Figure 4.12: Execution Time of Lattice creation

more URIs added. Indeed, the execution time of *Index+ASK* approach highly depends on the number of ASK queries and their response time. Consequently, in case of adding millions of URIs having a prefix that can be found only in one dataset, the number of ASK queries will not be increased. Therefore, the execution time will be increased less than linearly in that case. Finally, the *Index* approach is again very fast and its execution time ranges from 5 to 27 seconds. In these experiments it is evident that the *Index* approach is always faster than the other approaches. Indeed, all the data fit in memory, thereby, it is more efficient to store to the index all the URIs and remove the URIs that exists in one dataset in the end. However, in the experiments of §4.3.1, where the data was infeasible to fit in memory, we used the *Index+ASK* approach.

Computation of power set intersections. Here we compare the performance of the two lattice incremental algorithms and the *directCount* scan (*dcs*) approach for each subset which described in §4.2.2. We selected 24 datasets that are highly connected (i.e., each subset of lattice has $co_{\sim}(B) > 0$) and the number of *directCount* nodes for the lattice of these 24 datasets is approximately 1,000 (from over 4 million *rwo*). In Figure 4.12, we can see that each incremental approach is faster than the *dcs* approach. Concerning the incremental approaches, one can clearly see the trade-off between the two approaches. Indeed, for lower number of datasets the top-down approach is faster, since the cost of edges creation is lower than the cost for checking the $Up(B)$ of each subset B. As the number of datasets grows (for ≥ 17 datasets), the bottom-up approach is faster, since the number of edges (and their cost) increases greatly comparing to the cost of checking the $Up(B)$. Moreover, for $|\mathcal{D}| \geq 25$, it was infeasible to use the top-down approach due to memory limitations while with the bottom-up approach we computed the $co_{\sim}(B)$ for more than 2^{30} subsets as mentioned in §4.3.1.

Chapter 5

Publishing and Exchanging metrics

Here we discuss how the proposed metrics can be *published* and *exploited*. The entire life cycle of these metrics is illustrated in Figure 5.1. In brief, at first (i.e., step 1) we create the indexes (in case of LOD Cloud Datasets). Then, the metrics can be computed (i.e., step 2) using SPARQL queries (i.e., for the semantic warehouses) or Java code by using the proposed indexes and algorithms (i.e., for the LOD Cloud datasets). Afterwards, the resulting measurements can be represented using an extension of VoID (which is briefly described later) and can be published in *datahub.io* or in an *SPARQL endpoint* (i.e., step 3). Finally, we can query and exploit these measurements for a number of tasks (i.e., step 4). In this section, we focus on steps 3 and 4 of Figure 5.1. More specifically, in §5.1 we describe the proposed extension of VoID while in §5.2 we introduce a website having links to novel services that exploit these metrics for the aforementioned tasks.

5.1 Publishing metrics through VoIDWH ontology

For publishing and exchanging these metrics we have used *VoID* [37]. VoID is an RDF-based schema that allows expressing metadata about RDF datasets. It is intended as a bridge between the publishers and the users of RDF data by making the discovery and the usage of linked datasets an effective process. Note also that, according to [57], about 15% of LOD datasets use VoID. The design of VoID has been driven by representing a number of both domain-dependent features (e.g., which type of data it contains) and domain-independent ones (e.g., who published it). Conceptually, it has been built around the notions of *void:Dataset*, *void:Linkset* and *RDF Links*. A *void:Dataset* is a set of RDF triples that are published, maintained or aggregated by a single provider. A *void:Linkset* is a collection of RDF Links between two datasets. An RDF Link is an RDF triple whose subject and object are described in different void:Dataset. Based on Dublin Core [54], VoID provides properties that can be attached to both *void:Dataset* and

