UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCES AND ENGINEERING

Services for Connecting and Integrating Big Number of Linked Datasets

by

Michalis Mountantonakis

PhD Dissertation

Presented

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

Heraklion, March 2020

UNIVERSITY OF CRETE

DEPARTMENT OF COMPUTER SCIENCE

Services for Connecting and Integrating Big Number of Linked Datasets

PhD Dissertation Presented

by **Michalis Mountantonakis**

in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

APPROVED BY:

[illegible]

Author: Michalis Mountantonakis

[illegible]

Supervisor: Yannis Tzitzikas, Associate Professor, University of Crete, Greece

[illegible]

Committee Member: Dimitrios Plexousakis, Professor, University of Crete, Greece

[illegible]

Committee Member: Kostas Magoutis, Associate Professor, University of Crete, Greece

Antoniou

Committee Member: Grigoris Antoniou, Professor, University of Huddersfield, United Kingdom

[illegible]

Committee Member: Manolis Koubarakis, Professor, National and Kapodistrian University of Athens, Greece

Sören Auer

Committee Member: Sören Auer, Professor, Leibniz University of Hannover, Germany

[illegible]

Committee Member: Giorgos Flouris, Principal Researcher, FORTH-ICS, Greece

[illegible]

Department Chairman: Angelos Bilas, Professor, University of Crete, Greece

Heraklion, March 2020

Acknowledgments

First, I would like to express my deepest gratitude to my supervisor, Associate Professor Yannis Tzitzikas for everything that I have learned from him and for all the support and encouragement he has given to me. During the past four years, he has devoted a lot of time and effort for helping me with my PhD dissertation and for writing together several research papers that were published in prestigious journals and conferences. Indeed, his immense knowledge, motivation, passion and patience have given me more power and spirit to improve my research and presentation skills and to finish my dissertation.

Apart from my supervisor, I would like to deeply thank the other two members of my Advisory Committee, Professor Dimitris Plexousakis and Associate Professor Kostas Magoutis, for their valuable comments and suggestions all these years, which helped me to improve my PhD thesis and my research skills. I am also grateful to Professor Grigoris Antoniou, Professor Manolis Koubarakis, Professor Sören Auer and Principal Researcher Giorgos Flouris, for accepting the invitation to become members of my examination committee, and for their valuable comments and suggestions, that helped me to improve my dissertation. Finally, I would like also to thank the anonymous reviewers of our papers for providing us valuable feedback, which was crucial for improving our work.

I gratefully acknowledge the funding received towards my PhD thesis from Hellenic Foundation for Research and Innovation and the General Secretariat for Research and Technology for the period August 2017 - March 2020. In particular, this dissertation was supported by the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under the HFRI PhD Fellowship grant (GA. No. 166). Moreover, I would like to acknowledge the support of the Institute of Computer Science of the Foundation of Research and Technology (FORTH-ICS), and especially the Information Systems Laboratory (ISL), for the financial support (from the last year of my undergraduate studies until July 2017), for the facilities, and for giving me the opportunity to travel in several places for presenting our research work. In particular, I would like to thank the Head of ISL, Professor Dimitris Plexousakis, and also all the faculty and staff members of ISL. Furthermore, I would like to acknowledge the support of BlueBridge (H2020, 2015-2018), since part of my dissertation was supported by this project.

Additionally, I want to express my deepest thanks to the undergraduate and postgraduate students of Computer Science Department, and to the staff of University of Crete. Especially, I desire to thank the secretaries of Computer Science Department and ELKE, for helping me during my undergraduate and postgraduate studies. Moreover, I would like to thank okeanos GRNET's cloud service (`https://okeanos.grnet.gr/`), for giving me the opportunity to have access and to use 14 machines, during the past four years. By using these machines, I managed to run most of the experiments which are included in this dissertation and to host several online services.

Furthermore, I want to thank the people that supported me all these years. Especially in some difficult situations during my postgraduate studies, they encouraged me to continue and to complete my dissertation. First, I want to deeply thank my beloved Efstathia for her love, support and patience, and for the many wonderful moments we have spent together. Moreover, I want to thank my family and especially the following persons: my grandfather Antonis, who is my biggest supporter, my parents Rena and Lefteris and my sister Maria, for believing in me and encouraging me all these years. Finally, I want to thank all my friends for all the good times we have shared all the years.

Στην αγαπημένη μου Ευσταθία, στον παππού μου Αντώνη, στους γονείς μου Ρένα και Λευτέρη και στην αδερφή μου Μαρία, για την αμέριστη αγάπη, την ενθάρρυνση και την υποστήριξη τους.

Abstract

Linked Data is a method for publishing structured data that facilitates their sharing, linking, searching and re-use. A big number of such datasets (or sources), has already been published and their number and size keeps increasing. Although the main objective of Linked Data is linking and integration, this target has not yet been satisfactorily achieved. Even seemingly simple tasks, such as finding all the available information for an entity is challenging, since this presupposes knowing the contents of all datasets and performing cross-dataset identity reasoning, i.e., computing the symmetric and transitive closure of the equivalence relationships that exist among entities and schemas. Another big challenge is Dataset Discovery, since current approaches exploit only the metadata of datasets, without taking into consideration their contents.

In this dissertation, we analyze the research work done in the area of Linked Data Integration, by giving emphasis on methods that can be used at large scale. Specifically, we factorize the integration process according to various dimensions, for better understanding the overall problem and for identifying the open challenges. Then, we propose indexes and algorithms for tackling the above challenges, i.e., methods for performing cross-dataset identity reasoning, for finding all the available information for an entity, methods for offering content-based Dataset Discovery, and others. Due to the large number and volume of datasets, we propose techniques that include incremental and parallelized algorithms. We show that content-based Dataset Discovery is reduced to solving optimization problems, and we propose techniques for solving them in an efficient way.

The aforementioned indexes and algorithms have been implemented in a suite of services that we have developed, called `LODsyndesis`, which offers all these services in real time. Furthermore, we present an extensive connectivity analysis for a big subset of LOD cloud datasets. In particular, we introduce measurements (concerning connectivity and efficiency) for 2 billion triples, 412 million URIs and 44 million equivalence relationships derived from 400 datasets, by using from 1 to 96 machines for indexing the datasets. Just indicatively, by using the proposed indexes and algorithms, with 96 machines it takes less than 10 minutes to compute the closure of 44 million equivalence relationships, and 81 minutes for indexing 2 billion triples. Furthermore, the dedicated indexes, along with the proposed incremental algorithms, enable the computation of connectivity metrics for 1 million subsets of datasets in 1 second (three orders of magnitude faster than conventional methods), while the provided services offer responses in a few seconds. These services enable the implementation of other high level services, such as services for Data Enrichment which can be exploited for Machine-Learning tasks, and techniques for Knowledge Graph Embeddings, and we show that this enrichment improves the prediction of machine-learning problems.

Keywords: Linked Data, RDF, Data Integration, Dataset Discovery and Selection, Connectivity, Lattice of Measurements, Big Data, Data Quality

Supervisor: Yannis Tzitzikas
Associate Professor
Computer Science Department
University of Crete

Περίληψη

Τα Διασυνδεδεμένα Δεδομένα (Linked Data) είναι ένας τρόπος δημοσίευσης δεδομένων που διευκολύνει το διαμοιρασμό, τη διασύνδεση, την αναζήτηση και την επαναχρησιμοποίησή τους. Ήδη υπάρχουν χιλιάδες τέτοια σύνολα δεδομένων, στο εξής πηγές, και ο αριθμός και το μέγεθος τους αυξάνεται. Αν και ο κύριος στόχος των Διασυνδεδεμένων Δεδομένων είναι η διασύνδεση και η ολοκλήρωση τους, αυτός ο στόχος δεν έχει επιτευχθεί ακόμα σε ικανοποιητικό βαθμό. Ακόμα και φαινομενικά απλές εργασίες, όπως η εύρεση όλων των πληροφοριών για μία συγκεκριμένη οντότητα αποτελούν πρόκληση διότι αυτό προϋποθέτει γνώση των περιεχομένων όλων των πηγών, καθώς και την ικανότητα συλλογισμού επί των συναθροισμένων περιεχομένων τους, συγκεκριμένα απαιτείται ο υπολογισμός του συμμετρικού και μεταβατικού κλεισίματος των σχέσεων ισοδυναμίας μεταξύ των ταυτοτήτων των οντοτήτων και των οντολογιών. Η ανακάλυψη δεδομένων (Dataset Discovery) επίσης αποτελεί μεγάλη πρόκληση, διότι οι τρέχουσες προσεγγίσεις αξιοποιούν μόνο τα μεταδεδομένα των πηγών, και δεν λαμβάνουν υπόψη τα περιεχόμενα τους.

Σε αυτή τη διατριβή, αναλύουμε το ερευνητικό έργο που έχει παραχθεί στον τομέα της Ολοκλήρωσης Διασυνδεμένων Δεδομένων με έμφαση σε τεχνικές που μπορούν να εφαρμοστούν σε μεγάλη κλίμακα. Συγκεκριμένα παραγοντοποιούμε το πρόβλημα σε διαστάσεις που επιτρέπουν την καλύτερη κατανόηση του προβλήματος και τον εντοπισμό των ανοικτών προκλήσεων. Εν συνεχεία προτείνουμε ευρετήρια και αλγορίθμους για την αντιμετώπιση των παραπάνω προκλήσεων, συγκεκριμένα μεθόδους για συλλογισμό επί των ταυτοτήτων των πόρων, για εύρεση όλων των πληροφοριών για μία οντότητα, για ανακάληψη πηγών βάσει περιεχομένου και άλλων. Λόγω του μεγάλου πλήθους και όγκου των πηγών, οι τεχνικές που προτείνονται περιλαμβάνουν αυξητικούς και παράλληλους αλγορίθμους. Δείχνουμε ότι η ανακάλυψη πηγών βάσει περιεχομένου ανάγεται στην επίλυση προβλημάτων βελτιστοποίησης και προτείνουμε τεχνικές για την αποδοτική επίλυσή τους.

Τα παραπάνω ευρετήρια και αλγόριθμοι έχουν υλοποιηθεί στη σουίτα υπηρεσιών που αναπτύξαμε που αναφέρεται με το όνομα `LODsyndesis`, η οποία προσφέρει όλες αυτές τις υπηρεσίες σε πραγματικό χρόνο. Επιπροσθέτως, παρουσιάζουμε μία εκτενή ανάλυση συνδεσιμότητας για ένα μεγάλο υποσύνολο πηγών του νέφους Ανοικτών Διασυνδεδεμένων Δεδομένων (LOD Cloud). Συγκεκριμένα αναφέρουμε μετρήσεις (συνδεσιμότητας και αποδοτικότητας) που αφορούν 2 δισεκατομμύρια τριπλέτες, 412 εκατομμύρια URIs και 44 εκατομμύρια σχέσεις ισοδυναμίας που προέρχονται από 400 πηγές, χρησιμοποιώντας από 1 έως 96 μηχανήματα για την ευρετηρίαση. Ενδεικτικά, χρησιμοποιώντας 96 μηχανήματα χρειάστηκαν λιγότερα από 10 λεπτά για τον υπολογισμό του συμμετρικού και μεταβατικού κλεισίματος, και 81 λεπτά για την ευρετηρίαση 2 δισεκατομμυρίων τριπλετών. Επιπρόσθετα, χρησιμοποιώντας τα ευρετήρια μαζί με τους προτεινόμενους αυξητικούς αλγορίθμους, κατέστη εφικτός ο υπολογισμός των μετρήσεων συνδεσιμότητας για 1 εκατομμύριο υποσύνολα πηγών σε 1 δευτερόλεπτο (τρεις τάξεις μεγέθους γρηγορότερα σε σχέση με συμβατικές μεθόδους), ενώ οι προσφερόμενες υπηρεσίες έχουν απόκριση δευτερολέπτων. Οι υπηρεσίες αυτές καθιστούν εφικτή και την υλοποίηση υπηρεσιών υψηλότερου επιπέδου, όπως υπηρεσίες εμπλουτισμού πηγών για χρήση από τεχνικές Μηχανικής Μάθησης καθώς και τεχνικές για Διανυσματικές Αναπαστάσεις Γράφων Γνώσης (Knowledge Graph Embeddings) και δείχνουμε ότι ο εμπλουτισμός αυτός βελτιώνει της προβλέψεις σε προβλήματα μηχανικής μάθησης.

Λέξεις κλειδιά: Διασυνδεδεμένα Δεδομένα, Ολοκλήρωση Δεδομένων, Ανακάλυψη και Επιλογή Πηγών Δεδομένων, Συνδεσιμότητα, Πλέγμα Μετρήσεων, Μεγάλα Δεδομένα, Ποιότητα Δεδομένων

Επόπτης: Τζίτζικας Γιάννης
Αναπληρωτής Καθηγητής
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Context and Motivation

In recent years, there has been an international trend towards publishing open data and an attempt to comply with standards and good practices that make it easier to find, reuse and exploit open data. Linked Data is one such method for publishing structured data, that facilitates their sharing, linking, searching and re-use. It allows data to be interlinked (by using URIs instead of simple values) for assisting their integration. The main objective of Linked Data is linking and integration for easing data discovery process, for performing data analysis and for offering integrated query answering, and a big number of datasets (over 10,000 [91]) has already been published according to the principles of Linked Data (and their number and size keeps increasing). The large volume of these datasets necessitates their semantic integration, their connectivity, the preservation of their provenance, and the assessment of their quality and veracity, for fulfilling the requirements of e-science. In particular, the processing and the analysis of a large volume of integrated data is crucial for any scientific field, for providing novel and accurate scientific results.

However, the semantic integration of data from these datasets at a large (global) scale has not yet been achieved, and this is perhaps one of the biggest challenges of computing today. Indeed, the integration process still requires a number of steps, some of which are difficult or costly. As it is stated in [83], "Integration requires spending resources on mapping heterogeneous data items, resolving conflicts, cleaning the data, and so on. Such costs can also be huge. Actually, the cost of integrating some sources may not be worthwhile if the gain is limited, especially in the presence of redundant data and low quality data". Moreover, according to Mark Schrieber[1] "Data scientists spend even 95% of their time on data discovery and data integration", whereas it has been written[2] that "Data scientists spend from 50 percent to 80 percent of their time in collecting and preparing unruly

[1]https://amsterdamdatascience.nl/wp-content/uploads/2020/01/Top-10-Data-Science-Blunders-Michael-Stonebraker.pdf

[2]http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html?_r=0

digital data, before it can be explored for useful nuggets".

The main objective of this dissertation is to design and develop innovative indexes, methods, algorithms and tools for assisting the process of data discovery and semantic data integration at a large scale.

1.2 Related Problems

Although the ultimate objective of LOD (Linked Open Data) is linking and integration, for enabling discovery and integrated query answering and analysis, even some basic tasks are nowadays challenging due to the scale and heterogeneity of the datasets. Generally, Linking and Integration at scale are inherently difficult problems for various reasons. Initially, datasets are produced and managed by different organizations according to conceptualizations, models, schemas and formats based on their needs and choice, and these data are stored in different locations and systems. Another difficulty is that the same real-world entities or relationships are referred with different names or URIs and in different natural languages; the synonyms and homonyms of natural languages further perplex identification and matching. Moreover, quite often there is not enough data for automated matching because datasets contain complementary information about the same entities. Data integration also has to tackle the fact that datasets usually contain erroneous, out-of-date and conflicting data. Finally, integration is not a one shot task in the sense that everything can change as time passes (e.g., schemas, ontologies, naming conventions, etc.), necessitating additional actions for curating the integrated access.

Due to the aforementioned problems, the execution of various tasks that are related to data integration at large scale is quite difficult. These tasks are listed below.

• Task A. Obtaining complete information about one particular entity or a set of entities by applying cross-dataset identity reasoning. (i.e., *Object Coreference*).

• Task B. Assessing the connectivity among any combination of datasets, and monitoring their evolution over time (i.e., *Connectivity Analytics*).

• Task C. Discovering the K most relevant datasets to a given dataset or/and to a particular task, by exploiting the whole contents of datasets, and not only metadata (i.e., *Dataset Search, Discovery and Selection*).

• Task D. Combining information from several datasets (i.e., *Data Enrichment*), e.g., for improving the execution of Machine-Learning based tasks.

• Task E. Assessing the quality of one or more datasets and estimating the reliability of a specific fact for an entity (i.e., *Data Quality*).

All the aforementioned tasks require special methods, indexes and measurements that involve more than two datasets. However, they are not available although they are of primary importance for the integration process in an open and involving environment.

1.3 Analysis of the Problems and Related Challenges

The main objective of this dissertation is to offer advanced services for covering the needs of the aforementioned tasks. In particular, for each task we desire to provide services for fulfilling the requirements and needs of scientists of any scientific field (or even simple users).

Concerning task A, it is of primary importance to find all the URIs and the complete information for an entity (say "Aristotle"), in order one to be able to widen the information or/and to build an integrated dataset for that entity. Regarding task B, the exploitation of measurements, that concern the connectivity among any combination of datasets (e.g., between pairs or triads of datasets), is useful for assessing the degree up to which datasets are connected, e.g., for answering queries such as "How many entities share Wikidata, DBpedia and YAGO?". Concerning task C, we desire to answer complex *Dataset Discovery* queries that involve content-based measurements (and not only measurements based on metadata) among two or more datasets, such as "find the K most relevant datasets to a particular dataset", i.e., finding related datasets containing the same entities to a given one.

Concerning task D, one should be able to find datasets that can enrich the information of his/her dataset. Indeed, it is important to combine information from multiple datasets for several tasks, e.g., for producing more features or/and entity embeddings, which can be used for improving the predictions of Machine-Learning based tasks. Regarding task E, the cross-dataset identity reasoning allows spotting the contradictions that exist and on the same time provides information for data cleaning or for estimating and suggesting which data are probably correct or more accurate. Moreover, the content-based measurements can aid publishers to estimate and improve the quality of their datasets. More concrete motivating examples of tasks A-E are given in Chapter 3.

Below, we introduce the main challenges and the corresponding research questions (**RQ**) that this thesis aims at tackling, for being able to provide advanced services for the tasks A-E.

• **Challenge 1. Cross-Dataset Identity Reasoning.** The equivalence relationships that exist among entities and schemas of different datasets, such as `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass`, model an equivalence relation. Therefore, for finding all the equivalent URIs for a given URI, it is a prerequisite to compute the transitive and symmetric closure of these relationships. However, it presupposes knowledge of all datasets and most of the algorithms (such as Tarjan's connected components (*CC*) algorithm [233]) that compute the transitive and symmetric closure of equivalence relationships require a lot of memory, i.e., one should keep in memory all the binary relationships during the computation of closure. Therefore, the major research question (*RQ1*) is how to compute in an efficient way the cross-dataset identity reasoning, i.e., the computation of transitive and symmetric closure of `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass` relationships, by using either a single machine or a cluster of machines, since the number of equivalence relationships at scale is large. The corresponding research question are addressed in Chapter 4.

• **Challenge 2. Construction of Indexes at Large Scale by applying Cross-Dataset Identity Reasoning.** We desire to index the whole contents of multiple datasets, for enabling fast access to all the triples of a given URI *u* (including the triples that contain any of the equivalent URIs of *u*). Therefore, one major question (*RQ2*), is how to apply the result of the cross-dataset identity reasoning for constructing such semantics-aware indexes. Moreover, since the there are many datasets (hundreds or even thousands) and some of them are very big, one key question (*RQ3*) is how to parallelize in an efficient way the construction of these indexes. The corresponding research questions are addressed in Chapter 4.

• **Challenge 3. Dataset Discovery by using content-based measurements among two or more datasets.** For offering dataset discovery through content-based measurements, i.e., "find the K most relevant datasets to a given one", it is a prerequisite to take into account the whole contents of datasets (and not only metadata) and to solve maximization problems. The main problem is that the possible combinations of datasets is exponential in number, specifically the number of possible solutions for a given K is given by the binomial coefficient formula. Therefore, a major question (*RQ4*) is whether a standard W3C query language implementation (such as SPARQL) can be used for solving such maximization problems. Moreover, since set operations (i.e., intersection, union, complement) between large datasets are quite expensive, a key question (*RQ5*) is how we can reduce the number of set operations between different datasets. Finally, a main question (*RQ6*) is whether these content-based measurements can be parallelized. The corresponding research questions are addressed in Chapter 5.

1.4 Contributions of this Dissertation

The key contributions of this dissertation are the following:

- We introduce a comprehensive and clear landscape of large scale semantic integration approaches for better understanding the problem, for identifying the gaps as well as for identifying research directions. The results have been published in [173].

- We propose methods for performing cross-dataset identity reasoning at large scale, i.e., we focus on computing the transitive and symmetric closure of `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass` relationships. These methods rely on special indexes and algorithms and can be executed by using either a single machine or a cluster of machines (indicatively, by using 96 machines less than 9

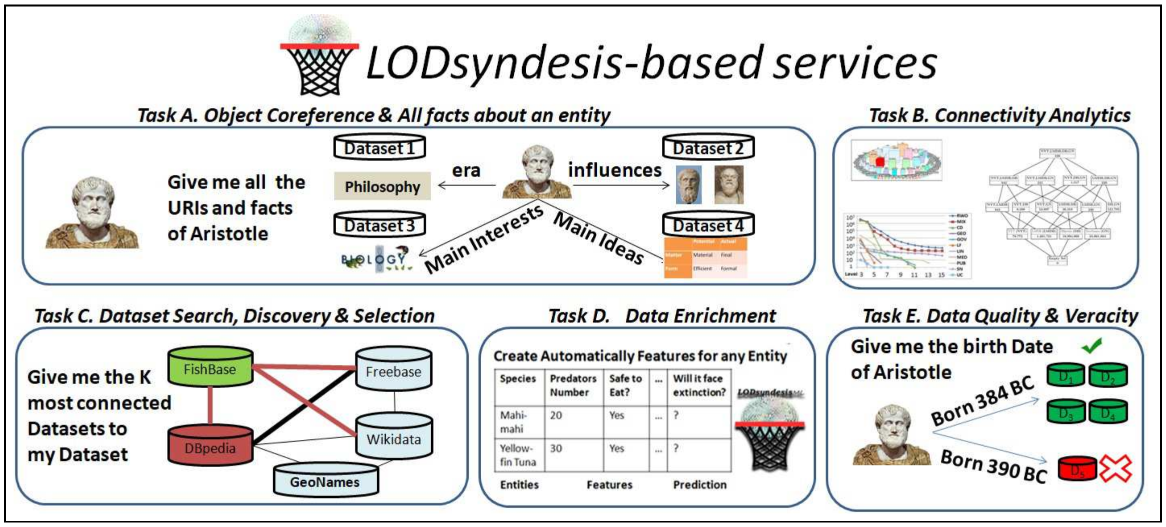

LODsyndesis-based services
Task A. Object Coreference & All facts about an entity
Give me all the URIs and facts of Aristotle
Dataset 1
Dataset 2
Dataset 3
Dataset 4
Philosophy
era
influences
Main Interests
Main Ideas
BIOLOGY
Task B. Connectivity Analytics
Task C. Dataset Search, Discovery & Selection
Give me the K most connected Datasets to my Dataset
FishBase
Freebase
DBpedia
Wikidata
GeoNames
Task D. Data Enrichment
Create Automatically Features for any Entity
Species
Predators Number
Safe to Eat?
Will it face extinction?
Mahi-mahi
Yellow-fin Tuna
Entities
Features
Prediction
Task E. Data Quality & Veracity
Give me the birth Date of Aristotle
Born 384 BC
Born 390 BC

Figure 1.1: Desired Services for Tasks A-E

minutes are needed for computing the transitive and symmetric closure of 44 million equivalence relationships). The results have been published in [165, 168, 171].

- We introduce scalable methods and algorithms that rely on MapReduce techniques, for constructing dedicated global semantics-aware indexes that cover the whole contents of datasets (just indicatively, 81.5 minutes are needed for constructing all the indexes for two billion triples by using 96 machines). The results have been published in [165, 168, 171].

- We introduce content-based intersection, union and complement metrics for dataset search and discovery over large number of linked datasets, which are formulated and tackled as maximization problems. Moreover, we exploit dedicated indexes, lattice-based incremental algorithms and set-theory properties for tackling the exponential complexity of the problems. Indicatively, the proposed algorithms are even more than 5000× faster than a straightforward method and can compute the metrics for 1 million subsets of datasets even in 1 second. The results have been published in [165, 168, 171, 174].

- We exploit the aforementioned indexes for various modern and forthcoming applications, such as for Dataset enrichment for Machine Learning tasks, entity embeddings over large number of datasets and Question answering over several datasets. Indicatively, by creating features and embeddings from multiple datasets simultaneously, we managed to improve the accuracy of predictions (we identified even 13% increase) for machine learning classification problems. The results have been published in [73, 166, 169, 172].

- We report statistics and connectivity analytics for a big subset of the current LOD cloud that comprises 400 datasets and 2 billion triples, which reveal the sparsity of LOD Cloud datasets. The results have been published in [165, 168, 169, 171, 174]
- We present our research prototype, called `LODsyndesis` (`http://www.ics.forth.gr/isl/LODsyndesis/`), that exploits the semantics-aware indexes and the content-based metrics for assisting the Data Integration process by offering services for the tasks A-E (which can be seen in Figure 1.1). Despite the large volume of data, we offer fast responses for millions of queries for 400 datasets, 412 million entities and 2 billion triples (e.g., 3.5 seconds are needed for retrieving all the triples of Aristotle). The results have been published in [169].

1.5 Publications

The work presented in this thesis has been published in international journals and conferences. These publications are listed below, ordered by date:

- M. Mountantonakis and Y. Tzitzikas, *On Measuring the Lattice of Commonalities Among Several Linked Datasets*, Proceedings of the VLDB Endowment (PVLDB), 2016
- M. Mountantonakis and Y. Tzitzikas *How Linked Data can aid Machine Learning-based Tasks*, 21st International Conference on Theory and Practice of Digital Libraries (TPDL), (pp. 155-168), September 2017, Thessaloniki, Greece
- M. Mountantonakis and Y. Tzitzikas *Scalable Methods for Measuring the Connectivity and Quality of Large Numbers of Linked Datasets*, ACM Journal of Data and Information Quality (JDIQ), 9(3), 15, 2018
- M. Mountantonakis and Y. Tzitzikas *High Performance Methods for Linked Open Data Connectivity Analytics*, Information 2018, 9, 134.(Special Issue Semantics for Big Data Integration)
- M. Mountantonakis and Y. Tzitzikas *LODsyndesis: Global Scale Knowledge Services*, Heritage. Open Access Journal (ISSN 2571-9408), 1(2), 335-348, MDPI.(Special Issue: On Provenance of Knowledge and Documentation: Selected Papers from CIDOC 2018), 2018.
- M. Mountantonakis and Y. Tzitzikas *Large Scale Semantic Integration of Linked Data: A survey*, ACM Computing Surveys, 52(5), Sept. 2019
- M. Mountantonakis and Y. Tzitzikas *Knowledge Graph Embeddings over Hundreds of Linked Datasets*, 13th International Conference on Metadata and Semantics Research, Rome, Italy, October 2019

- M. Mountantonakis and Y. Tzitzikas *Content-based Union and Complement Metrics for Dataset Search over RDF Knowledge Graphs*, ACM Journal of Data and Information Quality (JDIQ), 2020

1.6 Outline of Dissertation

The rest of this dissertation is organized in the following way:

- Chapter 2 introduces the background and it describes the context. Furthermore, it reviews the related literature, by presenting a survey of the work that has been done in the area of Linked Data integration in the last decade giving emphasis in the integration of large in numbers datasets.

- Chapter 3 discusses the research gaps derived by the analysis of Chapter 2, the main motivating scenarios and the queries and services that we desire to offer for the tasks A-E. Moreover, it describes the main differences between the work done in the context of the dissertation and the related approaches for each task.

- Chapter 4 presents algorithms for performing cross-dataset reasoning and for constructing semantics-aware indexes that cover the whole content (e.g., entities, schema, triples) of datasets. Finally, it reports comparative efficiency results by using different techniques and a cluster of 64 machines.

- Chapter 5 defines content-based intersection, union and complement metrics among any subset of datasets. It shows why current implementations of SPARQL language are not efficient for making such measurements, and it presents lattice-based incremental algorithms that rely on the semantics-aware indexes and exploit set-theory properties for computing these content-based metrics. Moreover, it introduces methods for performing such measurements in parallel. Finally, it contains comparative efficiency results and it presents indicative statistics and connectivity analytics (derived from the proposed indexes and measurements) for 400 real RDF datasets and 2 billion triples.

- Chapter 6 describes the suite of services and tools that have been developed in the context of this dissertation, which are referred by the name `LODsyndesis`. Emphasis is given in two tools, called $\texttt{LODsyndesis}_{\mathcal{ML}}$ and `LODVec`, that can be used for creating features and entity embeddings for aiding the execution of machine-learning based tasks.

- Chapter 7 concludes the thesis by providing a summary of the results and a discussion on possible future directions.

- Appendix A lists the publications resulted from this dissertation, and provides links to the prototypes and services that have been developed.
- Appendix B reports the acronyms used in this dissertation.

Chapter 2
Large Scale Semantic Integration of Linked Data: A survey

Data integration has been studied in the context of various data models, namely in the relational model, e.g., see [112] for a survey of information integration in the relational model, and in the semi-structured model, e.g., see [40] for a survey of approaches for integration for XML databases, while [84] surveys works for big data integration for both models. In this chapter, we survey the work that has been done in the area of Linked Data integration. There are surveys for various individual tasks of the general integration process, i.e. surveys for *distributed RDF processing* [121], for *ontology matching* [227], for *instance matching* [178], for *integration for OLAP* [27], for *query federation* [213], for *visualization and exploration* [39,66,239], for *quality* [192,261]. In the current survey, we aim at covering the topic holistically, i.e. from various perspectives, for better understanding the overall problem and process, and for making more obvious the dependence between the individual tasks. Moreover, since the LOD cloud already contains a large number of datasets (over 9,000 datasets according to [91]), we give emphasis on methods that can be applied to very large number of datasets. This distinction is important in the sense that a semi-automatic process which can be followed for integrating a few (say 5) datasets, is not affordable, due to the required human effort, for integrating thousands of datasets. Michael Stonebraker (a pioneer researcher in data management) has mentioned (`http://ilp.mit.edu/images/conferences/2013/ict/presentation/stonebraker.pdf`) that data integration at scale is a very big deal and probably the biggest problem that many enterprises face, since the traditional approaches cannot scale easily to more than 25 sources. For this reason, in this chapter we emphasize on tasks that could aid the integration of large number of datasets, and discuss the tools that are available for large in number RDF datasets. Overall this chapter provides a concise overview of the issues, methods, tools and systems for semantic integration of data.

In various places of this chapter, we shall use a running example from the marine domain. Moreover, please note that in several places we refer to `LODsyndesis`, which is the suite of services and tools that have been developed in this dissertation. The rest of this chapter is organized as follows: §2.1 provides the background and the context, discusses the Linked Data ecosystem and refers to past surveys. §2.2 introduces the difficulties of data integration while §2.3 describes the landscape of data integration. §2.4 surveys the integration methods while §2.5 discusses the different integration processes. §2.6 focuses on evaluation related to information integration, whereas §2.7 lists tools and services for large in number RDF datasets and what steps of the processes they cover and how. §2.8 provides a synopsis of this chapter.

2.1 Background and Context

In this section, in §2.1.1 we introduce the main principles of Linked Data, and in §2.1.3 we discuss the Linked Data Ecosystem.

2.1.1 Linked Data

Definition. "Linked Data 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" [42].

The roots of Linked Data. In the 1990s, Tim Berners-Lee, the inventor of the World Wide Web, discussed the vision for a Semantic Web [36], i.e., *"The first step is putting data on the Web in a form that machines can naturally understand, or converting it to that form. This creates what I call a Semantic Web - a web of data that can be processed directly or indirectly by machines"*. In 2001, Tim Berners-Lee and his colleagues described the main ideas of Semantic Web [37], e.g., representing data in RDF format, using ontologies that enable the creation of inference rules, and others, while the May 2006 paper [226], stressed the emerging need for semantic data integration and described most of the Linked Data principles, which are dissussed below.

Linked Data Principles. The major principles of Linked Data, which are required for reaching the goal for a "Web of Data" (or Semantic Web) [42], were officially proposed in July 2006 by Tim Berners-Lee[1]:

"1) use URIs as names for things, 2) use HTTP URIs so that people can look up those names, 3) when someone looks up a URI, provide useful information, using the standards (RDF, SPARQL) and 4) include links to other URIs, so that they can discover more things." The fourth principle, which refers to data interlinking, is of primary importance for data integration, since it suggests to the publishers to create relationships with URIs occuring in different datasets. The datasets can be linked through common URIs, which can refer either to schema elements (they are defined through RDF Schema and OWL [32]), or data el-

[1]https://www.w3.org/DesignIssues/LinkedData.html

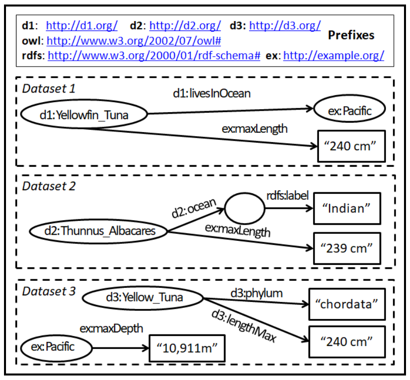


Figure 2.1:
Example of Linked Data with 9 triples

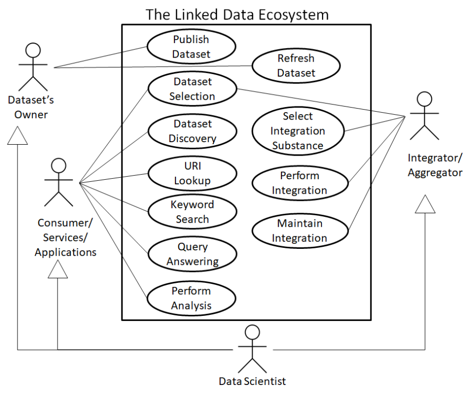


Figure 2.2:
Use case diagram for the Linked Data ecosystem

ements. SPARQL is a standard query language (`http://www.w3.org/TR/sparql11-query/`) for retrieving and managing RDF data, whereas queries can be expressed across different datasets. Moreover, the emergence for achieving linking and integration can be observed from proposals for rating *open data*, i.e., by using the 5-star Open Data (`http://5stardata.info/en/`), as well as proposals for rating *vocabularies* [125].

2.1.2 RDF

Resource Description Framework (RDF) [32] is a graph-based data model. It uses Internationalized Resource Identifiers (IRIs), or anonymous resources (blank nodes) for denoting resources, and constants (Literals), while triples are used for relating a resource with other resources or constants. Hereafter, we shall use the term URIs (Uniform Resource Identifiers) to refer to IRIs (since the term URI is more commonly used). A triple is a statement of the form subject-predicate-object ⟨s,p,o⟩, and it is any element of $\mathcal{T} = (\mathcal{U} \cup B_n) \times (\mathcal{U}) \times (\mathcal{U} \cup B_n \cup \mathcal{L})$, where $\mathcal{U}$, B_n and $\mathcal{L}$ denote the sets of URIs, blank nodes and literals, respectively, whereas an RDF graph (or dataset) is any finite subset of $\mathcal{T}$. For instance, the triple ⟨d1:Yellowfin_Tuna, d1:livesInOcean, ex:Pacific⟩, contains three URIs, where the first one (i.e., d1:Yellowfin_Tuna) is the subject, the second one (i.e., d1:livesInOcean) is the predicate (or property) and the last one (i.e., ex:Pacific) is the object. By using Linked Data, the linking of datasets can be achieved by the existence of common URIs or Literals, or by defining equivalence relationships, e.g., `owl:sameAs`, among different URIs (entities, properties and classes).

2.1.3 The Linked Data Ecosystem

To understand the world of Linked Data and the involved stakeholders, we could consider them as a single *ecosystem*. Figure 2.2 provides the Use Case Diagram and below we identify the main actors and use cases of that ecosystem.

Dataset's Owner. The owner of a dataset can be a public organization (e.g., a municipality, a research centre, a university), a private sector organization (e.g., a company, like BBC, NGOs, etc.) or even an individual that owns and is responsible for creating, keeping, maintaining and publishing the dataset.

Consumer, Services or Applications. This actor corresponds to entities, services or applications that consume data for various reasons, i.e., for providing services of various levels or granularity like *Dataset Discovery*, *Dataset Selection*, *URI Lookup*, *Keyword Search* and *Query Answering* services. For instance, *LODLaundromat* [207] offers URI Lookup and Keyword Search for over 650,000 documents. Other applications, such as *Swoogle* [74] and *Watson* [67], offer keyword search services for the Semantic Web datasets and resources, while *LODsyndesis* [165] offers services for *Dataset Discovery* and *Dataset Selection*. Finally, this actor includes end-user applications (like smart phone applications) that exploit Linked Data for supporting their functionality.

Integrator/Aggregator. This actor captures individuals or organizations whose objective is to integrate a number of datasets and provide integrated access services. The final output can be exploited by the members of the corresponding community. One example is Europeana Foundation which is responsible for *Europeana* [124], which is the European digital platform for cultural heritage. This platform combines data from more than 3,000 institutions across Europe while these data were transformed into Linked Data and are represented in the Europeana Data Model [124]. In other occasions, international projects play this role. Note that the use case "Perform Integration" is actually the process that will be analyzed in this survey.

Data Scientist. A Data Scientist can be considered as a special case of Consumer and Aggregator/Integrator. A data scientist usually has to find and select the appropriate datasets (or sub-datasets) for his/her needs and may have to aggregate and integrate data in order to perform the intended analysis. Moreover at the end, the data scientist can publish the results of this analysis as a new dataset (therefore it could be considered as dataset owner).

2.1.4 Related Surveys

Several surveys have been published in the database world about *Data Integration*, i.e., surveys assuming the relational data model (such as [112]), surveys assuming the semi-structured data model (e.g., [40] includes approaches for integrating XML databases), surveys that concern big data integration for both models (e.g., [84]), as well as surveys for semantic integration focusing on "ontology-based" approaches [134, 183]. Concerning Linked Data, there are surveys for various individual tasks of the general integration process. Specifically, [121] surveys techniques and approaches for scalable *distributed RDF processing, querying and reasoning*, e.g., search engines, federated query systems, rule-based reasoning and so forth. [227] provides a literature review for the field of *ontology matching* for the decade 2003-2013 whereas the authors in [178] compare the features of various tools and frameworks that perform *instance matching*. In [27], the objective was to survey how the "Semantic Web technologies can aid in data discovery, acquisition, integration, and analytical querying of external data, and thus serve as a foundation for exploratory on-line analytical processing (OLAP)". The authors analyzed the steps that should be carried out for creating data warehouses and answering *OLAP queries*. Moreover, [213] compares novel *SPARQL endpoint federation engines* in many dimensions and details the tools that have been created for this specific method of integration, while [131] surveys approaches that support scalable *distributed SPARQL query evaluation*. [202] highlights the strong need for holistic data integration approaches that can integrate many data sources (and not be limited only to pairwise matching). [66] surveys approaches for *visualizing* Linked Data, [39] surveys a number of systems for *data visualization* and exploration, while [239] surveys methods for supporting *faceted exploration* over RDF datasets. Concerning the *quality* of Linked Data, the authors in [261] survey 21 approaches and describe 26 data quality dimensions, e.g., *accuracy*, *interlinking*, *conciseness*, *consistency* and others, while they introduce corresponding metrics and approaches for each dimension. Moreover, [192] surveys approaches focusing on *knowledge graph refinement* (mainly on error detection and data completion), which are of primary importance for improving the quality of a single dataset (that could be an integrated dataset). Finally, *OCLC Research Team* has provided a survey for hundreds of projects and services from the domain of *digital libraries* that exploit Linked Data principles [229]. The key difference between our survey and the aforementioned ones is that they focus on various individual tasks of the general integration process whereas we emphasize on the whole integration process and methods that can be applied for large number of datasets (e.g., thousands of datasets).