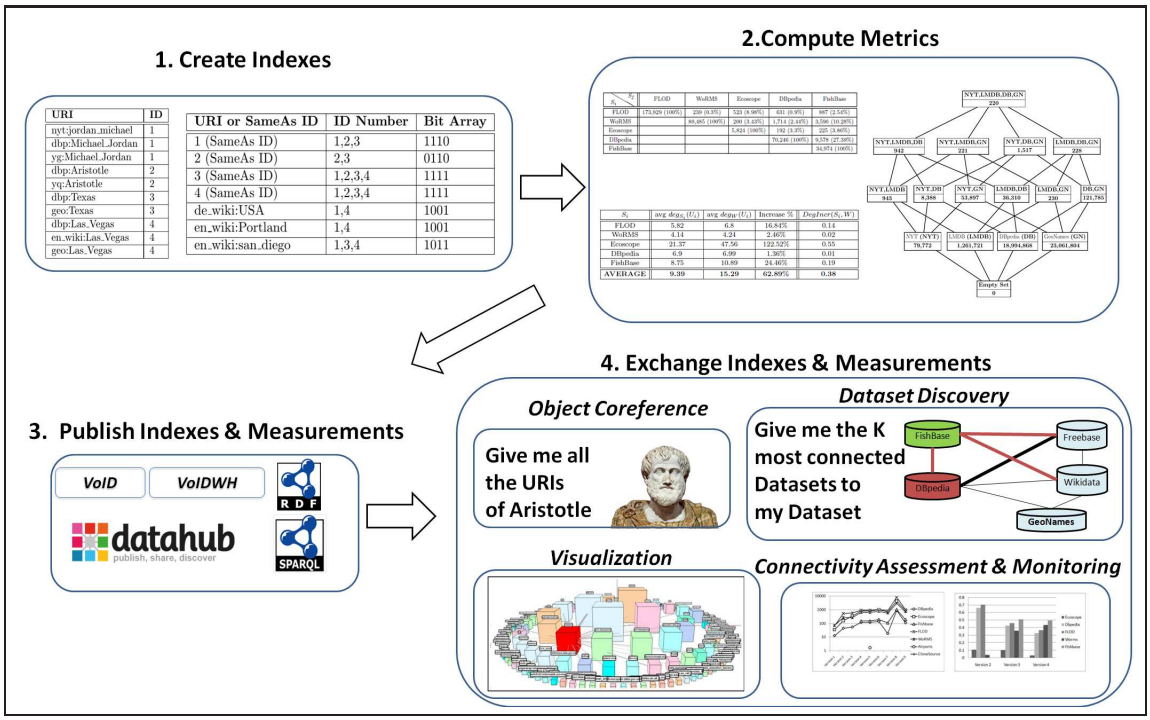


Figure 5.1: The process of computing, publishing and querying metrics.

void:Linkset to express metadata.

Apart from VoID, we have also used a particular extension [46], which is sufficient for describing our metrics. In brief, we could say that this extension allows the values of the various metrics to be expressed in a machine processable (and query-able) manner. If such information is exposed in a machine-readable format, it could be exploited in various ways, e.g.:

- For producing *visualizations* that give an overview of the LOD Cloud or contents of a warehouse.
- For *comparing different warehouses* and producing comparative reports.
- For aiding the *automatic discovery of related data* since software services/agents based on these metrics could decide which SPARQL endpoints to query based on time/cost constraints.
- For *crediting good sources* since these metrics make evident, and quantifiable, the contribution of a source to the warehouse.

An illustration of the main concepts we are using from VoID and its extension is depicted in Figure 5.2.

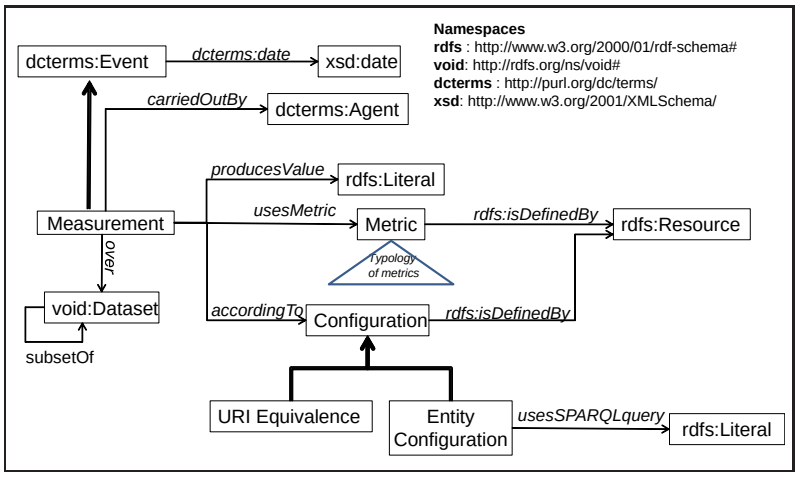


Figure 5.2: Schema for publishing and exchanging metrics.

We can see that there is the notion of *measurement* which is actually a specialization of *event* and therefore inherits the property *date*. A *measurement* is carried out by an *agent* using a specific *metric* according to one (or more) *configurations* over one (or more) *datasets* (atomic or composite) and produces a value (i.e., literal).

Each *metric* is an individual which means that it is assigned a URI and is *defined by* a resource (e.g., the DOI of the scientific paper that defined that *metric*). The notion of *configuration* concerns issues that explain how the *measurement* was done. Furthermore we use two specializations of the *configuration* class; the first concerns the way *URI equivalence* is defined, while the second

concerns how the *entities of interest* are defined. Regarding the latter the current modeling allows someone to specify the desired set of entities by providing a SPARQL query that returns them. The extension is accessible at `http://www.ics.forth.gr/isl/VoIDWarehouse`, as well as in Linked Open Vocabularies (LOV)[1]. More information about the exploitation of this extension for describing the `MarineTLO`-based warehouse can be found in [46].

5.2 Novel Services

In this section, we introduce ways to exploit measurements. In brief, we introduce a 3D visualization that exploit such metrics, and a website containing answerable queries that can be used for dataset discovery and for obtaining complete information about a URI (i.e., object coreference).

5.2.1 3D Visualization

Since there are many parameters the problem of understanding can be quite complex. To further alleviate the difficulty, below we propose a 3D visualization that can aid the user to get an overview.