2.2 Why Data Integration is Difficult

Information integration aims at offering unified access services over a set of information from heterogeneous datasets (structured, semi-structured or unstructured), that can have different conceptual, contextual and typographical representations. Integration is not easy for various reasons. Most of these reasons has attracted the interest of database community for decades, e.g., for relational model [230, 231] and in the area of *Semantic Integration* [134, 183]. Below we list six main reasons, each exemplified using the running example of Figure 2.3, which shows the three sources of Figure 2.1, i.e., D_1, D_2 and D_3, in

the context of four integration scenarios.
a) **Different Authorities**. The datasets are produced, kept or managed by different organizations in different formats, schema, models [134, 183], locations, systems and licenses. There is not any "centralized control system", therefore, each publisher (e.g., organization) decides how to produce, manage and publish a dataset based on their needs and choices. For instance, in the marine domain the dataset of *Fishbase* (`http://www.fishbase.org`), that contains information about the "taxonomy, geographical distribution, biometrics, population, genetic data and many more" [163], is accessible through an SQL server, therefore one needs a fetcher to download it and a transformer to RDF, for linking it with other datasets. On the contrary, for *Ecoscope* dataset (`http://ecoscopebc.mpl.ird.fr/joseki/ecoscope`), which contains "geographical data, pictures and information about marine ecosystems" [163], transformations are not required since this dataset has already been published in RDF format.

b) **Naming.** The same real-world entities or relationships are referred with different URIs and names and in different natural languages, while natural languages have synonyms and homonyms that make harder that automatic connection. For instance, the URI of the species *Thunnus Albacares* in DBpedia is `http://www.dbpedia.com/Thunnus_Albacares`, while in Ecoscope the corresponding URI is `http://www.ecoscope.com/thunnus_albacares`. Moreover, the aforementioned species has 348 common names in 82 different natural languages [236]. In addition, we often have to tackle the problem of homonyms, since the same name can describe two or more different real-world entities. For example, "Argentina" is used to refer to the country (`http://dbpedia.org/resource/Argentina`) but also to a fish genus (`http://dbpedia.org/resource/Argentina_(fish)`).

c) **Complementarity**. The datasets usually contain complementary information, e.g., consider two datasets about the same domain each modeling a different aspect of the domain. The commonalities between these datasets can be very few and this does not aid automated linking and integration. For example, a dataset can include data about the predators of marine species, and another one can contain data about the selling price of a species in a fish market. In the former, the species can be referred by its scientific name, while in the latter by its commercial code.

d) **Errors, Conflicts.** The datasets can contain data that are erroneous, out-of-date or conflicting. For example, in the marine domain *Fishbase* mentions that the max length of *thunnus albacares* is 239 cm while Wikipedia (whose content is used from many RDF sources such as DBpedia) states that its max length is 240 cm, meaning that conflicts can occur even because one dataset is more precise than another. Other conflicts are due to erroneous information, e.g., suppose that one dataset states that the capital of Australia is

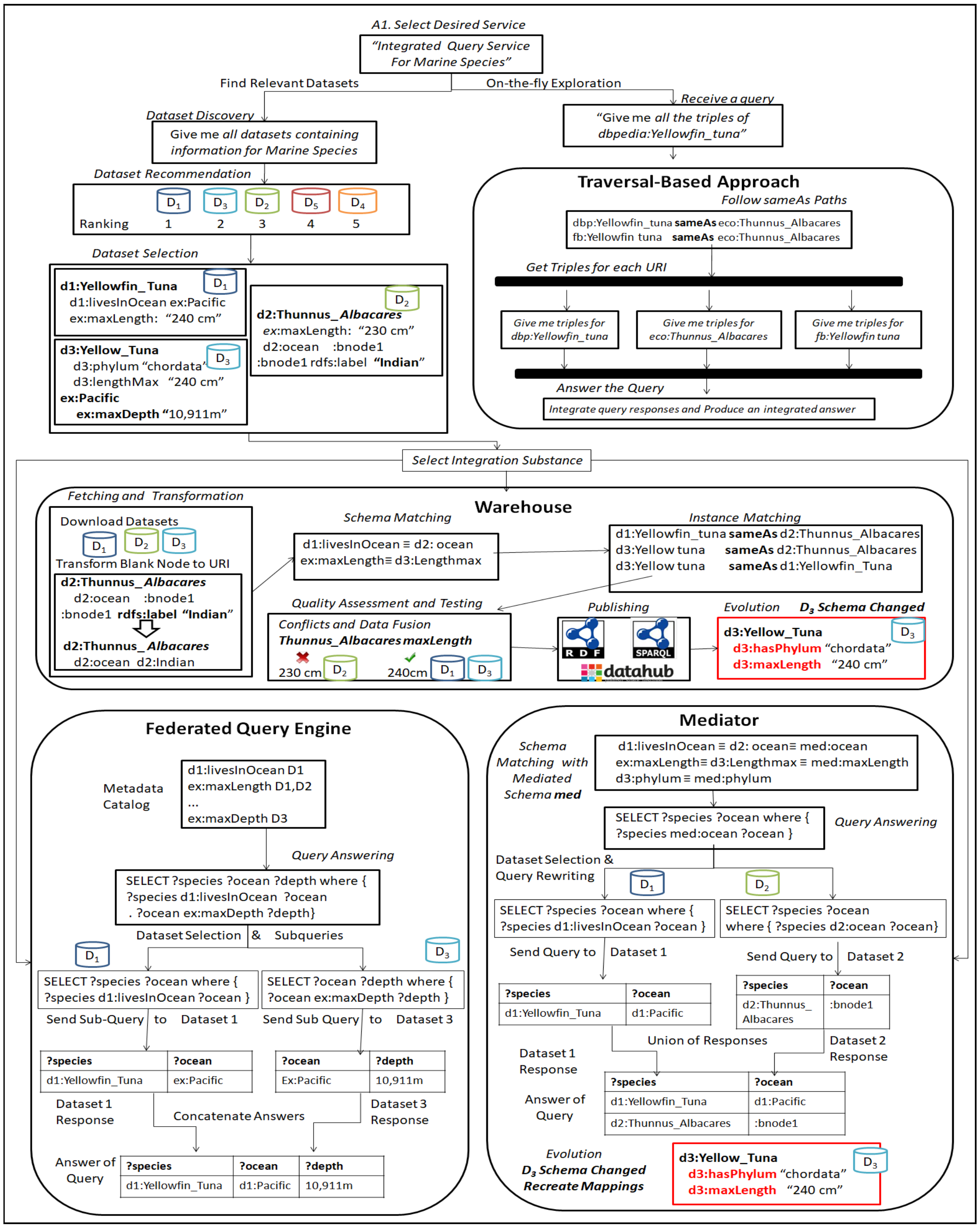

A1. Select Desired Service
"Integrated Query Service For Marine Species"
Find Relevant Datasets
On-the-fly Exploration
Dataset Discovery
Give me all datasets containing information for Marine Species
Receive a query
"Give me all the triples of dbpedia:Yellowfin_tuna"
Dataset Recommendation
Ranking
D1 D3 D2 D5 D4
1 2 3 4 5
Traversal-Based Approach
Follow sameAs Paths
dbp:Yellowfin_tuna sameAs eco:Thunnus_Albacares
fb:Yellowfin tuna sameAs eco:Thunnus_Albacares
Get Triples for each URI
Give me triples for dbp:Yellowfin_tuna
Give me triples for eco:Thunnus_Albacares
Give me triples for fb:Yellowfin tuna
Answer the Query
Integrate query responses and Produce an integrated answer
Dataset Selection
d1:Yellowfin_Tuna
d1:livesInOcean ex:Pacific
ex:maxLength: "240 cm"
d2:Thunnus_Albacares
ex:maxLength: "230 cm"
d2:ocean :bnode1
:bnode1 rdfs:label "Indian"
d3:Yellow_Tuna
d3:phylum "chordata"
d3:lengthMax "240 cm"
ex:Pacific
ex:maxDepth "10,911m"
Select Integration Substance
Fetching and Transformation
Warehouse
Download Datasets
Transform Blank Node to URI
d2:Thunnus_Albacares
d2:ocean :bnode1
:bnode1 rdfs:label "Indian"
d2:Thunnus_Albacares
d2:ocean d2:Indian
Schema Matching
d1:livesInOcean ≡ d2: ocean
ex:maxLength≡ d3:Lengthmax
Instance Matching
d1:Yellowfin_tuna sameAs d2:Thunnus_Albacares
d3:Yellow tuna sameAs d2:Thunnus_Albacares
d3:Yellow tuna sameAs d1:Yellowfin_Tuna
Quality Assessment and Testing
Conflicts and Data Fusion
Thunnus_Albacares maxLength
230 cm D2
240cm D1 D3
Publishing
RDF
SPARQL
datahub
Evolution
D3 Schema Changed
d3:Yellow_Tuna
d3:hasPhylum "chordata"
d3:maxLength "240 cm"
Federated Query Engine
Metadata Catalog
d1:livesInOcean D1
ex:maxLength D1,D2
...
ex:maxDepth D3
Query Answering
SELECT ?species ?ocean ?depth where { ?species d1:livesInOcean ?ocean . ?ocean ex:maxDepth ?depth}
Dataset Selection & Subqueries
SELECT ?species ?ocean where { ?species d1:livesInOcean ?ocean }
SELECT ?ocean ?depth where { ?ocean ex:maxDepth ?depth }
Send Sub-Query to Dataset 1
Send Sub Query to Dataset 3
?species ?ocean
d1:Yellowfin_Tuna ex:Pacific
?ocean ?depth
Ex:Pacific 10,911m
Dataset 1 Response
Concatenate Answers
Dataset 3 Response
Answer of Query
?species ?ocean ?depth
d1:Yellowfin_Tuna d1:Pacific 10,911m
Mediator
Schema Matching with Mediated Schema med
d1:livesInOcean ≡ d2: ocean≡ med:ocean
ex:maxLength≡ d3:Lengthmax ≡ med:maxLength
d3:phylum ≡ med:phylum
SELECT ?species ?ocean where { ?species med:ocean ?ocean }
Query Answering
Dataset Selection & Query Rewriting
SELECT ?species ?ocean where { ?species d1:livesInOcean ?ocean }
SELECT ?species ?ocean where { ?species d2:ocean ?ocean}
Send Query to Dataset 1
Send Query to Dataset 2
?species ?ocean
d1:Yellowfin_Tuna d1:Pacific
?species ?ocean
d2:Thunnus_Albacares :bnode1
Dataset 1 Response
Union of Responses
Dataset 2 Response
Answer of Query
?species ?ocean
d1:Yellowfin_Tuna d1:Pacific
d2:Thunnus_Albacares :bnode1
Evolution
D3 Schema Changed
Recreate Mappings
d3:Yellow_Tuna
d3:hasPhylum "chordata"
d3:maxLength "240 cm"

Figure 2.3: Running example containing the steps of four different integration substances

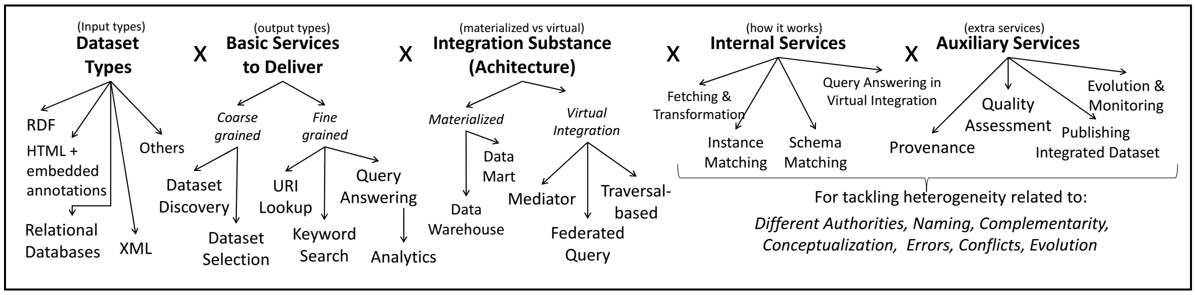


Figure 2.4: The dimensions of data integration landscape

Sydney and several other datasets that is Canberra. Finally, out-of-date data are very common in many domains, e.g., the current team of a football player (this is however related also to difficulty (f) described below).

e) **Different Conceptualizations.** The datasets about the same domain may follow different conceptualizations (or modeling decisions) of the domain [134, 183], i.e., they have different schemas (as we would say in the relational database world). For instance, some datasets conceptualize an address by using one property (e.g., ⟨`:Michael`,`:hasAddress`, `"Street1,Heraklion,71303"`⟩, others use one property per address part (e.g., ⟨`:Michael`,`:street`,`"Street1"`⟩,⟨`:Michael`,`:city`,`"Heraklion"`⟩and⟨`:Michael`,`:postCode`,`"71303"`⟩). Moreover, other datasets use a blank node for representing an address. Consequently, there is not a general pattern that the creators of the datasets follow for representing the information for a specific domain.

f) **Evolution.** Everything changes: the world, the ontologies (e.g., see [101] for a survey of ontology change), the data. Thereby, integration action which have taken place may have to be updated or revised. This is also a source of possible conflicts as stated earlier.

2.3 The Data Integration Landscape

The data integration landscape is wide and complex. For approaching it in a structured manner, we can describe an integration process through a multidimensional space. Such a space should allow describing each integration method as one or more points of the multidimensional space in a clear manner. Specifically, we introduce the space defined by the cartesian product of *five dimensions*

$$\begin{aligned} IntegrationLandScape &= (DatasetTypes \times BasicServicesToDeliver \times IntegrationSubstance \times \\ &\quad InternalServices \times AuxiliaryServices) \end{aligned}$$

Figure 2.4 illustrates the basic values that correspond to each dimension. Below we briefly describe each one, while a detailed description for each one is given in the next sections.
• *DatasetTypes* (or input types) refers to the different dataset types that can be used as input, e.g., RDF files, relational databases, HTML with embedded annotations (e.g., RDFa, JSON-LD). The *dataset's owner* (recall the Linked Data Ecosystem in §2.1.3) is responsible for the type of the dataset.
• *BasicServicesToDeliver* (or output types) refers to the main purpose of integration, i.e., what services we would like to offer by integrating several datasets, e.g., see those shown in Figure 2.4. This dimension corresponds to *Consumer, Services or Applications* actor of §2.1.3.
• *IntegrationSubstance* (or *Integration Architecture*) refers to the different integration substances, physical (materialized) versus virtual, and their specializations (e.g. see those shown in Figure 2.4). The responsible actor (according to §2.1.3) is the *integrator/aggregator* .
• *InternalServices* (i.e., how it works) refers to the services used during the integration process, e.g., transformations, bindings, etc., in order to "connect" the pieces of data, thereby, the responsible actor is the *integrator/aggegator* (see §2.1.3).
• *AuxiliaryServices* (i.e., extra output types, beyond the core ones) refers to services that can be optionally exploited/offered either before or after the integration process, related to provenance, evolution, quality and others (again *integrator/aggregator* is the responsible actor).

For tackling the various discrepancies (as mentioned in §2.2), which is actually the "duty" of the *InternalServices* mentioned before, the various integration approaches essentially attempt to "connect" the data of the underlying datasets. Generally, datasets can be connected (or linked) through a) instance links, b) schema concepts and c) constants, i.e., literals. According to [168], LOD cloud datasets are mainly linked through schema concepts (99% of datasets' pairs share RDF schema elements) and literals (78% of datasets' pairs share literals), while only 11.3 % of datasets' pairs contain common instances. One kind of connection is through *canonicalization*, e.g., an integration system can decide to transform every occurrence of "UK" and "Great Britain" to "United Kingdom". In this way, semantically equivalent elements can get the same single representation. Another kind of connection is through *binding* , i.e., by adding extra relationships (data in general) that connect these elements, e.g., "UK" ≡ "Great Britain". We use the term "binding" to refer to what is called *correspondence, mapping, link*, etc. We can distinguish the following cases based on the **semantics of these bindings** (in some parts of the discussion below we shall use examples from the two small datasets shown in Figure 2.5):

- *Taxonomy-based relations*
  - *Exact-ones.* Here we have relations expressing equivalence, e.g., `owl:sameAs`, `owl:EquivalentProperty`, skos:exactMatch, or difference e.g., `owl:DifferentFrom`.

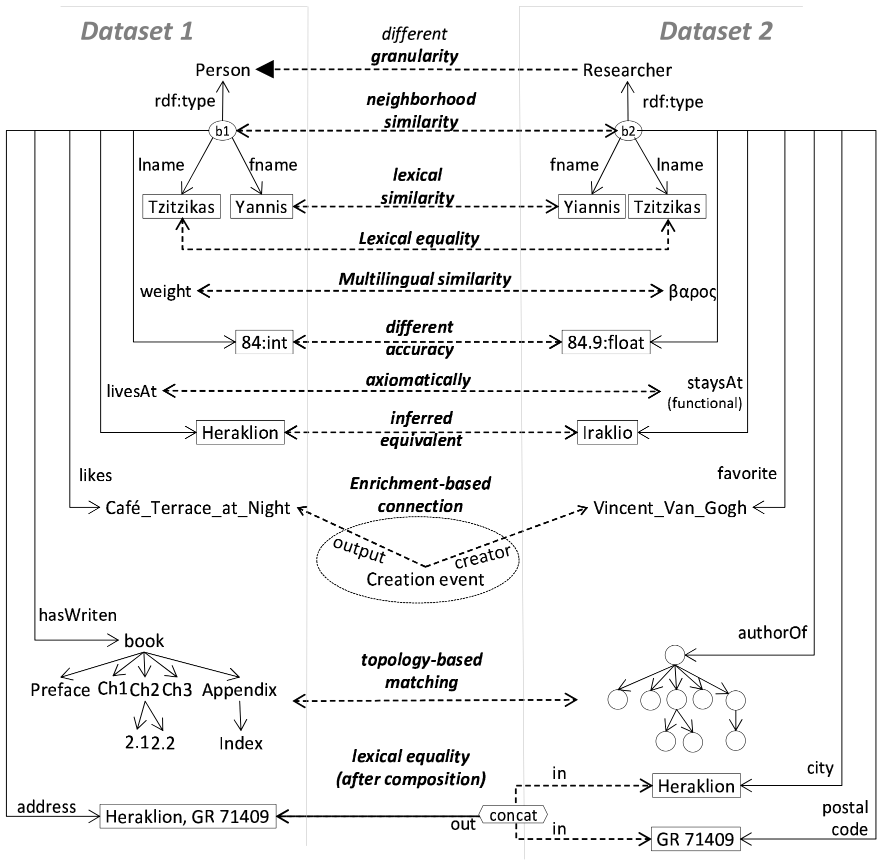


Figure 2.5: Example of two datasets and possible bindings

- *Inexact-ones.* Here we have connections between elements of different granularities, e.g., ⟨`Researcher,subclassOf,Person`⟩ and ⟨`livesAt,subpropertyOf, staysAt`⟩, connections between different accuracy levels, e.g., `84` vs `84.9`, or connections between similar concepts by using relations such as `skos:closeMatch`.

- *Enrichment-based.* Here the binding of two elements is achieved through non-taxonomic properties, or paths of properties, e.g., see how `Cafe_Terrace_at_Night` of Dataset1 can be connected with `Vincent_Van_Gogh` of Dataset2.

Now there are several **types or sources of evidence** for creating bindings of the aforementioned kinds:

- *Axiomatically* by users (designers, curators), e.g., the triple ⟨`leavesAt, subpropertyOf, staysAt`⟩ can be provided by a designer.
- *Name similarity* (string similarity, phonetic similarity, stemming), e.g., `Yannis` ∼ `Yiannis` can be detected through EditDistance.
- *Neighborhood (direct or indirect) similarity*, e.g., `b1` ∼ `b2` can be detected by blank node matching techniques.

- *Natural Language Processing-based*, e.g., entity or relation mining over the literals associated through `rdfs:label` with a URI can be used for connecting that URI with the URI of the identified (in the literal) entity(-ies). This type of evidence can be based on lexicons, thesaurus, translators, e.g., multilingual similarities, such as `weight` ∼ *βαρος* (i.e., weight in greek language) in Figure 2.5, can be detected by using a translator, and word embeddings (e.g., word2vec [153]), where words having similar meaning are represented in a similar way.
- *Common Instances*, for example two classes can be connected with a taxonomic relation based on the fact that they have common instances (through the ostensive method, or through inductive and machine learning-based methods).
- *Inference-based* (or *Reasoning-based*), e.g., the equivalence `Heraklion` ≡ `Iraklio` is inferred because the `staysAt` is a functional property and `livesAt` is a subproperty of `staysAt`.
- *Topology-based similarity*, e.g., all nodes of the book in the left side can be matched with the blank nodes subgraph in the right side (because these two graphs are isomorphic).
- *Usage-based*, e.g., frequently accessed together entities can be connected because of this. For example, `Yannis` could be connected with *Γιαννης* (which is the same name written in Greek characters) if these two words frequently co-occur in the log file of a search system.

We observe that in order to integrate properly these two very small and simple datasets, that contain information about the same (very narrow) domain, we have to combine various kinds of evidence and create various types of binding.

2.4 Surveying the Integration Methods

For surveying the various integration methods for Linked Data, apart from studying the literature [2008-2018] (the main works, since it is impossible, for reasons of space, to include the entire literature), we considered also our experience from various EU projects.

This section is structured according to the dimensions (or aspects) of the *Data Integration Landscape* introduced in §2.3. For each dimension, we describe its role, the related challenges, and the related methods, techniques and tools. Specifically, in §2.1.4 we mention related surveys, in §2.4.1 we discuss the different dataset types, in §2.4.2 we describe the basic services that can be delivered through an integration process, in §2.4.3 we focus on the integration substance, while in §2.4.4 we describe the internal services and in §2.4.5 the auxiliary services. Finally, in §2.4.6, we classify 18 integration tools according to the dimensions in §2.3.

2.4.1 Dataset Types

DatasetTypes refers to the different dataset types that can be used in an integration process as input. In this survey, we focus on datasets represented in RDF format. However, we should note that there are several ways and tools for mapping relational databases to RDF format [224] and for converting CSV (comma separated values) files to RDF [90]. Moreover, a huge volume of RDF data can be extracted through HTML web pages [184, 197]. Specifically, billions of web pages contain semantic annotations by using *Microformats*, *RDFa* and *HTML Microdata* mainly inside their *¡body¿* element, and *JSON-LD* scripts predominantly inside their *¡head¿* section.

2.4.2 Basic Services To Deliver

One aspect of primary importance is what is the *purpose* of integration, i.e. why we want to integrate data, what we want to achieve and to eventually deliver in terms of input and desired output. Some forms of the desired integrated access can be simple, while other forms can be more complex. To this end, below we dichotomize such services to *Fine Grained* and *Coarse Grained* services:

- *Fine Grained (FG) Services.* Here the objective is to find, select and assemble "pieces" of data. To this category of services we can distinguish three different levels: *Level I. Global URI Lookup Service* (analyzed in §2.4.2.1), *Level II. Global Keyword Search* (analyzed in §2.4.2.2) and *Level III. Integrated Query Answering Service* (analyzed in §2.4.2.3).

- *Coarse Grained (CG) Services.* Here the objective is to find or select *entire datasets.* In this category of services we have: *Dataset Discovery & Selection* (analyzed in §2.4.2.4).

2.4.2.1 FG: Level I. Global URI Lookup Service.

This kind of service can be used for finding all the URI containing a substring or all the equivalent (or similar) URIs of a given URI u. For realizing such a service, we have to tackle difficulty (b), i.e. the problem of synonyms and homonyms, which in turn requires the execution of various tasks including:

• *cross-dataset completion* of the `owl:sameAs` relationships for completing (with respect to symmetry and transitivity) the relationships that are already recorded, otherwise the response would be incomplete. The same is true for schema elements, e.g., for `owl:equivalentClass`.

• *matching methods* for individuals or schema elements (instance matching is described in §2.4.4.3 while schema matching is described in §2.4.4.2) for identifying new equivalences between URIs which have not been recorded. However we should note that not all entities (or concepts) have necessarily URIs, in the sense that some of them can be represented as literals, or even not modeled in the datasets themselves (they could be called hidden intermediate concepts or entities [78]).

2.4.2.2 FG: Level II. Global Keyword Search.

This kind of service can be used for finding URIs, triples, literals and datasets relevant to an information need that has been expressed as a keyword query. Such services can be based on Information Retrieval techniques, semantic-based techniques, or both. The responses of such services is a list of elements ranked according to their estimated relevance to the user query. It is not hard to see that having achieved an effective (of good quality) I-service, certainly aids the provision of II-services, since without complete URI lookup the II-services will miss relevant elements and thus will have low recall.

2.4.2.3 FG: Level III. Integrated Query Answering Service.

This kind of service answers complex queries containing data derived from more than one dataset over any integrated system. For instance, such a service should be able to answer the following query: ”Find the ecosystems, waterareas and countries that `http://www.dbpedia.com/Thunnus_Albacares` is native to, and the common names that are used for this species in each country, as well as their commercial codes”. One difference between level III and level II services is that a III-service takes as input a query expressed in a structured query language (SPARQL), while II-services receive as input queries expressed in natural language. Consequently, one major difference between III-services and I/II-services is that a III-service presupposes a common conceptualization that allows formulating the query and it also presupposes the existence of data according to this conceptualization (for evaluating the query). The latter is not always feasible, since we may miss the needed datasets mainly due to the different conceptualizations or the complementarity of datasets (difficulties c and e), i.e., we may not have enough common data to establish connections. Another difference between III-services and I/II-services is that it is much more difficult to achieve a *quality response* to a complex query. For example, suppose that an I-service has recall 80% (e.g., that it only finds 80% of the URIs or triples about a real-world entity). Now a query that contains a condition that involves 2 URIs (e.g., all x such that (u_1, X, u_2) is expected to have recall equal to 64% (=0.80 * 0.80), with k URIs, the recall would be 0.8^k. Clearly, the more complex the query becomes, the harder to achieve good quality.

2.4.2.4 CG: Dataset Discovery & Selection

It refers to the discovery of the most relevant datasets to a given keyword, dataset or URI and to the selection of the most desired datasets which fulfill the requirements that the

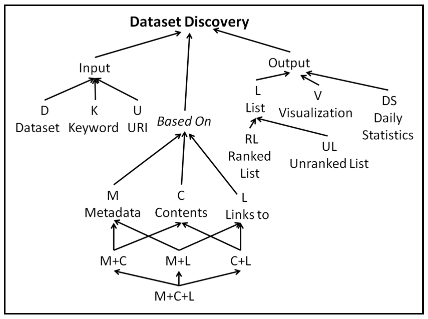


Figure 2.6:
The different criteria of dataset discovery

Approach	Input	Based on	Output
Nikolov et al. [182]	D,K	C	RL
Leme et al. [143]	D	M	RL
Wagner et al. [252]	D,K	M+C	RL, V
Ellefi et al. [87]	D	M	RL
LinkLion [179]	D	M	UL
RapidMiner Link [208]	U	L	UL
LODVader [33]	D	M	RL, V
LODsyndesis [165]	D, U	M+C	UL, V
WIMU [246]	U	M+C	RL
ExpLOD [133]	K	M	RL
Lotus [122]	K	C	UL
Swoogle [74]	K, U	M+C	RL
Sindice [187]	K, U	M+C	RL
SWSE [119]	K,U	M+C	RL
Watson [67]	K,U	M+C	RL
Datahub.io	K	M	UL
Google Dataset Search [184]	K	M	RL
SpEnD [258]	K	M	UL, DS, V
SPARQLES [248]	K	M	UL, DS, V
LODAtlas [198]	K,D	M	RL
LOV [247]	K	M	UL, V

Table 2.1:
Categorizing dataset discovery approaches according to the criteria of Figure 2.6.

selected integrated access system is intended to serve.

Context. If there is prior knowledge for the datasets that will be integrated (e.g., in MarineTLO warehouse [236], the authors already knew which datasets will be used for constructing the warehouse) there is no need to exploit such a *Dataset Discovery* service. On the contrary, one can exploit such a service if there is no knowledge about what datasets exist, or there is prior knowledge for some datasets that will be surely used, however, more datasets are needed. For instance, Figure 2.3 shows an example where the scientist (or user) desires to find datasets whose domain is about *Marine Species* by using a keyword search engine that returns a ranked list of datasets.

Difficulties. The heterogeneity in terms of format, schema, etc., the existence of different URIs referring to the same real-world entity, and datasets evolution, i.e., difficulties (a), (b) and (f), makes it difficult to discover valuable and relevant datasets. Additionally, difficulties (c), i.e., complementarity of information and (e), i.e., different conceptualizations, can complicate that process.

Categorization. Figure 2.6 depicts the different criteria that can be employed for characterizing the *Dataset Discovery* approaches. One criterion is how the information need is expressed: as a *Keyword-query*, as a *URI* or as a *Dataset*. Another criterion is the method which is used for discovering relevant datasets: there exist *Metadata-based* approaches which depend on metadata such as measurements, statistics and dataset descriptions, *Content-based* approaches, which exploit the contents of the datasets for discovering relevant datasets, and *Links-to* approaches which discover relevant URIs and datasets on-the-fly by traversing equivalent links. Regarding the output, the simplest one is an *Unranked*

List, which just returns a list of datasets (or links with their provenance) without ranking, while the output can be a *Ranked List*, where a score is assigned for each dataset, e.g., by exploiting *Dataset Recommendation* techniques. Finally, a *Dataset Visualization* output can be more informative for the human users for helping them to understand and explore the structure and the interconnections of Linked Data and lead to an efficient and intuitive interaction with them, while *Daily Statistics* (as an output) are important for checking the evolution of each dataset.

Approaches. Table 2.1 lists a number of approaches and categorizes them according to the dimensions of Figure 2.6. Below, we describe in brief these approaches.

Dataset-based Approaches. Leme et al. [143] proposed a probabilistic classifier based on Bayesian theory, for identifying the most relevant datasets that can be interlinked with a given dataset, i.e., they create a ranking w.r.t. the probability of a dataset to be relevant with a given one. *LinkLion* [179] is a service collecting pairwise mappings for 476 RDF datasets, while one can discover mappings for a given dataset. In [87], it is presented a dataset recommendation approach for detecting potential datasets, that can be linked to a given dataset and relies on the common schema elements among the datasets. *LODVader* [33] uses Bloom filters to compare, index and count links between RDF distributions in the streaming process. One can upload a dataset description file and that system compares its metadata with the existing datasets' metadata. Then, it provides to the users a LOD Diagram showing the discovered links for their datasets. Nikolov et al. [182] proposed a method that takes as input a single dataset and uses an index and keyword search for retrieving the most relevant datasets to the given one. *LODsyndesis* [165, 168, 171] offers content-based measurements for 400 datasets, e.g., one can find the "K most connected datasets to a given one".

Keyword-based Approaches. *Google Dataset Search* engine [184] crawls and indexes metadata from thousands of datasets and HTML web pages. By using that engine, one can retrieve the most relevant datasets to a set of keywords. Moreover, *Datahub.io* offers also a keyword search mechanism by exploiting the metadata of datasets that have been uploaded from several organizations and users. *Semantic Web Search Engine* (SWSE) [119] resembles a classical search engine, i.e., it crawls and indexes RDF data and provides a keyword search for easing the search, exploration and retrieval of RDF data. *Swoogle* [74] crawls and retrieves several semantic web documents by exploiting metadata and by traversing links. All the retrieved documents are indexed and analyzed by using several metrics for computing ranks (e.g., ranks for ontologies). The users can exploit the keyword search functionality for retrieving results about these documents. *Sindice* [187] uses also a crawler for discovering and fetching RDF files, while it connects to SPARQL endpoints for retrieving RDF datasets. It uses several indexes (e.g., for URIs and literals), while one can submit a keyword query in this system for finding relevant data. *Watson* [67] offers advanced functionality for searching semantic web resources, i.e., one can search in documents and ontologies, find metadata, explore ontologies, write their one SPARQL queries and use several metrics by selecting their own filters. *LODAtlas* [198] is a metadata-based engine, where one can type one or more keywords for finding datasets containing those keywords in their metadata, whereas *LOV* [247] offers a keyword search engine for retrieving relevant vocabularies. In [252], recommendations are provided through keyword search for integrating identical schema and instance elements of different datasets. The users can discover more sources, that are connected with the selected ones by exploiting measurements that try to find similarities between the datasets. *Lotus* [122] is a text engine returning URIs and their provenance for a given set of keywords. Finally, *SPARQLES* [248] and *SpEnD* [258] contain a catalog of SPARQL endpoints, however, their main objective is to monitor the evolution of SPARQL endpoints over specific time periods by providing daily statistics (and visualizations).

URI-based Approaches. The URI-based approaches are based either on indexes or on the existence of dereferencing HTTP URIs, where one can discover relevant URIs on-the-fly (by traversing `owl:sameAs` or other kinds of relationships). *LODsyndesis* [165, 171] provides a global entity lookup service based on a `owl:sameAs` catalog, which retrieves all the datasets (and triples) where a URI u (or an equivalent URI of u) exists. Moreover, *WIMU* [246] offer a service for returning to the user all the datasets where that URI occurs. ExpLOD [133] creates interlink usage summaries, which are used for understanding the contribution of each dataset for a real world entity. *RapidMiner Link Explorer* [208] takes as input a URI and discovers on-the-fly relevant URIs and datasets by following `owl:sameAs` paths. The advantage of on-the-fly approaches is that one can explore a large number of URIs for the same entity, while by using an index, the number of URIs for the same entity is stable. However, they strongly depend on dereferencing HTTP URIs, since a path terminates when a non-dereferencing URI is visited. Finally, a number of keyword searching services [67, 74, 119, 187] also offer a URI lookup service.

2.4.3 Integration Substance (Materialization vs Virtualization)

Here, we describe the *Integration Substance* (or *Integration Architecture*), which can be *materialized* or *virtual*. The corresponding integration approaches are described below and Figure 2.7 shows their main steps.

Materialized Approach. In the materialized (or warehouse) integration, the integrated data are stored in a single repository [52, 236]. As a first step (see upper part of Figure 2.7), one should select the datasets (or views of specific datasets) that are appropriate for fulfilling the requirements of the warehouse, which are defined in its “design” phase. In the “creation phase”, it is a prerequisite to download (or fetch) the underlying datasets and usually to transform them to a target model before uploading them in that repository, i.e., since datasets use different schemas, formats and so forth. Moreover, mappings among

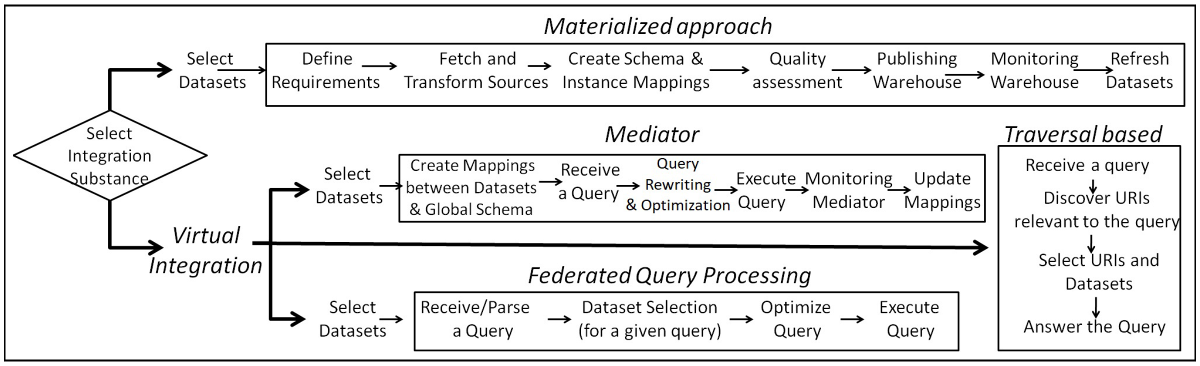


Figure 2.7: Steps for materialized and virtual integration

the underlying datasets in both schema and instance level should be defined, for enabling the answering of complex queries, that combine information from two or more datasets. A crucial step is the "testing phase", where one assess the quality of the constructed repository, by using several metrics and by taking into consideration the defined requirements. In the "monitoring phase", one should monitor the underlying datasets for identifying possible changes in one or more of them. Such a change can result to the reconstruction of the whole warehouse, i.e., "refresh phase". Regarding the benefits of this integration substance, (a) it is more flexible in terms of transformations, (b) the stability and robustness of the materialized repository do not rely on the datasets' servers, i.e., one should access such a server only either for fetching a dataset or for monitoring possible changes, instead of answering queries, and (c) it can offer faster responses, mainly in query evaluation, and secondarily in several other tasks, e.g., for applying techniques for instance or schema matching. Concerning the drawbacks, there is a cost for hosting such a repository, while it needs a periodical monitoring and refresh. Figure 2.3 contains an example with the required steps for building a *Semantic Warehouse*. Moreover, we should mention OLAP approaches, which is a special case of materialized data integration. In this case, data are described by using a star-schema (modeling entities of a single type), while "data are organized in cubes (or hypercubes), which are defined over a multidimensional space, consisting of several dimensions" [249]. That technology is mainly used by enterprises, for producing critical analytics (aggregate queries), by using internal data (e.g., sales of a company), or/and external data, e.g., through the exploitation of semantic web technologies [27]. OLAP queries mainly belong to *Analytics* service of *BasicServicesToDeliver* dimension (see Figure 2.4). In [27], several OLAP approaches are surveyed, while "*The RDF Data Cube Vocabulary*" (`http://www.w3.org/TR/vocab-data-cube`) can be exploited for publishing multi-dimensional data (results of aggregated queries) in RDF format. Finally, we should also mention *Data Marts*, which are "small units of a Data Warehouse that are dedicated to the study (analysis) of a specific problem" [47], i.e., they belong to *Analytics* service of *BasicServicesToDeliver* dimension (see Figure 2.4).

Mediator (Virtual Integration). In the mediator approach, the data remains in the original sources [52,236], while sources can be unaware that they are part of an integration system [121]. Concerning its "'design phase" (see central part of Figure 2.7), one should select the datasets that will be used and to define a mediated schema, which is essential for supporting query translation among the different models of the underlying datasets' schemas. Thereby, the mappings between the mediated schema and each dataset should be created. The core functionality of a mediator contains three main steps. Initially, a query, expressed by exploiting the mediated schema, is received. The query is disassembled in smaller sub-queries, where the mappings are exploited for performing *Query Rewriting*, i.e., as stated in [54], "the problem of query rewriting consists in reformulating the query into a (possibly) equivalent expression, called rewriting, that refers only to the source structures". Therefore, such a process makes it feasible each sub-query to be answered by a specific dataset, and for optimizing the query execution plan. Concerning the last step, each sub-query is sent to a specific dataset's server, which in turn sends a response with the answer of such a sub-query, and the responses of all the sub-queries are merged for providing to the user the answer of the initial query. Concerning the "monitoring phase", it is of primary importance to monitor the underlying sources for detecting possible changes that can result to the reconstruction of the mappings between the datasets and the mediated schema. Regarding the advantages of a mediator, there is no need to pay a cost for hosting the dataset, while it can access in real-time updates of datasets' content. On the contrary, its efficiency, quality and complexity relies mainly on the sources' servers. In Figure 2.3, we can see an example for a *Mediator* approach.

Federated Query Processing (Virtual Integration). "It refers to the process of running SPARQL queries over several SPARQL endpoints" [194]. The underlying sources either use terms from the same schemas for describing their data, or they use common URIs for describing specific entities. As a consequence, it is a prerequisite that specific schemas and URIs are reused for achieving federated query processing, i.e., for performing joins among two or more sources. As it can be seen in Figure 2.7, the first step is to select which datasets will be used for answering queries. Afterwards, the federated query processing contains the following steps: "*Query Parsing, Data Source Selection, Query Optimization* and *Query Execution*". *Query Parsing* is the process of parsing and transforming a given query expressed by using SPARQL query language into a query execution tree [186], while *Data Source Selection* is used for finding the relevant datasets (i.e., SPARQL endpoints) for a triple (or a set of triples) pattern of a given SPARQL query [108]. By having selected the datasets for a given query, *Query Optimization* process starts for placing the triple patterns into groups, and it is used for determining in an efficient way the order of joins and triple patterns. The last step, i.e., *Query Execution*, is performed for answering the initial query. Similarly to a mediator approach, the data remains in the original sources. However, a federated query approach does not depend on a global schema (and thus there is no need to create mappings among different sources); instead it assumes that the underlying sources describe their data by using a common data model (therefore, it is not capable to tackle heterogeneities such as different conceptualization). Moreover, sources can be aware that they participate in such a federation system [186]. In Figure 2.3, we can observe an example of a federated query engine.

Traversal-based Integration (Virtual Integration). The traversal based integration depends on the live exploration of data links at query execution time [115]. First, such an approach receives a query and searches for URIs given either in the query body or as additional parameters. Second, it discovers URIs that are relevant to that query by traversing paths (e.g., `owl:sameAs` paths), for finding more URIs that can be possibly exploited for enriching the results of the given query. Afterwards, the URIs and datasets that will be used for answering the query are selected, since it is not necessary to use all the relevant URIs (such a selection can be assisted by predefined rules). Finally, it collects the answers for each URI and returns the final answer (containing data from one or more datasets) to the user. One major advantage of this approach is the live exploration and discovery of relevant datasets that were unknown, since it does not need prior knowledge for answering a query. Consequently, there is no need to store and transform data in such systems, i.e. data remains in the original sources. On the contrary, the data access time can be a drawback due to the recursive process which is usually followed, while the existence of few available deferencable links makes it difficult to discover relevant data. Moreover, since datasets are explored at real time, it is difficult to measure their quality, while the possible chain of links can be huge. In Figure 2.3, we can see that a smaller number of steps is required for a *Traversal-based* approach. Comparing to other Virtual Integration approaches, it neither uses a global schema (like in a mediator approach), nor it knows a-priori all the candidate sources that can be used for answering a query.

Hybrid Integration. A *Hybrid Integration* approach can share characteristics from two or more integration substances, e.g., suppose an approach, where a part of data are fetched and transformed, while an other part of data is explored at query time (e.g., by following `owl:sameAs` paths).

2.4.4 Internal Services

This set of services are usually performed during the integration process. In §2.4.4.1 we discuss *Fetching and Transformation*, in §2.4.4.2 and §2.4.4.3 we discuss *Schema* and *Instance Matching*, respectively, whereas in §2.4.4.4 we describe the process of *Query Answering* in virtual integration.

2.4.4.1 Fetching and Transformation

It aims at fetching the data that will be used in a materialized approach and at transforming them into a common format.

Context. It can be applied in a Materialized approach, since in a Virtual approach, data are left in their original sources. We can see in Figure 2.3 (in the materialized approach) that the scientist/user downloaded the datasets and transformed the blank nodes of D_2 to URIs.

Difficulties. It can tackle difficulties (a) and (e), i.e., the heterogeneity of datasets in terms of format, schema, and modeling decisions that datasets follow for the same real-world objects.

Categorization. A variety of access methods can be offered from each dataset for *fetching* its contents, (all contents or a specific "slice" of a dataset). In particular, for many RDF datasets, a SPARQL endpoint or an RDF dump is provided, while alternative access methods include accessible files through HTTP, a JDBC connection and others. The datasets that are not provided in RDF format need to be transformed to that format, i.e., *format* transformation. For some datasets a *logical* transformation should be also performed, i.e., for enabling the representation of these datasets' content by using a core ontology. The *logical* transformation can contain rules that change the language of a literal, rules that transform the type of an RDF resource, e.g., transform a URI to a blank node, a URI to a literal, a blank node to a URI, etc., or rules for fixing syntactical errors.

Approaches. Concerning *Materialized* approaches, *LDIF* [222] can import data, that are represented in different formats (e.g., RDF/XML, turtle), from SPARQL endpoints or/and by using a crawler. It uses the *R2R* Framework [43] for performing complex *logical* transformations, in order to integrate data represented through different ontologies into a single one (i.e., conceptualization issues). `MatWare` [243] uses plugins for fetching data that are represented in different formats (e.g., RDF, JDBC, HTTP), and for providing *format transformation* for the fetched data. It also supports *logical transformation* through SPARQL queries and plugins, while it uses the *X3ML* framework [156], which handles in a state-of-the-art way the URI generation and the *logical* transformation. *ODCleanstore* [135] fetches RDF data through a SOAP webservice, and executes a defined set of transformers for offering a common data representation (i.e., logical transformation). *KARMA* [136] imports files in many different formats (csv files, RDF, etc.) and transforms all the different data formats into a nested relational data model. It supports also *logical* transformation, i.e., one can combine several ontologies for creating mappings between their data and standard ontologies, while it uses techniques for identifying and suggesting to the user possible mappings. Regarding *Hybrid Integration* approaches, *TopFed* [216] transforms billions of Cancer Genome Atlas (TCGA) data into RDF format, while the transformed sources are described by using the same schema (i.e., format and logical transformation).

RapidMiner LOD Extension [208] imports data in several formats (e.g., csv, excel), while it offers a SPARQL and an RDF Data Cube importer. It supports both *format* and *logical* transformation for representing data in a common format. FuhSen [62] fetches data from different formats (e.g., REST, RDF, SQL) by using wrappers, while data are transformed into RDF molecules, i.e., an RDF molecule is the cluster of all the triples of a specific subject. Furthermore, the authors in [197] fetched billions of HTML pages and they extracted the data expressed in Microdata format and RDFa for producing billions of RDF triples (the produced collection is accessible in `http://www.webdatacommons.org/`). Moreover, there exists approaches focusing mainly on fetching and transforming datasets. *LODLaundromat* [207] offers a common representation for over 650,000 RDF documents after cleaning them, i.e., they are free of syntactic errors. The authors of [224] have proposed automatic ways for publishing relational databases to RDF, while [225] surveys various approaches for mapping relational databases to RDF, e.g., *CSV2RDF* [90] transforms csv and excel files to RDF.

Evaluation Collections. LODIB [44] is a benchmark for evaluating whether a tool can detect several mapping patterns and perform the required logical transformations (transforming a literal to URI, renaming a class, etc.), for representing the data of three different sources by using a single target vocabulary. Moreover, it measures the performance of each tool in terms of execution time. Finally, in [44] several data transformation benchmarks are mentioned (except for LODIB).

2.4.4.2 Schema/Ontology Matching (or alignment)

It refers to the problem of determining mappings at schema level between schema concepts. i.e., classes (e.g., Person owl:equivalentClass Human) and properties (e.g., staysAt rdfs:subPropertyOf livesAt).

Context. An integration system can (a) use *predefined* (PD) ontology mappings (and possibly their closure) that have been declared (e.g., axiomatically) before the integration process or/and (b) exploit *ontology matching* techniques (OMT) for producing new mappings during the integration process. In the simplest case, when two or more datasets use the same schema (or ontology), there is no need to create mappings between them (e.g., federated query engines depend on datasets that share concepts from same schemas). However, by having two or more datasets with different schemas, mappings are essential for being able to answer queries over the integrated content. By deciding to build a warehouse, one can either a) create the mappings between each pair of datasets or b) can define a single global schema and create the mappings between that schema and each dataset, which enables the adequate mapping and integration for data, derived from distinct datasets. By choosing to build a mediator, a (single global) mediated schema is created, whereas mappings between that schema and each dataset's schema are also created.

The major advantage of using a single global schema is the less effort that is required for schema mappings: e.g., given K sources instead of having to construct $K * (K - 1)$ pairwise mappings, only K mappings are required (one for each source). Moreover, the focus is given on one model, rather than many. The most commonly used schemas for declaring mappings among ontologies are RDF/RDFS (e.g., rdfs:subClassOf and rdfs:subPropertyOf) and OWL (e.g., owl:equivalentProperty). In Figure 2.3, for the *Warehouse*, the user selected to create the mappings between the underlying sources, while for the *Mediator*, the mappings were created between a global schema and each underlying dataset.

Difficulties. This process is not trivial, mainly due to difficulties (a), i.e., datasets are usually published by using different and heterogeneous schemas and models, (e) different conceptualization of the same domain, and (f) evolution, i.e., several ontologies are frequently changed. This problem has attracted the interest of the community for decades. For instance, [230,231] extensively describe the difficulties of integrating different schemas in the area of relational databases and [134,183] analyze the difficulties of matching different ontologies in *Semantic Integration* area.

Categorization. We categorize matching tools (either schema or instance matching tools) in *manually specified linking configuration* tools (i.e., a user manually specifies some rules), *learning-based* tools (e.g., they can exploit machine-learning methods) and tools using *both techniques* [178], as it can be observed in Figure 2.8. Moreover, since we emphasize on big number of data (and datasets) we can categorize the approaches according to the type of the solution that they offer: a *centralized solution*, which means that the computation is performed in one machine, a *parallel solution*, i.e., a cluster of machines is used for speeding up the process, or *both solutions*, i.e., they provide both a centralized and a parallel solution. Furthermore, we can divide the tools according to the output that they produce, i.e., a *Schema Matching* approach produces always *Schema* mappings, an *Instance Matching* approach creates always *Instance* mappings (see §2.4.4.3), whereas some tools produce both instance and schema mappings. Moreover, we can distinguish the approaches according to the types of evidence (which were presented in §2.3) that they exploit for creating semantic bindings, and the resources that they use, i.e., only datasets' content (e.g., triples of datasets), or/and external resources (e.g., lexicons, translators) for improving the accuracy of mappings.

Approaches. Starting from the area of *Relational databases*, [230,231] has proposed a generic integration methodology (which can also be adopted by semantic web approaches) that concern the automatic resolution of conflicts (e.g., structural conflicts) and automatic generation of mappings among different schemas (or views) and the integrated schema by taking into consideration the underlying semantics of the different schemas. Fully automatic methods have also been proposed in various contexts, including query-oriented integration of relational databases [35] and methods for creating automatically mappings between taxonomy-based sources [241]. Below, we mention eight ontology matching tools, which are categorized in Table 2.2 by using the criteria of Figure 2.8.

YAM++ [180] is an ontology matching tool that provides several string similarity metrics, topology-based metrics, and metrics for identifying similar neighbors. It combines these metrics with machine learning methods for creating mappings, it uses inference-based rules for refining mappings, while it exploits a translator and Wordnet for tackling language issues (e.g., multi-lingual issues).

StringOpt [57] uses a large variety of string similarity metrics (such as Jaro, Levenstein, N-gram, etc.), external resources (e.g., lexicons, translators), and pre-processing strategies (e.g., stemming), while the authors tested and compared those string similarity metrics for several ontologies.

LogMap [127] is a tool that can be used by ontologies containing thousands of concepts and combines string similarity with topology-based metrics for identifying mappings between different ontologies. Moreover, it exploits external lexicons for finding synonyms and lexical variations, while it can detect and repair on-the-fly inconsistent mappings by exploiting inference rules.

SBOMT [185] is a scalable statistically-based ontology matching tool that can identify even 1:*n* relationships. It creates features by using several types of evidence (e.g., string similarity), which are used as input in a machine learning task along with historical mappings among ontologies, for learning and predicting new mappings, while WordNet is used for removing erroneous mappings.

AggrementMaker [65] is a tool that supports several structural and lexical matching methods (e.g., by using WordNet) and can produce even *n*:*m* relationships. Moreover, it offers manual and automatic user intervention while it can also produce mappings between instances.

WebPie [245] is a parallel engine that can be used for computing the inference of instances, classes and properties. It can compute the inference for several types of relations such as transitive, symmetric, inverseOf and so on, while the authors have tested the engine with billions of triples.

PARIS [232] is a learning-based system for discovering relationships between entities of different datasets and RDF ontologies. It mainly tries to identify class similarity between two different ontologies (e.g., exact or in-exact relations) by exploiting similarities between the instances.

XMAP++ [77] is a tool using lexical, structural and linguistics metrics for performing ontology matching. Specifically, it exploits external lexical databases and thesaurus, such as WordNet, for providing synonyms between two entities from different ontologies.

Furthermore, an “Expressive and Declarative Ontology Alignment Language” (EDOAL) can be exploited for describing complex relations of entities that are described by using different schemas (`http://ns.inria.org/edoal/1.0/`).

Evaluation Collections. The most popular benchmark (and competition) is the OAEI

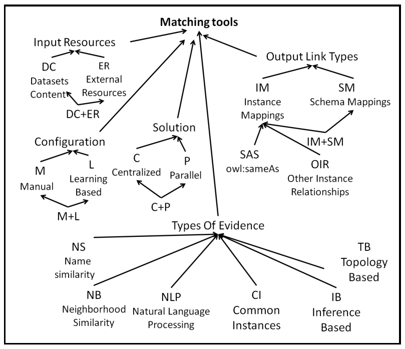


Figure 2.8:
The different criteria of (schema and instance) matching tools

Tool	Input Resources	Configuration	Solution	Output Link Types	Types of Evidence
Yam++ [180]	DC+ER	M+L	C	SM	NS, NB, NLP IB
LogMap [127]	DC+ER	M+L	C	SM	NS, NB, NLP, IB, TB
StringOpt [57]	DC+ER	M+L	C	SM	NS, NLP
SBOMT [185]	DC+ER	M+L	C	SM	NS, NB, NLP, CI, IB
Agreement Maker [65]	DC+ER	M+L	C	SM	NS, NB, NLP TB
XMAP++ [77]	DC+ER	M+L	C+P	SM	NS, NB, NLP, IB, TB
WebPie [245]	DC	M	P	SM+IM	IB
PARIS [232]	DC	L	C	SM+IM	NS, IB
Silk [250]	DC	M+L	C+P	SAS+OIR	NS, NLP,TB
LIMES [181]	DC	M+L	C+P	SAS+OIR	NS
LINDA [45]	DC	L	C+P	SAS	NS,NB,IB
MinoanER [85]	DC	L	P	SAS	NS,NLP
CoSum-P [262]	DC	L	C	SAS+OIR	NS, NB, TB
MINTE [63]	DC	L	C	SAS	NB, NS, TB

Table 2.2:
Categorizing Existing Schema & Instance Matching approaches for large-scale datasets according to the criteria of Figure 2.8

(Ontology Alignment Evaluation Initiative) which is a yearly evaluation event (`http://oaei.ontologymatching.org/`). It contains several test datasets, where each one focuses on different difficulties of ontology matching process, such as matching ontologies of the same or different domains, ontologies using different languages, and others. Finally, [188, 227] survey over 60 *Ontology Matching* tools and a number of benchmarks for comparing the performance of such tools.

2.4.4.3 Instance Matching (or alignment)

It refers to the problem of determining mappings at the data level, between real world entities (or instances/individuals), e.g., Thunnus Albacares ≡ Yellowfin Tuna since they refer to the same marine species.

Context. An integration system can (a) use *predefined* (PD) instance mappings (and possibly their closure) that have been declared (e.g., axiomatically) before the integration process or/and (b) exploit *instance matching* techniques (IMT) for producing new mappings during the integration process. A materialized approach can use any of these approaches, while a virtually integration system usually relies on existing equivalence relationships and on their closure (e.g., traversal-based engines follow `owl:sameAs` paths). The most commonly used mapping (or relationship) is the `owl:sameAs` relationship, i.e., an entity e_1 is same as entity e_2 (e_1 `owl:sameAs` e_2) when both entities refer to the same real entity, e.g., `http://www.dbpedia.com/Thunnus_Albacares` `owl:sameAs` `http://www.ecoscope.com/thunnus_albacares`, while such mappings, among the URIs of marine species *Thunnus Albacares*) are shown in Figure 2.3 (i.e., in the *Materialized* approach). Comparing to *Schema Matching*, *Instance Matching* process produces mappings for instances only (i.e., instances and schema concepts are two disjoint sets), whose number is usually much larger than the number of schema concepts [168]. However, both processes can be very complex due to several difficulties (e.g., different conceptualizations), while [55] provides more details about these two techniques.

Difficulties. It is related to the existence of many URIs for the same real entity (difficulty b).

Categorization. We categorize these tools by using the same criteria described in §2.4.4.2, which are shown in Figure 2.8. However, we can further categorize instance matching tools according to the mappings that they support (see Figure 2.8). Most tools usually produce `owl:sameAs` relationships, while some tools are also capable to produce other relationships, such as **foaf:based-near**, which declares that an entity has similar meaning with an other one but not exactly the same.

Approaches. Table 2.2 (i.e., see the last 8 tools) shows scalable tools performing instance matching which are briefly described below.

Silk [250] supports manually specified rules and supervised learning for producing `owl:sameAs` links (by default) or other user-specified relationships. It uses mainly several string similarity metrics, while it provides a parallel version for scaling out to very big datasets. Finally, *Silk* is used as a component from three materialized tools, i.e., OD-Cleanstore [135], *`MatWare`* [243] and LDIF [222].

LIMES [181] is a tool that supports both manual configuration and learning techniques (supervised and unsupervised). It offers different approximation techniques based on metric spaces for estimating the similarities between instances. It can produce both `owl:sameAs` and user-specified links, it offers both a centralized and a parallel version, while it is used by *TopFed* [216].

PARIS [232] can detect `owl:sameAs` relationships by exploiting functional properties. This system (which is used by *RapidMiner LOD Extension* [208]) does not require a configuration from the user and offers a centralized solution which have tested with a large number of triples and entities.

WebPie [245] can take as input `owl:sameAs` relationships and computes in a parallel way their transitive and symmetric closure in order to produce inferred `owl:sameAs` relationships.

LINDA [45] is a fully automatic system offering both a centralized and a parallel version and has been tested for over 100 millions of entities. It uses several techniques for identifying `owl:sameAs` relationships, while it checks the neighborhood of different entities for inferring relationships.

MinoanER [85] ia a system that discovers `owl:sameAs` relationships by placing similar descriptions into blocks, while it tries to discover mappings by using descriptions that occur in the same block. It has been built by using parallel frameworks and has been tested with millions of entities.

CoSum-P [262] is a generic framework that performs entity matching by solving a multi-type graph summarization problem. It receives an RDF graph and creates a summary graph, while it uses several structural similarity metrics, such as the number of common neighbors.

MINTE [63] (which is a component of *FuhSen* [62]) computes similarities between RDF molecules for matching equivalent entities by using semantic similarity functions, e.g., GADES [234], while except for perfoming instance matching, it integrates also all the triples of a specific real entity.

Recently, there is trend for approaches that use embeddings and Machine Learning techniques for finding similarities. *MateTee* [160] is an approach that creates a vector representation of entities and uses a Stohastic Gradient Descent method for computing similarities among entities. *RDF2VEC* [210] converts an RDF graph in sequences of entities and relations, and adapts neural language models, such as word2vec [153]. The produced vectors can be exploited for Machine Learning tasks (e.g., for identifying similarities between two entities). Finally, the task of matching different entities can become more complicated since there exists a remarkable percentage of unnamed entities (e.g., see the example with the book in Figure 2.5), i.e. blank nodes. For such a case, there have been proposed techniques based on signatures [140] for blank nodes matching.

Evaluation Collections. One can exploit the *OAEI* benchmark (as in the case of *Ontology Matching*) for evaluating the performance of *Instance Matching* systems with respect to various difficulties, such as detecting differences on the values or/and the structure of different datasets that contain information about the same entities. Moreover, it contains a track, which can be used a) for both *Ontology* and *Instance Matching*, and b) for creating mappings not only between pairs, but also among triads of datasets (`http://oaei.ontologymatching.org/2018/knowledgegraph`). Finally, [68] contains several real and synthetic benchmarks for evaluating instance matching systems.

2.4.4.4 Query Answering in Virtual Integration

It refers to the process of answering of a given query in virtual integration.

Context. In Figure 2.3, one can see specific examples of answering a query by using a mediator, a federated query engine and a traversal based approach.

Difficulties. For a mediator, the task of query rewriting requires the computation of mappings between the mediated schema and the underlying datasets, thereby, the heterogeneity in terms of schema, i.e., difficulty (a), and different conceptualizations, i.e., difficulty (e), can make this process more difficult. For a federated query or a traversal-based engine, it is important to select datasets that use the same model or/and URIs (they should be dereferencing in a traversal-based case), since they do not produce mappings, thereby, i.e., difficulties (a) and (b) should be overcome.

Approaches. We introduce approaches concerning the *Query Answering* process for each different virtual integration type.

Query Answering over a Mediator. In the database field, there exist two main approaches that can be defined for this process with respect to the way that the mappings have been defined. Specifically, they are called *LAV* (local as view) and *GAV* (global as view) while there exists approaches containing a combination of them, i.e., *GLAV* (global local as view) [54, 144]. *LAV* is an approach where "the source structures are defined as views over the global schema" [257] and most of the proposed solutions are based on query answering by using query rewriting, i.e., after the query rewriting process the query can be evaluated over the underlying sources for obtaining the results. *GAV* is an approach where "each global concept is defined in terms of a view over the source schemas" [257] and most solutions try to substitute each global relation with its corresponding definition concerning the sources. *GLAV* is an approach specifying the mappings between the source schemas and the global one. Finally, Maurizio Lenzerini, Diego Calvanese and their colleagues have extensively studied the aforementioned *LAV*, *GAV* and *GLAV* techniques and have surveyed a lot of approaches using these techniques [54, 144].

The above techniques have been also applied in the context of Linked Data [158]. In particular, *SemLAV* [159] is a LAV-based approach for processing in a scalable way SPARQL queries, by producing answers for SPARQL queries against large integration systems, whereas in [141], GAV SPARQL views are exploited for rewriting queries against a global vocabulary. Regarding other query rewriting approaches, in [244] techniques for query rewriting over taxonomy-based sources were proposed while in [148] the authors proposed a framework performing query rewriting for SPARQL queries. It takes as input a SPARQL query represented by a specific ontology, and transforms it to a semantically equivalent query, by using an other ontology. Furthermore, in [64] a technique is described for exploiting transformations between RDF graphs for enabling query rewriting.

Finally, we should mention the work that has been done by Diego Calvanese and his colleagues [51, 53, 256], for answering SPARQL queries over relational databases. Particularly, the *Ontop* Ontology-Based Data Access system [51] gives the opportunity to the users to access relational databases, by rewriting SPARQL queries into (federated) SQL queries over the underlying databases. Moreover, they have extended this work by proposing methods for assigning a single canonical IRI to entities [256]. In particular, they assign such a canonical IRI, for all the IRIs that refer to the same entity (i.e., these IRIs are connected through a direct or through an inferred `owl:sameAs` relationship). Their target is to avoid to provide redundant answers, i.e., avoid to include all the possible combinations of equivalent identifiers in the answers, and thus to achieve lower execution times.

Query Answering over a Federated Query Engine. The proposed engines can be distinguished in three different categories [213], according to the approach that they follow for selecting the datasets that can answer a given sub-query. First, there exists *Catalog/index-assisted* approaches, where they maintain a catalog for the available endpoints while indexes have been created and statistics have been collected during the construction of the virtually integrated system. Then, for each query the aforementioned indexes and metadata are exploited for selecting the most relevant datasets for a given triple pattern. Secondly, *Catalog/index-free* solutions maintain a catalog for the endpoints, however, they collect statistics on-the-fly (e.g., by exploiting SPARQL ASK queries) and the datasets are selected through the aforementioned statistics. Finally, *Hybrid solutions* combine both techniques for selected the datasets that can answer the initial query. Concerning *Catalog/index-assisted* approaches, *DaRQ* [201] system uses an index containing service descriptions for each source, which include the triple patterns that are answerable by each source. As regards *Catalog/index-free* approaches, *FedX* [223] system sends one SPARQL ASK query per triple pattern to the federated datasets' endpoints for detecting relevant datasets, e.g., for a triple pattern {?s foaf:name ?name}, it send the following query to each dataset D_i: "ASK {?s foaf:name ?name}", which returns true if D_i contains triples with this property. Finally, for decreasing the execution time of source selection task, query approximation techniques have been proposed [108].

On the contrary, most tools use a hybrid approach for answering a given query. In particular, *SPLENDID* [106] uses an index containing VoID descriptions for finding the candidate sources for answering a triple pattern, however, for triples containing variables that are not included in the VoID statistics, it sends SPARQL ASK queries. *HiBISCuS* [214] exploits an index for the subjects and the objects and SPARQL ASK queries for the predicates for discovering the relevant datasets for a given triple. *ANAPSID* [29] uses a catalog containing the available SPARQL endpoints and their ontology concepts, while statistics for the endpoints are updated on-the-fly in order to maintain up-to-date information. DAW [215] constructs indexes that contain information about a distinct predicate of a specific dataset (e.g., how many triples contain this predicate), and uses a novel source selection algorithm for ranking the sources based on their contribution, in order to select the most relevant sources for a given triple pattern. MULDER [88] sends SPARQL queries for collecting descriptions of RDF molecules templates, i.e., "descriptions of entities of the same RDF class". It mainly exploits that descriptions to select the datasets that can increase the completeness of the answer. Finally, more details about dataset selection of federated query engines and issues that concern query optimization (e.g., query planning, join strategy) are out of the scope of this survey, however, one can find surveys containing analytical details for these steps [186, 212, 213].

Query Answering over Traversal-based Integration Approaches. *SQUIN* [116] receives a query and tries to retrieve triples by using the URIs that can be found in that query. Then, a recursive iterative process (which stops only when a fixed point is reached) that relies on the aforementioned triples starts, and in each step, *SQUIN* tries to discover incrementally more URIs and triples that are related to the given query. *Linked-Data-Fu* [113] system supports traversal-based integration, by following links based on specific rules that can be declared by using *Data-Fu* Language. It exploits REST technology, i.e., it starts by sending a POST requests to a specific URI, and the results can be derived through a GET request. By sending such requests, it can discover on-the-fly links and it can send new requests for retrieving even more data. *SWGET* uses NautiLOD formal language [99], for retrieving in a recursive way data from several linked datasets. It relies on regular expressions and ASK SPARQL queries for selecting the most appropriate sources in order to continue the navigation (e.g., by traversing `owl:sameAs` paths), while one can control the navigation through SPARQL queries. Moreover, we should mention *SPARQL-LD* [93], an extension of SPARQL 1.1 that enables to combine in a single query (and perform whatever transformation is possible via SPARQL) data coming not only from external SPARQL endpoints, but also from external data stored in RDF, JSON-LD files, as well as data from deferenceable URIs, and others.

Query Answering over Hybrid-Integration Approaches. *TopFed* [216] is a hybrid approach (mainly a federated query engine) that contains biological data. We categorize *TopFed* as a hybrid approach, since data are transformed for being described by the same schema and they are linked with external sources through schema/instance matching techniques. For answering a specific query, this engine uses both a metadata catalog and ASK queries. *FuhSen* [62] takes as input a keyword query and a similarity threshold for creating a knowledge graph at query time. It uses a global vocabulary (called OntoFuhSen) for transforming the initial query to a SPARQL query or to a REST request, and sends federated queries to the relevant sources. Finally, the responses of the queries are enriched, by integrating all the triples of a specific entity (by using MINTE [63]). *RapidMiner LOD Extension* [208] is partially a traversal-based approach. It follows paths of a given type (e.g., `owl:sameAs` paths) for finding and integrating data over different sources, however, it can fetch data and perform instance/schema matching, while it offers data fusion mechanisms.

Evaluation Collections. There exists benchmarks like *FedBench* [220], LargeRDFBench, SP^{2}Bench and others [212], which cover several dimensions such as the result completeness, ranking of the returned answers and efficiency of each different step of the process (e.g., source selection).

2.4.5 Auxiliary Services

This set of services are auxiliary and can be exploited before or after the integration process. In §2.4.5.1 we analyze issues concerning data *Provenance*, while in §2.4.5.2 we introduce ways to measure and improve the *Quality* of one or more datasets. In §2.4.5.3 we show how one can *Monitor* the *Evolution* of datasets, whereas in §2.4.5.4 we discuss the process of *Publishing* an integrated dataset.

2.4.5.1 Provenance.

"It focuses on how to represent, manage and use information about the origin of the source or data to enable trust, assess authenticity and allow reproducibility" [261].

Context. Data provenance should be preserved regardless of the selected integration approach.

Difficulties. It mainly refers to difficulty (a), i.e., the initial format of the datasets can change and the contents of a dataset can be transformed (e.g., for being compatible with a global schema). In such cases, one should respect the datasets' licenses, and record the provenance of each triple.

Categorization. There are various levels of provenance support which are usually required: "i) Conceptual level, ii) URIs and Values level, iii) Triple Level and iv) Query level" [236]. Regarding level (i), the key point is that one can transform specific triples according to a conceptual model that models provenance, i.e., it is mainly applicable in a Materialized approach. Concerning level (ii), one can adopt the "namespace mechanism for URIs", i.e., the prefix of the URI can be exploited for providing information about the origin of the data (can be supported from any approach), while for literals one can use the extension "@Source" to the end of every literal (e.g., "Thunnus" @Dbpedia). Regarding level (iii), by storing each dataset in a separate graphspace, the origin of a triple can be obtained by asking for the graphspace containing that triple (applicable only in a Materialized approach), while N-Quads format can be exploited for storing each triple's provenance. Finally, level (iv) can be supported by offering query rewriting techniques, i.e., one can exploit the graphspaces' contents for showing the contribution of each dataset to the answer of the query. It can be supported from any integration substance (mainly for Virtual Integration approaches).

Approaches. In the first provenance challenge [161] (held on 2006), several teams selected to exploit Semantic Web technologies for creating systems in order to represent the provenance for a "functional magnetic resonance imaging workflow" and to answer some predefined provenance queries. A provenance model for RDF data, containing the dimensions of data creation and data access, is described in [114]. Specifically, the authors analyzed the major types and relationships of each dimension. Moreover, they showed ways for accessing provenance metadata, while they introduced properties form popular ontologies (such as Dublin Core and FOAF), that can be exploited for storing provenance information. The aforementioned model was used in [117] along with a method for assessing the trustworthiness of the Web data. By using that method, one can assess the timeliness of the existing provenance metadata and check whether important provenance information are missing for a given dataset. A provenance ontology, which can be used for describing the provenance for both data access and data creation is presented in [118]. In particular, they describe how one can include metadata provenance information to a specific dataset by using that ontology, and how to access and query such metadata, since it is important for several tasks, e.g., for evaluating the timeliness of provenance information. A generic provenance vocabulary, called PROV Model (`http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/`), has been standardised by the W3C community. By using the PROV Model, one can describe the main entities, agents and activities being part of the production of a specific dataset, while information about the conceptual model, the constraints and applications using the PROV Model, can be found in [157].

2.4.5.2 Quality Assessment

"It is the analysis of data in order to measure the quality of datasets by using relevant quality dimensions" [261].

Context. Quality is of primary importance for any integrated system. Figure 2.3 shows an example where a data fusion algorithm resolved a conflict, i.e., two sources agree that the max length of *Thunnus Albacares* is 240cm, while the remaining one contains another value for that fact.

Difficulties. It is predominantly related to difficulty (d) and secondarily to difficulty (f).

Categorization. There are various quality dimensions that can be measured in the context of an integrated system. Some of these dimensions can be used for assessing the quality of a single dataset, e.g., dimensions such as completeness, accuracy, data cleaning, consistency and others [261]. For example, the completeness of a dataset can be measured for detecting and correcting possible errors and inconsistencies. Other dimensions, such as *Data Interlinking, Connectivity* and *Data fusion* require information from two or more datasets (mainly evaluated by materialized approaches), while *Virtual Integration* systems mainly evaluate their *Query Performance* [186, 212, 213]. Below, we emphasize on quality dimensions requiring two or more datasets (mainly large number of datasets), however, we mention some novel approaches for the single dataset case, since the quality of an underlying dataset can affect the quality of the whole integrated content.

Data Interlinking. "It refers to the degree up to which entities that represent the same concept are linked to each other" [261]. For evaluating this quality dimension, one can check the quality of `owl:sameAs` links, the interlinking degree of a given dataset, and others. In [260], the authors distinguished two categories for dataset interlinking: interlinking of external websites, i.e., for measuring the availability of links between different sources, and interlinks with other datasets, i.e., for identifying possible mappings that are inaccurate or links containing not so useful information. [219] focused on crawling a large number of datasets and on providing statistics for them. The authors described data interlinking measurements such as the degree distribution of each dataset (how many datasets link to a specific dataset). LODStats [91] retrieves thousands of number of documents for providing useful statistics about how interlinked each document is, while [109] introduces network measures (e.g., interlinking degree and clustering coefficient), for assessing the quality of mappings and for detecting bad quality links (e.g., `owl:sameAs` links). Finally, there exists approaches, where one can find the number of common links [165, 179], and the number of common triples, literals and schema elements [168], between two or more datasets.

Connectivity Assessment. "*Connectivity* express the degree up to which the contents of a warehouse form a connected graph that can serve, ideally in a correct and complete way, the query requirements of a semantic warehouse, while making evident how each source contributes to that degree" [163, 164, 242]. *Connectivity* can occur both in schema and instance level. It is useful to measure connectivity "a) for assessing how much the aggregated content is connected, b) for getting an overview of the warehouse, c) for quantifying the value of the warehouse (query capabilities) since poor connectivity can result to less expressive queries, d) for making easier its monitoring after reconstruction and e) for measuring the contribution of each source to the warehouse" [163, 164, 242].

Data Fusion, Trust & Fact Checking. "*Data fusion* aims at resolving conflicts from different sources and find values that reflect the real world", according to [84]. In [135] *conflicts* are defined as the cases where two triples, belonging in different sources, contain conflicting object values for a specific subject-predicate pair. Regarding the differences with *Data Interlinking,* the latter aims at evaluating how connected two or more datasets are, i.e., whether they contain information about the same entities, while in *Data Fusion* case, such connections among datasets have already been discovered, and the goal is to find the triples containing URIs that represent the same real-world object and transform them into a single accurate representation by resolving conflicts. In the context of integrated (mainly non-structured) data and relational databases, there exists several proposed methods and approaches, depending on multiple techniques (e.g., probabilistic-based, IR models) [145]. Regarding approaches using Semantic Web notions, Google Knowledge Vault [79] stores the information in the form of RDF triples and identifies for such triple a confidence score. However, it extracts information from both RDF sources (i.e., FreeBase) and non-RDF ones.

Concerning the different strategies of data fusion [145, 152], the simplest case is to provide to the users all the conflicted objects for a given subject-predicate pair with their provenance, and let them decide whether an object is correct or not, i.e., *User-Based* approaches. Another technique called *Resolution-Based*, exploits a number of functions, such as majority or average for deciding which object to keep. Moreover, *Trust-Based* methods take into account the degree up to which we trust the provenance of data, i.e., dataset trustworthiness can be measured as a whole, by measuring the accuracy of all the triples of a dataset, however, it is usually difficult to know dataset's trustworthiness a priori. Concerning semantic integration tools, *LDIF* uses *Sieve* [152] for assessing the quality of the integrated data and exploits *Resolution-Based* techniques (e.g., average), *Trust-based* techniques (e.g., prefer data from trusted sources), and configurable metrics for deciding whether a specific value is correct or a transformation is required for improving its accuracy. *ODCleanStore* [135] offers several conflict resolution rules, where some of them select only one value among the conflicting values (e.g., MAX, ANY), while it can also compute a new value based on the conflicting values. *Fuhsen* uses *MINTE* [63], which applies three fusion policies for performing data fusion of two RDF datasets, while *RapidMiner LOD extension* [208] uses some simple data fusion operators (e.g., majority) for resolving conflicts. Finally, in [104] PageRank is computed for 319 RDF datasets for offering trust measurements that are also useful for evaluating dataset interlinking.

Regarding *Fact Checking*, it can be defined as the "area which focuses on computing which subset of a given set of statements can be trusted" [103, 190]. *DeFacto* [103] is a temporal approach that checks for the validity of facts. It takes RDF triples as input facts, and it exploits the web for searching for possible proofs, by supporting multiple languages. In particular, they propose supervised learning methods that require a large volume of training data while they use a combination of trustworthiness metrics and textual evidence for measuring an evidence score for a specific fact.

Crowdsourcing & Data Quality. "Crowdsourcing refers to the process of solving a problem formulated as a task by reaching out to a large network of (often previously unknown) people" [30]. There exists two different crowdsourcing mechanisms, called *content-based* and *micro-task* [30]. *Content-based* crowdsourcing concerns a group of Linked Data experts. In [139], each expert used TripleCheckMate tool, which enables users to select specific DBpedia resources [142], to detect problems for the triples containing that resources, and to categorize them by using a predefined set of quality issues (described in [261]). *Micro-task* mechanism concerns anonymous (even non-experts) users, thereby, the authors decided to restrict the scope of the possible errors that these users can detect [139]. For a specific triple, they can identify datatype or language errors, and incorrect links and object values (e.g., a person's birth place). Afterwards, for any mechanism that tool stores the incorrect triples (classified according to their quality issue) for further checking. Regarding other approaches, HARE [28] is a SPARQL query engine that exploits *micro-task* mechanism for completing missing values, which can occur at query execution time. Moreover, such a mechanism is used from ZenCrowd [71] for improving the quality of in-

Characteristics/ Systems	*LinkQA* [109]	*Luzzu* [70]	*ODCleanStore* [135]	*Sieve* [152]	*SWIQA* [102]	*RDFUnit* [138]	*MatWare* [243]	*SeaStar* [218]
Quality Dimensions [261]	Completeness, Interlinking	Consistency, Conciseness, and others	Accuracy, Completeness, Consistency, Conciseness	Completeness, Consistency, Conciseness	Accuracy, Completeness, Timeliness	Accuracy, Consistency	Connectivity, Interlinking, Relevancy	Accuracy, Interlinking, Connectivity
Sources Number	Set of Mappings	One Source	Collection of Quads	One Integrated Source	One Source	One Source	Set of Sources	One or Two Sources
Output Format	HTML	RDF	RDF, HTML	N-Quads	HTML	RDF, HTML	RDF, HTML	HTML
Progr. Lang.	JAVA	JAVA	JAVA	JAVA	-	-	JAVA	JAVA
Query Lang.	-	-	-	-	SPARQL	SPARQL	SPARQL	-

Table 2.3: Categorizing existing quality tools according to their characteristics

stance mappings (e.g., detecting incorrect `owl:sameAs` links), and from CrowdMap [217] for improving ontology alignment process.

Tools assessing the quality of RDF datasets. Here, we describe in brief several RDF quality assessment frameworks/tools, which are categorized by using several aspects in Table 2.3.

LinkQA [109] is a tool that can be exploited for measuring and improving the quality of RDF data concerning the dimensions of *interlinking* and *completeness*. It mainly assesses the quality of mappings (e.g., `owl:sameAs` relationships) by using network measurements.

Luzzu [70] evaluates the quality of a single RDF source for 10 quality dimensions, by using 25 predefined metrics, while it also supports user-defined metrics. The results are provided to the users in RDF format by exploiting the daQ ontology and the QPRO Ontology [70].

ODCleanStore [135] evaluates the quality of the integrated RDF dataset for four quality dimensions, whereas it offers methods for resolving the conflicts between the underlying datasets. The results of the measurements are produced in both an HTML page and in RDF format.

Sieve [152] (which is part of LDIF [222]) evaluates the quality of an integrated dataset. It offers configurable techniques for resolving conflicts and deciding the correctness of a specific value and it supports three quality dimensions, while the results are stored in N-Quads format.

SWIQA [102] is applicable for a unique dataset and relies on a set of rules for detecting possible errors in the data values. It uses several measurements for evaluating three quality dimensions, while it can also be adjusted for evaluating a relational database, by using D2RQ wrapping technology.

RDFUnit [138] relies on predefined or custom SPARQL queries, and its objective is to evaluate and improve the *accuracy* and the *consistency* of a single RDF dataset. It mainly tries to assess whether an ontology is correctly used and the results are offered in both an HTML page and in RDF format.

`MatWare` [243] computes connectivity metrics for assessing the connectivity among the integrated datasets. It produces as output an HTML page, and a file in RDF format by using VoIDWH ontology [163], i.e., an extension of the W3C standard VoID ontology (`http://www.w3.org/TR/void/`).

SeaStar [218] is a framework that assess the quality of one or two sources mainly in terms of *interlinking* dimension. Its major target is to evaluate the accuracy of several types of links (e.g., `owl:sameAs` links) between pairs of datasets.

2.4.5.3 Dynamics/Evolution & Monitoring.

"*Dynamics* quantifies the evolution of a dataset over a specific period of time and takes into consideration the changes occurring in this period" [76]. The objective of *Monitoring* is to observe and check the progress or quality of an integrated access system over a period of time.

Context. Since everything changes very fast, any integration system should take this dimension into account. For instance, we see in Figure 2.3 an example where the schema of a dataset changed, thereby, the mappings in a materialized or in a mediator approach should be regenerated.

Difficulties. It is predominantly related to difficulty (f) and secondarily to difficulty (d).

Categorization. We can distinguish the integration approaches in three categories according to how they are affected when a dataset changes: (a) approaches that needs to be updated *manually*, (b) approaches that can be semi-automatically updated (e.g., by modifying a part of a configuration file and by pushing a "reconstruction" button) and (c) approaches that are *automatically* updated (i.e., not affected from datasets' updates). Moreover, datasets' evolution can occur in *ontology level* and *instance level.* Evolution in ontology level occurs when the schema of at least one dataset changes or the global schema changes. By using a global schema, only the mappings between that dataset and the global schema should be recreated (i.e., in a mediator or/and in a warehouse). However, the construction of pair-wise mappings results to the recreation of the mappings between the aforementioned dataset and every other dataset (i.e., in a warehouse). For *Catalog/index-assisted* or *hybrid* federated query approaches (such as DaRQ [201]), the indexes containing statistics for properties/classes should be refreshed, while approaches such as FedX [223] (which relies on ASK queries), or ANAPSID [29] (which collects statistics on-the-fly) are automatically updated. On the contrary, evolution in instance level occurs when the policy (e.g., the prefix) of a dataset's URIs changes, or when more URIs are added in a specific dataset. For a materialized approach, instance matching rules should be reconstructed and all the mappings containing the dataset that changed should be recreated. Regarding virtual integration, it is essential to update indexes or/and statistics, that are affected through such an evolution (e.g., the number of triples where a property occurs can be changed when more URIs are added). Finally, by storing in a warehouse the integrated content as a single integrated dataset, the evolution of a dataset can result to its whole reconstruction. For avoiding this time consuming process, one can store the datasets in different graphspaces (e.g., like in MarineTLO warehouse [236]) and update only the dataset(s) that changed.

Approaches. In [137], the authors provide an approach which allows query answering in virtual integration systems under evolving ontologies without recreating mappings between the mediator and the underlying sources. In particular, it can be achieved by rewriting queries between different ontology versions and then forwarding them to the underlying sources in order to be answered. Moreover, in [128] a "Dynamic Linked Data Observatory" is introduced for monitoring Linked Data over a specific period of time, whereas [76] proposes a function for measuring the dynamics of a specific dataset, while they list several approaches related to datasets' dynamics. *SPARQLES* [248] and *SpEnD* [258] monitor the health of hundreds of public SPARQL endpoints, by sending SPARQL queries at regular intervals, e.g., for checking the availability of each endpoint over time. Finally, incosistencies can occur in the *specificity* of ontological instance descriptions, when such descriptions are migrated to the up-to-date version of a given ontology [237].

2.4.5.4 Publishing an Integrated Dataset

Its objective is to publish an integrated dataset in order to be reused by others for several tasks (e.g., query evaluation, data analysis, etc.).

Context. A scientist/user can publish an integrated dataset in order to be findable and reusable by others. However, Linked Data are not so "open" for the consumers or the scientists to be reused, since in many cases they are published under a license. According to [100, 120], each RDF dataset should contain such a license in order the datasets to be reused under legal terms. Moreover, the license should be both machine-readable (e.g., mentioned in the VoID description of a dataset) and human-readable (e.g., mentioned in the documentation of the dataset). According to [100] a license should contain information for the permission that concern the reproduction, distribution, modification or redistribution. Therefore, in case of publishing an integrated dataset, the publishers should respect both the provenance and the licenses of the constituent datasets. Such information should be included in the metadata of the dataset and published along with the dataset. Moreover, the publisher of any dataset (e.g., integrated dataset) should use recommended standards and follow specific principles, such as Linked Data principles [42] and FAIR principles [254], for making their dataset more discoverable, reusable and readable from both "'humans and computers". In Figure 2.3, we can see some ways for publishing an integrated dataset in the warehouse approach.

Difficulties. Datasets are produced by several organizations in different licenses, places, schemas, formats, and so forth (difficulty a).

Recommended Standards for Data Publishing. There exists several W3C standards that are recommended to be used during the creation and before publishing any dataset (a single or an integrated one), for enhancing *Data Exchange* and *Data Integration*. The key standards can be divided in a) standards for creating links among a single dataset and other datasets (e.g., relationships between instances and ontologies), and b) standards for creating basic metadata for a dataset (e.g., description, name, language, provenance). Concerning a), it is important each dataset to contain links to (schema/instance) URIs of other datasets, by using standard vocabularies such as *OWL*, *RDF/RDFs*, *SKOS*[2] and so forth. Regarding b), the basic metadata of each dataset should be desribed by using standard vocabularies, such as *DCAT*[3], *schema.org*, *VoID*[4], *Dublin Core*[5], *FOAF*[6], *PROV*[7], etc. Indeed, there is an emerging trend of using such vocabularies in several cases, i.e., they are used by billions of *Web Pages* [184, 197] (mainly through Microdata or/and JSON-LD), by *Digital Libraries* [229] and by *Governments* [75], e.g., Open Government Data from European Union, Unites States, United Kingdom and others, usually through the creation of descriptions using schema.org or/and DCAT vocabulary. Furthermore, Google's dataset search engine collects and indexes datasets' metadata that have been expressed by using such ontologies [184]. Therefore, it is clearly of primary importance for any publisher to use these standards, for enabling data sharing, and for easing the integration of their dataset with other ones. Moreover, it is worth noting that *VoID* vocabulary can be exploited for deciding whether two or more datasets are worth to be integrated, since it can be used for describing metadata concerning the links that exist between two datasets. Finally, we should also mention the Shapes Constraint Language, i.e., *SHACL*[8], which is a language for "validating RDF graphs against a set of conditions", that can used for several purposes, such as for data cleaning before publishing the dataset.

Where to Publish Datasets. First, dataset catalogs such as *datahub* (`http://datahub.io/`) and *Zenodo* (`http://zenodo.org/`), offer services for uploading a dataset, metadata about the dataset and so forth. Second, a SPARQL endpoint can be exploited in order the dataset to be directly published and accessed by humans and machines (e.g., for query answering). In particular, there are several available tools offering query evaluation such as *Virtuoso* (`http://virtuoso.openlinksw.com/`), *BlazeGraph* (`http://www.blazegraph.com`), *AllegroGraph* (`http://www.franz.com/agraph/`) and *Stardog* (`http://www.stardog.`

[2]`http://www.w3.org/2004/02/skos/`

[3]`http://www.w3.org/TR/vocab-dcat/`

[4]`http://www.w3.org/TR/void/`

[5]`http://dublincore.org`

[6]`http://xmlns.com/foaf/spec/`

[7]`http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/`

[8]`http://www.w3.org/TR/shacl/`

`com/`) which has been succesfully used in *NASA MARS mission*[9].

2.4.6 Integration Tools

Table 2.4 lists a number of integration tools and categorizes them according to the dimensions of §2.3, i.e., the integrated method that they offer, their input and output types and what internal (e.g., instance matching) and auxiliary services (e.g., provenance) they support. Although these tools can support millions or even billions of triples, to the best of our knowledge, they have been tested by using a small number (below 20) of real or synthetic datasets (see the last two columns of Table 2.4). Materialized approaches have mainly been tested by using real datasets [136, 222, 243], while federated approaches have been evaluated by using benchmarks such as LargeRDFBench [212] (having billions of triples) and FedBench [220] (having millions of triples). Materialized approaches are difficult to scale up to a big number of datasets, since some steps require manual effort, e.g., defining and configuring matching/transformation rules. On the contrary, virtual integrated systems do not offer transformation and data fusion mechanisms and mainly rely on a common schema (or/and common URIs), therefore, conceptualization, naming and conflicts issues are difficult to be tackled. Finally, when a dataset evolves, some steps that possibly require manual effort (e.g., defining new matching rules) should be repeated for most integration tools.

2.5 Processes for Integration

In the previous sections we have factorized the integration problem to various dimensions and we have analyzed the approaches and methods for each one of these dimensions. Regarding *InternalServices* dimension, one question is what processes (i.e., sequence of steps) are usually followed. For instance, given two or more sets of triples to be integrated, does one start from ontology matching or from instance matching? Of course various processes could be followed according to the context. Below, we distinguish the main ones, each accompanied by a real-world case.

P1. (Processes for) Top-Level Ontology-based or Competency Query-based Integration: Here, the desired Level III requirements are specified either as competency queries and/or by providing the ontology or schema that the integrated view of the datasets should have. Then, the data of the individual datasets should be transformed (physically or virtually) according to the integrated schema. Indeed, information integration is traditionally done in the context of databases by reconciling data coming from different sources under a common schema. An analogous process for the case of the RDF data (that follows the materi-

[9]`https://cdn2.hubspot.net/hubfs/2820685/Assets/CaseStudies/Stardog_NasaCaseStudy.pdf`

Tool/ Frame-work	**Integration Substance**	**Data-set Types**	**Out-put Types**	**Transf-orma-tions**	**Schema Match-ing**	**Instance Match-ing**	**VQA**	**Prov-enance Levels**	**Qua-lity**	**Evol-ution**	**Tested |D|**	**Tested |T|**
LDIF [222]	Materialized	RDF	Any	LT	PD+OMT	PD+IMT	✗	CL,UVL,TL	DF	S-Aut.	1-9	B
ODCleanstore [135]	Materialized	RDF	Any	LT	PD+OMT	PD+IMT	✗	CL,UVL,TL	DF	S-Aut.	1-9	M
MatWare [243]	Materialized	RDF+O	Any	LT, FT	PD+OMT	PD+IMT	✗	CL,UVL,TL	Con.	S-Aut.	1-9	M
KARMA [136]	Materialized	RDF+O	Any	LT, FT	PD+OMT	PD	✗	CL,UVL,TL	DC	S-Aut.	1-9	B
FuhSen [62]	Hybrid	RDF+O	KS	FT	PD	PD+IMT	✓	UVL,QL	DF	S-Aut.	1-9	M
TopFed [216]	Hybrid	RDF	QA	LT, FT	PD+OMT	PD+IMT	✓	UVL,QL	QP	S-Aut.	10-19	B
RapidMinerLOD [208]	Hybrid	RDF+O	Any	LT, FT	PD+OMT	PD+IMT	✓	UVL,QL	DF	Aut.	1-9*	M
SQUIN [116]	Traversal	RDF	QA	✗	PD	PD+C	✓	UVL,QL	QP	Aut.	1-9*	M
SWGET [99]	Traversal	RDF	QA	✗	PD	PD+C	✓	UVL,QL	QP	Aut.	1-9*	M
Linked-Data-Fu [113]	Traversal	RDF+O	Any	✗	PD	PD+C	✓	UVL,QL	QP	Aut.	10-19*	M
SEMLAV [159]	Mediator	RDF	QA	✗	PD+OMT	PD	✓	UVL,QL	QP	S-Aut.	1-9	M
DaRQ [201]	Federated	RDF	QA	✗	PD	PD	✓	UVL,QL	QP	S-Aut.	10-19	M
Splendid [106]	Federated	RDF	QA	✗	PD	PD	✓	UVL,QL	QP	S-Aut.	10-19	B
HiBISCuS [214]	Federated	RDF	QA	✗	PD	PD	✓	UVL,QL	QP	S-Aut.	10-19	B
FedX [223]	Federated	RDF	QA	✗	PD	PD	✓	UVL,QL	QP	Aut.	10-19	B
ANAPSID [29]	Federated	RDF	QA	✗	PD	PD	✓	UVL,QL	QP	Aut.	10-19	B
DAW [215]	Federated	RDF	QA	✗	PD	PD	✓	UVL,QL	QP	S-Aut.	1-9	M
MULDER [88]	Federated	RDF	QA	✗	PD	PD	✓	UVL,QL	QP	S-Aut.	10-19	M

Table 2.4: Categorizing Existing RDF integration tools. O=Other Formats, Any=Any Service, QA=Query Answering, KS=Keyword Search, FT=Format Transformation, LT=Logical Transformation, PD=Predefined Mappings, OMT= Ontology Matching Techniques, C=Closure, IMT=Instance Matching Techniques, VQA=Virtual Query Answering, CL=Conceptual Level, UVL=URIs and Values Level, TL=Triples Level, QL=Query Level, DF=Data Fusion, Con.=Connectivity, DC=Data Cleaning, QP=Query Performance, Aut.=Automatically, S-Aut.=Semi-Automatically, —D—=Datasets, *discovers more datasets on-the-fly, —T—=Triples, M=Millions, B=Billions

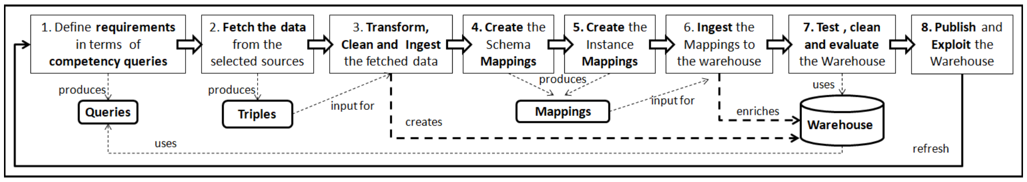

1. Define requirements in terms of competency queries
2. Fetch the data from the selected sources
3. Transform, Clean and Ingest the fetched data
4. Create the Schema Mappings
5. Create the Instance Mappings
6. Ingest the Mappings to the warehouse
7. Test , clean and evaluate the Warehouse
8. Publish and Exploit the Warehouse
produces
Queries
produces
Triples
input for
creates
produces
Mappings
input for
enriches
uses
Warehouse
uses
refresh

Figure 2.9: Top level - ontology based integration

alized approach), is the one that is followed by *LDIF* [222] and *ODCleanStore* [135], which are generic frameworks for integrating Linked Data. `MatWare` [243] follows a similar process and it has been used for building and maintaining real and operational warehouses, i.e., MarineTLO warehouse [236] and the ongoing GRSF warehouse [240]. In Figure 2.9, one can observe the process that is usually followed by such frameworks and tools. In general, we could say that the process starts with the specification of a kind of "integration template" that the data of the underlying datasets should "fill". This template can be specified by (competency) queries, schemas/ontologies or both.