We have adopted a quite familiar metaphor, specifically that of a *urban area*. The main idea is to visualize each source of the warehouse as a building, while the proximity of the buildings is determined by their common URIs and literals (note that the number of `sameAs` relationships between their URIs could also be considered). Finally, the number of `sameAs` relationships created by instance matching, can be visualized as bridges (or strings) between the buildings.

In particular, each source S_i corresponds to a building b_i. The volume of the building corresponds to $|triples(S_i)|$. Since $|triples(S_i)| \approx (|U_i| + |Lit_i| + |BN_i|) * Deg(U_i)$, (where BN_i denotes the set of blank nodes of the source i) we decided to set the height of the building analogous to $|U_i| + |Lit_i| + |BN_i|$, and the footprint of the building analogous to $deg_{S_i}(U_i)$. Specifically, assuming square footprints:

$$height(b_i) = |U_i| + |Lit_i| + |BN_i| \tag{5.1}$$

$$width(b_i) = \sqrt{deg_{S_i}(U_i)} \tag{5.2}$$

In this way, the volume of the building b_i approximates $|triples(S_i)|$; if its degree is low it will become a high building with a small footprint, whereas if its degree is high then it will become a building with big footprint but less tall. For getting building sizes that resemble those of a real urban area, a calibration is required, specifically we use an additional parameter K, through which we can obtain the desired average ratio of height/width of the buildings. The new formulas are:

$$height(b_i) = (|U_i| + |Lit_i| + |BN_i|)/K \tag{5.3}$$

[1] `http://lov.okfn.org/dataset/lov/details/vocabulary_voidwh.html`

$$width(b_i) = \sqrt{deg_{S_i}(U_i) * K} \tag{5.4}$$

Urban building usually have height > width, so one approach for setting automatically the value for K is to select the smallest K such $height_{avg} > width_{avg}$ (if one prefers to have 3 floor buildings he can select the smallest K such that $h_{avg} > 3 * width_{avg}$). In our case for a desired ratio of around 5, we used $K = 500$. Table 5.1 shows the building sizes of warehouse version 4.

S_i	$\|triples(S_i)\|$	$\|U_i\|$	$\|Lit_i\|$	$\|BN_i\|$	avg $deg_{S_i}(U_i)$	**Height**	**Width**
FLOD	904,691	159,042	115,573	27,812	6.28	**604,8**	**167.9**
WoRMS	1,382,748	51,649	217,759	271,352	4.43	**1081.2**	**67.2**
Ecoscope	61,597	6,145	14,122	896	10.78	**42.3**	**163.7**
DBpedia	782,479	114,556	130,287	32,890	6.90	**555,5**	**144.6**
FishBase	2,332,739	35,089	146,394	490,940	5.99	**1344.9**	**68.9**
CloneSource	641,586	47,794	46,716	54,932	9.86	**298.9**	**135.6**
Airports	88,714	12,027	25,901	0	7.55	**75.9**	**62.5**

Table 5.1: Buildings' sizes.

Each building has a location, i.e., x and y coordinates. The more common URIs and literals two source have, the closer the corresponding buildings should be. Below we describe the approach for computing the locations of the buildings. We can define the similarity between two sources (based on their common URIs/literals and sameAs relationships), denoted by $sim(S_i, S_j)$, as follows:

$$sim(S_i, S_j) = \frac{1}{2}\left(\frac{|U_i \cap U_j|}{min(|U_i|, |U_j|)} + \frac{|Lit_i \cap Lit_j|}{min(|Lit_i|, |Lit_j|)}\right) \tag{5.5}$$

S_i \ S_j	FLOD	WoRMS	Ecoscope	DBpedia	FishBase	CloneSource	Airports
FLOD	1	0.0870	0.1007	0.03884	0.1330	0.0955	0.0516
WoRMS		1	0.0381	0.0914	0.3207	0.7971	0.0003
Ecoscope			1	0.0264	0.0337	0.0248	0.0059
DBpedia				1	0.1999	0.1436	0.0051
FishBase					1	0.5220	0.0155
CloneSource						1	0.00002
Airports							1

Table 5.2: Computing the similarity of sources using $sim(S_i, S_j)$.

From $sim(S_i, S_j)$ we can compute the distance as follows:

$$dist(S_i, S_j) = \frac{1}{sim(S_i, S_j)} \tag{5.6}$$

Then we adopt a force-directed placement algorithm (similar in spirit with that of [70]) for computing x and y coordinates. Let $\equiv_{i,j}$ denote the set of `sameAs` relationships between the URIs of S_i and S_j. Each $\equiv_{i,j}$ can be visualized as a bridge that connects b_i and b_j. The volume of the bridge can be analogous to

$| \equiv_{i,j} |$. Three snapshots from different points of view of the produced 3D model are shown in Figure 5.3. The model can give an overview of the situation in a quite intuitive manner. The relative sizes of the buildings allow the user to understand the relative sizes in triples of the sources. The proximity of the buildings and the bridges make evident the commonalities of the sources, while the side size of each building indicates the average degree of the graph of each source. Furthermore, the user can navigate into the model using a Web browser, in this example through the following address `http://62.217.127.128:8080/warehouseStatistics/Statistics.html`.