P2. (Processes for) General purpose integration (fully automatic): In this case we do not

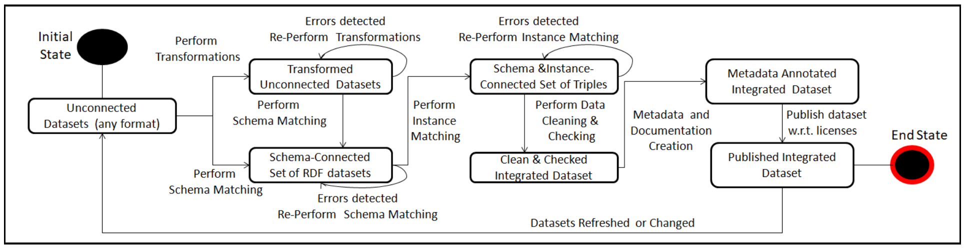

Initial State
Perform Transformations
Errors detected Re-Perform Transformations
Transformed Unconnected Datasets
Errors detected Re-Perform Instance Matching
Schema &Instance-Connected Set of Triples
Metadata Annotated Integrated Dataset
Unconnected Datasets (any format)
Perform Schema Matching
Perform Instance Matching
Perform Data Cleaning & Checking
Metadata and Documentation Creation
Publish dataset w.r.t. licenses
End State
Perform Schema Matching
Schema-Connected Set of RDF datasets
Clean & Checked Integrated Dataset
Published Integrated Dataset
Errors detected Re-Perform Schema Matching
Datasets Refreshed or Changed

Figure 2.10: Lifecycle of an integrated dataset

have an “integration template”, either because we are not aware about the contents of the underlying datasets, and/or because the integration does not have a specific purpose, i.e., we aim at building a general purpose integrated view of a set of datasets. Therefore, here we try to do the best that we can, and usually in such cases it is more difficult to guarantee that the integrated view satisfies the criteria of completeness, validity and accuracy. We can say that approaches and systems that fall in this category include, *LODLaundromat* [207], *LOD-a-Lot* [96], *LODsyndesis* [165, 171]. In these systems the process followed varies.

Specifically, *LODLaundromat* [207] starts by collecting URLs denoting dataset dumps and downloads the datasets by connecting to the hosting servers. Afterwards, it performs data cleaning by finding and correcting syntax errors, by replacing blank nodes with well-known URIs and by removing duplicate triples. Then, it stores the documents in a uniform serialization format (each document is stored in a different file), it creates the metadata for the underlying datasets and it publishes the data for being reused for various purposes in N-Triples and HDT format [97]. *LOD-a-Lot* [96] is a service that integrate all the documents of *LODLaundromat* (over 650K documents) into a single indexed HDT file [97] for easing the process of accessing and querying the full corpus as a whole. *LODsyndesis* [165, 171] starts by collecting hundreds of available datasets from online catalogs (such as datahub.io) and then it computes the transitive and symmetric closure of `owl:sameAs` relationships in order to find all the equivalent entities among the datasets. Afterwards, it creates indexes and performs measurements that are exploited for offering several services such as object coreference dataset discovery and selection, connectivity assessment and others. Figure 2.10 illustrates an indicative lifecycle model of an integrated dataset in the form of UML State Diagram. Each integration process can be conceived as a series of transitions between the illustrated states.

P3. Composite Processes: Composite processes are also possible. A composite process can be seen as a sequence of points of the multidimensional space that we defined. For example, suppose an integration process that comprises two subprocesses: the first aims at discovering and selecting the 10 most related datasets to one information need, and the second aims at fine-level integration of these 10 datasets of the first step. In general, workflow management systems could be exploited for specifying and enacting complex processes for data integration.

We should note that *Data Enrichment* is a special composite data integration process, since it involves dataset discovery and selection, and fine grained integration. In that case, the context is usually different since the main target of such approaches to produce an enriched dataset for feeding Machine-Learning algorithms, and not to provide query answering services. There are several tools performing data enrichment from several sources for improving Machine-Learning tasks, such as *RapidMiner Semantic Web Extension* [208], FeGeLOD [193], `LODsyndesis`$_{\mathcal{ML}}$ [166] and *LODVec* [172]. We will further elaborate on this topic in Chapter 6.

Remark about evaluation. It is easier to evaluate how successful a P1 process was. It is harder for P2 processes, and even harder for P3 processes.

2.6 Evaluation of Integration

How an integration approach can be evaluated? In general, we could identify two basic evaluation aspects, *quality* and *cost* (the latter includes the human effort required), and ideally we would like optimal quality and fully automatic (and cost-effective) integration processes. Of course an integration process comprises various steps, each one could be evaluated independently of the others, and in §2.4 we referred to such benchmarks. However, there is a need for evaluating also the *overall process* and its outcome, and for this reason below we discuss how each kind of services (that can be delivered through an integration process), either fine-grained or coarse-grained (as distinguished in §2.4.2), can be evaluated.

Evaluation of Fine-grained Services. Here, we describe ways and we introduce evaluation collections concerning the fine-grained services.

• **FG: Level I.** For offering complete *Global URI Lookup Services*, i.e., for finding all the datasets and the equivalent URIs of a URI u, it is important to produce high quality mappings among URIs and to offer cross-dataset completion (see §2.4.2.1). Incorrect `owl:sameAs` mappings result to low quality services, i.e., URIs referring to different entities are considered as equivalent. The quality of mappings is mainly evaluated by using metrics such as precision and recall [109], whereas cross-dataset completion is evaluated through connectivity metrics [163, 165]. Concerning the cost, several tools automate such processes and produce mappings having high precision (see §2.4.4.2 and §2.4.4.3), however they are not free of errors. Consequently, human effort is required for interpreting the results of the measurements and the produced mappings. For this reason, crowdsourcing approaches are used for improving the precision and recall of matching process [28, 30, 71, 139, 217].

• **FG: Level II.** The evaluation of *Global Keyword Search Services* relates more to *Information Retrieval techniques.* In particular, a straightforward way for evaluating such a service is to use reference collections [221]. LUBM benchmark [110] contains an Concerning *Virtual Integration Systems*, there exists benchmarks like *FedBench* [220], LargeRDFBench, SP^{2}Bench and others [212], which cover several dimensions such as the result completeness, ranking of the returned answers and efficiency (e.g., execution time). evaluation collection with keyword queries (and their corresponding answer) that can be evaluated over RDF datasets. INEX Linked Data track provides a collection, which contains Wikipedia articles, and *DBpedia* and *YAGO* links (and triples) for the entities that occur in each article [86]. The main task of that track was to find the most relevant articles for a keyword query. Finally, in *SemSearch* workshops the challenge was to evaluate semantic web *Keyword Search* engines at large scale, by using the "Billion Triples Challenge 2009" dataset, which contains 1.4 billion of triples [86].

• **FG: Level III.** The evaluation of *Integrated Query Answering Services* differs from the previous level, since the input for such an evaluation is a query expressed in a structured query language. One way to evaluate such an approach, is to predefine some competency queries [163] (MarineTLO warehouse [236] and GSRF warehouse [240] are evaluated in this way). "A competency query is a query that is useful for the community at hand, e.g., for a human member (e.g., a scientist)" [163], and sketches the desired scope and structuring of the information before the creation of the integrated system. For evaluating correctly each competency query, the expected results should be predefined. However, this type of evaluation requires human effort for defining the requirements of the integrated access system. Concerning *Virtual Integration Systems*, in §2.4.4.4 we mentioned several benchmarks for evaluating such systems. General question answering benchmarks are also used for testing such services, e.g., *Free917* [49] is an open question answering benchmark containing 917 natural language questions which can be answered by using the content of Freebase dataset. Moreover, *QALD* [147] is an evaluation series campaigns on question answering over Linked Data. The major challenge for the participants, is to use as input several RDF datasets and natural language questions, for returning the desired answers or a SPARQL query that is able to produce the correct answers. *QALD* is used for evaluating several aspects such as *multilinguality*, *large-scale question answering* and others, while for each aspect it contains several questions and tasks. Finally, *LC-QuAD* [235] is a corpus containing 5,000 questions for testing several different cases, while it provides the corresponding SPARQL queries which are essential for answering questions over DBpedia [142]. Finally, a framework is provided for generating natural language questions and their corresponding SPARQL queries for reducing the manual effort.

Evaluation of Coarse-grained Services. Concerning *Dataset Discovery & Selection Services*, the results of such services for a specific integration task, are usually evaluated by measuring the marginal gain of the selected sources [83], or/and by measuring the level of connectivity among any set of datasets [168], i.e., for evaluating whether the selected datasets contain information for the same entities. Additional quality dimensions can be

Tool/Service	Total Triples	Include > Datasets	Global URI Lookup	Dataset Discovery	Dataset Visualization	Connectivity	Fetching Transforming	Keyword Search	Dataset Analysis	Querying Datasets	Dataset Evolution
LODsyndesis [165]	2 Bil.	400	✓	✓	✓	✓					
LODLaundromat [207]	38 Bil.	>650,000*	✓	✓			✓	✓			
LOD-a-lot [96]	28 Bil.	>650,000*					✓		✓	✓	
LODStats [91]	130 Bil.	9,960		✓					✓		
Datahub.io	Unk.	>1,270		✓			✓		✓		
LinkLion [179]	77 Mil.	476		✓		✓	✓				
DyLDO [128]	Unk.	86,696*									✓
LODCache	4 Bil.	346						✓		✓	
LODCloud [219]	Unk.	1,239			✓	✓	✓		✓		
sameAs.org [105]	Unk.	>100	✓								
WIMU [246]	Unk.	>650,000*	✓				✓				
LOV [247]	Unk.	637**	✓					✓		✓	
Linghub [150]	Unk.	272		✓				✓		✓	
SPARQLES [248]	Unk.	557		✓	✓						✓
SpEnD [258]	Unk.	1,487		✓	✓						✓

Table 2.5: Existing Services for large numbers of RDF datasets, *Documents, **Vocabularies, Mil.=Million, Bil.=Billion, Unk.=Unknown

considered, such as timeliness, accuracy, coverage and efficiency [205], as well as constraints such as the cost of integration (e.g., in terms of money amount for private datasets).

2.7 Semantic Integration On a Large Scale

At first, in §2.7.1 we list services for integrating large in number RDF datasets, while in §2.7.2 we show some successful paradigms of integration on a large scale.

2.7.1 Services for Large in Number RDF Datasets

Table 2.5 compares the running LOD cloud tools/services that contain information about over a hundred LOD datasets, according to the services that they provide (each service can be exploited for different dimensions of Data Integration), while each tool/service is described in brief, below. Please note that `LODsyndesis` is the suite of services and tools that have been developed in this dissertation.

LODsyndesis [165, 168, 171] offers services for object coreference, i.e., find the datasets, the equivalent URIs and all the triples of a given entity [168], services for content-based dataset discovery and selection, and services for performing connectivity analytics for large numbers of RDF datasets.

LODLaundromat [207] (`http://lodlaundromat.org`) is a set of services for fetching, transforming, cleaning, compressing and querying 650,000 RDF documents. By using this service, one can search for specific URIs, while it provides a keyword search engine (called

Lotus [122]), where one can find all the triples and URIs where a specific keyword occurs.

LOD-a-Lot [96] (`http://lod-a-lot.lod.labs.vu.nl/`) has integrated the set of *LOD-Laundromat* documents in a single self-indexed file in HDT format [97]. This integrated dataset can be used for query resolution at Web scale.

LODStats [91] (`http://stats.lod2.eu/`) is a service including some basic metadata for over 9,000 RDF datasets. For example, one can find the number of datasets' triples, or the languages that each dataset uses, while one can find all the datasets for specific schema elements (i.e., properties and classes).

Datahub.io (`http://datahub.io`) is a portal that provides several datasets in different formats, and one can find some major statistics for each dataset (e.g., number of triples, number of links to other datasets). Moreover, one can fetch datasets provided in dumps and get updates from datasets.

LinkLion [179] (`http://linklion.org/portal`) contains dumps of mappings between pairs of datasets and statistics about them.

DyLDO [128] (`http://km.aifb.kit.edu/projects/dyldo/`) is a Dynamic Linked Data Observatory whose target is to monitor the evolution of Linked Datasets over time. This service collects frequent, continuous snapshots of a subset of RDF documents and provides interesting insights in how Linked Data evolve over time.

LODCache (`http://lod.openlinksw.com`) is a SPARQL endpoint service, based on Virtuoso database engine [89], that includes several datasets and billions of triples. By using this service, one can send queries that can be answered from two or more datasets, while one can type queries for measuring the connectivity among several datasets. However, in that case the closure of equivalence relations is performed on query time which can be time-consuming (`http://docs.openlinksw.com/virtuoso/rdfsameas/`).

LOD Cloud [219] provides statistics about the datasets and the domains where they belong, while one can see all the connections between pairs of datasets through the popular *LOD Cloud Diagram* (`http://lod-cloud.net`). Moreover, the datasets can be fetched through that webpage.

SameAs.org [105] (`http://sameas.org`) is a URI lookup service containing over 203 million URIs, where one can find all the equivalent URIs for a given URI.

WIMU [246] (`http://w3id.org/where-is-my-uri/`) is a service that uses the datasets of *LODLaundromat* and *LODstats*, and one can search for a specific URI. This service returns a ranked list of documents, where a particular URI occurs. Moreover, one can download the documents containing that URI, or/and the triples for a URI from a specific document.

LOV (Linked Open Vocabularies) [247] (`http://lov.okfn.org`) is a service containing over 600 vocabularies from different datasets. It offers a keyword search and a SPARQL endpoint, which contains schema triples and it can be used in order to search for schema elements.

Linghub [150] (`http://linghub.lider-project.eu`) is a portal that collects thousands of datasets from linguistic domain (272 of them in RDF format), and uses the Linked Data Principles for displaying the metadata of datasets. This portal offers a faceted browsing service, a keyword search service and a SPARQL endpoint, for easing the process of discovering datasets according to different users' requirements.

SPARQLES [248] (`http://sparqles.ai.wu.ac.at`) and *SpEnD* [258] (`http://wis.etu.edu.tr/spend/`) are services containing metadata and statistics for hundreds of SPARQL endpoints.

2.7.2 Integration Cases on a Large Scale

Here, we mention some real, and noteworthy, cases of integration in a large scale, and we divide them in domain-specific and domain-independent cases. These cases either contain large number of datasets, or the number of datasets is not big but the integrated dataset is itself big (and popular).

2.7.2.1 Domain-Specific Integration Use Cases.

Here, we introduce use cases of successful data integration that have been applied for a specific domain. *Europeana* [124] is a digital library for cultural heritage combines data from than 3,000 institutions across Europe while these data were transformed into Linked Data (over 1.8 billion of triples). One can query the integrated content by using SPARQL queries. *OCLC* (`http://www.oclc.org`) is a "global library cooperative" that supports data from thousands of libraries. *OCLC* has developed *WorldCat* (`http://worldcat.org`) which contains 2.7 billion records with bibliographic metadata in Linked Data format. For making the data more findable by search engines (e.g., Google Dataset Search [184]) and reusable from other users, data are expressed by using standard formats and vocabularies, such as *RDFa* and *schema.org*. *PhLeGrA* [129] has integrated data from four heterogeneous large scale biomedical datasets. The authors analyzed the integrated graph for discovering associations between drugs, and they used that graph and machine learning techniques for improving the accuracy of predictions of adverse drug reactions. *Bio2RDF* is "the largest network of Linked Data for Life Sciences" [50]. It contains approximately 11 billion triples from 35 datasets, and it can execute federated queries among the underlying sources, since it creates mappings between the ontology of each dataset and a global schema, called *Semanticscience Integrated Ontology (SIO).* Finally, *Open Phacts* [107] has integrated over 3 billion triples from approximately 20 datasets containing information about drugs, and it exploits a mediator mechanism for supporting complex SPARQL queries.

2.7.2.2 Domain Independent Integration Use Cases.

Here, we introduce domain-independent use cases of successful data integration. *Google Knowledge Graph* contains over 18 billions of facts for over 570 million entities [79], and exploits Semantic Web technologies for integrating information for any domain. It integrates both semi-structured data (e.g., from organizations and companies) and data from millions of HTML web pages that are mainly expressed in standard formats and vocabularies, such as *JSON-LD*, *Microdata* and *schema.org*. Except for *Google Knowledge Graph*, *Google* uses similar mechanisms for providing integration for specific domains, such as for *Shopping* (`http://developers.google.com/search/docs/data-types/product`), for providing *Maps* (`http://developers.google.com/search/docs/data-types/local-business`) and others. *Wikidata* [251] is a free and open knowledge base that contains over 26 millions of entities and it is readable and editable by both humans and machines. It contains data in over 350 different languages, while it mainly integrates data from Wikipedia (different languages) and users' data. *Freebase* [46] is a former knowledge base which integrated data from four sources: Wikipedia, MusicBrainz (`http://musicbrainz.org`), FMD (`http://www.fashionmodeldirectory.com`) and the dataset of NNDB (`http://www.nndb.com`), and contained approximately 2 billions of triples. The users had the opportunity to contribute to the knowledge base by editing structured data. *YAGO* [204] is a multilingual knowledge base that integrates data from Wikipedia, Wordnet (`http://wordnet-rdf.princeton.edu/`), and Geonames (`http://www.geonames.org`). It includes over 1 billions of triples for approximately 17 millions of entities. It is constructed through information extraction and merging and not by community effort. *DBpedia* [142] is a knowledge base that integrates multilingual data extracted from Wikipedia (`http://www.wikipedia.org`) in 125 languages. It contains over 3 billions of triples, and describe over 38 millions of things. For easing the integration process with other sources, it contains large number of links to other datasets in the LOD cloud. In a comparative survey for the aforementioned knowledge bases [95], the authors defined 35 aspects according to which knowledge bases can be analyzed, while in [94] they analyzed the quality of these knowledge bases in several dimensions.

2.8 Discussion

The presented integration tools (see §2.4.6) have been applied over a small number of datasets (even for billions of data), however, they cannot easily scale up to large number of datasets, since manual effort is usually needed for defining/configuring transformation rules, matching rules, and other tasks. Moreover, virtual integration tools do not offer conflicts resolution mechanisms, and most of them rely on the usage of a common schema or/and URIs, thereby, issues, such as different naming and conceptualizations, cannot be fully tackled. Apart from that, the evolution of the underlying datasets cannot be tackled straightforwardly by most integration approaches, i.e., some steps with manual effort should be repeated (e.g., mappings between datasets) for applying such changes.

For tackling the integration difficulties (presented in §2.2) that cannot be solved at a large scale by the integration tools of §2.4.6, there is a recent trend for services for large in number RDF datasets (see §2.7.1), each focusing on one aspect of the problem. For instance, *LODLaundromat* [207] faces the problem of different *formats* (and syntactic errors), by fetching and transforming thousands of documents, *LODsyndesis* [168] tackles the problem of *naming*, by finding all the equivalent URIs of instances and schema elements of billion of data, while *SpEnD* [258] and *SPARQLES* [248] check the *evolution* of datasets by monitoring a large number of online SPARQL endpoints. Moreover, due to the large volume of available data, we have observed a) a proliferation of dataset discovery and selection approaches (e.g., [33, 87, 143, 165]), since it is important to select the most valuable datasets for integration [165, 205], b) the exploitation of the available knowledge and the usage of external resources, such as predefined equivalence relationships [165,168,171], web pages [103], lexicons and translators [57,65,77,127,180,185], and knowledge bases [79], for tackling quality and matching issues, c) the exploitation of novel machine learning-based methods for identifying similarities [160,210], and d) the increase of crowdsourcing approaches [28, 30, 71, 139, 217] for identifying and solving errors/conflicts. However, we are still far from achieving full semantic integration at large scale [84], and we should also take into account the evolving requirements for "Web semantics" [38].

Synopsis. Overall, a combination of the a) above state-of-the-art methods (e.g., machine learning and crowdsourcing approaches), and b) the usage of as much as possible available knowledge on the web (e.g., knowledge bases, web pages), is required for achieving integration at large scale. Successful integration examples include i) domain specific use cases, such as *Europeana* [124] and *OCLC* from cultural heritage domain, and *Bio2RDF* [50], *Open Phacts* [107] and *PhLeGrA* [129] from life science domain, and ii) domain independent use cases, i.e., *Google Knowledge graph* [79], *DBpedia* [142], *Wikidata* [251], *YAGO* [204] and *Freebase* [46]. Indicatively, *Google Knowledge graph* integrates data from millions of annotated web pages and semi-structured data, for offering information for billions of facts. Moreover, Wikidata [251] can be also considered as a successful integration example; it combines knowledge from several datasets (e.g., Wikipedia) and it can be edited from both humans and machines, and this facilitates its evolution and the correction of errors. In general, success stories of data integration at large scale in *Semantic Web* mainly involve centralized knowledge bases, however, there are proposals for more effective decentralization [199]. Overall, it seems that hybrid approaches, that combine various sources of evidence, are more likely to drive to effective, and sustainable, large scale data integration.

Research Gaps. In Chapter 3, we present the major research gaps which are derived by the analysis of the presented survey and how our approach tries to tackle them. These research gaps correspond to the tasks A-E, where we also introduce several motivation scenarios about these tasks and we describe the main differences of our thesis comparing to the related approaches.

Chapter 3

Research Gaps & Motivating Scenarios

In this chapter, we introduce the major research gaps, derived by the analysis of Chapter 2, whereas we describe several motivating scenarios (for the tasks A-E) and the novelty of this thesis. Moreover, in Table 3.1 we introduce some key indicative queries that we desire to answer (for the tasks A-E). In particular, in §3.1, we describe the placement of this dissertation to the Data Integration Landscape (see Chapter 2). In §3.2-§3.6, for each of the tasks A-E, we introduce the major research gaps, motivating scenarios and the novelty of this dissertation comparing to the existing approaches. Finally, in §3.7, we describe in brief the process which will be followed for achieving the targets of this dissertation.

3.1 Placement of dissertation in the Data Integration Landscape

Here, we describe the placement of this dissertation in the Data Integration Landscape (see §2.3). Concerning the dimension of *Dataset Types*, the input of our approach is a large number of *RDF* datasets. As regards the other dimensions, our work predominantly belongs to the dimensions of *Basic Services to Deliver*, since the objective is to provide advanced services belonging to both *Coarse-grained* (i.e., *Dataset Discovery & Selection*) and *Fine Grained*, such as URI lookup services and *Question Answering* services (e.g., related to Machine-Learning tasks). Moreover, it belongs to the dimension of *Auxiliary* services, since the proposed methods can be exploited for *Quality Assessment* and for *Evolution & Monitoring* (e.g., through connectivity analytics). Finally, the proposed work secondarily

Table 3.1: Indicative Queries for the tasks A-E

Query Name	Tasks	Query Description
$Q_{objectCor}$	A	"Give me all the equivalent URIs of a given URI u (e.g., "http://www.dbpedia.org/resource/Aristotle")"
Q_{prov}	A	"Give me all the datasets containing information for a given entity (or property, literal, etc.) (e.g., Aristotle)"
$Q_{allFacts}$	A	"Give me all the triples (and their provenance) of a given URI u"
$Q_{connectivity}$	B,C	"Give me the K datasets having the most common entities (or triples, literals, etc.)"
$Q_{datConnectivity}$	C,E	"Give me the K most connected datasets to my dataset (according to the number of entities, triples, literals, etc.)"
$Q_{coverage}$	C	"Give me the K datasets that maximize the information (i.e., triples) for an entity (or a set of entities)"
$Q_{enrichment}$	C,D	"Give me the K datasets that contain the most complementary information (e.g., maximum number of triples) for the entities of my dataset"
$Q_{uniqueness}$	C,E	"Give me the K datasets, that my dataset contributes the most (or less) unique content"
$Q_{veracity}$	A,E	"Compare all the different values (and their provenance) for a specific fact (e.g., the birth date of Aristotle)"
$Q_{datQuality}$	C,E	"How unique/redundant is the content of my dataset comparing to others?"
$Q_{datReuse}$	C,E	"Does my dataset enrich the content of other datasets?"

belongs to the dimension of *Internal Services* (i.e., for *Schema/Instance Matching*), since we perform cross-dataset identity reasoning for both instance and schema elements.

3.2 Task A. Object Coreference & All Facts about an Entity

This task belongs to the *Fine Grained* services (see §2.4.2), since the objective is to find, select and assemble "pieces" of data. Our target is to be able to answer the queries $Q_{objectCor}$, Q_{prov}, $Q_{allFacts}$ and $Q_{veracity}$ (see Table 3.1) for any given entity (URI) at global scale.

3.2.1 Research Gaps

As we have seen in §2.7.1, there exists global scale services offering object coreference, such as LODLaundromat [207] and *WIMU* [246], where one can find all the datasets (and documents) containing a specific URI. However, one is not able to find all the equivalent URIs and triples for a given URI or/and schema element. Furthermore, there exist approaches, such as *sameAs.org* [105], where one can find the equivalent URIs of a given one, however, they do not provide such functionality for schema elements and do not collect all the available information for a given entity.

3.2.2 Motivating Scenarios

Suppose a scenario where one user or an application wants to find all the available data and their provenance associated with "http://www.dbpedia.org/resource/Aristotle", including URIs being `owl:sameAs` with this entity (i.e., object coreference), coming from multiple sources. For making it feasible, one should compute the transitive and symmetric closure of `owl:sameAs` relationships. Except for the entities, the publishers tend to use also different URIs for representing schema elements (i.e., properties and classes). For instance for the fact "Which was the birth date of Aristotle?", different datasets can use different URIs to represent the entity "Aristotle" and the schema element "birth date". Therefore, for collecting all the available data (and their provenance) for an entity and for not missing facts that occur in two or more datasets, it is also mandatory to compute the transitive and symmetric closure for the corresponding equivalence relationships in schema level, i.e., `owl:equivalentProperty` and `owl:equivalentClass`.

3.2.3 Our placement

To the best of our knowledge, this is the first work that computes the transitive and symmetric closure in both instance and schema level for large number of linked datasets. Therefore, by using the methods which are proposed in this dissertation, we are able to provide all the equivalent URIs and all the available information for any entity. Moreover, the authors in [51, 53, 256] follow a virtual approach for answering SPARQL queries over relational databases, and query rewriting techniques for keeping a single IRI (or identifier) for each real entity. On the contrary, we use a set of RDF datasets (and not relational databases), and we follow a "materialized-based" approach for collecting all the data for a given entity, since we construct and store several semantics-aware indexes.

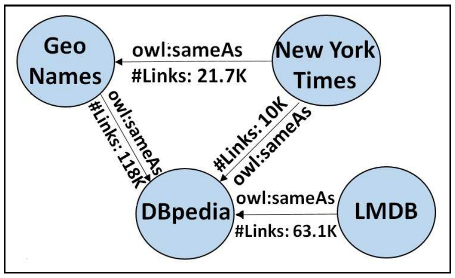


Figure 3.1: LargeRDFBench Cross Domain Datasets

3.3 Task B. Connectivity Analytics

This task belongs to the *Auxiliary Services* (see §2.4.5), i.e., it can be used for measuring how connected the datasets are. Moreover, it also belongs to the *Coarse Grained* services (i.e., see §2.4.2.4), because connectivity analytics can be used for finding the most connected datasets to a given one. The objective is to be able to answer queries, such as $Q_{connectivity}$ (see Table 3.1), for any subset of datasets.

3.3.1 Research Gaps

There are only few approaches that offer global scale measurements for large number of linked datasets (see §2.7.1). Moreover, current global scale services do not provide measurements among more than two datasets (see Table 2.5), although such measurements are useful for several tasks (e.g., connectivity analytics). Indeed, the community uses catalogs that contain some very basic metadata, and diagrams like the Linking Open Data cloud diagram[1], as well as LargeRDFBench[2] (an excerpt is shown in Figure 3.1). 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, Literal, and so forth.

3.3.2 Motivating Scenarios

The target is to compute connectivity measurements that involve more than two datasets. The results can be visualized as Lattices, like that of Figure 3.2 which shows the lattice of

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

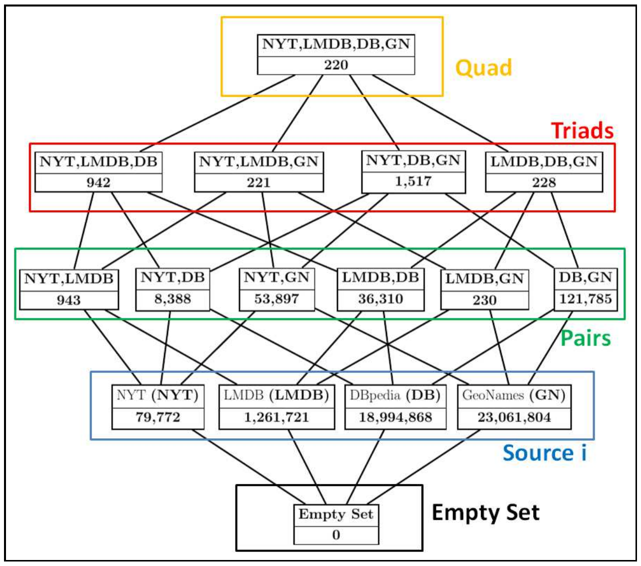


Figure 3.2: Lattice of four datasets showing the common Real World Objects

the four datasets of Figure 3.1. From this lattice one can see the number of common real world entities in the triads of datasets, e.g. it is evident that the triad of *DBpedia*, *GeoNames* and *NYT* shares 1,517 real world objects, and there are 220 real world objects shared in all four datasets. Instead, the classical visualizations of the LOD cloud, like that of Figure 3.1, stop at the level of pairs, therefore it is not an easy task to estimate how connected Linked Datasets are and to assess their quality.

3.3.3 Our Placement

This is the first work that provides connectivity measurements among any subset of linked datasets (and not only between pairs of datasets as in [33, 143, 179, 207, 252]). Moreover, even for measurements between pairs of datasets, this is the first work that takes into account the transitive and symmetric closure of equivalence relationships (for not missing connections).

3.4 Task C. Dataset Search, Discovery & Selection

This task belong to the *Coarse Grained* services (i.e., see §2.4.2.4), since its major target is to find or select entire datasets. As we have discussed in Chapter 2, dataset search and discovery are the first steps of an integration process and it is really difficult to find whether there are related datasets for a given application/task. Therefore, the objective is to be able to answer queries such as $Q_{datConnectivity}$, $Q_{coverage}$,$Q_{enrichment}$ and $Q_{uniqueness}$.

3.4.1 Research Gaps

Current methods for dataset search and discovery exploit metadata (see §2.4.2.4) for aiding the discovery of datasets. Such metadata usually describe generic information about a dataset, like its description, a set of keywords, etc. By using services such as *Google Dataset search* (`http://toolbox.google.com/datasetsearch`) or *datahub.io*, one can find relevant datasets according to some keywords. In particular, they show to the users several datasets according to a ranking, however, the user does not know whether these datasets are worth to be combined. Another drawback of these approaches is that they contain several datasets with unstructured data, which makes that problem even harder, i.e., it is not easy to find all the datasets containing information about the same entities.

The problem of unstructured data can be alleviated by adopting *Linked Data*, since linking can be achieved through common URIs or/and through equivalence relationships (e.g., `owl:sameAs` relationships). As we have seen in §2.4.2.4 and §2.7.1, existing Linked Data content-based approaches, such as LODStats [91] and LODLaundromat [207], offer services for searching and discovering datasets containing specific URIs or/and keywords, while SPARQLES [248] and SpEnD [258] exploit metadata for searching for SPARQL Endpoints. However, the main target of these approaches is to offer statistics, preservation and quality assessment services, i.e., they do not focus on assessing the *quality among combinations of datasets*. Thereby, it is impossible to get back hits each corresponding to a set (or "cluster") of datasets.

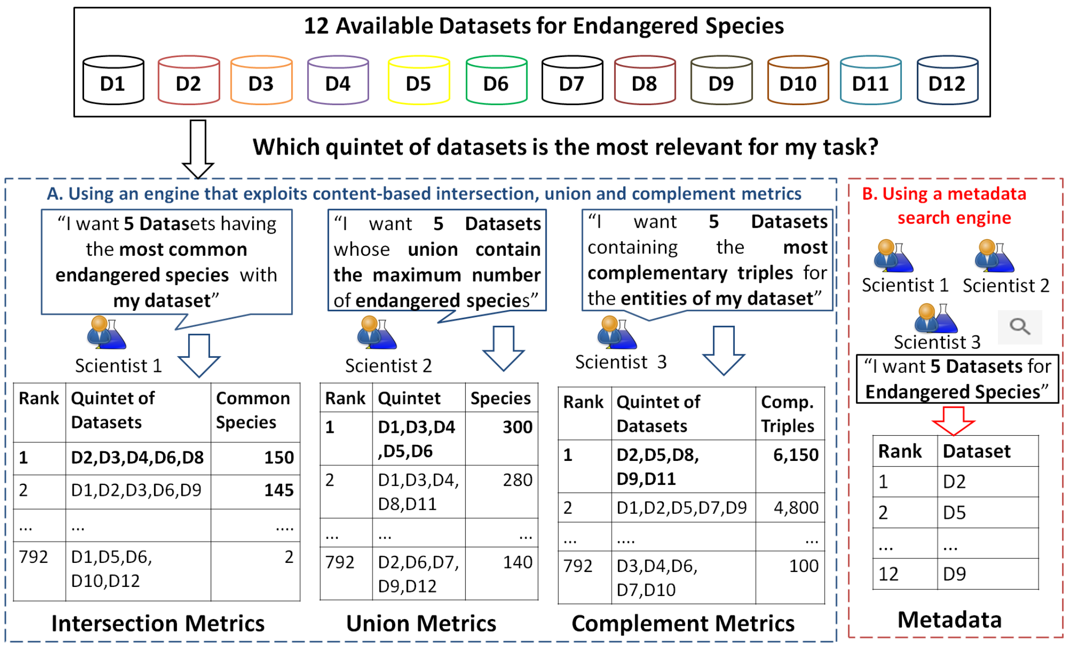
Moreover, a key problem is that current approaches do not take into account the cross-dataset reasoning among datasets (see §3.2) for discovering relevant ones, therefore connections can be missed. Another problem is that "the reuse and take-up of rich data sources is often limited and focused on a few well-known and established RDF datasets" [34]. Thereby, a key challenge is how to reveal the importance of high quality under-recognized datasets. For example, suppose that an under-recognized dataset, called D_{Tiger}, contains valuable information only for the species "Tiger". However, since the famous RDF datasets, e.g., DBpedia [142], Wikidata [251], YAGO [204], contain information about this species, in a global dataset ranking, the dataset D_{Tiger} would be in a very low position comparing to the popular datasets, even if the desired task is to collect information about "Tiger".

3.4.2 Motivating Scenarios

The target is to assist the task of Dataset Search, Discovery and Selection by offering content-based services which are based on intersection, union and complement metrics. In particular, we desire to propose measurements that can be directly used for answering *Dataset-based* queries such as "find the K datasets that are more connected to a particular dataset" or/and "find the K datasets that maximize the information of a given dataset". Such queries are important for finding related datasets containing the same entities and/or triples, literals and schema elements, either for constructing a warehouse or for mediator-based query answering. Moreover, they can be used for answering *Entity-based* queries, i.e., "Give me the K datasets containing the maximum number of triples for a set of entities".

For being able to provide such services, it is crucial not to miss connections between the datasets, i.e., we should compute the cross-dataset identity reasoning. For instance, suppose that you publish a dataset and for connecting it to the rest datasets of the LOD cloud you establish relationships with DBpedia (by having triples that refer to URIs from DBpedia). Without the computation of closure, 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 equivalence 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 (or triples, literals, etc.) with your dataset.

As a motivating example, suppose that in Figure 3.3 three scientists want to perform an analysis for endangered species, i.e., for predicting the possibility those species to be-


12 Available Datasets for Endangered Species
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
Which quintet of datasets is the most relevant for my task?
A. Using an engine that exploits content-based intersection, union and complement metrics
"I want 5 Datasets having the most common endangered species with my dataset"
Scientist 1
"I want 5 Datasets whose union contain the maximum number of endangered species"
Scientist 2
"I want 5 Datasets containing the most complementary triples for the entities of my dataset"
Scientist 3
Intersection Metrics
Union Metrics
Complement Metrics
B. Using a metadata search engine
Scientist 1
Scientist 2
Scientist 3
"I want 5 Datasets for Endangered Species"
Metadata

Rank	Quintet of Datasets	Common Species
1	D2,D3,D4,D6,D8	150
2	D1,D2,D3,D6,D9	145
...	...	....
792	D1,D5,D6, D10,D12	2

Rank	Quintet	Species
1	D1,D3,D4 ,D5,D6	300
2	D1,D3,D4, D8,D11	280
...	...	...
792	D2,D6,D7, D9,D12	140

Rank	Quintet of Datasets	Comp. Triples
1	D2,D5,D8, D9,D11	6,150
2	D1,D2,D5,D7,D9	4,800
...	....	...
792	D3,D4,D6, D7,D10	100

Rank	Dataset
1	D2
2	D5
...	...
12	D9

Figure 3.3: Motivating Example. Three scientists want to find K datasets (say 5) from the 12 available ones, for performing an analysis for endangered species.

come extinct in the current century. For this reason, they desire to search and integrate at least K (say five) different datasets containing information about such species, from the 12 available datasets for these species. However, each of these scientists has different requirements (i.e., see their queries in the left side of Figure 3.3).

In the first case, the scientist wants to enrich the content of his dataset (say D_j), however, his requirement is the five desired datasets to contain the most common endangered species with D_j. The second scientist has not collected any data, and he desires to discover the five datasets, whose union will produce an integrated dataset that will contain the maximum number of endangered species, i.e., comparing to any other combination of five datasets. In the third case, the scientist has already created a dataset, say D_i, containing information about several endangered species, and he desires to find five more datasets for enriching the information for the same species of D_i, i.e., by finding complementary information for those species. In that example, since there are available 12 different related datasets, which result to 792 possible quintets of datasets, thereby, it is time-consuming for the scientists to check all the possible quintets for discovering the best one for them.

The above user needs should be taken into account, however, they are not covered from existing approaches that exploit metadata (see §2.4.2.4), e.g., in Figure 3.3 (right side) such methods return the same ranking list of single datasets for all three scientists.

On the contrary, in Figure 3.3, an engine that computes such (intersection, union and complement) metrics, returns a ranking of quintets of datasets (instead of a ranking of single datasets), while the ranking is different for each scientist, according to their requirements, e.g., for *Scientist 2* the best quintet of datasets contains $\{D_1, D_3, D_4, D_5, D_6\}$, while for *Scientist 3*, the five best datasets for his needs are $\{D_2, D_5, D_8, D_9, D_{11}\}$. Therefore, we can obviously see that for different users (and even for the same task), (i) different combinations of datasets and (ii) different "slices" of a specific dataset can have different quality and value, e.g., even a small "slice" of an under-recognized dataset can be of primary importance for being used in a specific application.

3.4.3 Our Placement

To the best of our knowledge, this is the first work proposing content-based intersection, union and complement metrics for returning ranking lists of multiple datasets (instead of single datasets or pairs of datasets as in [33, 143, 179, 207, 252]), which is quite important, since "Many tasks involving datasets require the stitching of several datasets to create a whole that is fit for purpose" [56]. Comparing to other dataset discovery approaches, we exploit the contents of datasets (and not metadata, e.g., as in [198, 248, 258]). Comparing to content-based approaches [207, 246, 252], we use indexes enriched with inferred equivalence relationships.

3.5 Task D. Data Enrichment

This task mainly belongs to the composite services (see §2.5), since it involves dataset discovery and selection, and fine grained integration. The main objective is exploit "pieces" of data, for enriching the content of a given dataset. The objective is to be able to answer queries such as $Q_{enrichment}$ (see Table 3.1), and to provide data enrichment services that can be exploited for machine learning tasks.

3.5.1 Research Gaps

For enriching the data for a given entity (or all the entities of a given dataset), it is a prerequisite to perform cross-dataset identity reasoning, however it is not supported from current global scale approaches (see §2.7.1). Moreover, as we will explain in Chapter 6, there are not available data enrichment tools for Machine-Learning tasks, which combine data from hundreds of datasets simultaneously, by taking into account the cross-dataset identity reasoning.

3.5.2 Motivating Scenarios

It is important to collect information about the same real world entities from several sources in order to "widen" the information for the entities of interest. Technically, this reduces to constructing a dataset with high "pluralism factor", i.e., a consolidated dataset where the number of datasets that provide information about each entity is high. An indicative query for discovering relevant datasets for such a scenario follows: "find the K datasets that maximize the pluralism factor of the entities of a particular dataset".

Moreover, by collecting all the available data for an entity, we can exploit that functionality in the context of Machine Learning tasks, specifically for finding more features or/and for creating word embeddings about a given set of entities from many datasets, simultaneously. The target is to create a new enriched dataset, that can exploited for improving machine learning tasks. Below, we mention a possible use case for the cultural domain. Suppose that one wants to classify a set of paintings according to their genre [228], e.g., Impressionist, Renaissance and others, by using a machine learning algorithm. However, there are either few or even no available information for these entities, therefore, one should search on the web for those paintings to create more features and to perform cross-dataset reasoning for collecting data from multiple sources. Such a process can be time-consuming, while the discovered data often should be transformed before being used in a Machine Learning task. On the contrary, by collecting all the data for a given entity and for offering mechanisms for creating features and word embeddings automatically, such a process can be much easier.

3.5.3 Our Placement

We propose queries that can aid the publishers of datasets to find other datasets that can enrich their content (i.e., such as $Q_{enrichment}$), which are not covered from related approaches (see §3.4). Moreover, we propose two tools, i.e., `LODsyndesis`$_{\mathcal{ML}}$ [166] and `LODVec` [172], which offer data enrichment for creating features and word embeddings for machine learning tasks for any set of entities. Their novelty is that they can enrich a given dataset by using hundreds of datasets, simultaneously. More details about their novelty (and a comparison with related approaches) are given in Chapter 6.

3.6 Task E. Data Quality Assessment

This task mainly belongs to the *Auxiliary* services (see §2.4.5), since the objective is to assess data quality. The target is to be able to answer queries such as $Q_{veracity}$, $Q_{datQuality}$ and $Q_{datReuse}$ (see Table 3.1).