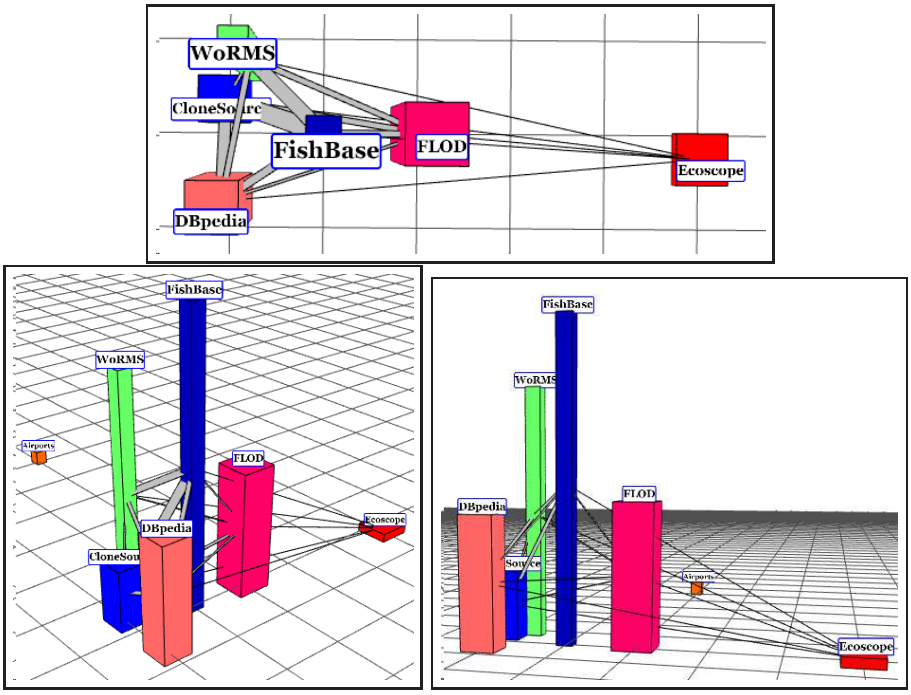


Figure 5.3: Three snapshots from different points of view of the produced 3D model.

5.2.2 LODsyndesis Website

We provide a website `http://www.ics.forth.gr/isl/LODsyndesis/` that contains links to (a) *datahub.io*[2] where we have published the measurements, (b) a page that lists queries that are now answerable, (c) a running SPARQL endpoint on which one can run such demo queries or his own queries, (d) a link to an ongoing interactive 3D visualization `http://www.ics.forth.gr/isl/3DLod/` that exploits some of the measurements. Below we show some of these queries.

[2]http://datahub.io/dataset/connectivity-of-lod-datasets

Number	Query
Query 1	Give me the top-K most connected datasets to my dataset
Query 2	Give me all the connected sources with FishBase, and how many URIs these datasets share with datasets from the geographical domain
Query 3	Give me all pairs of sources that were not connected, but now they are connected due to closure and the number of their common RWO
Query 4	Give me the increase of the commonalities of all pairs of sources due to closure in descending order
Query 5	Give me the K datasets that maximize the pluralism factor of the entities in my dataset
Query 6	Give me the connected triads from datasets coming from three different domains

Table 5.3: Queries for Dataset Discovery

5.2.2.1 Queries for Dataset Discovery

Here we provide more information about the queries that can be used for dataset discovery, while Table 5.3 shows six queries that can be used for this task. Below, we provide details for some of these queries.

Regarding *Query 1*, suppose that you publish a dataset and for connecting it to the rest datasets of the LOD you establish relationships with DBpedia (by having triples that refer to URI's from DBpedia). Now suppose that you would like to find related datasets (about the same real world objects) either for constructing a warehouse or for mediator-based query answering. Specifically, suppose that you would like to find the "K" more connected datasets (to your own). Without the proposed approach, you would get that only 1 dataset is connected to your dataset (i.e. only DBpedia). With the proposed indexes and measurements (that include the computation of the transitive closure of `sameAs` relationships), you could get much more datasets, i.e. datasets that you could not easily discover because they could have in common only few, or even none, URIs with your dataset. Moreover, the impact of the computation of transitive closure of `sameAs` relationships is shown in *Queries 3 and 4*, where we can see that the 19 million more `sameAs` relationships resulted to more than two thousand more connected datasets while for some pairs that were already connected, the increase of the number of their common *rwo* due to closure was huge.

We should also note that sometimes it is important to collect information about the same real world entity from several sources in order to verify or clean that information and eventually produce a more accurate or correct consolidated dataset or "widen" the information for a URI (i.e. to get more property-value pairs for that URI). Technically this reduces to constructing a dataset with high complementarity factor (or pluralism factor), i.e. a consolidated dataset where the number of datasets that provide information about each entity is high. For example, suppose that one would like to do this for the entities of a particular dataset, say D_1. To such tasks it is important to be able to answer queries, like

Number	Query
Query 7	Give me all the datasets that contain information about http://dbpedia.org/resource/Aristotle
Query 8	Give me all the equivalent URIs of http://dbpedia.org/resource/Aristotle
Query 9	Give me all the datasets from the publication domain containing information about yago:Socrates
Query 10	Give me all the URIs that are equivalent with the URIs of my dataset
Query 11	Give me the datasets that contain information for both Aristotle and Socrates
Query 12	Give me all the common RWO between Wikidata, DBpedia and Yago

Table 5.4: Queries for Object Coreference

Query 5, of the form: Give me the K datasets that maximize the pluralism factor of the entities in D_1. To find the sought datasets, it is important to compute the commonalities of all sets of K datasets that include D_1. Specifically, we need all nodes of the lattice that contain K datasets and one of these is D_1. Only the returned datasets can maximize the number of entities that have complementarity factor equal to K.

5.2.2.2 Queries for Object Coreference

Here we provide more information about the queries that can be used for object coreference, while Table 5.4 shows six queries that can be used in this case. In particular, we transform the `sameAs` catalog and the element index to RDF triples for achieving our target. Regarding the Queries, indicatively we can find which datasets contain information for an entity of interest, such as *Aristotle* (i.e., *Query 7*), the equivalent URIs for a given entity (i.e., *Query 8*) while we can find which of the URIs of our dataset are equivalent with URIs from coming from different datasets (i.e., *Query 10*). Moreover, one can find which datasets from specific domains (i.e., *Query 9*) contain information about an entity of interest.

Chapter 6

Conclusion

In this thesis, we proposed metrics, indexes and algorithms, for computing and quantifying the connectivity among several datasets. The proposed metrics can help us (a) obtain complete information about one particular URI (or a set of URIs), (b) discover a dataset which is relevant to another one, (c) compute and visualize the degree of connectivity between two or more datasets. At first, we applied and evaluated the approach in the context of domain specific semantic warehouses (in our case for a warehouse for the marine domain). In this case, the metrics were used for assessing the quality of the semantic warehouse and its underlying sources, and for monitoring the quality of the semantic warehouse after a reconstruction. The main metrics proposed are: (a) the matrix of percentages of the common URIs and/or literals, (b) the complementarity factor of the entities of interest, (c) the table with the increments in the average degree of each source, (d) the unique triple contribution of each source, and (e) a single-valued metric for quantifying the value of a source.

The values of (a),(b),(c) allow valuating the warehouse, while (c),(d) and (e) mainly concern each particular source. In addition we introduced metrics and plots suitable for monitoring the evolution of a warehouse.

Regarding experiments in larger number of datasets, since it would be prohibitively expensive to compute measurements that involve more than two datasets without special indexes, in this thesis we introduced a namespace-based *prefix index*, a *sameAs catalog* for computing the symmetric and transitive closure of the `sameAs` relationships encountered in the datasets, a semantics aware *element index* (that exploits the aforementioned indexes) and two lattice-based incremental algorithms for speeding up the computation of the intersection URIs of any set of datasets. We showed that with the proposed algorithm it takes only 45 seconds to compute the reflexive and transitive closure for 13 millions of `sameAs` relationships. As regards the computation of intersections we showed that the combination of the element index and the bottom-up incremental lattice-based algorithm is much faster (even 100 times faster for 20 datasets) than a straightforward method whose time complexity increases exponentially as the number of datasets and their size

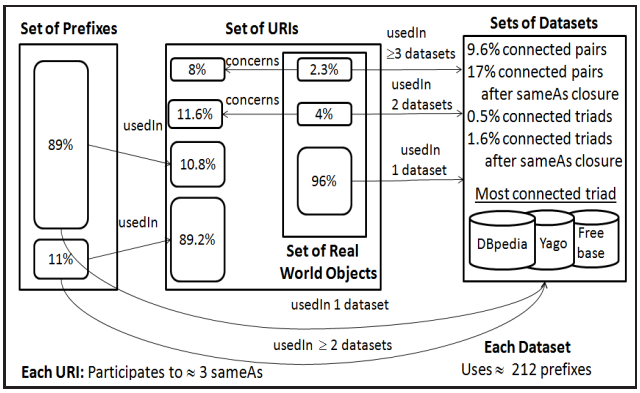


Figure 6.1: Results Synopsis

increase. We exploited the introduced indexes for making various measurements over the entire LOD. A few indicative are illustrated in Figure 6.1. The measurements showed that a dataset contains on average URIs with 212 different prefixes, and that 89% of the prefixes are used in 10.8% of URIs and occur in only one dataset. Concerning `sameAs` relationships, for a real world object exist on average 3 URIs. The transitive and symmetric closure of the `sameAs` relationships of all datasets yielded more than 19 million new `sameAs` relationships, and this increases the connectivity of the datasets: 35.6% of the 6,636 connected pairs of datasets are due to these new relationships. Regarding the *rwo* of element index, 60% of them exist in ≥ 3 datasets.