3.6.1 Research Gaps

There is a lack of approaches that are scalable for assessing the quality (see §2.4.5.2) of large in number RDF datasets. For *Data Fusion* only a few approaches exist for RDF datasets, while the existing ones have been tested for a small number of datasets (less than ten datasets). Moreover, in the context of web pages, in [145] the authors mentioned as a gap for the tasks of *Data Fusion* and *Data Trust*, that the different fractions of data occurring in the same dataset can possibly have different quality, e.g., one dataset can provide more accurate information for the instances belonging in class *Marine Species* comparing to the instances belonging in class *Countries*.

3.6.2 Motivating Scenarios

Below, we provide some motivating scenarios for several quality aspects.

Data Veracity. As it is stated in [82], a mandatory component for resolving conflicts and improving the correctness (and in this way the quality) of data is to collect all the records (in our case URIs) referring to the same real world entity, to fuse them into a single representation and then to apply dedicated algorithms over the "integrated" content. Indeed, the "global scale"-like semantic indexing that we propose can be beneficial for estimating the veracity of data. The key point is that the collection of all information about an entity, and the cross-dataset inference that is offered, aids spotting the contradictions that exist and on the same time provides information for data cleaning or for estimating and/or suggesting which data are probably correct or more accurate. After having spotted such cases, several options are possible for improving quality: to inform the owners of the datasets for correcting the spotted errors, or to produce an aggregated dataset with no conflicts by adopting a method for estimating the probability of correctness of each fact and/or selecting the more probable one (as it is also stated in [81, 82]). Specifically, errors can occur in the values of specific real world objects. For instance, in *d-nb.info*[3] it is written that *Aristotle's* birth year was 384 BC while in *DBpedia*[4] the value of the property *birthDate* for *Aristotle* is -383-1-1. The aggregation of all information enables estimating the more probable one. Moreover, it is not hard to see that the equivalence relationships can be exploited for improving the effectiveness of instance matching [191], which is an important requirement for effective integration [59].

Dataset Quality in terms of Interlinking and Reusability. A dataset's publisher should be able to assess the quality of his dataset comparing to other datasets, e.g., to assess whether his dataset is worth to be reused. For assisting this task, union and complement metrics can be exploited, for answering queries like, i) "How unique/redundant is the content of my dataset comparing to others?", ii) "What percentage of the available information for a set of entities is offered only from my dataset?", and iii) "Does my dataset enrich the content of other datasets?". These queries can assist publishers to improve their dataset's

[3]`http://d-nb.info/gnd/118650130`, accessed date: November 1, 2017
[4]`http://dbpedia.org/resource/Aristotle`, accessed date: November 1, 2017

quality in a long run, however, they cannot be answered from current systems.

Quality of a Possible Integration. Since the integration process is quite challenging and requires a big effort and possibly a monetary cost, it is important to estimate the quality of a possible integration by using such metrics, e.g., for answering queries like "What is the gain of integrating dataset D_j with datasets D_k and D_m?".

3.6.3 Our Placement

Comparing to the *Data Quality* services presented in §2.4.5.2, we provide metrics for Dataset Quality in terms of Interlinking and Reusability among any subset of datasets (and not only for pairs (e.g., as in [143, 179]). Moreover, with our approach it is feasible to compare all the values for a specific fact (e.g., the birth date of Aristotle) from hundreds of datasets. Therefore our approach can be exploited for performing *Data Fusion* for large number of datasets, since most current methods have been tested for a small number of datasets (i.e., see §2.4.5.2).

3.7 The Proposed Process

In the rest of this thesis, we describe how to design and developed novel indexes, methods and tools, for tackling the above research gaps and for assisting the process of semantic integration of data at large scale (i.e., for improving the tasks A-E). In Figure 3.4, we show the process that is followed, which is also described in brief, below. For performing the steps of Figure 3.4, we should tackle the challenges introduced in §1.3.

The input is a subset of LOD Cloud datasets and equivalence relationships in schema and instance level (i.e., `owl:equivalentProperty`, `owl:equivalentClass` and `owl:sameAs`). The first step is to tackle the challenge of performing cross-dataset identity reasoning over large number of datasets, for constructing a set of equivalence catalogs. The second step is to construct indexes at large scale by applying cross-dataset identity reasoning, for producing several semantics-aware indexes that cover the whole contents of datasets. These two steps will be introduced in Chapter 4.

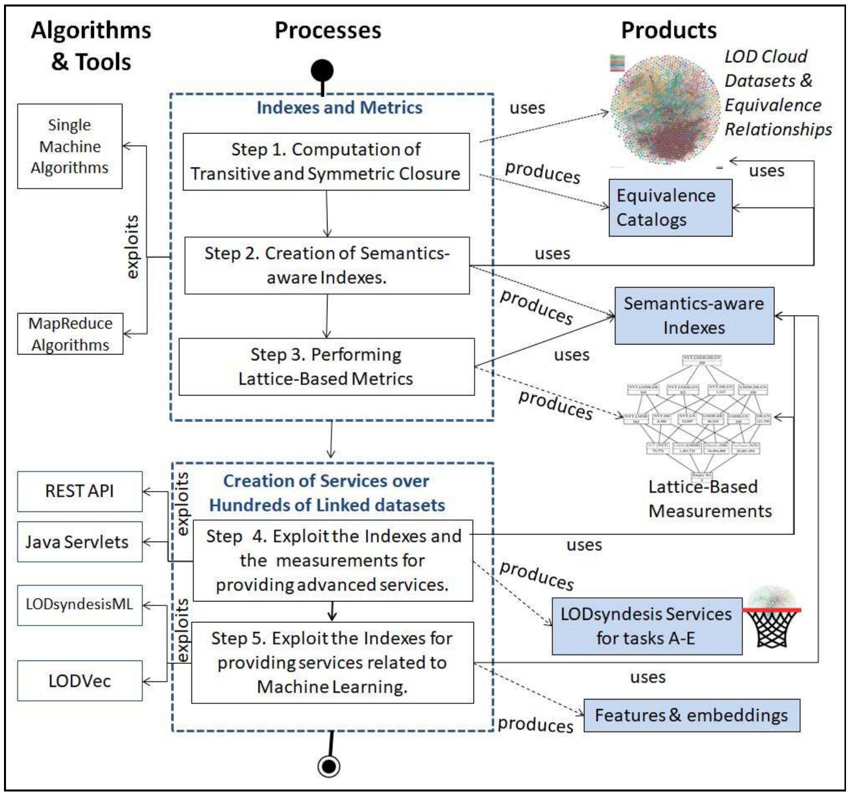

Algorithms & Tools
Processes
Products
LOD Cloud Datasets & Equivalence Relationships
Indexes and Metrics
Single Machine Algorithms
Step 1. Computation of Transitive and Symmetric Closure
uses
produces
Equivalence Catalogs
uses
exploits
Step 2. Creation of Semantics-aware Indexes.
uses
produces
Semantics-aware Indexes
MapReduce Algorithms
Step 3. Performing Lattice-Based Metrics
uses
produces
Lattice-Based Measurements
Creation of Services over Hundreds of Linked datasets
REST API
Java Servlets
exploits
Step 4. Exploit the Indexes and the measurements for providing advanced services.
uses
produces
LODsyndesis Services for tasks A-E
LODsyndesisML
LODVec
exploits
Step 5. Exploit the Indexes for providing services related to Machine Learning.
uses
produces
Features & embeddings

Figure 3.4: The steps of the process

The third step (which will be described in Chapter 5) concerns the exploitation of the constructed indexes, for performing lattice-based measurements among any subset of datasets. All these metrics will be exploited for offering connectivity analytics and content-based Dataset Discovery over large number of linked datasets.

The fourth step is to use the semantics-aware indexes for offering through LODsyndesis website[5], several advanced services for the tasks that are represented in this chapter (i.e., tasks A-E) and for answering all the queries that are included in Table 3.1. The last step is to exploit the indexes for offering advanced services for machine-learning tasks over large number of datasets, i.e., we introduce two research prototypes, called LODsyndesis$_{\mathcal{ML}}$ and LODVec that use the semantic-aware indexes for creating features and word embeddings for machine learning tasks. All these tools and services are introduced in Chapter 6.

[5]http://www.ics.forth.gr/isl/LODsyndesis

Chapter 4
Cross-dataset Identity Reasoning & Semantics-aware Indexes at Global Scale

In this chapter, we propose methods for tackling *Challenge 1* and *Challenge 2*, i.e., performing cross-dataset identity reasoning and constructing semantics-aware indexes at large scale. We focus on answering the research questions *RQ1*, *RQ2* and *RQ3*, for providing indexes that can be directly used for answering the queries related to task A (i.e., *Object Coreference and Finding all the Facts for an Entity*). In particular, by using the methods that are presented in this chapter, it is feasible to answer the queries $Q_{objectCor}$, Q_{prov}, $Q_{allFacts}$, and $Q_{veracity}$ for any given entity. For the rest queries of Table 3.1, it is also a prerequisite to exploit the indexes of this chapter, however, they require also special algorithms, methods and tools which will be introduced in the next chapters. Concerning the contributions of this chapter:

- we show how to perform cross-dataset identity reasoning, i.e., we propose methods for computing the transitive and symmetric closure for `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass` relationships, either by using a single machine or a cluster of machines,
- we exploit the result of the aforementioned closure for constructing in parallel (by using MapReduce [69] techniques) semantics-aware triples, where we replace all the URIs belonging in the same class of equivalence with a unique identifier,
- we construct in parallel five semantics-aware indexes, i.e., an entity-based triples index, for enabling the immediate access to the available data for any entity, and indexes for finding fast the provenance of entities, classes, properties and literals,
- we measure the speedup and scalability obtained by the proposed indexes and algorithms, by using 400 real RDF datasets containing over 2 billion triples.

The rest of this chapter is organized as follows. In §4.1, we introduce the MapReduce framework and several notations, whereas in §4.2, we describe the requirements for constructing the semantics-aware indexes. Moreover, in §4.3, we introduce the problem statement and we describe in brief the proposed indexing process. In §4.4 we describe ways to compute the transitive and symmetric closure of equivalence relationships by using either a single or a cluster of machines and we show how to exploit the result of the closure for creating semantics-aware triples. In §4.5, we show how to construct semantics-aware indexes for entities, literals, triples and schema elements, whereas §4.6 compares the parallel algorithms that are used (in terms of the number of iterations that are required and their communication cost). Moreover, §4.7 reports experimental results, that concern mainly the efficiency of the proposed methods, and statistics about the constructed indexes. Finally, §4.8 concludes this chapter.

Publications related to this chapter. The work presented in this chapter has been published in [165, 168, 171].

4.1 Background & Notations

4.1.1 Background - MapReduce Framework

MapReduce [69] is a popular distributed computation framework which is widely applied to large scale data-intensive for processing, primarily in the big-data domain. The computation takes as input a set of key-value pairs and produces a set of output key-value pairs. Each *MapReduce* program (or job) is carried out in two phases: a map followed by a reduce phase. Map and Reduce are two different functions which are user-defined that can become tasks that are executed in parallel. The Map function takes as input key-value pairs, performs a user-defined operation over an arbitrary part of the input, partitions the data and produces a set of intermediate key-value pairs. Subsequently, all key-value pairs produced during the map phase are grouped by their key and are passed (shuffled to the appropriate tasks and sorted) to the reduce phase. During the reduce phase, a reduce function is called for each unique key, it processes the corresponding list of values and finally produces a set of output key-value pairs.

4.1.2 Notations

First, let $\mathcal{D} = \{D_1, \ldots, D_n\}$ be a set of datasets, $\mathcal{P}(\mathcal{D})$ denote the power set of $\mathcal{D}$, and B is any set of datasets $B \subseteq \mathcal{D}$. Moreover, we divide the URIs in three categories, i.e., $\mathcal{U} = E \cup P \cup C$, where E refers to the entities, P to the properties and C to the classes, where these sets of URIs are pairwise disjoint. In particular, a property describes a relation between a subject and an object and the set of properties P is defined as $P = \{p \mid \langle s, p, o \rangle \in \mathcal{T}\}$. Concerning the classes, they are used for grouping entities into classes, e.g., Humans, Philosophers, Actors, and so forth. They can be found through triples of the form ⟨s,rdf:type,c⟩, where s is an entity and c is a class, i.e., $C = \{c \mid \langle s, rdf{:}type, c \rangle \in \mathcal{T}\}$. Finally, the remaining URIs are defined as entities, i.e., $E = \mathcal{U} \setminus (P \cup C)$. For a single dataset $D_i \in \mathcal{D}$, $U_i \subset \mathcal{U}$ is the set of its URIs (where E_i, P_i and C_i are the sets of entities, properties and classes, respectively), $triples(D_i) \subset \mathcal{T}$ is the set of its triples and L_i is the set of its literals. It is worth mentioning that we ignore triples containing blank nodes, since we do not apply blank node matching techniques for finding common blank nodes, like those proposed in [238].

4.2 Requirements

Here, we introduce the main requirements, which are essential for constructing semantically enriched indexes that cover the whole contents of datasets.

4.2.1 Considering Equivalence Relationships

Our major requirement is to take into consideration the equivalence relationships between the datasets, i.e., to compute the cross-dataset identity reasoning. We consider the following equivalences in schema and instance level in a subset of datasets $B \subseteq \mathcal{D}$. Let $Equiv(B, r)$ be all the triples that contain a property $r \in Eqv$, $Eqv = \{\texttt{owl:sameAs}, \texttt{owl:equivalentProperty}, \texttt{owl:equivalentClass}\}$, that defines a specific equivalence between two URIs: $Equiv(B, r) = \{(u, u') | (u, \mathrm{r}, u') \in triples(D_i), D_i \in B, r \in Eqv\}$. The `owl:sameAs` property denotes that two URIs refer to the same entity, while the remaining ones denote equivalences among schema elements (i.e., properties and classes). These types of equivalences are transitive, symmetric and reflexive, and our target is to compute their closure, in order not to miss equivalence relationships among different datasets.

If R denotes a binary relation, consequently, we use $C(R)$ to denote the transitive, symmetric and reflexive closure of R, while $C(Equiv(B, r))$ stands for this type of closure of a relation r in all datasets in B. For instance, if $U_{temp} = \{u_1, u_2, u_3, u_4, u_5\}$ and there are two `owl:sameAs` relationships, u_1 `owl:sameAs` u_3 and u_1 `owl:sameAs` u_4, then their closure derives the following classes of equivalence $U_{temp}/\sim = \{\{u_1, u_3, u_4\}, \{u_2\}, \{u_5\}\}$. We can now define the equivalent URIs (considering all datasets in B) of an entity $u \in E$, a property $p \in P$ and a class $c \in C$ as follows:

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

4.2.2 Creation of Real World Objects & Triples

Here, we show how to exploit the result of the computation of cross-dataset identity reasoning, for keeping a single representation for each real world object.

4.2.2.1 From URIs to Real World Objects

For taking into account the equivalences, our target is to replace any URI with its corresponding class of equivalence, where the URIs that are "semantically" the same belong to the same class of equivalence, e.g., all the URIs referring to "Aristotle" belong to the same equivalence class. We denote as $[u]_e$, $[u]_{pr}$ and $[u]_{cl}$ the class of equivalence of a URI u, if it belongs to entities, properties or classes, respectively. In particular, for a dataset $D_i \in \mathcal{D}$, we define its' real world entities, properties and classes, as follows:

- **Real World Entities:** $RWE(D_i) = \cup_{u \in E_i}[u]_e$, where $\forall u' \in EqEnt(u, \mathcal{D})$, $[u]_e = [u']_e$.
- **Real World Properties:** $RWP(D_i) = \cup_{u \in P_i}[u]_{pr}$, where $\forall u' \in EqProp(u, \mathcal{D})$, $[u]_{pr} = [u']_{pr}$.
- **Real World Classes:** $RWC(D_i) = \cup_{u \in C_i}[u]$ where $\forall u' \in EqClass(c, \mathcal{D})$, $[u]_{cl} = [u']_{cl}$.

Finally, for all the datasets $\mathcal{D}$ we define the following sets: $RWE = \bigcup_{D_i \in \mathcal{D}} RWE(D_i)$, $RWP = \bigcup_{D_i \in \mathcal{D}} RWP(D_i)$ and $RWC = \bigcup_{D_i \in \mathcal{D}} RWC(D_i)$, for entities, properties and classes, respectively.

4.2.2.2 Literals Conversion

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

4.2.2.3 From Triples to Real World Triples.

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

$$s' = \begin{cases} [s]_e, s \in E_i \end{cases} \quad p' = \begin{cases} [p]_{pr}, p \in P_i \end{cases} \quad o' = \begin{cases} o_{conv}, o \in L_i \\ [o]_e, o \in E_i \\ [o]_{cl}, o \in C_i \end{cases}$$

Table 4.1: Finding the provenance of different elements.

Element	**Datasets Where an Element Occurs**
Entity u	$dsetsEnt_{\sim}(u,B) = \{D_i \in B \mid [u]_e \in RWE(D_i)\}$
Property p	$dsetsProp_{\sim}(p,B) = \{D_i \in B \mid [p]_{pr} \in RWP(D_i)\}$
Class c	$dsetsClass_{\sim}(c,B) = \{D_i \in B \mid [c]_{cl} \in RWC(D_i)\}$
Literal l	$dsetsLit(l,B) = \{D_i \in B \mid l_{conv} \in LIT(D_i)\}$
Triple t	$dsetsTr_{\sim}(t,B) = \{D_i \in B \mid \mu(t) \in RWT(D_i)\}$

As a consequence, the real world triples for a dataset D_i is defined as $RWT(D_i) = \cup_{t \in triples(D_i)} \mu(t)$, and for all the datasets in $\mathcal{D}$ we define $RWT = \bigcup_{D_i \in \mathcal{D}} RWT(D_i)$. Finally, for a set of entities E', we define $RWT_{E'} = \{\langle s,p,o \rangle \in RWT \mid s = [u]_e \text{ } or \text{ } o = [u]_e, u \in E'\}$ as the set of real world triples where at least one $u \in E'$ (or an equivalent URI of u) occurs.

4.3 Problem Statement & Process

4.3.1 Problem Statement

Our major target is to be able to answer the queries $Q_{objectCor}$, Q_{prov}, $Q_{allFacts}$, and $Q_{veracity}$. Concerning $Q_{objectCor}$, the objective is to compute the transitive and symmetric closure of equivalence relationships at global scale, for being able to answer such queries for any $u \in \mathcal{U}$, i.e., for finding the $EqEnt(u,B)$, $EqProp(p,B)$ and $EqClass(c,B)$.

Regarding Q_{prov}, the target is to be able to find all the datasets that contain information (i.e., the provenance) about a specific element (or an equivalent one), by exploiting the real world objects and triples which were introduced above. In Table 4.1, we define the set of datasets where a specific element (or an equivalent one) occurs, e.g., for the URI d1:Aristotle, $dsetsEnt_{\sim}$(d1:Aristotle, $\mathcal{D}$) returns all the datasets where d1:Aristotle (or any equivalent URI of d1:Aristotle) occurs.

Finally, for the queries $Q_{allFacts}$, and $Q_{veracity}$, the target is to retrieve fast all the real world triples for a given entity, e.g., for being able to compare conflicting values from different datasets. In the rest of this chapter we present methods for constructing several semantics-aware indexes that can be directly exploited for answering the aforementioned queries.

4.3.2 The Proposed Process of Global Indexing

Here, we describe the process of global indexing in brief. In particular, it comprises of four different steps, which can be seen in the running example of Figure 4.1. The first step is the collection of input datasets, which is a set of datasets' triples and a set of RDF relationships (e.g., `owl:sameAs` relationships). In our running example, the input contains 4 datasets, each one having six triples (in the upper left side), and several instance and schema relationships (in the upper right side). The next step includes the computation of closure in schema and instance level, where catalogs containing for each URI an identifier are produced. In the third step, we use the initial datasets and the aforementioned catalogs for creating "semantically" enriched triples (i.e., real world triples). In the fourth step, we create semantically enriched inverted indexes for different sets of elements (e.g., triples, entities), where we collect all the information of each different entity, and we store the dataset IDs (i.e., a posting list) where real world entities, properties, classes, triples and literals occur.

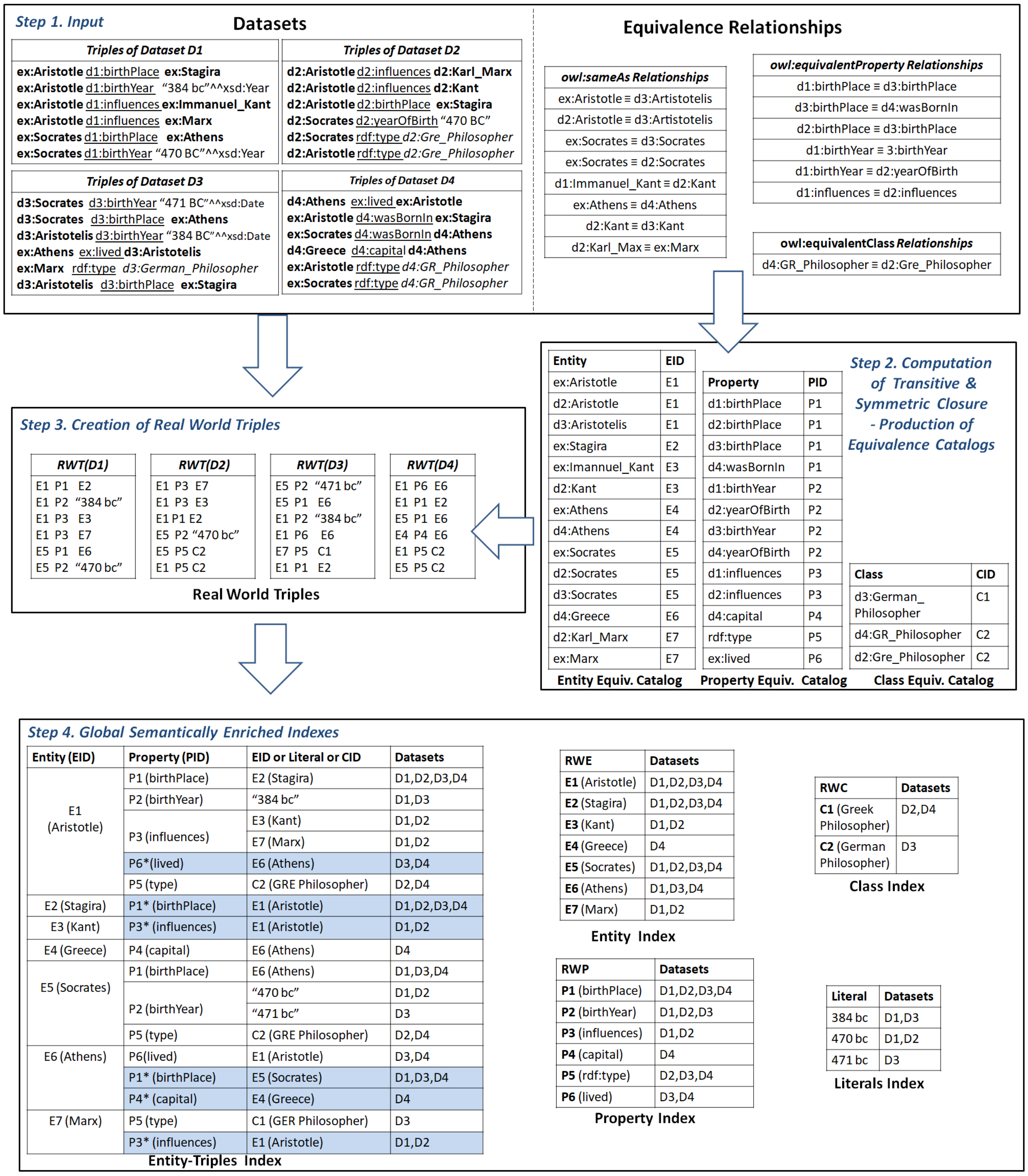
4.4 Cross-dataset Identity Reasoning at Global Scale

Rationale: It is required for finding all the URIs that are equivalent to a given one, i.e, for finding $EqEnt(u, B)$, $EqProp(p, B)$, and $EqClass(c, B)$, which were defined in §4.2.1. Moreover, the result of this process will be exploited for creating the sets RWE, RWP, RWC and RWT (i.e., see §4.2.2). Here, we introduce methods where each class of equivalence will get a signature (id) and the signature is constructed incrementally during the computation. We create one such catalog for each different URI's category, i.e., properties, classes and entities, as it is described below:

- Entity Equivalence Catalog (`EntEqCat`): For each of the entities $u \in E$ we assign a unique ID, where EID denotes this set of identifiers (i.e., a binary relation $\subseteq EID$). For constructing this catalog, we read the URIs of each dataset (marked in bold in Figure 4.1) and the `owl:sameAs` relationships (see the upper right side of Figure 4.1), and we compute the transitive, symmetric and reflexive closure of `owl:sameAs` relationships, for computing the classes of equivalence. All the entities belonging to the same class of equivalence (e.g., all the URIs of "Aristotle") will be assigned the same identifier, e.g., see the `EntEqCat` in the running example of Figure 4.1. This catalog will be exploited for replacing each URI that occurs in a triple with an identifier.

- Property Equivalence Catalog (`PropEqCat`): For each of the properties $p \in P$, we store a unique ID, where PID denotes this set of identifiers (i.e., a binary relation $\subseteq PID$). For constructing it, one should read the properties of each dataset (underlined in Figure 4.1), the `owl:equivalentProperty` relationships (see the upper right side of Figure 4.1), and compute the closure of these relationships for producing the classes of equivalence for properties. Afterwards, all the properties belonging to the same class of equivalence are assigned the same identifier, e.g., in Figure 4.1, we can observe the `PropEqCat` of our running example. As we shall see, this catalog will be used for replacing each property with its corresponding identifier.

- Class Equivalence Catalog (`ClEqCat`): For any class $c \in C$, we store a unique ID, where CID denotes this set of identifiers (i.e., a binary relation $\subseteq CID$). For constructing it,


Step 1. Input
Datasets
Equivalence Relationships
Step 2. Computation of Transitive & Symmetric Closure - Production of Equivalence Catalogs
Step 3. Creation of Real World Triples
Real World Triples
Step 4. Global Semantically Enriched Indexes

Triples of Dataset D1
ex:Aristotle d1:birthPlace **ex:Stagira**
ex:Aristotle d1:birthYear "384 bc"^^xsd:Year
ex:Aristotle d1:influences **ex:Immanuel_Kant**
ex:Aristotle d1:influences **ex:Marx**
ex:Socrates d1:birthPlace **ex:Athens**
ex:Socrates d1:birthYear "470 BC"^^xsd:Year

Triples of Dataset D2
d2:Aristotle d2:influences **d2:Karl_Marx**
d2:Aristotle d2:influences **d2:Kant**
d2:Aristotle d2:birthPlace **ex:Stagira**
d2:Socrates d2:yearOfBirth "470 BC"
d2:Socrates rdf:type *d2:Gre_Philosopher*
d2:Aristotle rdf:type *d2:Gre_Philosopher*

Triples of Dataset D3
d3:Socrates d3:birthYear "471 BC"^^xsd:Date
d3:Socrates d3:birthPlace **ex:Athens**
d3:Aristotelis d3:birthYear "384 BC"^^xsd:Date
ex:Athens ex:lived **d3:Aristotelis**
ex:Marx rdf:type *d3:German_Philosopher*
d3:Aristotelis d3:birthPlace **ex:Stagira**

Triples of Dataset D4
d4:Athens ex:lived **ex:Aristotle**
ex:Aristotle d4:wasBornIn **ex:Stagira**
ex:Socrates d4:wasBornIn **d4:Athens**
d4:Greece d4:capital **d4:Athens**
ex:Aristotle rdf:type *d4:GR_Philosopher*
ex:Socrates rdf:type *d4:GR_Philosopher*

owl:sameAs Relationships
ex:Aristotle ≡ d3:Artistotelis
d2:Aristotle ≡ d3:Artistotelis
ex:Socrates ≡ d3:Socrates
ex:Socrates ≡ d2:Socrates
d1:Immanuel_Kant ≡ d2:Kant
ex:Athens ≡ d4:Athens
d2:Kant ≡ d3:Kant
d2:Karl_Max ≡ ex:Marx

owl:equivalentProperty Relationships
d1:birthPlace ≡ d3:birthPlace
d3:birthPlace ≡ d4:wasBornIn
d2:birthPlace ≡ d3:birthPlace
d1:birthYear ≡ 3:birthYear
d1:birthYear ≡ d2:yearOfBirth
d1:influences ≡ d2:influences

owl:equivalentClass *Relationships*
d4:GR_Philosopher ≡ d2:Gre_Philosopher

Entity	EID
ex:Aristotle	E1
d2:Aristotle	E1
d3:Aristotelis	E1
ex:Stagira	E2
ex:Imannuel_Kant	E3
d2:Kant	E3
ex:Athens	E4
d4:Athens	E4
ex:Socrates	E5
d2:Socrates	E5
d3:Socrates	E5
d4:Greece	E6
d2:Karl_Marx	E7
ex:Marx	E7

Entity Equiv. Catalog

Property	PID
d1:birthPlace	P1
d2:birthPlace	P1
d3:birthPlace	P1
d4:wasBornIn	P1
d1:birthYear	P2
d2:yearOfBirth	P2
d3:birthYear	P2
d4:yearOfBirth	P2
d1:influences	P3
d2:influences	P3
d4:capital	P4
rdf:type	P5
ex:lived	P6

Property Equiv. Catalog

Class	CID
d3:German_ Philosopher	C1
d4:GR_Philosopher	C2
d2:Gre_Philosopher	C2

Class Equiv. Catalog

RWT(D1)	*RWT(D2)*	*RWT(D3)*	*RWT(D4)*
E1 P1 E2	E1 P3 E7	E5 P2 "471 bc"	E1 P6 E6
E1 P2 "384 bc"	E1 P3 E3	E5 P1 E6	E1 P1 E2
E1 P3 E3	E1 P1 E2	E1 P2 "384 bc"	E5 P1 E6
E1 P3 E7	E5 P2 "470 bc"	E1 P6 E6	E4 P4 E6
E5 P1 E6	E5 P5 C2	E7 P5 C1	E1 P5 C2
E5 P2 "470 bc"	E1 P5 C2	E1 P1 E2	E5 P5 C2

Entity (EID)	Property (PID)	EID or Literal or CID	Datasets
E1 (Aristotle)	P1 (birthPlace)	E2 (Stagira)	D1,D2,D3,D4
	P2 (birthYear)	"384 bc"	D1,D3
	P3 (influences)	E3 (Kant)	D1,D2
		E7 (Marx)	D1,D2
	P6*(lived)	E6 (Athens)	D3,D4
	P5 (type)	C2 (GRE Philosopher)	D2,D4
E2 (Stagira)	P1* (birthPlace)	E1 (Aristotle)	D1,D2,D3,D4
E3 (Kant)	P3* (influences)	E1 (Aristotle)	D1,D2
E4 (Greece)	P4 (capital)	E6 (Athens)	D4
E5 (Socrates)	P1 (birthPlace)	E6 (Athens)	D1,D3,D4
	P2 (birthYear)	"470 bc"	D1,D2
		"471 bc"	D3
	P5 (type)	C2 (GRE Philosopher)	D2,D4
E6 (Athens)	P6(lived)	E1 (Aristotle)	D3,D4
	P1* (birthPlace)	E5 (Socrates)	D1,D3,D4
	P4* (capital)	E4 (Greece)	D4
E7 (Marx)	P5 (type)	C1 (GER Philosopher)	D3
	P3* (influences)	E1 (Aristotle)	D1,D2

Entity-Triples Index

RWE	Datasets
E1 (Aristotle)	D1,D2,D3,D4
E2 (Stagira)	D1,D2,D3,D4
E3 (Kant)	D1,D2
E4 (Greece)	D4
E5 (Socrates)	D1,D2,D3,D4
E6 (Athens)	D1,D3,D4
E7 (Marx)	D1,D2

Entity Index

RWP	Datasets
P1 (birthPlace)	D1,D2,D3,D4
P2 (birthYear)	D1,D2,D3
P3 (influences)	D1,D2
P4 (capital)	D4
P5 (rdf:type)	D2,D3,D4
P6 (lived)	D3,D4

Property Index

RWC	Datasets
C1 (Greek Philosopher)	D2,D4
C2 (German Philosopher)	D3

Class Index

Literal	Datasets
384 bc	D1,D3
470 bc	D1,D2
471 bc	D3

Literals Index

Figure 4.1: Running example containing four datasets.

one should read the classes (marked in italics in Figure 4.1), the `owl:equivalentClass` relationships, and compute their closure for finding the classes of equivalence. Finally, all the classes that refer to the same real world thing will take the same identifier. The resulted `ClEqCat` for our running example can be seen in Figure 4.1. We will exploit this catalog for replacing each class with the corresponding identifier.

In the rest of this section, we introduce algorithms for computing the transitive and symmetric closure of equivalence relationships by using a single machine (see §4.4.1) and by using a cluster of machines (see §4.4.2).

4.4.1 Computation of Equivalence Relationships - Single Machine Algorithm

Here, we introduce a method for computing the transitive and symmetric closure by using a single machine.

Construction Method: We introduce a signature-based algorithm (see Alg. 1) 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. Alg. 1 computes the symmetric and transitive closure of `owl:sameAs` relationships, however, the same algorithm can be used for computing the closure of `owl:equivalentProperty` and `owl:equivalentClass` relationships. First, it uses two maps, a) `EntEqCat` where for each URI it stores a unique identifier (an integer) , and b) `Cle` where it stores all the URIs having the same key (see lines 1-2). Afterwards, it assigns to each pair $(u, u') \in Equiv(B, r)$ an identifier according to the following rules:

1. If both URIs have not an identifier, a new identifier is assigned to both of them (see lines 5-9). E.g. Table 4.2 contains two classes of equivalence and four URIs. In the next step, a new `owl:sameAs` pair containing two URIs without identifier is inserted resulting to a new class of equivalence which is shown in Table 4.3.
2. If u has an identifier while u' has not, u' gets the same identifier as u (see lines 10-13). Table 4.4 shows such an example where a new URI (u_7) is assigned 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' (see lines 14-17).
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 (see lines 18-23), while the entry of the highest signature of `Cle` is deleted (see line 24). In case of Table 4.5, both URIs exist in the `EntEqCat` and have a different identifier. For this reason, the classes of equivalence of these identifiers are merged.

Finally, Alg. 1 returns the constructed `EntEqCat` (see line 25). We can say that the algorithm constructs incrementally chains of `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass` URIs where each URI becomes a member of a chain if and only if there is a such relationship with a URI which is already member of this chain. Its correctness is based on the following proposition.

Prop. 1. If $(a, b) \in Equiv(B, r)$ and $(a, c) \in Equiv(B, r)$ then $(b, c) \in C(Equiv(B, r))$.

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

Table 4.2:
Classes of Equivalence

ID	URIs
1	u_1, u_2
2	u_3, u_4

Table 4.3:
Insert u_5 `owl:sameAs` u_6

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

Table 4.4:
Insert u_3 `owl:sameAs` u_7

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

Table 4.5:
Insert u_1 `owl:sameAs` u_3

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

Time and Space Complexity. Its benefit is that it needs one pass for each equivalence pair to compute the transitive and symmetric closure, i.e., where n is the number of all the $Equiv(B, r)$ pairs. By using a TreeMap for the `EntEqCat`, the keys are sorted, therefore the cost of the functions lookup (i.e., $u \notin Left($`EntEqCat`$)$), insert, and for replacing the value of an entry equals $O(logk)$, where k is number of elements that are part of `EntEqCat` in a specific step. The range of k is $0 \leq k \leq 2 * n$, and k increases as we read more pairs. For the `Cle`, by using also a TreeMap, the cost of the functions lookup, insert and deleteKey equals $O(logr)$, where r is number of elements that are part of `Cle`in a specific step. The range of r is $0 \leq r \leq n$. Therefore, the time complexity of Alg. 1 is $O(n\ logn)$,

Concerning space complexity, it keeps in memory chains of $Equiv(B, r)$ in order to connect such chains with new `owl:sameAs` relationships. Indeed, the space complexity is $O(m)$, where m is the number of unique URIs that occur in $Equiv(B, r)$ pairs, since in the worst case each URI is stored both in the catalog and in the classes of equivalence map, 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 catalog is equal to the number of unique real

ALGORITHM 1: Equivalence Catalog Creation

```
Input: $Equiv(B, \texttt{owl:sameAs})$ a binary relation containing owl:sameAs relationships
Output: A catalog containing for each class of equivalence a signature.
1  $map$ EntEqCat$\{key, value\}$  $(key : URI, value : integer)$
2  $map$ Cle$\{key, value\}$  $(key : integer, value : set\ of\ URIs)$
3  $ID \leftarrow 1$
4  forall $(u, u') \in Equiv(B, \texttt{owl:sameAs})$ do
5      if $u \notin Left(\texttt{EntEqCat})$ and $u' \notin Left(\texttt{EntEqCat})$ then
6          EntEqCat.$insert(u, ID)$
7          EntEqCat.$insert(u', ID)$
8          Cle.$insert(ID, \{u, u'\})$
9          $ID \leftarrow ID + 1$
10     else if $u \in Left(\texttt{EntEqCat})$ and $u' \notin Left(\texttt{EntEqCat})$ then
11         $u_{ID} \leftarrow \texttt{EntEqCat}(u)$
12         EntEqCat.$insert(u', u_{ID})$
13         $\texttt{Cle}(u_{ID}) \leftarrow \texttt{Cle}(u_{ID}) \cup \{u'\}$
14     else if $u \notin Left(\texttt{EntEqCat})$ and $u' \in Left(\texttt{EntEqCat})$ then
15         $u'_{ID} \leftarrow \texttt{EntEqCat}(u')$
16         EntEqCat.$insert(u, u'_{ID})$
17         $\texttt{Cle}(u'_{ID}) \leftarrow \texttt{Cle}(u'_{ID}) \cup \{u\}$
18     else if $\texttt{EntEqCat}(u) \neq \texttt{EntEqCat}(u')$ then
19         $min_{ID} \leftarrow min(\texttt{EntEqCat}(u), \texttt{EntEqCat}(u'))$
20         $max_{ID} \leftarrow max(\texttt{EntEqCat}(u), \texttt{EntEqCat}(u'))$
21         $\texttt{Cle}(min_{ID}) \leftarrow \texttt{Cle}(min_{ID}) \cup \texttt{Cle}(max_{ID})$
22         forall $u_j \in \texttt{Cle}(max_{ID})$ do
23             $\texttt{EntEqCat}(u_j) \leftarrow min_{ID}$
24         Cle.$deleteKey(max_{ID})$
25 return EntEqCat
```

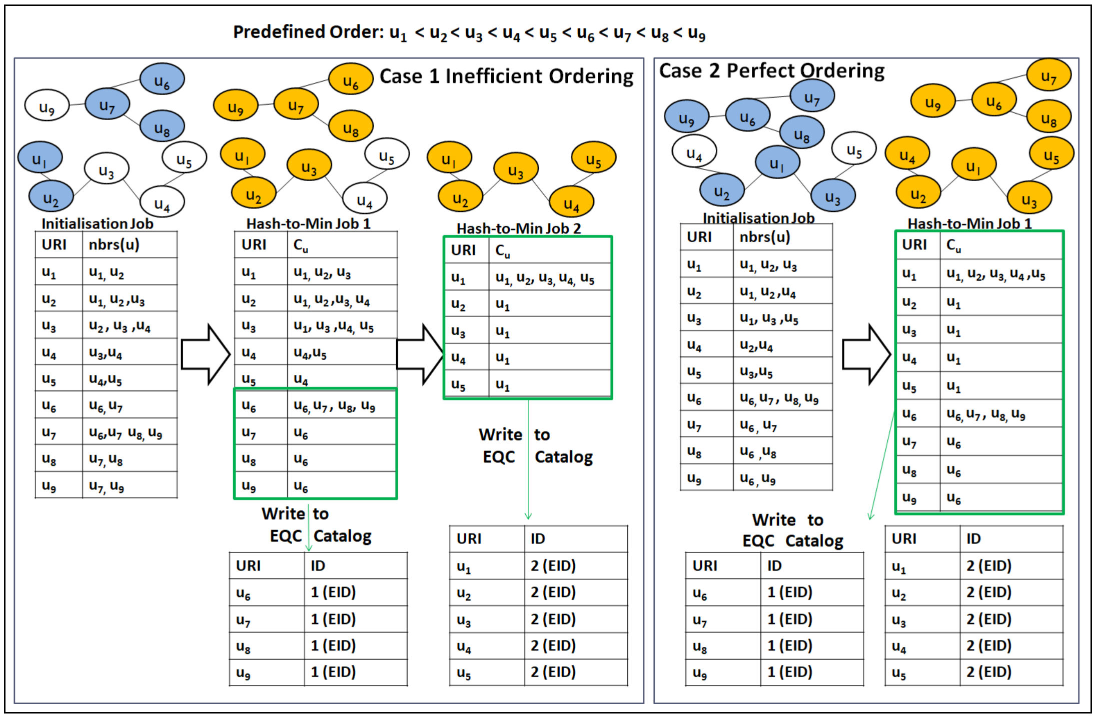
world objects of $Equiv(B, r)$. In our running example, the produced catalogs are shown in the upper right side Figure 4.1 (see step 2).

Alternatively, one can use Tarjan's connected components (CC) algorithm [233] that uses Depth-First Search (DFS). The input of CC algorithm is a graph which should have been created before running the algorithm. For this reason, we have to read all the $Equiv(B, r)$ pairs once $O(n)$ (i.e., each $Equiv(B, r)$ pair represents an edge) in order to construct the graph. The time complexity of CC algorithm is $O(m + n)$. 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 $O(m)$, since in the worst case it needs to keep all the nodes in DFS stack (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 $O(n + m)$. In §4.7 we compare the execution time of the two aforementioned approaches.

4.4.2 Computation of Equivalence Relationships - Parallel Algorithm

The limitations of single-node process and therefore motivation for parallelization using several machines are the following: the signature based algorithm needs a lot of memory, thereby, as we will see in §4.7, we are unable to compute the closure for over 14 million of equivalence relationships. For this reason, we propose an algorithm for computing the cross-dataset identity reasoning in parallel.

Parallelization Overview. Let m be the set of available machines and m_i be a single machine. Moreover, let $EQ = \{EQ_1, \ldots, EQ_m\}$ be a partition of the equivalence relationships, i.e., `owl:sameAs`, `owl:equivalentClass` and `owl:equivalentProperty`, while we need also partitions of URIs, where $\mathcal{U} = \{UR_i, UR_j, .., UR_m\}$ and $(UR_i \cup UR_j \cup \ldots \cup UR_m = \mathcal{U})$. Each machine m_i is assigned the responsibility to read a subset of equivalence pairs EQ_i, and a subset of URIs UR_i, and to compute a partial function eq_i: $\mathcal{U} \rightarrow ID$. In the reducer, any equivalence catalog ($EqCat$) can be derived as $EqCat = eq_1 \cup \ldots \cup eq_m$. From a wide range of proposed parallel algorithms, we selected to use Hash-to-Min algorithm [203] since it computes the connected component in logarithmic rounds of *MapReduce* Jobs and has linear communication cost per round. However, we propose some optimizations that we apply to our context for reducing the number of *MapReduce* jobs and the size of intermediate data.