The measurements also reveal the "sparsity" of the current LOD cloud and make evident the need for better connectivity. Only 2.3% of real world objects (in number 3,293,248 real world objects) are part of three or more datasets. Most of these real world objects (belonging in ≥ 3 datasets) are part of cross domain datasets such as *DBpedia* and geographical datasets like *GeoNames*. Additionally, many datasets from the social networking domain are highly connected.

Finally, we proposed an extension of VoID for publishing, sharing and exploiting such measurements and we provided links to novel services that exploit the measurements.

Regarding future work, there are several topics that are worth investigating. One is to generalize the lattice-based approach for measuring other kinds of RDF features (such as common literals and triples). Another is to exploit the derived measurements for better visualizations and monitoring services of the LOD Cloud. An interesting direction is to investigate how to make the lattice of measurements adaptive, i.e. to avoid computing all measurements but be able to build those parts of the lattice that are more important and/or users/applications pose queries (in a way similar in spirit with [74] for data series). Another direction is to investigate the speedup that can be achieved by parallelizing the measurements in a MapReduce context, or by developing methods for approximate measurements. Finally, it would be also interesting to run the measurements for billions of triples and URIs (identically for the whole LOD Cloud).

Bibliography

[1] Ecoscope - Knowledge Base on Exploited Marine Ecosystems. `http://www.ecoscopebc.ird.fr/`.

[2] FishBase. `http://www.fishbase.org/`.

[3] FLOD - Fisheris Linked Open Data. `http://www.fao.org/figis/flod/`.

[4] Linked Data Integration Framework (LDIF). `http://www4.wiwiss.fu-berlin.de/bizer/ldif/`.

[5] WoRMS - World Register of Marine Species. `http://www.marinespecies.org/`.

[6] Charu Aggarwal, Yan Xie, and Philip S Yu. Gconnect: A connectivity index for massive disk-resident graphs. *Proceedings of the VLDB Endowment*, 2(1):862–873, 2009.

[7] Sören Auer, Jan Demter, Michael Martin, and Jens Lehmann. LODStats - an Extensible Framework for High-Performance Dataset Analytics. In *Knowledge Engineering and Knowledge Management*, pages 353–362. Springer, 2012.

[8] Donald P Ballou and Giri Kumar Tayi. Enhancing data quality in data warehouse environments. *Communications of the ACM*, 42(1):73–78, 1999.

[9] Nikos Bikakis and Timos K. Sellis. Exploration and visualization in the web of big linked data: A survey of the state of the art. In *EDBT/ICDT Workshops*, volume 1558, 2016.

[10] Christian Bizer. *Quality-Driven Information Filtering in the Context of Web-Based Information Systems (PhD Thesis)*. PhD thesis, Freie Universität Berlin, Germany, 2007.

[11] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009.

[12] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia-a crystallization point for the web of data. *Web Semantics: science, services and agents on the world wide web*, 7(3):154–165, 2009.

[13] Leonardo Candela, Donatella Castelli, and Pasquale Pagano. Making virtual research environments in the cloud a reality: the gcube approach. *ERCIM News*, 2010(83):32, 2010.

[14] Richard Cyganiak, Simon Field, Arofan Gregory, Wolfgang Halb, and Jeni Tennison. Semantic Statistics: Bringing Together SDMX and SCOVO. *LDOW*, 628, 2010.

[15] Mathieu d'Aquin and Enrico Motta. Watson, more than a semantic web search engine. *Semantic Web*, 2(1):55–63, 2011.

[16] F. Darari, W. Fariz, W. Nutt, G. Pirro, and S.Razniewski. Completeness Statements about RDF Data Sources and their Use for Query Answering. In *The Semantic Web–ISWC 2013*, pages 66–83. Springer, 2013.

[17] Jeremy Debattista, Christoph Lange, and Sören Auer. daq, an ontology for dataset quality information. *Linked Data on the Web (LDOW)*, 2014.

[18] Jeremy Debattista, Santiago Londoño, Christoph Lange, and Sören Auer. Luzzu-a framework for linked data quality assessment. *arXiv preprint arXiv:1412.3750*, 2014.

[19] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659. ACM, 2004.

[20] Renata Dividino, Thomas Gottron, Ansgar Scherp, and Gerd Gröner. From Changes to Dynamics: Dynamics Analysis of Linked Open Data Sources. In *PROFILES'14: Proceedings of the Workshop on Dataset ProfiIling and Federated Search for Linked Data*, 2014.

[21] Mohamed Ben Ellefi, Zohra Bellahsene, Stefan Dietze, and Konstantin Todorov. Dataset recommendation for data linking: an intensional approach. In *ESWC*, 2016.

[22] P. Fafalios and Y. Tzitzikas. X-ENS: Semantic Enrichment of Web Search Results at Real-Time. In *SIGIR'13*, pages 1089–1090, Dublin, Ireland, 2013. ACM.

[23] Christian Fürber and Martin Hepp. Using sparql and spin for data quality management on the semantic web. In *Business Information Systems*, pages 35–46. Springer, 2010.