Predefined Order: $u_1 < u_2 < u_3 < u_4 < u_5 < u_6 < u_7 < u_8 < u_9$
Case 1 Inefficient Ordering
Case 2 Perfect Ordering
u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9
Initialisation Job
Hash-to-Min Job 1
Hash-to-Min Job 2
Write to EQC Catalog

Case 1 Inefficient Ordering

Initialisation Job

URI	nbrs(u)
u_1	u_1, u_2
u_2	u_1, u_2, u_3
u_3	u_2, u_3, u_4
u_4	u_3, u_4
u_5	u_4, u_5
u_6	u_6, u_7
u_7	u_6, u_7, u_8, u_9
u_8	u_7, u_8
u_9	u_7, u_9

Hash-to-Min Job 1

URI	C_u
u_1	u_1, u_2, u_3
u_2	u_1, u_2, u_3, u_4
u_3	u_1, u_3, u_4, u_5
u_4	u_4, u_5
u_5	u_4
u_6	u_6, u_7, u_8, u_9
u_7	u_6
u_8	u_6
u_9	u_6

URI	ID
u_6	1 (EID)
u_7	1 (EID)
u_8	1 (EID)
u_9	1 (EID)

Hash-to-Min Job 2

URI	C_u
u_1	u_1, u_2, u_3, u_4, u_5
u_2	u_1
u_3	u_1
u_4	u_1
u_5	u_1

URI	ID
u_1	2 (EID)
u_2	2 (EID)
u_3	2 (EID)
u_4	2 (EID)
u_5	2 (EID)

Case 2 Perfect Ordering

Initialisation Job

URI	nbrs(u)
u_1	u_1, u_2, u_3
u_2	u_1, u_2, u_4
u_3	u_1, u_3, u_5
u_4	u_2, u_4
u_5	u_3, u_5
u_6	u_6, u_7, u_8, u_9
u_7	u_6, u_7
u_8	u_6, u_8
u_9	u_6, u_9

Hash-to-Min Job 1

URI	C_u
u_1	u_1, u_2, u_3, u_4, u_5
u_2	u_1
u_3	u_1
u_4	u_1
u_5	u_1
u_6	u_6, u_7, u_8, u_9
u_7	u_6
u_8	u_6
u_9	u_6

URI	ID
u_6	1 (EID)
u_7	1 (EID)
u_8	1 (EID)
u_9	1 (EID)

URI	ID
u_1	2 (EID)
u_2	2 (EID)
u_3	2 (EID)
u_4	2 (EID)
u_5	2 (EID)

Figure 4.2: Hash-to-Min Algorithm Example - Impact of Ordering

Construction method: Let G be a graph $G = \{V, E\}$, where V is the set of vertices, in our case $V = \{u \mid \{(u, r, u') \cup (u', r, u)\} \in \mathcal{T}, r \in Eqv\}$, and E is the set of edges, in our case $E = Equiv(B, r)$. The key notion of Hash-to-Min is that it creates one cluster C_u, where $C_u = Equiv(u, \mathcal{D}) \cup \{u\}$, for all the URIs belonging in the same class of equivalence (or connected component), and for each $u \in C_u$ it assigns to them the same signature (see an example in step 2 of Figure 4.1). Hash-to-Min [203] algorithm takes as initial input a set of key-value pairs having key a URI u and value a set $nbrs(u) = \{u' \mid \{(u, r, u') \cup (u', r, u)\} \cup \{u\}\}$ ($r \in Eqv$). Consequently, we need a single job that reads a "slice" of `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass` pairs in the mapper and produces the set $C_u = nbrs(u)$, $\forall\ u \in V$ in the reducer, e.g., see the initialization job of the example in Figure 4.2 (the vertices with the blue color represent the neighbors of u_{min}). It is worth mentioning that when $|nbrs(u)| = 1$, u is equivalent only to itself, e.g., in the running example of Figure 4.1, d4:Greece does not belong to an `owl:sameAs` relationship. Therefore, we can immediately assign to this URI a unique identifier (i.e., E6 for "d4:Greece").

In the example of Figure 4.2, we have nine URIs and suppose that there exists the following predefined order $u_1 < u_2 < u_3 < u_4 < u_5 < u_6 < u_7 < u_8 < u_9$, thereby, for the cluster of vertices u_1 to u_5, u_1 it the u_{min} while for the cluster u_6 to u_9, u_6 is the u_{min}. It is worth noting that this example contains two cases having connected components with exactly the same structure. However, due to the impact of ordering (which is described in §4.4.2.1) the connected components of case 1 need one additional job in order to be computed comparing to the connected components of case 2.

Concerning Hash-to-Min algorithm, it sends in its first job the entire cluster C_u to a reducer u_{min}, where u_{min} is the smallest node in the cluster C_u according to a global ranking (e.g., lexicographical order or any other pre-specified order). Moreover, it sends a key-value pair (u', u_{min}), for each $u' \in (C_u \setminus u)$ in order to "inform" the other nodes of the cluster C_u about the smallest node inside the aforementioned cluster, i.e. u_{min}. In the reducer, the algorithm computes for each URI u_{min} the union of the cluster $C_{u_{min}}$ for a specific u_{min}, e.g., see job 1 in Figure 4.2 (the vertices having orange color represent the cluster of u_{min} in each job). When the cluster of the vertex with the minimum id contains the entire connected component, while the cluster of other vertices in the component is a singleton having the minimum vertex, the connected component has been computed. Afterwards, for all the URIs $u \in C_{u_{min}}$, we assign them a new signature $[u]$ (i.e., they belong to the same class of equivalence) and we store $\forall u \in C_{u_{min}}$ in the corresponding catalog (e.g., in `EntEqCat`) an entry comprising of URI u and [u]. For instance, in the first Hash-to-Min job of case 1 in Figure 4.2, the computation of connected component containing the URIs u_6 to u_9 has finished, since u_6 which is the u_{min} in this connected component contains the entire connected component while at the same time each of the other vertices of the connected component (i.e., u_7, u_8 and u_9) is a singleton having the minimum vertex (i.e., u_6). Afterwards, an entry in the corresponding catalog is created for each of these URIs, i.e., each of these URIs is assigned the same signature since they belong to the same class of equivalence.

On the contrary, for the connected components that have not finished after a specific job, a new *MapReduce* job is performed (e.g., see the second Hash-to-Min job of case 1 in Figure 4.2 for the URIs u_1 to u_5), where we send the cluster again to the u_{min} while we “inform” the other nodes about the u_{min} (which is possibly different). In the reducer, the same process is performed again (i.e., we merge the clusters for u_{min}).

Iterations and Communication Cost. The complexity, regarding the *MapReduce* iterations needed by this algorithm, is $O(logn)$, where n is the number of nodes in the largest component of a path graph (i.e. a tree with only nodes of degree 2 or 1 like in the case of URIs u_1 to u_5 in Figure 4.2) while its communication cost (i.e., number of key-value pairs that we send to the reducers) is $O(logn|V| + |E|)$. The proofs about the time and space complexity of this algorithm are given in [203].

The result of this algorithm is a set of clusters, where each cluster C_u contains all the URIs of the same entity. For instance, in the running example of Figure 4.1, the cluster of “Aristotle” entity is the following one: $C_{Aristotle}$ = {ex:Aristotle,d2:Aristotle,d3:Aristotle}, while the cluster for “birthPlace” property contains the following four URIs, i.e., $C_{birthPlace}$ = {d1:birthPlace,d2:birthPlace,d3:birthPlace,d4:wasBornIn}. For all the URIs of the same class of equivalence, it assigns to each of them the same unique identifier, e.g., the identifier of all URIs referring to Aristotle is E1, while P1 is the identifier of all the URIs referring to the property “birthPlace”.

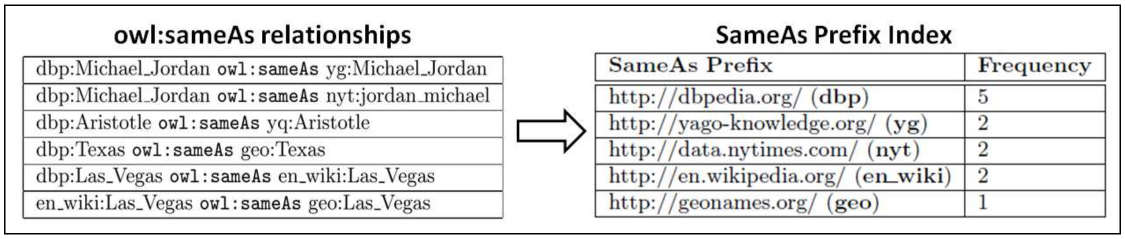
Combination with Signature Based Algorithm. In many cases, the size of the classes of equivalence is small for most of real world objects (as we shall see in Figure 4.7) while there exists only a few number of connected components that need more *MapReduce* jobs in order to be finished. To avoid performing *MapReduce* jobs for few number of large connected components, we can use a threshold t and when the number of remaining URIs is lower than t, we can send the remaining data to one machine and use the signature-based algorithm described in §4.4.1. In §4.7 we introduce comparative results showing the gain by combining the Hash-to-Min algorithm with the signature-based one.

4.4.2.1 Deciding the Global Ranking of Nodes For Connected Components - SameAs Prefix Index

Rationale: For many connected components parallel algorithms [203], it is important to decide a global ranking in order to “foresee” the centre of the connected component. For instance, in Figure 4.2 we can observe that for two graphs with the same structure, in the first case we need two Hash-to-Min jobs for computing the connected components of the graph while in the second case we need one Hash-to-Min job (except for the initialisation job). The difference is that in the second case we always select as u_{min} the centre of each of the connected components, while in the first one we select as u_{min} the URI having the lowest degree. Generally, if there is available information about the URI occurring as the center of a connected component (URI with the minimum radius), we can select as u_{min} this URI, however, such a pre-processing step can be expensive [130]. As it is stated in [130], nodes with high degree usually have small radii and is efficient to be selected as the minimum node, i.e. u_{min}, of a connected component. In our case, instead of computing the degree of each URI (that requires an additional *MapReduce* job and to store and read the degree of each URI in each step) we can use the `SameAsPrefixIndex`. In Figure 4.3, we can see an example of `SameAsPrefixIndex` for the `owl:sameAs` relationships which are shown in the left side of the figure. The `SameAsPrefixIndex` is defined as follows:

`SameAsPrefixIndex`: It is essentially a function $sp : SAP(\mathcal{D}) \rightarrow \mathbb{Z}_{>0}$ where $SAP(\mathcal{D})$ is the set of prefixes of the URIs occurring in the `owl:sameAs` relationships in $\mathcal{D}$. Generally, a prefix is the initial part of a URI, which usually indicates it's provenance (e.g., *http://dbpedia.org/*). The `SameAsPrefixIndex` is defined as follows: $SAP(\mathcal{D}) = \{\, pre(u) \mid u \in (u, \texttt{owl} : \texttt{sameAs}, u'), D_i \in \mathcal{D}\} \cup \{\, pre(u) \mid u \in (u', \texttt{owl} : \texttt{sameAs}, u), D_i \in \mathcal{D}\}$, and $\mathbb{Z}_{>0}$ is the set of positive integers.

Therefore, this index stores the frequency of each prefix in all the `owl:sameAs` relationships. For instance, in the example of Figure 4.3, we can observe that the prefix *dbp* exists in five `owl:sameAs` relationships, thereby, we store this information in `SameAsPrefixIndex`.



owl:sameAs relationships

dbp:Michael_Jordan `owl:sameAs` yg:Michael_Jordan
dbp:Michael_Jordan `owl:sameAs` nyt:jordan_michael
dbp:Aristotle `owl:sameAs` yq:Aristotle
dbp:Texas `owl:sameAs` geo:Texas
dbp:Las_Vegas `owl:sameAs` en_wiki:Las_Vegas
en_wiki:Las_Vegas `owl:sameAs` geo:Las_Vegas

SameAs Prefix Index

SameAs Prefix	Frequency
http://dbpedia.org/ (**dbp**)	5
http://yago-knowledge.org/ (**yg**)	2
http://data.nytimes.com/ (**nyt**)	2
http://en.wikipedia.org/ (**en_wiki**)	2
http://geonames.org/ (**geo**)	1

Figure 4.3: Example of `SameAsPrefixIndex`

By exploiting this index, we use the following heuristic rules in order to foresee the "center" of the connected component: (a) select always the URI whose prefix occurs in the most `owl:sameAs` relationships (this information can be taken from `SameAsPrefixIndex`) (b) if more than two URIs contain a prefix that occur in the same number of `owl:sameAs` relationships, compare the URIs lexicographically. Therefore, by using this heuristic we suppose that a URI with a prefix that occurs in many `owl:sameAs` relationships will have a high degree.

Construction Method: For getting the prefixes that occur in `owl:sameAs` relationships and their frequency, we read each `owl:sameAs` pair once in order to get the prefixes and then we count in how many `owl:sameAs` pairs this prefix exists. It needs a single job while its communication cost is $O(n)$, where n is the number of prefixes occurring in all the `owl:sameAs` relationships.

Note. We can also create such an index for `owl:equivalentProperty` and `owl:equivalentClass` relationships, however, for our implementation there was no need to create such indexes, since the number of these relationships was small. For this reason, we are able to create the equivalence catalogs, `PropEqCat` and `ClEqCat`, by using a single machine.

4.4.3 Creation of Semantics-Aware RDF Triples

Rationale: The objective of this step is to create the set of semantics-aware triples, which were defined in §4.2.2. These triples will be exploited for constructing semantically enriched indexes. This step includes the replacement of each URI with its corresponding identifier by exploiting the equivalence catalogs, i.e., `EntEqCat`, `PropEqCat` and `ClEqCat` (i.e., see §4.2.2.1), and the conversion of each literal (i.e., see §4.2.2.2). Concerning the output of this algorithm, it replaces each triple of a dataset D_i with its corresponding semantics-aware triple by performing the above replacements and conversions (i.e., see §4.2.2.3). There exists three different types of triples according to the type of their object, i.e., (a) triples with a literal as object; (b) triples with a class as object; and (c) triples with an entity as object. As we will see, for the third type of objects, an additional *MapReduce* job will be performed. The resulted real world triples (of each dataset) for our running example, is shown in the left side of Figure 4.1.

Parallelization Overview. We use a partition of triples $TR = \{tr_1, ..., tr_m\}$, where $tr_1 \cup \ldots \cup tr_m = T'$ (where $T' = \bigcup_{D_i \in \mathcal{D}} triples(D_i)$). Each machine m_i reads a subset of triples tr_i, and a subset of `EntEqCat`, and in the reducer each triple is replaced with its corresponding semantical-enriched triple. Since some triples need two steps to be transformed (i.e., it depends on object type), two *MapReduce* jobs are needed. After the execution of the aforementioned jobs, each machine will have a specific part of real world triples (rwt_i), i.e., the real world triples can be derived as $RWT = rwt_1 \cup \ldots \cup rwt_m$.

Construction Method. Algorithm 2 shows the steps for producing the set of real world triples. First, we read in parallel a set of triples and the `EntEqCat`, since their size is huge. On the contrary, since the number of properties and classes is low, we keep in memory the other catalogs, i.e., `PropEqCat` and `ClEqCat`, which contain the identifiers of each property and class, respectively. We check whether the input is a triple or an entry of the `EntEqCat` (see line 3), and then we check the object type. In the first case (see lines 4–5), i.e., the object is a literal, we replace the property with its identifier (by using `PropEqCat`), we convert each literal to lower case, and we remove its language type and its datatype (when such information is included). Afterwards, we put as a key the subject of the triple and as value the rest part of the triple along with the dataset ID (for preserving the provenance). For example, for the triple ⟨ex:Aristotle,d1:birthYear,384 BC^^xsd:Year⟩ of dataset D_1 (see Figure 4.1), we replace d1:birthYear with $P1$, we convert the literal into lower case, we remove its

ALGORITHM 2: Creation of Real World Triples.

```
Input: All triples and equivalence catalogs, EntEqCat, PropEqCat, and ClEqCat
Output: The set of Real World Triples
1  function Mapper (input = $tr_i \cup$ EntEqCat )
2      forall $inp \in input$ do
3          if $inp = \langle s, p, o\rangle, D_i \in tr_i$ then
4              if $o \in L$ then
5                  emit $(s, \{[p]_{pr}, o_{conv}, D_i\})$ ;        // PropEqCat used and literal
                   converted
6              else if $o \in C$ then
7                  emit $(s, \{[p]_{pr}, [o]_{cl}, D_i\})$ ;        // PropEqCat and ClEqCat used
8              else if $o \in E$ then
9                  emit $(s, \{[p]_{pr}, o, D_i\})$ ;                        // PropEqCat used
10         else if $inp = (u, [u]_e) \in$ EntEqCat then
11             emit $(u, [u]_e)$
12
13 function SubjectReducer (URI key, values = $list(\{[p]_{pr}, o, D_i\}), [key]_e$)
14     forall $v \in values$ do
15         if $(o \notin E)$ then
16             $t' \leftarrow \langle [key]_e, [p]_{pr}, o\rangle$
17             store $(t', D_i)$ ;          // All conversions finished.  $t' \in RWT(D_i)$
18         else
19             emit $(o, \{[key]_e, [p]_{pr}, D_i\})$ ;              // Object Replacement Needed
20     emit$(key, [key]_e)$
21
22 function ObjectReducer (URI key, values = $list(\{s', p', D_i\}), [key]_e$)
23     forall $v \in values$ do
24         $t' \leftarrow \langle s', p', [key]_e\rangle$
25         store $(t', D_i)$ ;              // All conversions finished.  $t' \in RWT(D_i)$
```

datatype (e.g., "384 BC"^^xsd:Year → "384 bc"), and finally we emit a tuple (ex:Aristotle, {P2,384 bc,D_1}). In the second case (see lines 6–7), i.e., the object is an RDF class, we use PropEqCat and ClEqCat for replacing the property and the object with their corresponding identifiers, while in the third case (see lines 8–9), we replace only the property, and we emit a key-value pair, having as a key the subject of the triple, and as a value the rest part of the triple. For instance, for the triple ⟨d2:Aristotle,rdf:type,d2:Gre_Philosopher⟩ we will emit the following key-value pair: (d2:Aristotle, {P5,C2,D_2}), while for the triple ⟨ex:Aristotle,d1:influences,ex:Marx⟩, the following key-value pair will be sent to the reducers: (ex:Aristotle, {P3,ex:Marx,D_1}). On the contrary, when the input is an entry of EntEqCat, we put as a key the URI, and as a value its corresponding class of equivalence, e.g., we create a tuple (ex:Aristotle, E1).

In a reduce function (see SubjectReducer in Algorithm 2), we just replace each URI occurring as a subject with its identifier. For instance, the tuple (ex:Aristotle, E1) and all the tuples having as subject ex:Aristotle, e.g., (ex:Aristotle, {P2,384 bc,D_1}), will be sent to the same reducer, therefore we can replace ex:Aristotle with E1, e.g., (E1,P2,384 bc,D_1). After replacing the URIs of the subjects with their corresponding identifier, for the triples containing literals or classes as objects, we can store their corresponding real world triple (and its provenance), i.e., $\langle E1, P1, 384\ bc\rangle, D_1$, since we have finished with all the conversions (see lines 15–17).

On the contrary, for the triples containing objects that belong to entities, we should also replace these URIs with their class of equivalence (see lines 18–19). For instance, after the execution of the first *MapReduce* job, the triple ⟨ex:Aristotle,d1:influences,ex:Marx⟩ has been transformed into $\langle E1, P3,$ex:Marx$\rangle$; however, we should also replace the URI ex:Marx with its corresponding identifier. In particular, we put as a key the object and as a value the rest part of the triple and the dataset ID, e.g., (ex:Marx, {E1,P3, D_1}), while we create again a tuple containing each URI and its corresponding identifier, e.g., (ex:Marx, E7) (see lines 19–20 of Algorithm 2). By using an other reduce function (see ObjectReducer in Algorithm 2), we replace the URIs occurring as objects with their identifier, and we store the real world triple (and its provenance), i.e., $\langle E1, P3, E7\rangle, D_1$.

Iterations and Communication Cost. In general, two *MapReduce jobs* are needed, where in the first job we transfer all the triples and all the entries of `EntEqCat` from the mappers to the reducers, i.e., communication cost is $O(|T'|+|\texttt{EntEqCat}|)$ ($T' = \bigcup_{D_i \in \mathcal{D}} triples(D_i)$), while in the second job, we transfer only the triples containing an entity as an object (and the `EntEqCat`), i.e., communication cost (i.e., number of key-value pairs that we send to the reducers) is $O(|T''|+|\texttt{EntEqCat}|)$, where $T'' = \{\langle s,p,o\rangle \mid \langle s,p,o\rangle \in T', o \in E\}$ (all the triples having an entity as an object).

4.5 The Set of Semantics-Aware Indexes $\mathcal{I}$

Here, we define the five semantically enriched indexes that we construct, i.e., $\mathcal{I}$ = {*Entity Index, Property Index, Class Index, Entity-Triples Index, Literals Index*}.

- *Entity Index*: it is a function $ei : RWE \rightarrow \mathcal{P}(\mathcal{D})$, where for a $[u] \in EID$, $ei([u]) = dsetsEnt_{\sim}(u, \mathcal{D})$, i.e., for each different real world entity, this index stores all the datasets where it occurs (see the *Entity Index* in Figure 4.1).
- *Property Index*: it is a function $pi : RWP \rightarrow \mathcal{P}(\mathcal{D})$,where for a $[p] \in RWP$, $pi([p]) = dsetsProp_{\sim}(p, \mathcal{D})$, i.e., it stores all the datasets where each different real world property occurs (see the *Property Index* in Figure 4.1).
- *Class Index*: it is a function $ci : RWC \rightarrow \mathcal{P}(\mathcal{D})$, where for a $[c] \in RWC$, $ci([c]) = dsetsClass_{\sim}(c, \mathcal{D})$, i.e., it stores the datasets where a real world class occurs (see the

Class Index in Figure 4.1).

- *Literals Index*: it is a function $li : LIT \rightarrow \mathcal{P}(\mathcal{D})$, where for a $l \in$, $li(l) = dsetsLit(l, \mathcal{D})$, i.e., it stores all the datasets where a converted literal occurs (see the *Literals Index* in Figure 4.1).
- *Entity-Triples Index*: it is a function $eti : EID \rightarrow list(RWP \rightarrow list((RWE \cup RWC \cup LIT) \rightarrow \mathcal{P}(\mathcal{D})))$, i.e., for a specific $t = \langle s, p, o\rangle \in triples(D_i), D_i \in B$, $eti([s], [p], [o]) = dsetsTr_{\sim}(t, \mathcal{D})$. Therefore, it stores all the datasets where a real world triple occurs (see the *Entity-Triples Index* in Figure 4.1).

For creating all these indexes, we will use the desired information from the semantically enriched triples (i.e., real world triples). At first, we introduce the *Entity-Triples Index*, and afterwards we show how to construct indexes for specific sets, i.e., literals, entities, classes and properties. In general, with $Left(r)$ we will denote the set of elements that occur in the left side of a binary relation. Finally, remind that the sets of *EID*, *PID* and *CID* denote the identifiers of real world entities, properties and classes, respectively.

Candidate Data Structure for each Index. We can identify two basic candidate data structures for such indexes: (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 [264], in which for each element we store the ID of the datasets (a distinct arbitrary number). A bitmap index keeps for each element e, a bit array of length $|\mathcal{D}|$, therefore its total size is $|e| * |\mathcal{D}|$ bits. An inverted index for each element of e keeps a posting list of dataset identifiers. If ap is the average size of the posting lists, then the total size is $|e| * ap * \log|\mathcal{D}|$ 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 $ap > \frac{|\mathcal{D}|}{\log|\mathcal{D}|}$.

Note. Since the average size of posting lists, for the real datasets that we use in the experiments is quite small (see [165, 168]), we decided to use an inverted index containing posting lists of dataset identifiers (instead of a bit array).

Single Machine versus a Cluster of Machines. We should note that in this dissertation, we show how to construct the semantics-aware indexes by using parallel *MapReduce* methods. In our past work, we have proposed methods for creating the indexes by using a single computer (for more details see [165]). However, we were unable to create the indexes for a large number of entities, since it was a prerequisite a) to load in memory the `EntEqCat`, and b) to send millions of ASK queries to a SPARQL endpoint. Indicatively, by using these methods, we needed approximately 7 hours for constructing a single index (i.e., for entities) for 172 million URIs and 658 million triples, whereas as we will see later in this chapter, by using the proposed parallel methods and 96 machines, we can create all the five indexes approximately in 81.5 minutes for 412 million URIs and 2 billion triples.

4.5.1 Constructing the Entity-Triples Index

Rationale. We construct this index for collecting all the available real world triples (and their provenance) for any real world entity (i.e., see §4.3.1).

Parallelization Overview. We will use a partition of real world triples $RWTR = \{rwt_1, ..., rwt_m\}$, where the set of real world triples can be derived as $RWT = rwt_1 \cup ... \cup rwt_m$. Each machine m_i reads a subset of real world triples rwt_i and each reducer collects all the real world triples for a subset of entities E'. After the execution of the aforementioned jobs, each machine will have a specific part of this index (eti_i), i.e., the entity-triples index can be derived as $eti = eti_1 \cup ... \cup eti_m$.

Construction Method. For each real world entity we create a multi-level index. This index stores in the first level an entity identifier (i.e., belonging in EID) and a pointer to a list of its (real world) properties (a set of PIDs), whereas in the second level each property points to a list of values. Finally, in the third level, it stores for a specific entity-property pair, all its values and the datasets (i.e., a posting list of dataset identifiers) where each different triple (entity-property-value) occurs. In the left side of Figure 4.1, we can see the *Entity-Triples* index of our running example. We selected to store together all the values of a property for a given entity, for enabling the comparison of the values of each property, e.g., in Figure 4.1, we can see that two datasets support that the birth date of "Socrates" is "470 bc" and one that is "471 bc". Such a design can be useful for data fusion algorithms, i.e., for comparing the conflicting values of a given subject and predicate (e.g., the birth date of a person) for deciding which value is the correct one [80].

It is worth noting that for finding fast all the available information for a specific entity, this index stores together all the triples for a specific entity, either if that entity occurs in a triple as a subject, or as an object, which means that some triples will be stored twice in the index, i.e., those having entities as objects. Moreover, we also store the position of an entity in the triple. Specifically, when an entity occurs in a triple as an object (see the entry for "Stagira" in *Entity-Triples* index of Figure 4.1), we add an $*$ in the property ID of such a real world triple. Consequently, we define this set of property IDs as $PID*$. It is essential to store such triples twice, for being able to answer queries like "Give me the all the triples for Aristotle" (i.e., for retrieving also triples where the entity "Aristotle" occurs as an object).

We use $eti(e)$ for denoting the entry of a specific entity e, e.g., in Figure 4.1, $eti(E1)$ is the entry for "Aristotle", we use $eti(e.p)$ for denoting the sub-entry containing all the values for a specific entity-property pair, e.g., $eti(E1.P3)$, is the sub-entry for the combination "Aristotle", "influences" while $eti(e.p.o)$ denotes the sub-entry that contain all the dataset for a specific combination of an entity-property-value, e.g., $eti(E1.P3.E3)$ is the sub-entry for the combination"Aristotle", "influences", "Kant".

The Steps of the Algorithm. Algorithm 3 describes the steps for producing this index. In the map phase, we read a set of real world triples and we emit a key-value pair consisting

ALGORITHM 3: Construction of *Entity-Triples* Index.

```
Input: Real World Triples
Output: Entity-Triples Index
1  function Entity-Triples Index-Mapper (input = $rwt_i$)
2      forall $\langle s, p, o\rangle, D_i \in rwt_i$ do
3          if $s \in EID$ then
4              emit $(s, \{p, o, D_i\})$
5          if $o \in EID$ then
6              emit $(o, \{p*, s, D_i\})$
7
8  function Entity-Triples Index-Reducer (Entity e, values = $list(\{p, k, D_i\})$)
9      $eti(e) \rightarrow \emptyset$
10     forall $v = p, k, D_i \in values$ do
11         if $(p \notin Left(eti(e)))$ then
12             $eti(e) \rightarrow eti(e) \cup \{p, \{\{k, \{D_i\}\}\}\}$
13         else
14             if $(k \in Left(eti(e.p)))$ then
15                 $eti(e.p.k) \rightarrow eti(e.p.k) \cup \{D_i\}$
16             else
17                 $eti(e.p) \rightarrow eti(e.p) \cup \{k, \{D_i\}\}$
18     store $eti(e)$
```

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

In the reduce phase (of Algorithm 3), we collect all the real world triples for a specific entity e, i.e., we construct the whole entry of e in the *Entity-Triples* index. We iterate over all the values, where each value contains a property p, the corresponding value k for that property and the dataset where the aforementioned information occurs. We first check (see line 11) whether p exists in the sub-entries of entity e. When it is false (see line 12), we create a sub-entry, i.e., $eti(e.p.k)$, and we add the dataset where this triple, i.e., $e.p.k$, occurs. For instance, suppose that we first read the $\{P3, E3, D_1\}$ for the entity $E1$. In such a case, we just create an entry where $eti(E1.P3.E3) = \{D_1\}$. When the property p exists in the sub-entry of e (see lines 13–17), we check whether the value k also exists in the sub-entry of $eti(e.p)$. In such a case (see lines 14–15), we just update the posting list of $eti(e.p.k)$. Suppose that for the entity $E1$, we read the following value: $\{P3, E3, D_2\}$. Since the entry $E1.P3.E3$ already exists, we just update its posting list, i.e., $eti(E1.P3.E3) = \{D_1, D_2\}$. Otherwise, we create a new sub-entry for $eti(e.p)$, where we add the value k and its provenance (see lines 16–17). For instance, if we read the value $\{P3, E7, D_2\}$, we have already created an entry for $E1.P3$; however it does not contain $E7$. For this reason, we create a new entry for storing also this information, i.e., $eti(E1.P3)$={{E3,{D_1, D_2}}, {E7,{D_2}}}. Finally, we can store the entry of a specific entity on disk (see line 18).

Iterations and Communication Cost. A single *MapReduce* job is needed, while all the real world triples are passed from the mappers to the reducers at least once, whereas the set of triples $RWT' = \{\langle s,p,o\rangle | \langle s,p,o\rangle \in RWT \text{ and } o \in EID\}$, which contains an entity in the object position, is transferred twice. Consequently, the communication cost (i.e., number of key-value pairs that we send to the reducers) is $O(|RWT| + |RWT'|)$.

4.5.2 Constructing Semantically Enriched Indexes for Entities, Properties, Classes and Literals

Rationale. We introduce the indexes for entities, properties, classes and literals. By using these indexes, the target is to make it feasible to find all the datasets where a specific entity, property, class and literal occurs (i.e., see §4.3.1).

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

Construction Method. For constructing the index of each different set of elements (entities, properties, etc.), one can use Algorithm 4. In particular, we read all the real world triples and we select the part(s) of the triple that we are interested in (see lines 3–16), e.g., for constructing the class index, we need only the objects referring to a real world class. In any case, we emit a key-value pair, having as key an element (e.g., a literal, a real world entity) and as value the dataset where the triple, containing this element, occurs. In the reducer, for any (inverted) index, we just create a posting list with dataset identifiers for a specific element. In particular, we read a set of dataset identifiers, and we just update the posting list of each element (see line 21). Finally, we store in the corresponding index the element and its posting list (see line 22).

Iterations and Communication Cost. For each index, one *MapReduce* job is needed; however, it is worth mentioning that all the five indexes (or any subset of them) can be constructed simultaneously by using one *MapReduce* job, since the input in any case is the real world triples. Regarding the communication cost (i.e., number of key-value pairs that we send to the reducers), it depends on the inverted index that we construct each time. In particular, for the *Entity Index*, we need to transfer each subject or object referring to a real world entity, i.e., we define the sum of these real world entities as $allRWE = |\{t \in rwt_i, D_i \in \mathcal{D} \mid t = \langle s,p,o\rangle, s \in EID\}| + |\{t \in rwt_i, D_i \in \mathcal{D} \mid t = \langle s,p,o\rangle, o \in RWE\}|$, and the communication cost is $O(allRWE)$. The notion is similar for the rest three indexes. Concerning *Property Index* , i.e., we define the sum of real world properties as $allRWP = |\{t \in rwt_i, D_i \in \mathcal{D} \mid t = \langle s,p,o\rangle, p \in PID\}|$, whereas the corresponding communication cost is $O(allRWP)$. As regards *Literals Index*, we define the sum of literals as $allLIT = |\{t \in rwt_i, D_i \in \mathcal{D} \mid t = \langle s,p,o\rangle, o \in LIT\}|$ and the communication cost as $O(allLIT)$, whereas for the *Class Index*, we define the sum of real world classes as $allRWC = |\{t \in rwt_i, D_i \in \mathcal{D} \mid t = \langle s,p,o\rangle, o \in CID\}|$, and the corresponding communication cost as $O(allRWC)$.

A problem derived by the above analysis, is that some real world entities, classes, properties and literals of the same Dataset D_i can be transferred to the reducers several times, which results to multiple key-value pairs that are exactly the same. One possible solution for decreasing the communication cost is a) to keep in the cache memory of each mapper the last n key-value pairs, that were previously transferred, and b) to send a key-value pair in a reducer only if the mentioned key-value pair cannot be found in cache memory.

4.5.3 Additional Indexes. Namespace Index

Except for the set of indexes $\mathcal{I}$, we also construct an inverted index for the prefixes, i.e., namespaces, which is defined as:

`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. $Pre(\mathcal{D}) = \{\, pre(u) \mid u \in U_i, D_i \in \mathcal{D}\}$ ($pre(u)$ is the prefix of URI u).

Generally, most data providers publish their data using prefixes that indicate their company or university (e.g., *DBpedia* URIs starts with prefix *http://dbpedia.org*), whereas the prefix in a schema element (a property or a class) indicates which ontology is used. Therefore, a `PrefixIndex` can be important for finding all the datasets that use either entities or schema elements from a specific dataset, ontology, etc. Moreover, it can also be exploited for greatly reduce the cost of finding common URIs, as it is described in [165].

Construction Method: It can be easily constructed by reading the URIs of each dataset once, as it is explained in [165, 171], by using either a single or a cluster of machines.

4.6 Comparison of Parallel Algorithms

Table 4.6 shows the number of iterations and the communication cost for each parallel algorithm. Regarding the number of iterations, the execution of Hash-to-Min algo-

ALGORITHM 4: Creation of a Semantically-Enriched Inverted Index ind_F for any set of elements.

```
Input: Real World Triples
Output: An inverted index for a set of specific elements
function Inverted Index-Mapper (input = $rwt_i$)
    forall $\langle s, p, o\rangle, D_i \in rwt_i$ do
        /*For constructing the Entity Index, include lines 4-7 */
        if $s \in EID$ then
            emit $(s, D_i)$
        if $o \in EID$ then
            emit $(o, D_i)$
        /*For constructing the Property Index, include lines 9-10*/
        if $p \in PID$ then
            emit $(p, D_i)$
        /*For constructing the Class Index, include lines 12-13*/
        if $o \in CID$ then
            emit $(o, D_i)$
        /*For constructing the Literals Index, include lines 15-16*/
        if $o \in LIT$ then
            emit $(o, D_i)$

function Inverted Index-Reducer (Element $t$, values = $\{D_i, D_j, \ldots, D_n\}$)
    $ind_F(t) \leftarrow \emptyset$
    forall $D_i \in values$ do
        $ind_F(t) \leftarrow ind_F(t) \cup \{D_i\}$
    store $ind_F(t)$
```

Table 4.6: Comparison of Parallel algorithms.

Algorithm	**Iterations Number**	**Communication Cost**	**Explanation**
Hash-to-Min (Closure)	$O(logn)$	$O(logn\|V\| + \|E\|)$	n: number of nodes in the largest component of a path graph, V: nodes, E: edges
Creation of *Real World Triples*	2	1st iteration: $O(\|T'\| + \|\texttt{EntEqCat}\|)$, 2nd Iteration $O(\|T''\| + \|\texttt{EntEqCat}\|)$	T': all the triples, T'': all the triples having entities as objects, `EntEqCat`: entity equivalence catalog
Creation of *Entity-based Triples Index*	1	$O(\|RWT\| + \|RWT'\|)$	RWT: real world triples, RWT': real world triples with entities as objects
Creation of *Entity Index*	1	$O(allRWE)$	$allRWE$: all the real world entities
Creation of *Property Index*	1	$O(allRWP)$	$allRWP$: all the real world properties
Creation of *Class Index*	1	$O(allRWC)$	$allRWC$: all the real world classes
Creation of *Literals Index*	1	$O(allLIT)$	$allLIT$: all the literals

rithm [203] can result to the most iterations, comparing to any other step, especially when we have large components containing path graphs. On the contrary, for the rest steps we need a single *MapReduce* job, except for the Creation of Real World Triples, where we need two iterations. Regarding the communication cost, for the Hash-to-Min algorithm, the communication cost can be increased if we have a large component of path graphs, and of course if we select as u_{min} which is not located near to the centre of the connected component. However, one solution is to exploit the heuristics which we proposed in §4.4.2 and in §4.4.2.1. Concerning the other steps, the most expensive step is the creation of real world triples, since we need to read each triple at least once (and at most two times), whereas we have to transfer to the reducers the `EntEqCat`. On the contrary, for the remaining indexes, we need to transfer either some parts of the triples (e.g., properties), or all the real world triples (for the *Entity-based Triples Index*). However, in the previous cases, we do not read and we do not transfer to the reducers the `EntEqCat`. In the section about experimental evaluation (in §4.7), we will see that the most time-consuming step is to create the set of real world triples, and we will see whether our optimizations can reduce the number of iterations and the communication cost of Hash-to-Min algorithm.

4.7 Experimental Evaluation - Efficiency

Here, we report the results concerning the computation of the closure of equivalence relationships and the construction of indexes, for quantifying the speedup obtained by the introduced algorithms and methods for over 2 billion triples and 400 LOD Cloud datasets.

4.7.1 Datasets Used.

In Table 4.7, we can see some basic metadata for 400 real datasets which are used in our experiments (belonging in 9 domains), i.e., it shows the number of datasets, triples, URIs, literals, unique subjects and unique objects for each domain (in descending order with respect to their size in triples), which were manually derived (i.e., for this set of 400 datasets, an RDF dump was provided). In particular, we have collected over 2 billion triples, 412 million URIs and 429 million literals manually through *datahub.io*, and 44 million of equivalence relationships collected from these 400 datasets and from the LinkLion webpage [179], i.e., we did not use any instance/schema matching tool. Moreover, the number of unique subjects (by taking into account all the triples) is 308 million, while the corresponding number for the unique objects is 691 million. Most datasets belong to the social network domain; however, most triples occur in cross-domain datasets (i.e., 48% of triples), while a large percentage of triples belong to datasets from publication domain (i.e., 33% of triples). Concerning entities and literals, again most of them occur in datasets from cross-domain and publications (i.e., 79.2% of entities and 86.5% of literals). Finally,

Table 4.7: Metadata of datasets which are used in the experiments.

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

the size of all the triples on disk is 251 GB, while the size of equivalence relationships is 3.45 GB. In `LODsyndesis` website (`http://www.ics.forth.gr/isl/LODsyndesis`) one can find the data which were used for performing the experiments, the code and guidelines for reproducing the results, and metadata for each of the 400 datasets that were used in the experiments.

4.7.2 Hardware & Details about the Implementation.

In our implementation, we used a cluster in okeanos cloud computing service [16], containing 12 machines, each of them having 8 cores, 8 GB main memory and 60 GB disk space (in total 96 cores, 96 GB main memory and 720 GB disk space). By using these 12 machines, we created 96 different virtual machines, each one having 1 core and 1 GB memory.

We used the previously mentioned set of 400 RDF datasets and the set of 44 million equivalence relationships. First, we used as input this set of equivalence relationships, and by using a cluster of machines and the algorithms described in this chapter, we computed their transitive and symmetric closure for inferring more relationships (i.e., we inferred over 73 million new `owl:sameAs` pairs, as we shall see in §5.8.2).

However note that even a single incorrect relationship can produce several erroneous inferred ones [171], e.g., suppose that there exists an erroneous relationship, such as the following one: ⟨ex:Aristotle,`owl:sameAs`, d2:Socrates⟩. Due to closure, we would infer that every URI referring to “Aristotle” is equivalent to any URI of “Socrates”. For tackling this issue, we checked the quality of the inferred relationships and we removed the erroneous ones in a semi-manually way. In particular, we supposed that an existing equivalence relationship is probably incorrect, if we inferred through this relationship that two or more URIs from the same dataset are equivalent. Indeed, we identified such cases programmatically (i.e., we found all the classes of equivalence that contain two or more URIs from the same dataset), we removed the incorrect relationships manually, and then we created the

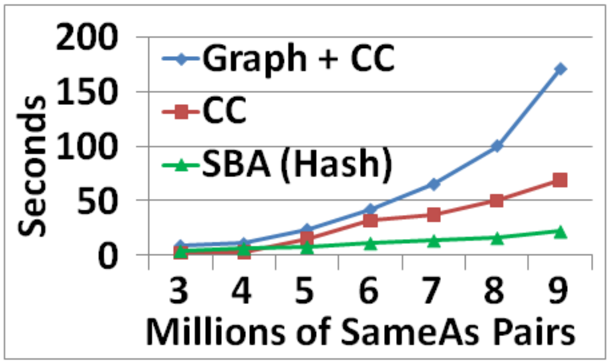


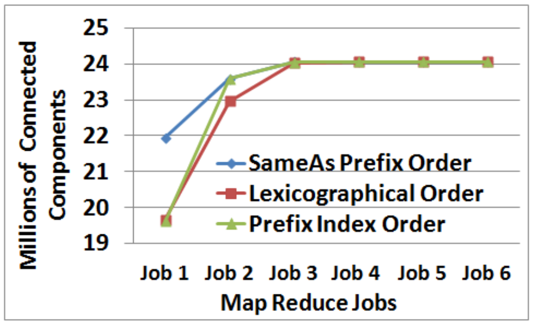
Figure 4.4: `EntEqCat` Catalog Construction time

final versions of equivalence catalogs. Afterwards, we used as input the triples of these 400 datasets and the equivalence catalogs for creating the semantics-aware triples and the set of indexes *I*, whose construction algorithms were described in this chapter. Below, we provide some results concerning efficiency, whereas in the next Chapters, we will show several statistics, connectivity analytics and services that were derived by exploiting the constructed indexes.

4.7.3 Efficiency of Cross-Dataset Identity Reasoning

4.7.3.1 Efficiency in a Single Machine

Here we report measurements that quantify the speedup obtained by the introduced techniques for computing the transitive and symmetric closure of equivalence relationships by using a single machine having an i5 core, 8 GB main memory and 1TB disk space. In particular, we compare the signature-based algorithm (*SBA*) versus Tarjan's connected components (*CC*) algorithm [233] that uses Depth-First Search (*DFS*) and was described in §4.4.1. We performed experiments for 3 to 9 million randomly selected `owl:sameAs` relationships and the results are shown in Figure 4.4. 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 `owl: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 [31] to overcome this limitation. The signature-based algorithm needed only 45 seconds to compute the closure of 13 million of `owl:sameAs` pairs.



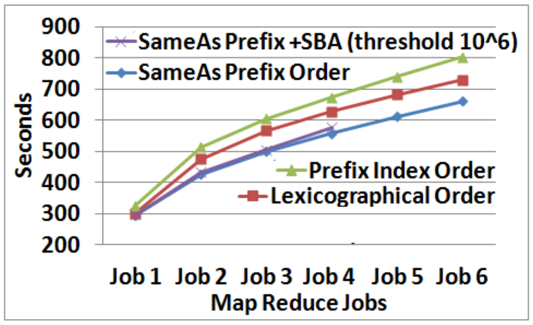


Figure 4.5:
Number of Connected Components having computed after the execution of each *MapReduce* Job per Different Order

Figure 4.6:
Total Execution time (in seconds) after the execution of each *MapReduce* Job per Algorithm Variation

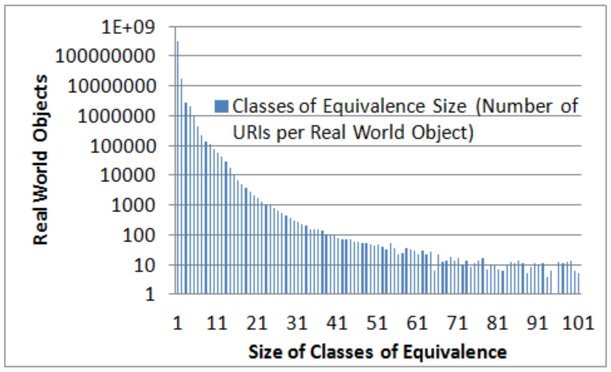


Figure 4.7: Number of Connected Components per Real world Object

4.7.3.2 Efficiency in a Cluster of Machines

Here, we report measurements performed by using the cluster in okeanos cloud computing service [16], which was described previously, by using Hash-to-Min algorithm (see §4.4.2). We should note that we evaluate the performance of the algorithm by using also two additional indexes for deciding the order, i.e., a `PrefixIndex` which contains 197,826 prefixes (and their provenance) and a `SameAsPrefixIndex` that includes 3,895 prefixes (and their frequency). Since both indexes are small in size, we are able to load these indexes in memory.