[24] Christian Fürber and Martin Hepp. Swiqa-a semantic web information quality assessment framework. In *ECIS*, volume 15, page 19, 2011.

[25] Christophe Guéret, Paul Groth, Claus Stadler, and Jens Lehmann. Assessing linked data mappings using network measures. In *The Semantic Web: Research and Applications*, pages 87–102. Springer, 2012.

[26] Andreas Harth and Sebastian Speiser. On completeness classes for query evaluation on linked data. In *AAAI*. Citeseer, 2012.

[27] Andreas Harth, Jürgen Umbrich, Aidan Hogan, and Stefan Decker. YARS2: A federated repository for querying graph structured data from the web. In *The Semantic Web, 6th International Semantic Web Conference ISWC , Busan, Korea*, pages 211–224, 2007.

[28] Olaf Hartig. Provenance information in the web of data. In *LDOW*, 2009.

[29] Olaf Hartig and Jun Zhao. Using web data provenance for quality assessment. CEUR Workshop Proceedings, 2009.

[30] Olaf Hartig and Jun Zhao. Publishing and Consuming Provenance Metadata on the Web of Linked Data. In *Provenance and Annotation of Data and Processes*, pages 78–90. Springer, 2010.

[31] James Hendler. Data integration for heterogenous datasets. *Big data*, 2(4):205–215, 2014.

[32] A. Hogan, A. Harth, J. Umbrich, S. Kinsella, A. Polleres, and S. Decker. Searching and Browsing Linked Data with SWSE: The Semantic Web Search Engine. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4), 2011.

[33] Aidan Hogan, Andreas Harth, and Stefan Decker. Performing object consolidation on the semantic web data graph. In *Proceedings of the WWW2007 Workshop I^3,Banff, Canada*, 2007.

[34] Aidan Hogan, Andreas Harth, Jürgen Umbrich, Sheila Kinsella, Axel Polleres, and Stefan Decker. Searching and browsing linked data with swse: The semantic web search engine. *Web semantics: science, services and agents on the world wide web*, 9(4):365–401, 2011.

[35] Aidan Hogan, Jürgen Umbrich, Andreas Harth, Richard Cyganiak, Axel Polleres, and Stefan Decker. An empirical survey of linked data conformance. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:14–44, 2012.

[36] Thomas Jech. *Set theory*. Springer Science & Business Media, 2013.

[37] Michael Hausenblas Keith Alexander, Richard Cyganiak and Jun Zhao. Describing linked datasets with the void vocabulary, w3c interest group note, 2011.

[38] Graham Klyne and Jeremy J Carroll. Resource description framework (rdf): Concepts and abstract syntax. 2006.

[39] Tomáš Knap and Jan Michelfeit. Linked Data Aggregation Algorithm: Increasing Completeness and Consistency of Data, `http://www.ksi.mff.cuni.cz/~knap/files/aggregation.pdf`.

[40] Tomáš Knap, Jan Michelfeit, Jakub Daniel, Petr Jerman, Dušan Rychnovskỳ, Tomáš Soukup, and Martin Nečaskỳ. ODCleanStore: a Framework for Managing and Providing Integrated Linked Data on the Web. In *Web Information Systems Engineering-WISE 2012*, pages 815–816. Springer, 2012.

[41] Shirlee-ann Knight and Janice M Burn. Developing a framework for assessing information quality on the world wide web. *Informing Science: International Journal of an Emerging Transdiscipline*, 8(5):159–172, 2005.

[42] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd international conference on World Wide Web*, pages 747–758. ACM, 2014.

[43] Pablo N Mendes, Hannes Mühleisen, and Christian Bizer. Sieve: Linked Data Quality Assessment and Fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 116–123. ACM, 2012.

[44] Jan Michelfeit and Tomas Knap. Linked Data Fusion in ODCleanStore. In *International Semantic Web Conference (Posters & Demos)*, 2012.

[45] Paolo Missier, Khalid Belhajjame, and James Cheney. The w3c prov family of specifications for modelling provenance metadata. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 773–776. ACM, 2013.

[46] M. Mountantonakis, C. Allocca, P. Fafalios, N. Minadakis, Y. Marketakis, C. Lantzaki, and Y. Tzitzikas. Extending void for expressing the connectivity metrics of a semantic warehouse. In *1st International Workshop on Dataset Profiling & Federated Search for Linked Data (PROFILES'14)*, Anissaras, Crete, Greece, May 2014.

[47] M. Mountantonakis, N. Minadakis, Y. Marketakis, P. Fafalios, and Y. Tzitzikas. Quantifying the connectivity of a semantic warehouse and understanding its evolution over time (accepted for publication). *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2016.

[48] M. Mountantonakis and Y. Tzitzikas. On measuring the lattice of commonalities among several linked datasets. *Proceedings of the VLDB Endowment*, 2016.

[49] Markus Nentwig, Tommaso Soru, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm. Linklion: A link repository for the web of data. In *The Semantic Web: ESWC 2014 Satellite Events*, pages 439–443. Springer, 2014.

[50] Thomas Neumann and Gerhard Weikum. The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, 19(1):91–113, 2010.