We compare the execution time of the creation `EntEqCat` and the number of connected components computed after each job by selecting different global rankings of the URIs: a) a lexicographical order b) an order by the frequency of each prefix in `SameAsPrefixIndex` and c) an order by the frequency of each prefix in `PrefixIndex` (where we take into account all the URIs and not only the URIs being part of `owl:sameAs` relationships as in the case of

`SameAsPrefixIndex`).

In Figure 4.5, we show the number of connected components that were computed after each *MapReduce* job by using different order for “foreseeing” the center of the connected component. By using `SameAsPrefixIndex`, Hash-to-Min algorithm was able to compute in the first job correctly approximately 22 millions of connected components while with the two other orders, it managed to compute approximately 19.5 millions of connected components. It is worth noting that the size of approximately 19 million connected components is two, i.e., there exists exactly two URIs for each of these 19 million real world objects. For instance, in the example of Figure 4.1, for the entity *Athens* there exists two URIs, therefore the size of the connected component of this entity is two, while for the entity *Aristotle* there exists three URIs, thereby, its connected component size its three. The computation of connected components having size two requires only one job (except for the initialisation job) regardless of the order that we use. However, the proposed order seems very effective comparing to the other two approaches, especially for the connected components containing three or more URIs.

After the second job, both algorithm variations that use the order by the frequency of each prefix in either `SameAsPrefixIndex` or `PrefixIndex` had computed approximately 23.5 million of connected components while the variation with the lexicographical order less than 23 million. In the remaining four jobs, there were a few number of connected components left, since most real world objects belong to a connected component having either size two or three as it can be observed in Figure 4.7.

Concerning the execution time, as we can see in Figure 4.6, the variations of the algorithm using the `SameAsPrefixIndex` are faster comparing to the other ones. This is justified since more connected components had been finished after the execution of each job. The best execution time achieved by combining the algorithm with the *Signature-Based* one, where the computation of connected components finished in less than 10 minutes in four *MapReduce* jobs. More specifically, when less than 1 million of URIs left, we used the *Signature-Based* for constructing the classes of equivalence for these URIs, instead of continuing to perform additional Map-Reduce jobs. Finally, the remaining variations finished in 6 jobs and needed 10-13 minutes.

4.7.4 Efficiency of Constructing the set of Semantics-aware Indexes $\mathcal{I}$.

Here, we report measurements that quantify the speedup obtained by the proposed methods for constructing the indexes. First, we report the execution time for computing the closure and for constructing the real world (RW) triples and the semantics-aware indexes. In Table 4.8, we can see the execution time for constructing the equivalence catalogs, the real world triples and the indexes by using 96 machines, the size of each index on disk and the number of its’ index entries.

Table 4.8: Construction time and size of catalogs and indexes.

Index/Catalog	Execution Time (96 Machines)	Size on Disk	Entries
Equivalence Catalogs	9.35 min	24 GB	413,567,083
Real World Triples	33.5 min	82.4 GB	1,826,224,504
Entity-Triples Index	17 min	70.3 GB	2,498,223,345
Entity Index	13.2 min	6 GB	368,295,245
Properties Index	5 sec	2.5 MB	247,713
Class Index	8 sec	6 MB	544,250
Literals Index	8.5 min	16 GB	379,043,131
All	81.55 min	198.7 GB	5,486,145,271

Table 4.9: Exact creation time by using different number of virtual machines (VMs)

Index	12 VMs	24 VMs	48 VMs	96 VMs
Equivalence Catalogs	43 min	23.7 min	13.4 min	9.3 min
Real World Triples	190 min	98 min	56.7 min	33.5 min
Entity-Triples Index	85 min	47 min	27 min	17 min
Entity Index	79 min	41 min	22 min	13.2 min
Literals Index	49 min	25 min	13.5 min	8.5 min
All	446 min	234.7 min	132.6 min	81.5 min

For constructing the equivalence catalogs, the real world triples and the *Entity-Triples* Index, 60 minutes are needed by using 96 machines, while we need additionally 21 minutes for constructing the other four indexes (for entities, literals, classes and properties), i.e., the execution time for performing all the jobs (in total 10 *MapReduce* jobs) is 81.55 minutes. The most time-consuming job is the creation of real world triples, where we replace all the URIs with an identifier, and we transform the literals. In particular, 33.5 minutes are needed for this job by using 96 machines, whereas the second most time-consuming job was to create the *Entity-Triples Index*, which has the most entries comparing to any other index.

Moreover, we managed to achieve scalability as it can be seen in Figure 4.8 and in Table 4.9. Specifically, we report the execution time for constructing the catalogs and indexes by using 12, 24, 48 and 96 machines. As we can observe, each time that we double the number of machines, the execution time is almost reduced in half in many cases, especially in the construction of real world triples. Generally, by using 96 (virtual) machines instead of 12 machines, we identified from $4.62\times$ to $6\times$ speedup.

We didn't manage to achieve the ideal speedup (i.e., $8\times$) due to the following reasons. Concerning the computation of cross-dataset identity closure, we did not achieve the maximum speedup, predominantly due to the existence of some large connected components. Concerning all the indexes, the key problem was that we used 96 virtual machines instead of real ones. Therefore, we expect that by using 96 real machines, the execution time can be further decreased.

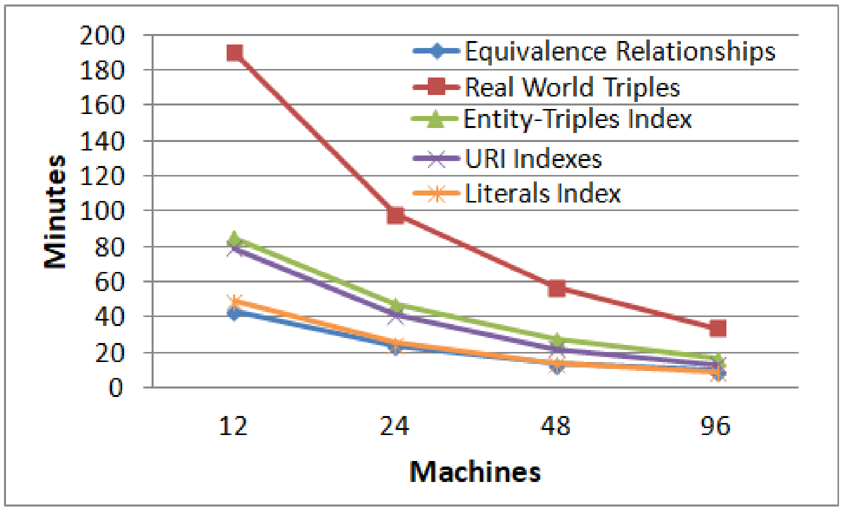


Figure 4.8: Creation time of indexes and catalogs for different number of machines.

4.7.5 Contents of Indexes

For storing the real world triples and the equivalence catalogs, which are the input for creating the indexes, we needed 106.4 GB. In particular, we created 1.82 billion real world triples and we needed 82.4 GB for storing them (and their provenance). Concerning `EntEqCat` it contains one entry for each unique URI and we needed 24 GB for saving it on disk. On the contrary, the size of `PropEqCat` is only 13.6 MB and of `ClEqCat` is 40.9 MB, since they contain only 247 thousand and 544 thousand entries, respectively.

The size of all the indexes on disk are 92.3 GB, where the size of *Entity-Triples* index, which contains all the triples (occurring either as a subject or as an object) for each unique entity is 70.3 GB. It is worth mentioning that approximately 672 million triples contain an entity as object, therefore these triples can be found twice in the Entity-Triples index. For this reason, this index contains approximately 2.5 billion triples, i.e., approximately 6.7 triples per entity. Concerning *Entity Index*, it contains 339 million entries (i.e., unique real world entities), and its size is 6 GB, which is less smaller than the size of `EntEqCat` since we have replaced each URI with a unique identifier, which is usually quite smaller than a URI (e.g., E1 instead of "http://www.dbpedia.org/reource/Aristotle"). Regarding *Literals Index*, it contains 379 million entries, however since some literals are quite large, the size of the index on disk is 16 GB. Finally, the rest two indexes, i.e., *Property Index* and *Class Index* are quite small.

4.8 Epilogue

In this chapter, we introduced algorithms for computing the cross-dataset identity reasoning among different datasets, i.e., the transitive and symmetric closure of equivalence relationships, either by using a single machine or a cluster of machines. Moreover, we introduced methods for creating in parallel semantics-aware indexes for different sets of elements, i.e., entities, triples, classes, properties and literals. By using these indexes, one can find and compare all the triples of a given entity from large number of datasets.

As regards the experiments, we showed that with the proposed algorithm for cross-dataset identity reasoning, it takes only 45 seconds to compute the transitive and symmetric closure for 13 million `owl:sameAs` relationships. However, since this algorithm is not highly scalable, by using the parallel algorithm that we proposed, we were able to compute the closure for 44 millions of `owl:sameAs` relationships in less than 10 minutes by using 96 virtual machines. Moreover, we managed to construct all the indexes for 2 billions of RDF triples from 400 real datasets in 81 minutes. Finally, we performed experiments by using different number of machines, which showed that the algorithms for performing cross-dataset identity reasoning and for constructing the semantics-aware indexes are scalable. In the next chapters, we will exploit all these indexes and equivalence catalogs for performing metrics among any subset of datasets and for creating several services, which are related to tasks A-E. Moreover, we will introduce much more statistics derived from these indexes and equivalence catalogs in §5.8, where we provide a wide range of connectivity analytics for 400 real RDF datasets.

Chapter 5
Content-based Intersection, Union and Complement Metrics Among Several Linked Datasets

In this chapter, our target is to propose methods for tackling *Challenge 3* , i.e., to provide *Dataset Discovery* services, by using content-based measurements among two or more datasets. Specifically, we focus on answering the research questions *RQ4*, *RQ5* and *RQ6*. By answering these research questions, we desire to provide services predominantly for the tasks of *Connectivity Analytics* and *Dataset Search, Discovery and Selection* (i.e., tasks B-C) and secondarily for the tasks of *Data Enrichment* and *Data Quality* (i.e., tasks D-E). In particular, our main focus is to answer the queries $Q_{connectivity}$, $Q_{coverage}$, $Q_{enrichment}$ and $Q_{uniqueness}$. For each of these queries, we provide a running example in Figure 5.1, which contains queries from scientists that desire to perform a study for endangered species (see the left part of Figure 5.1). Specifically, this example includes an *Entity Index* that contains seven species and an *Entity-Triples* Index which contains data for three different endangered species, “Tiger”, “Giant Panda” and “Snow Leopard”. Concerning the contributions of this chapter:

- We introduce content-based metrics (formalized as maximization problems) among any possible subset of datasets, which rely on intersection, union and complement, i.e., we introduce the metrics *commonalities*, *coverage*, *information enrichment* and *uniqueness*,
- we describe why plain SPARQL implementations are not enough for performing such measurements,
- we propose *lattice-based incremental algorithms* that exploit the posting lists of semantics-aware indexes, set theory properties and pruning and regrouping methods, for speeding up the computation of the intersection, union and complement among *any sub-*

set of datasets,

- we introduce methods for performing the lattice-based measurements in parallel even for trillions of lattice nodes,
- we measure the efficiency and the scalability of the proposed methods through comparative results,
- we report connectivity measurements for a subset of the current LOD Cloud that comprises 400 datasets and 2 billion triples.

The rest of this chapter is organized as follows: §5.1 introduces the problem statement, §5.2 shows the limitations of SPARQL implementations for computing these measurements, and §5.3 introduces the lattice of measurements and provides some notations that are required for the algorithms which are proposed. Moreover, in §5.4 we show how to compute the content-based metrics by exploiting the set of semantics-aware indexes $\mathcal{I}$, in §5.5 we introduce lattice-based algorithms for computing the metrics among any subset of datasets incrementally, whereas in §5.6 we show how to compute the metrics in parallel. §5.7 reports comparative results for all the metrics for evaluating the efficiency of the proposed algorithms, while §5.8 introduces connectivity analytics for 400 RDF real datasets. Finally, §5.9 concludes the chapter.

Publications related to this chapter. The work presented in this chapter has been published in [165, 168, 171, 174].

5.1 Problem Statement

Remind that $\mathcal{D} = \{D_1, ..., D_n\}$ be a set of datasets, $\mathcal{P}(\mathcal{D})$ denote the power set of $\mathcal{D}$, and B is any set of datasets $B \subseteq \mathcal{D}$. Let $\mathcal{F} = \{RWE, RWP, RWC, LIT, RWT, RWT_{E'}\}$ be the measurements types that we focus on. We shall use F to denote a specific measurement type ($F \in \mathcal{F}$), and $F(D_i)$ to denote the measurement type F applied to a dataset D_i, e.g., $RWE(D_i)$ correspond to all the entities that can be found in D_i. Our objective is to be able to answer the queries: $Q_{connectivity}$,$Q_{coverage}$, $Q_{enrichment}$ and $Q_{uniqueness}$ (introduced in Table 3.1) for any measurement type $F \in \mathcal{F}$. Each is a maximization problem that is defined formally below.

5.1.1 Commonalities

The answer of $Q_{connectivity}$ queries corresponds to $cmnBest(K, F)$ defined as:
Input: An integer K, where $2 \leq K < |\mathcal{D}|$ and a measurement type $F \in \mathcal{F}$.
Output: A subset $B \subseteq \mathcal{D}$, s.t. $|B| = K$, that maximizes the following criterion:
Maximization Criterion:

$$cmnBest(K, F) = arg_B \max\ |cmn(B, F)| \text{ where } cmn(B, F) = \cap_{D_i \in B} F(D_i) \quad (5.1)$$

Note. For answering queries such as $Q_{datConnectivity}$ for a given dataset D_i, it is a requirement that the subset B that maximizes the criterion $cmnBest(K,F)$ contains the dataset D_i.

5.1.2 Coverage

The answer of $Q_{coverage}$ queries corresponds to $covBest(K,F)$ defined as:
Input: An integer K, where $1 \leq K < |\mathcal{D}|$ and a measurement type $F \in \mathcal{F}$.
Output: A subset $B \subseteq \mathcal{D}$, s.t. $|B| = K$, that maximizes the following criterion:
Maximization Criterion:

$$covBest(K,F) = arg_B \max\ |cov(B,F)| \text{ where } cov(B,F) = \cup_{D_i \in B} F(D_i) \tag{5.2}$$

5.1.3 Information Enrichment

The answer of $Q_{enrichment}$ queries corresponds to $enrichBest(K,F,D_m)$ defined as:
Input: An integer K $(1 \leq K < |\mathcal{D}| - 1)$, a measurement type $F \in \mathcal{F}$ and a dataset D_m $(D_m \in \mathcal{D})$.
Output: A subset $B \subseteq \mathcal{D}$, s.t. $|B| = K$ and $D_m \notin B$, that maximizes the following criterion:
Maximization Criterion:

$$\begin{aligned} enrichBest(K,F,D_m) &= arg_B \max\ |enrich(B,F,D_m)| \text{ where} \\ enrich(B,F,D_m) &= cov(B,F) \setminus F(D_m) \end{aligned}$$

5.1.4 Uniqueness

The answer of $Q_{uniqueness}$ queries corresponds to $uniqBest(D_m,F,K)$, defined as:
Input: An integer K $(1 \leq K < |\mathcal{D}| - 1)$, a measurement type $F \in \mathcal{F}$, and a dataset D_m $(D_m \in \mathcal{D})$.
Output: A subset $B \subseteq \mathcal{D}$, s.t. $|B| = K$ and $D_m \notin B$, that maximizes the following criterion:
Maximization Criterion:

$$\begin{aligned} uniqBest(D_m,F,K) &= arg_B \max\ |uniq(D_m,F,B)| \text{ where} \\ uniq(D_m,F,B) &= F(D_m) \setminus cov(B,F) \end{aligned}$$

5.1.5 Difficulty and Challenge

From the above definitions we can see that the computation of $cmn(B,F)$, $cov(B,F)$, $enrich(B,F,D_m)$ and $uniq(D_m,F,B)$ is reduced to the computation of *intersection, union, absolute complement* and *relative complement*, respectively. However, even for a single subset of datasets B and a measurement type F, these set operations are quite expensive if B contains datasets whose $F(D_i)$ is very big. Moreover, the number of possible solutions (for finding the datasets that maximize the required formulas), can be exponential in number (specifically the possible solutions for a given K is given by the binomial coefficient formula: ${}_nC_K \equiv \begin{pmatrix} n \\ K \end{pmatrix} \equiv \frac{n!}{(n-K)!K!}$), which is prohibitively expensive for such maximization problems.

For tackling these issues, the objective is to solve the above maximization problems by reducing the number of set operations between different datasets for any measurement type F and subset of datasets B. For this reason, we propose a solution based on the set of dedicated indexes $\mathcal{I}$ (introduced in Chapter 4) for each measurement type F, and we use set theory properties and pruning and regrouping methods for further reducing the number of set operations, i.e., we reuse the measurements $|cmn(B,F)|$, $|cov(B,F)|$, $|enrich(B,F,D_m)|$ and $|uniq(D_m,F,B)|$, between two subsets of datasets B and B' (where $B \subset B'$).

5.2 Why plain SPARQL implementations are not enough

Here, we show how one can exploit SPARQL query language [200] for computing the metrics among any subset of datasets for real world entities, properties, classes, triples and literals. Suppose that there exists $|\mathcal{D}|$ datasets. At first, one should upload and store in a SPARQL endpoint, e.g., Virtuoso [89] or Blazegraph [2], all the triples of each dataset (a separate graph is needed for each dataset), and upload also all the equivalence relationships. By using SPARQL syntax, it is possible for one to write a query for computing the metrics for several subsets of datasets; however, each subset can contain exactly L datasets, e.g., $L = 2$ corresponds to pairs of datasets, $L = 3$ to triads of datasets, and so forth. For instance, for computing the measurements for all the pairs and triads of datasets, two different queries are needed, and for $|\mathcal{D}|$ datasets, there exists $|\mathcal{D}|+1$ such levels. Alternatively, one can use one SPARQL query for each different subset of datasets, i.e., $2^{|\mathcal{D}|}$ queries.

Below, we show nine SPARQL queries, for computing the metrics for properties, classes, entities, literals, and triples, respectively, which can be executed by using *OpenLink Virtuoso* database engine [89] or/and *Blazegraph* (`http://www.blazegraph.com`). Concerning *OpenLink Virtuoso*, one should follow some steps for enabling the computation of closure for schema elements on query time (`http://vos.openlinksw.com/owiki/wiki/VOS/VirtSPARQLReasoningTutorial\#Step4.:SettingUpInferenceRules`). For instance elements, one should put the command “define input:same-As “yes”” for enabling the computation of closure on query time for `owl:sameAs` relationships. On the contrary, *Blazegraph* does not support inference in the quads mode out of the box (more details are given here[1]). For avoiding the computation of transitive and symmetric closure on query time, one can exploit the semantics-aware triples that we have already constructed (which are described in Chapter 4).

[1]`https://wiki.blazegraph.com/wiki/index.php/InferenceAndTruthMaintenance`

5.2.1 Real World Properties

For computing the metrics for the real world properties, among all the combinations containing exactly L datasets, one should use the complex queries which are shown in Listings 5.1-5.4, where for $\mathcal{D}$ datasets, we need $|\mathcal{D}| - 1$ queries. Moreover, it is required to compute the closure of `owl:equivalentProperty` relationships on query time (for Virtuoso case), if we do not use the set of semantics-aware triples. In particular, for enabling the computation of closure on query time, in the query of Listing 5.1, we have defined an inference rule with name "SchemaEquivalence".

Listing 5.1: SPARQL Query for computing the cardinality of commonalities for F=RWP

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

Listing 5.2: SPARQL Query for computing the cardinality of coverage for F=RWP.

```
DEFINE input:inference ''SchemaEquivalence''
select ?Di ?Dj ... ?Dn (count (distinct ?property)
as  ?coverageOfProperties)
where {
{graph ?Di {?s ?property ?o}} union
{graph ?Dj {?s1 ?property ?o1}} union
...
{graph ?Dn {?sn ?property ?on}} union
filter(?Di>?Dj && ... && ?Dn-1>?Dn)}
group by ?Di ?Dj ... ?Dn
```

For all the queries, one should include a filter statement for denoting that each dataset is different with the other ones, otherwise, it will even compute the metric for the same dataset D_i, e.g., $|cmn(D_i, RWP)| \cap |cmn(D_i, RWP)|$. We use the > operator instead of the in-equivalence one (i.e., ! =), for not computing many times the metrics of a specific combination of datasets. For instance, for pairs of datasets, if we put $?D_i$!= $?D_j$ (instead of $?D_i > ?D_j$), the query will compute the metric of all the pairs of datasets twice, i.e., it will compute the metrics for all the possible permutations, e.g., for two datasets D_1 and D_2, it will compute the metrics for both orders $< D_1, D_2 >$ and $< D_2, D_1 >$. Moreover, a *group by* clause should be used, where each such group corresponds to a subset of datasets.

Listing 5.3: SPARQL Query for computing the cardinality of information enrichment for F=RWP and for a given dataset Dm

```
DEFINE input:inference ''SchemaEquivalence''
select ?Di ?Dj ... ?Dn (count (distinct ?property)
as ?enrichmentToDm)
where {
{graph ?Di {?s ?property ?o}} union
{graph ?Dj {?s1 ?property ?o1}} union
...
{graph ?Dn {?sn ?property ?on}} union
.FILTER NOT EXISTS{graph <Dm> {?sm ?property ?om}}
filter(?Di>?Dj && ... && ?Dn-1>?Dn )}
group by ?Di ?Dj ... ?Dn
```

Concerning the differences for the four metrics, for coverage (see Listing 5.2) the key difference is that we should use the “union” term instead of the dot “.”, which is used for the case of commonalities (see Listing 5.1). Concerning information enrichment (see Listing 5.3), we should also use term “union”, however, we should also include a “FILTER NOT EXISTS” statement, for not counting properties that belong to the dataset D_m (since we desire to find $enrich(B, F, D_m)$). Finally, for uniqueness (see Listing 5.4), we count only properties belonging in a dataset D_m but not in any other dataset, therefore, we add a “FILTER NOT EXISTS” statement for each other dataset (except for D_m).

Listing 5.4: SPARQL Query for computing the uniqueness of a dataset Dm for F=RWP.

```
DEFINE input:inference ''SchemaEquivalence''
select ?Di ?Dj ... ?Dn (count (distinct ?property)
as ?uniquenessOfDm)
where {
{graph <Dm> {?s property ?o}}
.FILTER NOT EXISTS {graph ?Di {?s ?property ?o}}
.FILTER NOT EXISTS {graph ?Dj {?s1 ?property ?o1}}
...
.FILTER NOT EXISTS {graph ?Dn {?sn ?property ?on}} union
filter(?Di>?Dj && ... && ?Dn-1>?Dn )}
group by ?Di ?Dj ... ?Dn
```

5.2.2 Real World Classes

Here, we show indicatively how to compute the commonalities for $F = RWC$. In particular, one should use the query which is introduced in Listing 5.5, where it is a requirement to compute the closure of `owl:equivalentClass` relationships on query time (i.e., for Virtuoso).

Listing 5.5: SPARQL Query for computing the cardinality of commonalities for F=RWC.

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

5.2.3 Real World Entities

Here, we show indicatively how to compute the commonalities for $F = RWE$, i.e., see Listing 5.6. For this query, we need a unique line for each dataset for finding the union of all its subjects and objects that are URIs (but not classes), which can be time-consuming. Finally, for *Virtuoso* case, one should put the command "define input:same-As "yes"" for enabling the computation of closure on query time for `owl:sameAs` relationships.

Listing 5.6: SPARQL Query for computing the cardinality of commonalities for F=RWE.

```

DEFINE input:same-As ''yes''

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

5.2.4 Literals

Here, we show indicatively how to compute the commonalities for $F = RWE$, i.e., see Listing 5.7.

Listing 5.7: SPARQL Query for computing the cardinality of commonalities for F=LIT.

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

5.2.5 Real World Triples

For the computation of metrics of real world triples, we need to compute the closure for finding the equivalences for both instance and schema relationships (which is computed on query time). Listing 5.8 shows the corresponding query for measuring the number of common triples among several datasets.

Listing 5.8: SPARQL Query for computing the cardinality of commonalities for F=RWT.