[51] Damla Oguz, Belgin Ergenc, Shaoyi Yin, Oguz Dikenelli, and Abdelkader Hameurlain. Federated query processing on linked data: a qualitative survey and open challenges. *Knowledge Eng. Review*, 30(5):545–563, 2015.

[52] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com: a Document-Oriented Lookup Index for Open Linked Data. *Int. J. Metadata Semant. Ontologies*, 3(1):37–52, 2008.

[53] Peng Peng, Lei Zou, M Tamer Özsu, Lei Chen, and Dongyan Zhao. Processing sparql queries over distributed rdf graphs. *The VLDB Journal*, pages 1–26, 2015.

[54] A. Powell, M. Nilsson, A. Naeve, and P. Johnston. Dublin core metadata initiative - abstract model, 2005. White Paper.

[55] Eric Prud' Hommeaux, Andy Seaborne, et al. Sparql query language for rdf. *W3C recommendation*, 15, 2008.

[56] Laurens Rietveld, Wouter Beek, and Stefan Schlobach. Lod lab: Experiments at lod scale. In *Proceedings of the International Semantic Web Conference (ISWC). Springer*, 2015.

[57] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. Adoption of the linked data best practices in different topical domains. In *The Semantic Web–ISWC*, pages 245–260. Springer, 2014.

[58] Graeme G Shanks and Peta Darke. Understanding Data Quality and Data Warehousing: A Semiotic Approach. In *Third Conference on Information Quality (IQ'98)*, pages 292–309, 1998.

[59] Robert Tarjan. Depth-first search and linear graph algorithms. In *Twelfth Annual Symposium on Switching and Automata Theory*, pages 114–121. IEEE, 1971.

[60] Yannis Theoharis, Yannis Tzitzikas, Dimitris Kotzinos, and Vassilis Christophides. On graph features of semantic web schemas. *Knowledge and Data Engineering*, 20(5):692–702, 2008.

[61] E. Tsiflidou and N. Manouselis. Tools and Techniques for Assessing Metadata Quality. In *7th Metadata and Semantics Research Conference (MTSR'13)*, 2013.

[62] Giovanni Tummarello, Eyal Oren, and Renaud Delbru. Sindice.com: Weaving the open linked data. In *Proceedings of the International Semantic Web Conference (ISWC)*, volume 4825, pages 547–560, 2007.

[63] Y. Tzitzikas, C. Alloca, C. Bekiari, Y. Marketakis, P. Fafalios, M. Doerr, N. Minadakis, T. Patkos, and L. Candela. Integrating Heterogeneous and Distributed Information about Marine Species through a Top Level Ontology. In *Proceedings of the 7th Metadata and Semantic Research Conference (MTSR'13)*, Thessaloniki, Greece, November 2013.

[64] Y. Tzitzikas, N. Minadakis, Y. Marketakis, P. Fafalios, C. Alloca, and M. Mountantonakis. Quantifying the connectivity of a semantic warehouse. In *4th International Workshop on Linked Web Data Management (LWDM'14)*, Athens, Greece, 2014.

[65] Y. Tzitzikas, N. Minadakis, Y. Marketakis, P. Fafalios, C. Allocca, M. Mountantonakis, and I. Zidianaki. Matware: Constructing and exploiting domain specific warehouses by aggregating semantic data. In *11th Extended Semantic Web Conference (ESWC'14)*, Anissaras, Crete, Greece, May 2014.

[66] Yannis Tzitzikas, Mary Kampouraki, and Anastasia Analyti. Curating the Specificity of Ontological Descriptions under Ontology Evolution. *Journal on Data Semantics*, pages 1–32, 2013.

[67] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk - A Link Discovery Framework for the Web of Data. In *Proceedings of the WWW'09 Workshop on Linked Data on the Web*, 2009.

[68] Richard Y Wang and Diane M Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, pages 5–33, 1996.

[69] John Wieczorek, David Bloom, Robert Guralnick, Stan Blum, Markus Döring, Renato Giovanni, Tim Robertson, and David Vieglais. Darwin core: An evolving community-developed biodiversity data standard. *PLoS One*, 7(1):e29715, 2012.

[70] Stamatis Zampetakis, Yannis Tzitzikas, Asterios Leonidis, and Dimitris Kotzinos. Star-like auto-configurable layouts of variable radius for visualizing and exploring RDF/S ontologies. *Journal of Visual Languages & Computing*, 23(3):137 – 153, 2012.

[71] Amrapali Zaveri, Dimitris Kontokostas, Mohamed A Sherif, Lorenz Bühmann, Mohamed Morsey, Sören Auer, and Jens Lehmann. User-driven quality evaluation of dbpedia. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 97–104. ACM, 2013.

[72] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for linked data: A survey. *Semantic Web Journal*, 7(1):63–93, 2016.

[73] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM computing surveys*, 38(2):6, 2006.

[74] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. Indexing for interactive exploration of big data series. In *Proceedings of the ACM SIGMOD*, pages 1555–1566. ACM, 2014.