```
DEFINE input:inference ‘‘SchemaEquivalence’’
DEFINE input:same−As ‘‘yes’’
select ?Di ?Dj ... ?Dn (count (*) as ?commonTriples)
where {
{graph ?Di {?s ?p ?o}} .
{graph ?Dj {?s ?p ?o}} .
...
{graph ?Dn {?s ?p ?o}} .
filter(?Di>?Dj && ... && ?Dn−1>?Dn)
}
group by ?Di ?Dj ... ?Dn
```

5.2.6 Comparison of a SPARQL implementation with the proposed lattice-based incremental approaches.

The queries above can be exploited for performing such measurements, however, this approach has many limitations comparing to the proposed one. This set of limitations concern four different categories, i.e., **A. Computation of Closure**, **B. Indexes**, **C. Number of Joins**, and **D. Set theory Properties**. Below, we analyze each of these categories.

A. Computation of Closure. At first, by using *Virtuoso* the computation of closure is performed on query time, which can be time consuming. Therefore, for n queries, the closure will be computed n times. On the contrary, we have computed the closure of equivalence relationships once. Moreover, *Blazegraph* does not support inference in the quads mode. One possible solution for both tools is one to use the semantics-aware triples (where we have pre-computed the closure) and then to upload all the triples (including the inferred ones) to *Virtuoso*.

B. Indexes. The indexes offered by *Virtuoso* and *Blazegraph* are applicable for offering fast response to specific queries, such as for queries where only the subject or object is specified. For this reason, they offer indexes having as a primary key a subject, a predicate or an object, i.e., the indexing process of *Virtuoso* is explained here[2] and of *Blazegraph* in this link[3]. In this way, it is not so easy to have access to all the distinct literals, entities, etc. For instance, for the case of literals, the whole index having as key the objects should be scanned, which contains both literals and URIs. Therefore, the cost is increased, because such an implementation should check which objects are literals. On the contrary, our constructed indexes enable the fast access to the provenance of distinct entities, triples, literals, and schema elements.

C. Number of Joins For both SPARQL implementations, one (complex) query is needed for each different size of datasets combinations (e.g., pairs, triads), i.e., in total $|\mathcal{D}| + 1$ queries are needed for all the possible combinations of datasets, while as the number of datasets grow, such a query can be huge. Moreover, several joins (and comparisons) should be performed for each subset of datasets for computing the metrics. For example, suppose that we want to find the intersection of entities between a pair of datasets D_i and D_j and suppose that each dataset contains million of URIs and triples. For each of these two datasets, a SPARQL implementation scans all the triples and keeps the distinct URIs (that are not classes) occurring as a subject or an object. Afterwards, a join should be performed between the distinct URIs of these datasets, which is very time-consuming when the number of (distinct) URIs is large. On the contrary, we will use incremental algorithms taking as input the distinct posting lists of a specific index, whose size is very small comparing to the distinct URIs, in our indexes their size is less than 0.02%.

[2]`http://docs.openlinksw.com/virtuoso/rdfperfrdfscheme/`

[3]`https://wiki.blazegraph.com/wiki/index.php/About_Blazegraph`

D. Exploitation of Set Theory Properties. For both SPARQL implementations, each query computes the metrics for a fixed number of combinations of datasets, i.e., pairs, triads, etc. Therefore, it is not possible to exploit set theory properties, which hold between two or more subsets of datasets, B and B', where $B \subseteq B'$. On the contrary, we take into consideration set theory properties for reusing the measurements in an incremental way.

In §5.7, we introduce some indicative experiments containing the execution time for measuring the number of commonalities among several datasets, by using *Virtuoso* query engine, *Blazegraph*, and an incremental "lattice"-based approach (which is described later in this chapter).

5.3 The Lattice of Measurements by using the set of Semantics-aware Indexes $\mathcal{I}$

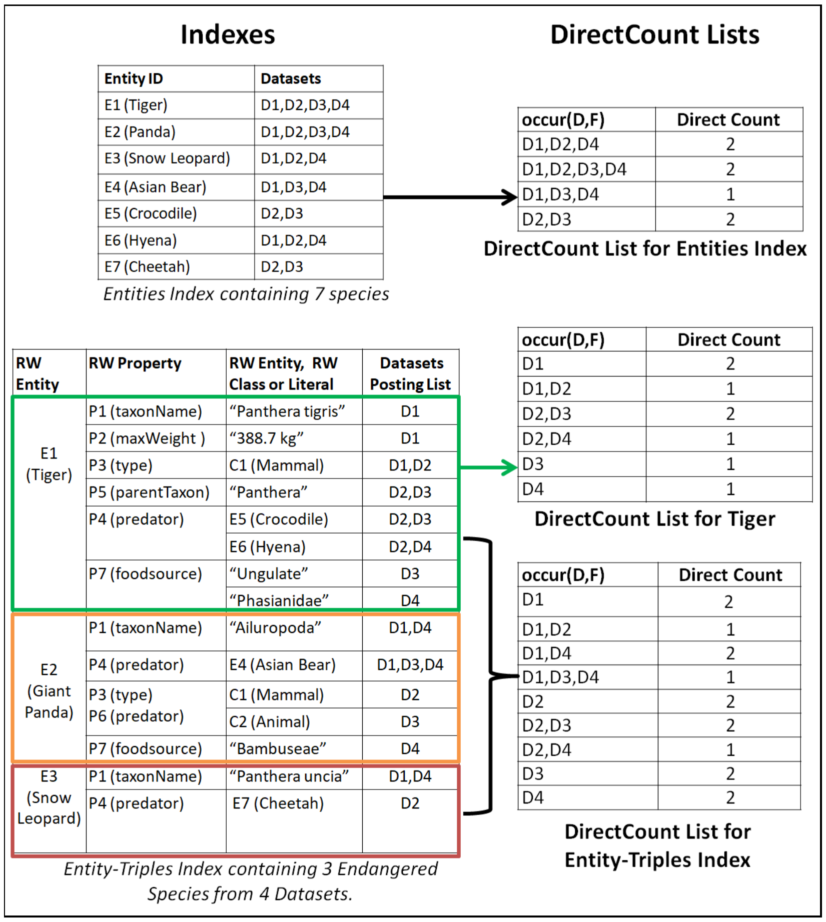
Our target is to use the semantics-aware indexes $\mathcal{I}$, for computing the metrics between any subset of datasets in $\mathcal{D}$. For (a) speeding up the computation of the metrics and (b) visualizing these measurements (for aiding understanding), we propose a method that is based on a lattice (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 as 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 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 $|V| = 2^{|\mathcal{D}|}$ nodes (see examples in Figures 5.2 and 5.3) , where each node corresponds to a subset of datasets $B \in \mathcal{P}(\mathcal{D})$, and $|E| = |\mathcal{D}| * 2^{|\mathcal{D}|-1}$ edges, where each edge points towards the direct supersets, i.e., an edge is created from a subset B to a superset B', where $B' = B \cup \{D_k\}, D_k \notin B$. Moreover, it contains $|\mathcal{D}|$+1 levels, and the value of each level L $(0 \leq L \leq |\mathcal{D}|)$ indicates the number of datasets that each subset of level L contains, e.g., $L = 2$ corresponds to pairs of datasets.

5.3.1 Direct Counts

Here, we show how to exploit an index $ind_F \in \mathcal{I}$ for computing the metrics for a measurement type F. Let $k \in Left(ind_F)$ be a single entry in the left part of the index ind_F, meaning that k is a key while $ind_F(k)$ corresponds to the value of this key (i.e., it is a posting list).

Below we define the number of times a subset B occurs in the posting lists of an index ind_F.



Entity ID	Datasets
E1 (Tiger)	D1,D2,D3,D4
E2 (Panda)	D1,D2,D3,D4
E3 (Snow Leopard)	D1,D2,D4
E4 (Asian Bear)	D1,D3,D4
E5 (Crocodile)	D2,D3
E6 (Hyena)	D1,D2,D4
E7 (Cheetah)	D2,D3

Entities Index containing 7 species

occur(D,F)	Direct Count
D1,D2,D4	2
D1,D2,D3,D4	2
D1,D3,D4	1
D2,D3	2

DirectCount List for Entities Index

RW Entity	RW Property	RW Entity, RW Class or Literal	Datasets Posting List
E1 (Tiger)	P1 (taxonName)	“Panthera tigris”	D1
	P2 (maxWeight)	“388.7 kg”	D1
	P3 (type)	C1 (Mammal)	D1,D2
	P5 (parentTaxon)	“Panthera”	D2,D3
	P4 (predator)	E5 (Crocodile)	D2,D3
		E6 (Hyena)	D2,D4
	P7 (foodsource)	“Ungulate”	D3
		“Phasianidae”	D4
E2 (Giant Panda)	P1 (taxonName)	“Ailuropoda”	D1,D4
	P4 (predator)	E4 (Asian Bear)	D1,D3,D4
	P3 (type)	C1 (Mammal)	D2
	P6 (predator)	C2 (Animal)	D3
	P7 (foodsource)	“Bambuseae”	D4
E3 (Snow Leopard)	P1 (taxonName)	“Panthera uncia”	D1,D4
	P4 (predator)	E7 (Cheetah)	D2

Entity-Triples Index containing 3 Endangered Species from 4 Datasets.

occur(D,F)	Direct Count
D1	2
D1,D2	1
D2,D3	2
D2,D4	1
D3	1
D4	1

DirectCount List for Tiger

occur(D,F)	Direct Count
D1	2
D1,D2	1
D1,D4	2
D1,D3,D4	1
D2	2
D2,D3	2
D2,D4	1
D3	2
D4	2

DirectCount List for Entity-Triples Index

Figure 5.1: Running example for four datasets $D_1 - D_4$ and two indexes.

Definition 1. $directCount(B, F) = |\{k \in Left(ind_F) \mid ind_F(k) = B\}| \diamond$

The value of $directCount(B, F)$ is computed by scanning the corresponding index (ind_F) once. The result of this process is a *directCount* list, where each entry contains in the left side a subset B and in the right side the *directCount* score of B. In Table 5.1, we can observe how to find $directCount(B, F)$ for any measurement type F, which are essential for performing the measurements of the desired measurement type (which were introduced in §5.1). For instance, in Figure 5.1, for finding the *directCount* list for the Entity Index, we scan that index and we compute how many times each subset of datasets exists in the right part of the index, e.g., the *directCount* value of $\{D_1, D_2, D_3, D_4\}$ is 2, since it exists two times in the index. Moreover, in Figure 5.1, one can see in the middle right side the *directCount* list for the entry of “Tiger” in the *Entity-Triples* index, and in the lower right side the *directCount* list for the whole *Entity-Triples* index. If m is the total number of entries of an index, then the time complexity for constructing that list is $O(m)$.

How to Use Only a Part of the *directCount* List? In many cases, i.e., for computing

Table 5.1: *directCount* of different measurement types.

Measurement Type F	**Which** $directCount(B, F)$ **to Use.**
RWE	$directCount(B, RWE) = \lvert\{[u] \in Left(ei) \mid ei([u]) = B\}\rvert$
RWC	$directCount(B, RWC) = \lvert\{[u] \in Left(ci) \mid ci([u]) = B\}\rvert$
RWP	$directCount(B, RWP) = \lvert\{[u] \in Left(pi) \mid pi([u]) = B\}\rvert$
LIT	$directCount(B, LIT) = \lvert\{l \in Left(li) \mid li(l) = B\}\rvert$
RWT	$directCount(B, RWT) = \lvert\{s.p.o \in Left(eti) \mid eti(s.p.o) = B\}\rvert$
$RWT_{E'}$	$directCount(B, RWT_{E'}) = \lvert\{s.p.o \in Left(eti) \mid eti(s.p.o) = B, s \in E' \text{ or } o \in E'\}\rvert$

$|cmn(B, F)|$, $|enrich(B, F, D_m)|$, $|uniq(D_m, F, B)|$ and for the incremental algorithms that will be introduced in this chapter, it is a prerequisite to exclude or to keep some posting lists containing specific datasets. For this reason, we introduce four definitions, while for each of them we show an example for the *directCount* list of *Entity Index* (see the upper right side of Figure 5.1).

First, for a given measurement type F, we define the posting lists (i.e., subsets of datasets) which contain at least one dataset $D_i \in B$, i.e., we define the set $occur(B, F)$ as follows:

Definition 2. $occur(B, F) = \{B_i \in \mathcal{P}(\mathcal{D}) \mid directCount(B_i, F) > 0, B_i \cap B \neq \emptyset\}$

Obviously, $occur(B, F)$ is a finite set. In the extreme case when $B = \mathcal{D}$, $occur(\mathcal{D}, F)$ contains all the entries in the left side of the *directCount* list for a given F, e.g., for the *Entity Index* of Figure 5.1, $occur(\mathcal{D}, F) = \{\{D_1, D_2, D_4\}, \{D_1, D_2, D_3, D_4\}, \{D_1, D_3, D_4\}, \{D_2, D_3\}\}$.

For the computation of $|cmn(B, F)|$, we will see that the target is to find all the posting lists that are supersets of B. For this reason, we define the set $Up(B, F)$ as follows:

Definition 3. $Up(B, F) = \{B_i \in \mathcal{P}(\mathcal{D}) \mid B \subseteq B_i, directCount(B_i, F) > 0\}$

Obviously, $Up(B, F)$ is a finite set. For instance in the *Entity Index* of Figure 5.1, if $B = \{D_1, D_2\}$, $Up(B, F) = \{\{D_1, D_2, D_4\}, \{D_1, D_2, D_3, D_4\}\}$.

In many cases, especially for computing $|enrich(B, F, D_m)|$ and for the incremental algorithms of §5.5, we should keep only the posting lists containing at least one $D_j \in B'$, but not any $D_i \in B$ (i.e., we exclude such posting lists). For this reason, we define the set $occur(B', F)_{\setminus B}$ as follows:

Definition 4. $occur(B', F)_{\setminus B} = occur(B', F) \setminus occur(B, F)$ ⋄

Obviously, $occur(B', F)_{\setminus B}$ is a finite set. For instance, in the *directCount* list which is shown in the upper right of Figure 5.1, if we want to exclude the posting lists containing D_3 (and to keep all the remaining ones), we will construct the set $occur(\mathcal{D}, F)_{\setminus\{D_3\}} = \{\{D_1, D_2, D_4\}\}$.

Finally, for computing $|uniq(D_m, F, B)|$ (in §5.4.4), we should keep all the posting lists containing the dataset D_m and at least one dataset $D_i \in B$, i.e., we define the set $occur(B, F)_{D_m}$ as follows:

Definition 5. $occur(B, F)_{D_m} = occur(B, F) \cap occur(\{D_m\}, F)$ $\diamond$

Obviously, $occur(B, F)_{D_m}$ is a finite set. For the *directCount* list of the *Entity Index* of Figure 5.1, if $B = \{D_1, D_3\}$ and $D_m = D_2$, then $occur(\{D_1, D_4\}, F)_{D_2} = \{\{D_1, D_2, D_4\}, \{D_1, D_2, D_3, D_4\}\}$, i.e., we keep a posting list if it contains D_2 and at least one dataset of B, i.e., D_1 or D_4.

5.3.2 Lemmas

Here, we provide some lemmas that are used for providing the proofs for the propositions in the rest of this chapter.

Lemma 1. If $k \in F(D_i)$, then $k \in Left(ind_F), ind_F(k) = B_i, D_i \in B_i$. In particular, since we store any element k and its provenance, k occurs also in the left side of the corresponding index, and its posting list $B_i \subseteq \mathcal{D}$ contains D_i.

Lemma 2. For a subset of datasets $B \in \mathcal{P}(\mathcal{D})$, if $\forall D_i \in B, k \in F(D_i)$, then $k \in Left(ind_F), ind_F(k) = B_i, B \subseteq B_i$. In particular, the posting list B_i of k contains at least all the datasets $D_i \in B$ (i.e., B_i is a superset of B) .

Lemma 3. If $B_i \neq B_j$ ($B_i \in \mathcal{P}(\mathcal{D}), B_j \in \mathcal{P}(\mathcal{D})$), then $\{k \in Left(ind_F) | ind_F(k) = B_i\} \cap \{k \in Left(ind_F) | ind_F(k) = B_j\} = \emptyset$, i.e., they are disjoint. It holds because each element $k \in Left(ind_F)$ has a unique posting list of dataset IDs.

Lemma 4. From the set theory, we know that for n disjoint sets $\{A_1, A_2, ..., A_n\}$, $|A_1 \cup A_2 \cup \ldots \cup A_n| = |A_1| + |A_2| + \ldots + |A_n|$ *(Proof can be found in [126]).*

Lemma 5. From the set theory, we know that for n sets $\{A_1, A_2, ..., A_n\}$, $|A_n \setminus \{A_1 \cup A_2 \cup \ldots \cup A_{n-1}\}| = |A_n| - |A_n \cap \{A_1 \cup A_2 \cup \ldots \cup A_{n-1}\}|$ *(Proof can be found in [126]).*

5.4 Content-Based Intersection, Union and Complement Metrics

Here we show how to compute the metrics by exploiting a *directCount* list in a trivial way, for computing *Commonalities* (in §5.4.1), *Coverage* (in §5.4.2), *Information Enrichment* (in §5.4.3) and *Uniqueness* (in §5.4.4).

5.4.1 Computation of Commonalities

Here, we show how to exploit a pre-constructed *directCount* list for computing the commonalities, i.e., the value $|cmn(B, F)|$, for any measurement type $F \in \mathcal{F}$.

Prop. 2. The key point is that the sum of the *directCount* of $Up(B,F)$ gives the intersection value of each subset B for a measurement type F.

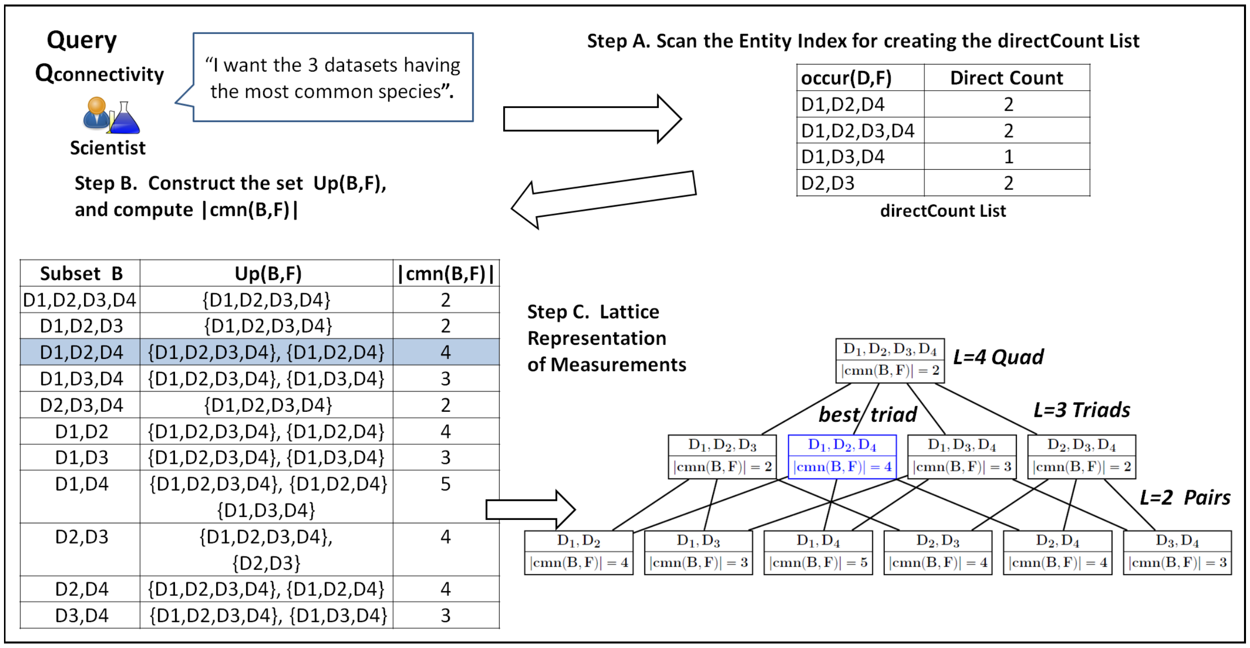
$$|cmn(B,F)| = \sum_{B' \in Up(B,F)} directCount(B',F) \tag{5.3}$$

$$\begin{aligned}
|cmn(B,F)| &= |\cap_{D_i \in B} F(D_i)| = |\cap_{D_i \in B} \{k \mid k \in F(D_i)\}| = |\{k \mid \forall D_i \in B, k \in F(D_i)\}| \\
&\overset{(Lemma\ 2)}{=} |\{k \in Left(ind_F) \mid ind_F(k) = B_i, B \subseteq B_i\}| \\
&\overset{(Lemma\ 1)}{=} |\cup_{B_i \in \mathcal{P}(\mathcal{D})} \{k \in Left(ind_F) | ind_F(k) = B_i, B \subseteq B_i\}| \\
&\overset{(Def.\ 3)}{=} |\cup_{B_i \in Up(B,F)} \{k \in Left(ind_F) | ind_F(k) = B_i\}| \\
&\overset{(Lemma\ 3)}{=} |\{k \in Left(ind_F) | ind_F(k) = B_1\} \cup \ldots \cup \{k \in Left(ind_F) | ind_F(k) = B_n\}| \\
&\overset{(Lemma\ 4)}{=} |\{k \in Left(ind_F) | ind_F(k) = B_1\}| + \ldots + |\{k \in Left(ind_F) | ind_F(k) = B_n\}| \\
&\overset{(Def.\ 1)}{=} directCount(B_1,F) + \ldots + directCount(B_n,F) \\
&\overset{(Def.\ 3)}{=} \sum_{B_i \in Up(B,F)} directCount(B_i,F) \diamond
\end{aligned}$$

5.4.1.1 Baseline Model for Computing $|cmn(B,F)|$ (*BM* method).

For each different subset B, we iterate over the set $occur(\mathcal{D},F)$ once, for creating the set $Up(B,F)$. In particular, $\forall B_i \in occur(\mathcal{D},F)$, we check if $B \subseteq B_i$, and if it holds, we add B_i to $Up(B,F)$. Afterwards, we read the set $Up(B,F)$ once, for computing $|cmn(B,F)|$ (see Prop. 2). For computing the $|cmn(B,F)|$ for a finite set of subset of datasets $BV = \{B_i, ..., B_m\}$, the time complexity is $O(|BV| * |occur(\mathcal{D},F)|)$, since we traverse once the whole set $occur(\mathcal{D},F)$ for each $B \in BV$ (in the worst case $|BV| = 2^{|\mathcal{D}|}$). The space complexity is $O(|occur(\mathcal{D},F)|)$ (i.e., we keep in memory the *directCount* list).

Example. In Figure 5.2, we show the steps for computing the number of common species among all the different subsets of datasets (we exclude the measurements concerning single datasets), therefore $F = RWE$. In particular, we scan the *Entity Index* for creating the *directCount* list. The next step is for each subset B to scan the whole set $occur(\mathcal{D},F)$ for finding the set $Up(B,F)$ (in Step B of Figure 5.2), and we compute $cmn(B,F)$ by taking the sum of the *directCount* value of the $Up(B,F)$. For example, for $B = \{D_1, D_2, D_4\}$, we constructed the set $Up(\{D_1, D_2, D_4\}, F) = \{\{D_1, D_2, D_3, D_4\}, \{D_1, D_2, D_4\}\}$. By taking the sum of their *directCount* score, i.e., $directCount(\{D_1, D_2, D_3, D_4\}, F) = 2$ and $directCount(\{D_1, D_2, D_4\}, F) = 2$, we computed $|cmn(\{D_1, D_2, D_4\}, F)| = 4$. As we can see in the the results in Step C of Figure 5.2, the triad with the most common species for the scientist is $cmnBest(3,F) =$


Query
Qconnectivity
Scientist
"I want the 3 datasets having the most common species".
Step A. Scan the Entity Index for creating the directCount List
Step B. Construct the set Up(B,F), and compute |cmn(B,F)|
Step C. Lattice Representation of Measurements
L=4 Quad
L=3 Triads
L=2 Pairs
best triad
D_1, D_2, D_3, D_4 $|cmn(B,F)| = 2$
D_1, D_2, D_3 $|cmn(B,F)| = 2$
D_1, D_2, D_4 $|cmn(B,F)| = 4$
D_1, D_3, D_4 $|cmn(B,F)| = 3$
D_2, D_3, D_4 $|cmn(B,F)| = 2$
D_1, D_2 $|cmn(B,F)| = 4$
D_1, D_3 $|cmn(B,F)| = 3$
D_1, D_4 $|cmn(B,F)| = 5$
D_2, D_3 $|cmn(B,F)| = 4$
D_2, D_4 $|cmn(B,F)| = 4$
D_3, D_4 $|cmn(B,F)| = 3$

occur(D,F)	Direct Count
D1,D2,D4	2
D1,D2,D3,D4	2
D1,D3,D4	1
D2,D3	2

directCount List

Subset B	Up(B,F)	|cmn(B,F)|
D1,D2,D3,D4	{D1,D2,D3,D4}	2
D1,D2,D3	{D1,D2,D3,D4}	2
D1,D2,D4	{D1,D2,D3,D4}, {D1,D2,D4}	4
D1,D3,D4	{D1,D2,D3,D4}, {D1,D3,D4}	3
D2,D3,D4	{D1,D2,D3,D4}	2
D1,D2	{D1,D2,D3,D4}, {D1,D2,D4}	4
D1,D3	{D1,D2,D3,D4}, {D1,D3,D4}	3
D1,D4	{D1,D2,D3,D4}, {D1,D2,D4} {D1,D3,D4}	5
D2,D3	{D1,D2,D3,D4}, {D2,D3}	4
D2,D4	{D1,D2,D3,D4}, {D1,D2,D4}	4
D3,D4	{D1,D2,D3,D4}, {D1,D3,D4}	3

Figure 5.2: Query $Q_{connectivity}$. Computation of Commonalities

$\{D_1, D_2, D_4\}$, since these three datasets contain four species in common, while any other possible triad of sources share a smaller number of entities.

5.4.2 Computation of Coverage

Here, we show how to exploit a pre-constructed *directCount* list (i.e., the set $occur(\mathcal{D}, F)$ and the corresponding *directCount* scores) for computing coverage, i.e., $|cov(B, F)|$.

Prop. 3. The sum of the *directCount* score of all $B_i \in occur(B, F)$ gives the cardinality of coverage for a subset B and a given measurement type F:

$$|cov(B, F)| = \sum_{B_i \in occur(B,F)} directCount(B_i, F) \qquad (5.4)$$

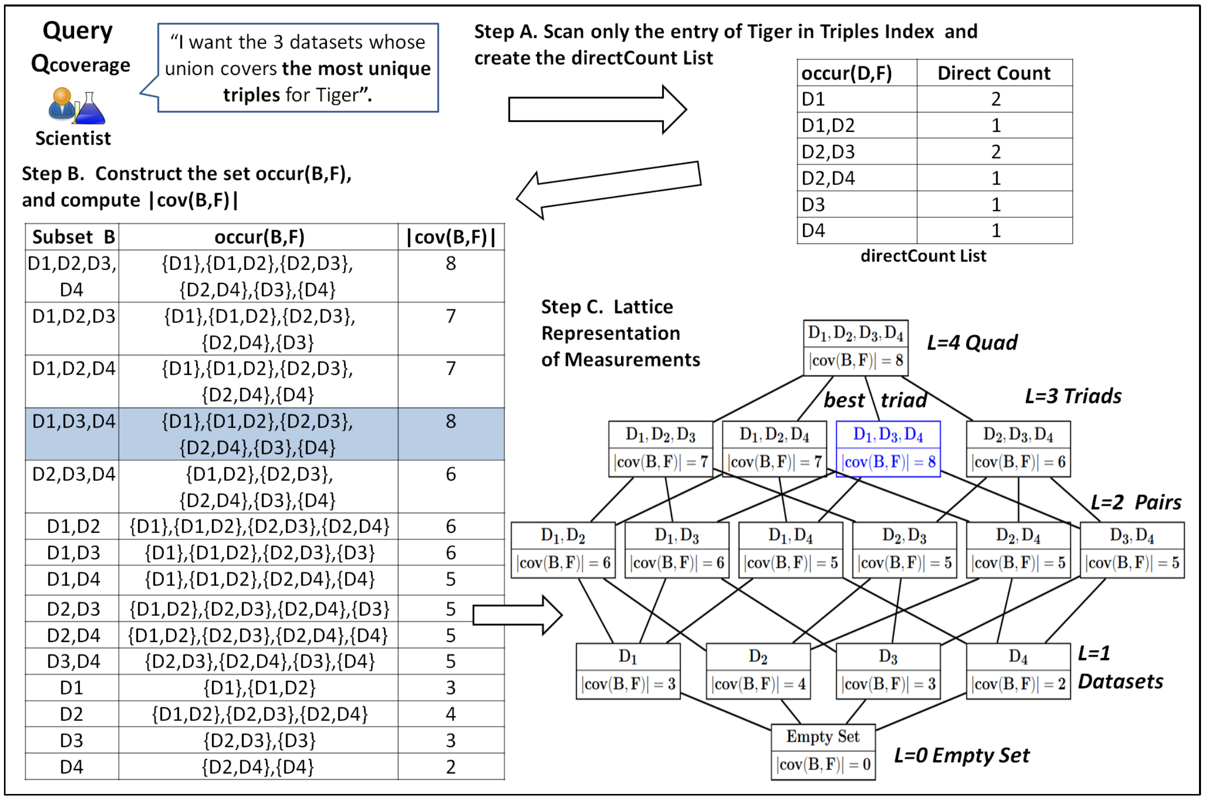
$$
\begin{aligned}
|cov(B,F)| &= |\cup_{D_i \in B} F(D_i)| = |\cup_{D_i \in B} \{k | k \in F(D_i)\}| \\
&\overset{(Lemma\ 1)}{=} |\cup_{B_i \in \mathcal{P}(\mathcal{D})} \{k \in Left(ind_F) | ind_F(k) = B_i, B \cap B_i \neq \emptyset\}| \\
&\overset{(Def.\ 2)}{=} |\cup_{B_i \in occur(B,F)} \{k \in Left(ind_F) | ind_F(k) = B_i\}| \\
&\overset{(Lemma\ 3)}{=} |\{k \in Left(ind_F) | ind_F(k) = B_1\} \cup \ldots \cup \{k \in Left(ind_F) | ind_F(k) = B_n\}| \\
&\overset{(Lemma\ 4)}{=} |\{k \in Left(ind_F) | ind_F(k) = B_1\}| + \ldots + |\{k \in Left(ind_F) | ind_F(k) = B_n\}| \\
&\overset{(Def.\ 1)}{=} directCount(B_1, F) + \ldots + directCount(B_n, F) \\
&\overset{(Def.\ 2)}{=} \sum_{B_i \in occur(B,F)} directCount(B_i, F) \diamond
\end{aligned}
$$

5.4.2.1 Baseline Model for Computing $|cov(B,F)|$ (*BM* method).

For each different subset B, we iterate over the set $occur(\mathcal{D}, F)$ once, for creating the set $occur(B,F)$. In particular, $\forall B_i \in occur(\mathcal{D}, F)$, we check if $B_i \cap B \neq \emptyset$, and if it holds, we add B_i to $occur(B,F)$. Afterwards, we read the set $occur(B,F)$ once, for computing $|cov(B,F)|$ (see Prop. 3). For computing the $|cov(B,F)|$, for a finite set of subset of datasets $BV = \{B_i, \ldots, B_m\}$, the time complexity is $O(|BV| * |occur(\mathcal{D}, F)|)$, since we traverse once the whole set $occur(\mathcal{D}, F)$ for each $B \in BV$ (in the worst case $|BV| = 2^{|\mathcal{D}|}$). The space complexity is $O(|occur(\mathcal{D}, F)|)$ (i.e., we keep in memory the whole *directCount* list).

Concerning the check $B_i \cap B \neq \emptyset$ it has time complexity $O(m * log_n)$ where $m = \max(|B_i|, |B|)$ and $n = \min(|B_i|, |B|)$, since both B_i and B_i are ordered sets. We ignore this complexity in $O(2^{|\mathcal{D}|} * |occur(\mathcal{D}, F)|)$, since even in the worst case (when $m = n = |\mathcal{D}|$), the cost is quite smaller comparing to the number of possible subsets ($2^{|\mathcal{D}|}$) and to the size of $occur(\mathcal{D}, F)$, i.e., in the extreme case when the index contains any $B \in \mathcal{P}(\mathcal{D})$ as a posting list, its size equals $2^{|\mathcal{D}|}$.

Example. In Figure 5.3, we show the steps for computing the coverage of triples for "Tiger" species (i.e., $F = RWT_{\{E1\}}$). For each subset of datasets B, we scan the whole set $occur(\mathcal{D}, F)$ (in Step A of Figure 5.3), for creating the set $occur(B,F)$, and we compute $|cov(B,F)|$ (in Step B in Figure 5.3). For example, for $B = \{D_1, D_2\}$, we constructed the set $occur(\{D_1, D_2\}, F) = \{\{D_1\}, \{D_1, D_2\}, \{D_2, D_3\}, \{D_2, D_4\}\}$. By taking the sum of their *directCount* score (i.e., $directCount(\{D_1\}, F) = 2$, $directCount(\{D_1, D_2\}, F) = 1$, $directCount(\{D_2, D_3\}, F) = 2$, $directCount(\{D_2, D_4\}, F) = 1$) we computed $|cov(\{D_1, D_2\}, F)| = 6$. By seeing the results in Step C of Figure 5.3, we observe that the best triad for the scientist is $covBest(3, F) = \{D_1, D_3, D_4\}$, since its union contains the maximum number of triples for "Tiger".


Query
Qcoverage
"I want the 3 datasets whose union covers the most unique triples for Tiger".
Scientist
Step A. Scan only the entry of Tiger in Triples Index and create the directCount List
Step B. Construct the set occur(B,F), and compute |cov(B,F)|
directCount List
Step C. Lattice Representation of Measurements
D_1, D_2, D_3, D_4 $|cov(B,F)| = 8$
L=4 Quad
best triad
L=3 Triads
D_1, D_2, D_3 $|cov(B,F)| = 7$
D_1, D_2, D_4 $|cov(B,F)| = 7$
D_1, D_3, D_4 $|cov(B,F)| = 8$
D_2, D_3, D_4 $|cov(B,F)| = 6$
L=2 Pairs
D_1, D_2 $|cov(B,F)| = 6$
D_1, D_3 $|cov(B,F)| = 6$
D_1, D_4 $|cov(B,F)| = 5$
D_2, D_3 $|cov(B,F)| = 5$
D_2, D_4 $|cov(B,F)| = 5$
D_3, D_4 $|cov(B,F)| = 5$
L=1 Datasets
D_1 $|cov(B,F)| = 3$
D_2 $|cov(B,F)| = 4$
D_3 $|cov(B,F)| = 3$
D_4 $|cov(B,F)| = 2$
Empty Set $|cov(B,F)| = 0$
L=0 Empty Set

occur(D,F)	Direct Count
D1	2
D1,D2	1
D2,D3	2
D2,D4	1
D3	1
D4	1

Subset B	occur(B,F)	|cov(B,F)|
D1,D2,D3, D4	{D1},{D1,D2},{D2,D3}, {D2,D4},{D3},{D4}	8
D1,D2,D3	{D1},{D1,D2},{D2,D3}, {D2,D4},{D3}	7
D1,D2,D4	{D1},{D1,D2},{D2,D3}, {D2,D4},{D4}	7
D1,D3,D4	{D1},{D1,D2},{D2,D3}, {D2,D4},{D3},{D4}	8
D2,D3,D4	{D1,D2},{D2,D3}, {D2,D4},{D3},{D4}	6
D1,D2	{D1},{D1,D2},{D2,D3},{D2,D4}	6
D1,D3	{D1},{D1,D2},{D2,D3},{D3}	6
D1,D4	{D1},{D1,D2},{D2,D4},{D4}	5
D2,D3	{D1,D2},{D2,D3},{D2,D4},{D3}	5
D2,D4	{D1,D2},{D2,D3},{D2,D4},{D4}	5
D3,D4	{D2,D3},{D2,D4},{D3},{D4}	5
D1	{D1},{D1,D2}	3
D2	{D1,D2},{D2,D3},{D2,D4}	4
D3	{D2,D3},{D3}	3
D4	{D2,D4},{D4}	2

Figure 5.3: Query $Q_{coverage}$. Computation of Coverage

5.4.2.2 Extra/Derived Coverage-related Metrics

Here we describe some extra coverage-related metrics.

Coverage Percentage. We define the coverage percentage of a subset of datasets B for a given F:

$$covPer(B, F, \mathcal{D}) = \frac{|cov(B, F)| * 100}{|cov(\mathcal{D}, F)|} \tag{5.5}$$

It can be used for answering questions, such as, "What percentage of all the available triples for "Tiger" species is covered from the union of DBpedia and YAGO datasets?". For example, in Figure 5.3, the $covPer(\{D_1, D_3, D_4\}, RWT_{\{E1\}}, \mathcal{D}) = 100\%$, since the union of these datasets contain all the available triples for "Tiger" (in total 8 triples), in our running example of Figure 5.1.

Coverage Gain. We define the coverage gain between a subset B and its superset B', $(B \subset B')$, for a given F as follows:

$$covGain(B', F, B) = \frac{(|cov(B', F)| - |cov(B, F)|) * 100}{|cov(B, F)|} \tag{5.6}$$

We measure whether there is an increase in coverage, if we add (i.e., integrate) more datasets to a given subset of datasets. For instance, suppose that in Figure 5.3 we want to measure the gain between $B = \{D_1, D_3, D_4\}$ and $B' = \{D_1, D_2, D_3, D_4\}$. Since $|cov(B, RWT_{\{E1\}})| = 8$ and $|cov(B', RWT_{\{E1\}})| = 8$, the coverage gain is $covGain(B', RWT_{\{E1\}}, B) = 0\%$. It means that for this task, the addition of D_2 to the subset $\{D_1, D_3, D_4\}$ is useless, i.e., D_2 does not offer additional unique triples for "Tiger".

5.4.3 Computation of Information Enrichment given a Dataset D_m

Here, we exclude any posting list containing D_m (since we want to compute the complement to a dataset D_m), i.e., we use as input the set $occur(\mathcal{D}, F)_{\setminus\{D_m\}}$ instead of $occur(\mathcal{D}, F)$.

Prop. 4. For a subset B, a dataset D_m and a given F, the sum of the *directCount* score of all the $B_i \in occur(B, F)_{\setminus\{D_m\}}$ gives the cardinality of information enrichment:

$$|enrich(B, F, D_m)| = \sum_{B_i \in occur(B,F)_{\setminus\{D_m\}}} directCount(B_i, F) \tag{5.7}$$

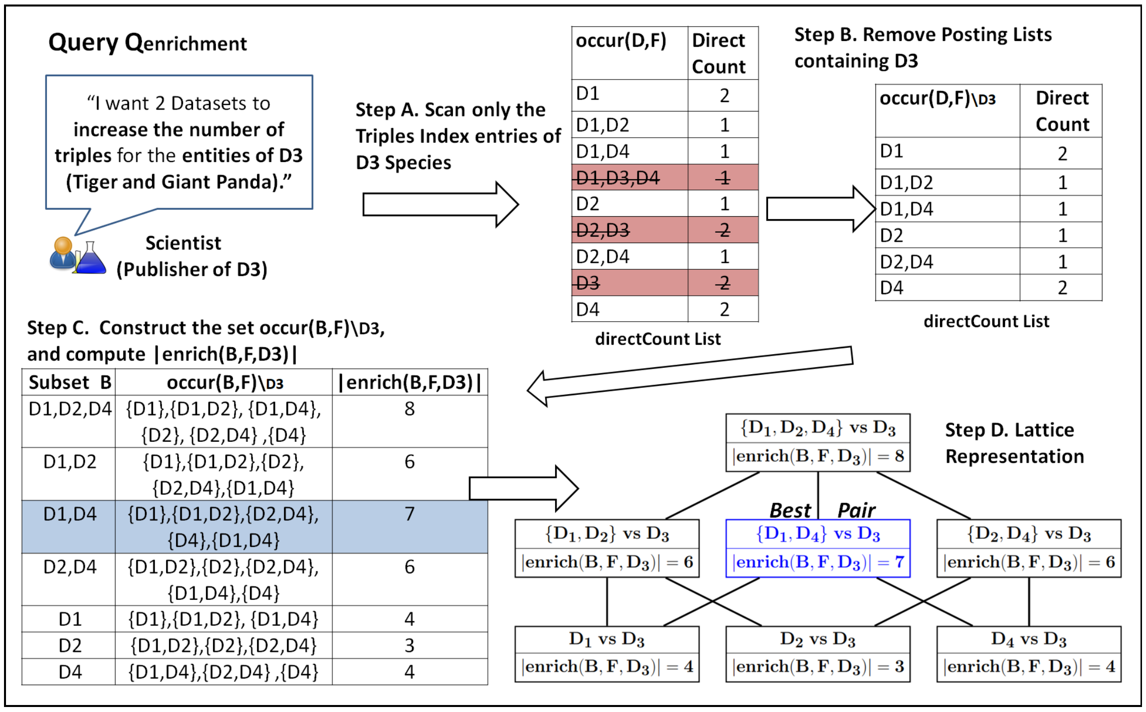
Proof.

$$\begin{aligned}|enrich(B, F, D_m)| &= |(cov(B, F) \setminus F(D_m))| = |\{\cup_{D_j \in B} F(D_j)\} \setminus F(D_m)| \\ &= | \cup_{D_j \in B} \{k | k \in F(D_j), k \notin F(D_m)\}| \\ &\overset{(Lemma\ 1)}{=} | \cup_{B_i \in \mathcal{P}(\mathcal{D})} \{k \in Left(ind_F) | ind_F(k) = B_i, B_i \cap B \neq \emptyset, D_m \notin B_i\}| \\ &\overset{(Def.\ 4)}{=} | \cup_{B_i \in occur(B,F)_{\setminus\{D_m\}}} \{k \in Left(ind_F) | ind_F(k) = B_i\}| \\ &\overset{(Similarly\ to\ Proof\ of\ Prop.3)}{=} \sum_{B_i \in occur(B,F)_{\setminus\{D_m\}}} directCount(B_i, F) \diamond\end{aligned}$$

□

5.4.3.1 Baseline Model for Computing $|enrich(B, F, D_m)|$ (*BM* method).

For a single subset B, we iterate over the set $occur(\mathcal{D}, F)_{\setminus\{D_m\}}$ once, for constructing the set $occur(B, F)_{\setminus\{D_m\}}$. In particular, for each $B_i \in occur(\mathcal{D}, F)_{\setminus\{D_m\}}$, we check if $B_i \cap B \neq \emptyset$, and if it holds, we add B_i to $occur(B, F)_{\setminus\{D_m\}}$. In the end, we sum the $directCount(B_i, F)$ score of each $B_i \in occur(B, F)_{\setminus\{D_m\}}$, for computing $|enrich(B, F, D_m)|$ (see Prop. 2). For a finite set of subset of datasets $BV = \{B_i, ..., B_m\}$, the time complexity is $O(|BV| * |occur(\mathcal{D}, F)_{\setminus\{D_m\}}|)$, where in the worst case $|BV| = 2^{|\mathcal{D}|-1}$, i.e., we compute $|enrich(B, F, D_m)|$ for all the possible subsets of datasets that do not contain D_m. Moreover, the space complexity is $O(|occur(\mathcal{D}, F)_{\setminus\{D_m\}}|)$. For the complexity of checking non-null intersection, i.e., $B_i \cap B \neq \emptyset$, see §5.4.2.


Query Qenrichment
"I want 2 Datasets to increase the number of triples for the entities of D3 (Tiger and Giant Panda)."
Scientist (Publisher of D3)
Step A. Scan only the Triples Index entries of D3 Species
Step B. Remove Posting Lists containing D3
Step C. Construct the set occur(B,F)\D3, and compute |enrich(B,F,D3)|
Step D. Lattice Representation
$\{D_1, D_2, D_4\}$ vs D_3 $|enrich(B, F, D_3)| = 8$
Best Pair
$\{D_1, D_2\}$ vs D_3 $|enrich(B, F, D_3)| = 6$
$\{D_1, D_4\}$ vs D_3 $|enrich(B, F, D_3)| = 7$
$\{D_2, D_4\}$ vs D_3 $|enrich(B, F, D_3)| = 6$
D_1 vs D_3 $|enrich(B, F, D_3)| = 4$
D_2 vs D_3 $|enrich(B, F, D_3)| = 3$
D_4 vs D_3 $|enrich(B, F, D_3)| = 4$

occur(D,F)	Direct Count
D1	2
D1,D2	1
D1,D4	1
~~D1,D3,D4~~	~~1~~
D2	1
~~D2,D3~~	~~2~~
D2,D4	1
~~D3~~	~~2~~
D4	2

directCount List

occur(D,F)\D3	Direct Count
D1	2
D1,D2	1
D1,D4	1
D2	1
D2,D4	1
D4	2

directCount List

Subset B	occur(B,F)\D3	|enrich(B,F,D3)|
D1,D2,D4	{D1},{D1,D2}, {D1,D4}, {D2}, {D2,D4} ,{D4}	8
D1,D2	{D1},{D1,D2},{D2}, {D2,D4},{D1,D4}	6
D1,D4	{D1},{D1,D2},{D2,D4}, {D4},{D1,D4}	7
D2,D4	{D1,D2},{D2},{D2,D4}, {D1,D4},{D4}	6
D1	{D1},{D1,D2}, {D1,D4}	4
D2	{D1,D2},{D2},{D2,D4}	3
D4	{D1,D4},{D2,D4} ,{D4}	4

Figure 5.4: Query $Q_{enrichment}$. Computation of Information Enrichment

Example. In Figure 5.4, the publisher of D_3, desires to find two datasets containing additional information for D_3 endangered species (i.e., "Tiger" and "Giant Panda"). Therefore, we focus on $F = RWT_{\{E1,E2\}}$, since E1 corresponds to "Tiger" and E2 to "Giant Panda". For this reason, in Step A, we scan only their entries in the *Entity-Triples* index, i.e., green and orange boxes in Figure 5.1, for constructing the set $occur(\mathcal{D}, F)$. However, since our target is to find the complementary triples for D_3 entities, in Step B we exclude all the posting lists containing D_3, i.e., we construct the set $occur(\mathcal{D}, F)_{\setminus\{D_3\}} = \{\{D_1\}, \{D_1, D_2\}, \{D_1, D_4\}, \{D_2\}, \{D_2, D_4\}, \{D_4\}\}$, and we store their corresponding *directCount* score. By traversing the set $occur(\mathcal{D}, F)_{\setminus\{D_3\}}$, for each subset B, we create the set $occur(B, F)_{\setminus\{D_3\}}$, and we compute the corresponding $|enrich(B, F, D_3)|$, e.g., see Step C of Figure 5.4. For instance, for computing $|enrich(\{D_1, D_2\}, F, \{D_3\})|$, we constructed $occur(\{D_1, D_2\}, F)_{\setminus\{D_3\}} = \{\{D_1\}, \{D_1, D_2\}, \{D_1, D_4\}, \{D_2\}, \{D_2, D_4\}\}$ and we computed the sum of their *directCount* score, for finding $|enrich(\{D_1, D_2\}, F, \{D_3\})| = 6$. In Step D of Figure 5.4, we observe that the best pair is $enrichBest(2, F, D_3) = \{D_1, D_4\}$, since it offers 7 additional triples for the species of D_3.

5.4.3.2 Extra/Derived Information Enrichment related Metrics

Here, we show one extra metric related to information enrichment.

Information Enrichment Percentage for a single dataset D_m. We define the increase percentage of information enrichment for a dataset D_m in a subset of datasets B, for a given F, as follows:

$$enrichPer(B, F, D_m) = \frac{|enrich(B, F, D_m)| * 100}{|cov(\{D_m\}, F)|} \tag{5.8}$$

This metric shows the percentage that the information of a given dataset D_m increases, in a possible integration with the datasets of subset B. For example, in Query $Q_{enrichment}$ of Figure 5.4, by integrating $\{D_1, D_2\}$ with D_3, $enrichPer(\{D_1, D_2\}, RWT_{\{E1,E2\}}, D_3) = 120\%$, since $\{D_1, D_2\}$ offers 6 new triples for the entities of D_3, i.e., $|enrich(\{D_1, D_2\}, RWT_{\{E1,E2\}}, D_3)| = 6$, which is divided by the total number of D_3 triples of these species, i.e., $|cov(\{D_3\}, RWT_{\{E1,E2\}})| = 5$.

5.4.4 Computation of Uniqueness given a Dataset D_m

Here, the input is only the set $occur(\{D_m\}, F)$ (and not $occur(\mathcal{D}, F)$), since we focus only on elements that can be found in D_m.

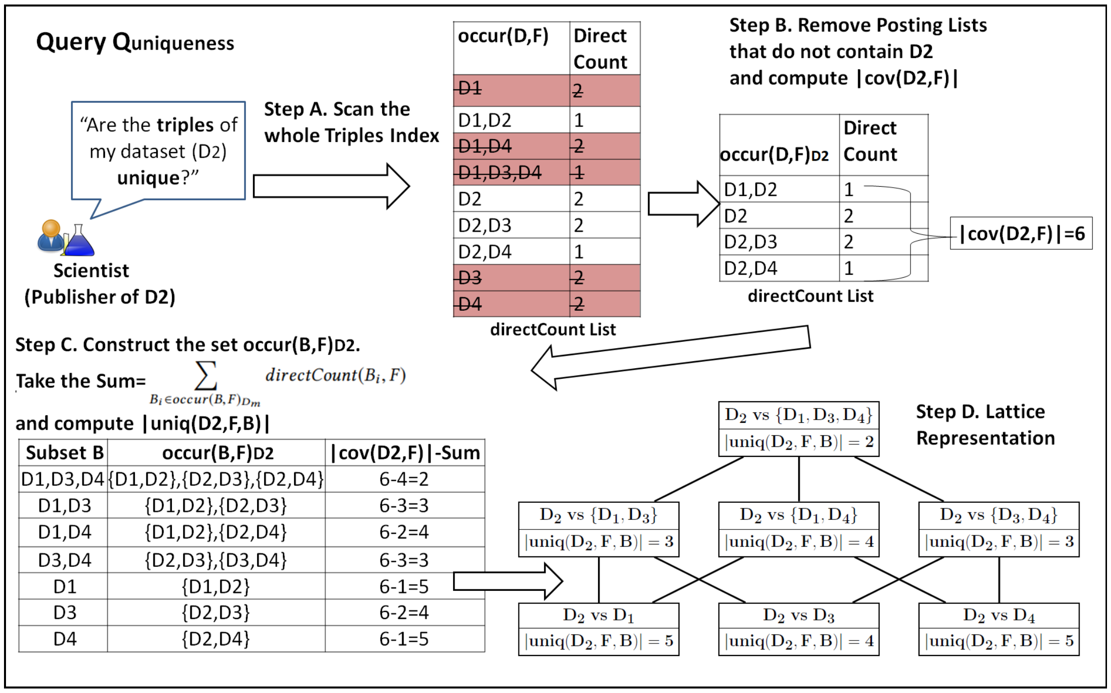
Prop. 5. For a subset B, a dataset D_m and a given F, the uniqueness of a dataset D_m to a subset B can be given by subtracting from the cardinality of all the elements of D_m, the cardinality of all the elements that co occur in D_m and in at least one $D_i \in B$.

$$|uniq(D_m, F, B)| = |cov(\{D_m\}, F)| - \sum_{B_i \in occur(B,F)_{D_m}} directCount(B_i, F) \tag{5.9}$$

$$\begin{aligned}
|uniq(D_m, F, B)| &= |(F(D_m) \setminus cov(B, F))| \overset{(Lemma\ 5)}{=} |F(D_m)| - |(F(D_m) \cap cov(B, F))| \\
&= |cov(\{D_m\}, F)| - |\cup_{D_j \in B} \{k | k \in F(D_j), k \in F(D_m)\}| \\
&= |cov(\{D_m\}, F)| - |\cup_{B_i \in \mathcal{P}(\mathcal{D})} \{k \in Left(ind_F) | ind_F(k) = B_i, D_m \in B_i, B \cap B_i \neq \emptyset\}| \\
&\overset{(Def.\ 5)}{=} |cov(\{D_m\}, F)| - |\cup_{B_i \in occur(B,F)_{D_m}} \{k \in Left(ind_F) | ind_F(k) = B_i\}| \\
&\overset{(Similarly\ to\ Proof\ of\ Prop. 3)}{=} |cov(\{D_m\}, F)| - \sum_{B_i \in occur(B,F)_{D_m}} directCount(B_i, F)
\end{aligned}$$

5.4.4.1 Baseline Model for Computing $|uniq(D_m, F, B)|$ (*BM* method).

First, we should compute $|cov(\{D_m\}, F)|$ by scanning the $occur(\{D_m\}, F)$ once, since it is required for the computation of $|uniq(D_m, F, B)|$, for any subset of datasets B (see Prop. 3). For a single subset B, we traverse the set $occur(\{D_m\}, F)$ once, for constructing the set $occur(B, F)_{D_m}$. For each $B_i \in occur(\{D_m\}, F)$, we check if $B_i \cap B \neq \emptyset$, and if it holds, we add B_i to $occur(B, F)_{D_m}$. In the end, we compute the sum of the $directCount(B_i, F)$, for each


Query Quniqueness
"Are the triples of my dataset (D2) unique?"
Scientist (Publisher of D2)
Step A. Scan the whole Triples Index
Step B. Remove Posting Lists that do not contain D2 and compute |cov(D2,F)|
|cov(D2,F)|=6
Step C. Construct the set occur(B,F)D2.
Take the Sum= $\sum_{B_i \in occur(B,F)_{D_m}} directCount(B_i, F)$
and compute |uniq(D2,F,B)|
Step D. Lattice Representation
D_2 vs $\{D_1, D_3, D_4\}$ $|uniq(D_2, F, B)| = 2$
D_2 vs $\{D_1, D_3\}$ $|uniq(D_2, F, B)| = 3$
D_2 vs $\{D_1, D_4\}$ $|uniq(D_2, F, B)| = 4$
D_2 vs $\{D_3, D_4\}$ $|uniq(D_2, F, B)| = 3$
D_2 vs D_1 $|uniq(D_2, F, B)| = 5$
D_2 vs D_3 $|uniq(D_2, F, B)| = 4$
D_2 vs D_4 $|uniq(D_2, F, B)| = 5$

occur(D,F)	Direct Count
~~D1~~	~~2~~
D1,D2	1
~~D1,D4~~	~~2~~
~~D1,D3,D4~~	~~1~~
D2	2
D2,D3	2
D2,D4	1
~~D3~~	~~2~~
~~D4~~	~~2~~

directCount List

occur(D,F)D2	Direct Count
D1,D2	1
D2	2
D2,D3	2
D2,D4	1

directCount List

Subset B	occur(B,F)D2	|cov(D2,F)|-Sum
D1,D3,D4	{D1,D2},{D2,D3},{D2,D4}	6-4=2
D1,D3	{D1,D2},{D2,D3}	6-3=3
D1,D4	{D1,D2},{D2,D4}	6-2=4
D3,D4	{D2,D3},{D3,D4}	6-3=3
D1	{D1,D2}	6-1=5
D3	{D2,D3}	6-2=4
D4	{D2,D4}	6-1=5

Figure 5.5: Query $Q_{uniqueness}$. Computation of Uniqueness

$B_i \in occur(B, F)_{D_m}$ Finally, we subtract this sum from the value $|cov(\{D_m\}, F)|$ for computing the $|uniq(D_m, F, B)|$ (see Prop. 3). For a finite set of subset of datasets $BV = \{B_i, \ldots, B_m\}$, the time complexity is $O(|BV| * |occur(\{D_m\}, F)|)$, where in the worst case $|BV| = 2^{|\mathcal{D}|-1}$ (we exclude the subsets containing D_m), and the space complexity is $O(|occur(\{D_m\}, F)|)$. For the complexity of checking non-null intersection ($B_i \cap B \neq \emptyset$) see §5.4.2.

Example. In Figure 5.5, the publisher of D_2 desires to know how unique are the triples of D_2 comparing to the other datasets. In Step A, we scan the whole *Entity-Triples* index of Figure 5.1 for creating the *directCount* list, i.e., we focus on $F = RWT$. However, we keep only the subsets of the *directCount* list containing the dataset D_2, i.e., we create the set $occur(\{D_2\}, F)$. In Step B of Figure 5.5, we compute the value $|cov(\{D_2\}, F)| = 6$, while in Step C, we construct the set $occur(B, F)_{D_2}$ and we compute $|uniq(D_2, F, B)|$ for each subset B. Finally, in Step D we show a visualization of the results. For example, for measuring the uniqueness of D_2 triples versus $\{D_1, D_3\}$, we created the set $occur(\{D_1, D_3\}, F)_{D_2} = \{\{D_1, D_2\}, \{D_2, D_3\}\}$, where the sum of the *directCount* score of the entries of $occur(\{D_1, D_3\}, F)_{D_2}$ equals 3. We subtracted that value from $|cov(\{D_2\}, F)|$ (which equals 6), for computing $|uniq(D_2, F, \{D_1, D_3\})| = 3$, i.e., there are 3 triples of D_2 that cannot be found either in D_1 or in D_3.

5.4.4.2 Extra/Derived Uniqueness-related Metrics

Here, we show two additional metrics related to uniqueness for a given dataset D_m.

Unique Content Percentage of a dataset D_m. We define the uniqueness of a dataset D_m for a given measurement type F, comparing to a subset of datasets B as follows:

$$uniqPer(D_m, F, B) = \frac{|uniq(D_m, F, B)| * 100}{|cov(\{D_m\}, F)|} \tag{5.10}$$

It can be used for answering questions like "What it the percentage of the triples of DBpedia, that cannot be found in Wikipedia, YAGO, and Freebase?". For example, in Query $Q_{uniqueness}$ of Figure 5.5, for $B = \{D_1, D_3\}$ the $uniqPer(D_2, RWT, \{D_1, D_3\}) = 50\%$, since 3 out of 6 triples of D_2 cannot be found in any of the datasets of B, i.e., neither to D_1 nor to D_3.

Unique Contribution of a Dataset D_m We define the "unique contribution" percentage that a dataset D_m offers to the "universe" for a measurement type F as follows:

$$uniqCont(D_m, F, \mathcal{D}) = \frac{|uniq(D_m, F, \mathcal{D})| * 100}{|cov(\mathcal{D}, F)|} \tag{5.11}$$

It can be used for answering queries, such as "What percentage of triples of "Tiger" species is offered only by DBpedia?". For example, for the"Tiger" species of Figure 5.1, dataset D_3 offers only one triple that cannot be found in any other dataset (i.e., the triple ⟨Tiger, foodSource, Ungulate⟩). In total, there are 8 available triples for "Tiger", thereby, the percentage is the following: $uniqCont(D_3, RWT_{\{E1\}}, \mathcal{D}) = 12.5\%$.

5.5 Incremental Computation of Intersection, Union & Complement Metrics

Since the possible subsets of datasets for solving the maximization problems (described in §5.1) can be exponential in number, it is very time-consuming (as we shall see experimentally in §5.7) to traverse the whole set $occur(\mathcal{D}, F)$ for each possible subset of datasets B, which is a requirement in the baseline method (BM) of §5.4 for all the metrics. For being able to solve such maximization problems, the target is to reduce the number of *directCount* entries that we read for each subset B. In this way, we propose incremental algorithms, based on set-theory properties and pruning methods, that reuse the measurements between two subsets of datasets B and B' (where $B \subset B'$). In particular, in §5.5.2, we propose two incremental algorithms and a pruning method for computing the commonalities, and in §5.5.3, we propose an incremental algorithm, and pruning methods for computing the coverage. Moreover, in §5.5.4 and §5.5.5, we show how to compute incrementally the information enrichment and uniqueness, respectively.

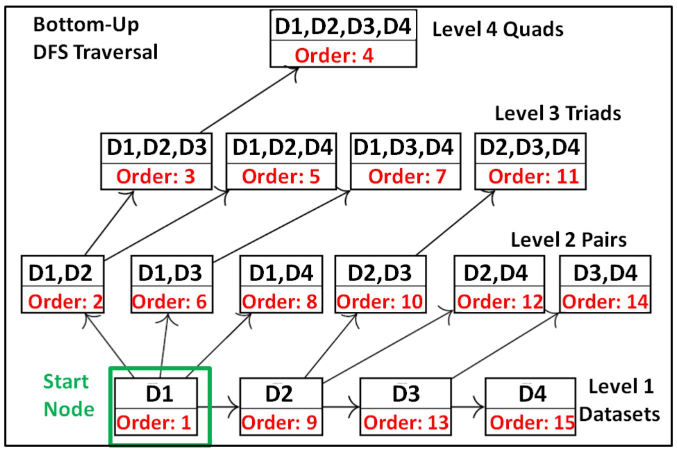


Figure 5.6: The bottom-up depth first search traversal

5.5.1 Lattice Traversals

Here, we introduce two different lattice traversals, a bottom-up depth-first search traversal, which will be used for all the four metrics, i.e., *Commonalities*, *Coverage*, *Information Enrichment* and *Uniqueness*, whereas we describe also a top-down breadth first search approach, which will be used in the case of *Commonalities*.

5.5.1.1 The Bottom-Up Depth-First Search (DFS) Traversal

Figure 5.6 shows the exact visiting order for four datasets, which is used for performing the measurements through the bottom-up approach. For visiting each lattice node (subset of datasets) once, we create a directed edge from B to B', only if $B = prev(B')$. In particular, it holds that $B = prev(B')$, only if $B' = B \cup \{D_k\}$ ($D_k \notin B$), and $\forall D_i \in B, k > i$, therefore, we follow a strict numerical ascending order. By following such a *DFS* traversal, we always go upwards (see Figure 5.6), i.e., we compute first the metrics for all the supersets of D_1, then for all the supersets of D_2 that do not contain D_1 and so on, and we create $|E| = |V| - 1$ directed edges (instead of $|E| = |\mathcal{D}| * 2^{|\mathcal{D}|-1}$). Due to this traversal, we visit and compute the metrics for any $B \in \mathcal{P}(\mathcal{D})$ at run time, i.e., there is no need to pre-construct a data structure for the lattice, whereas it allows us to stop the computation at any desired level L, e.g., for computing the metrics for triads of datasets, we can stop at level $L = 3$.

5.5.1.2 The Top-Down Breadth-First Search (BFS) Traversal

An alternative approach is to follow a top-down breadth-first search *BFS* traversal. As we can see in Figure 5.7, by using this traversal we start from the top of the lattice, and we compute the metrics for all the subsets of a specific level L, before continuing with the subsets of the previous level of L, i.e., $L_{prev} = L - 1$. Therefore, we first start with the level

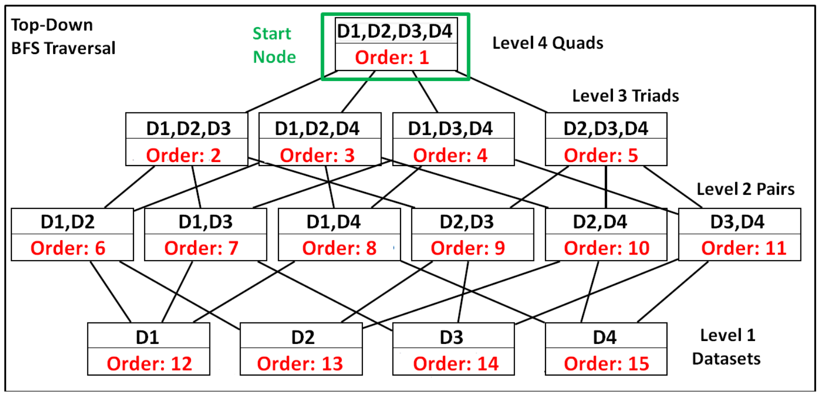


Figure 5.7: The top-down breadth first search traversal

$L = |\mathcal{D}|$, we compute the metrics and then we continue with the previous level, i.e., $|\mathcal{D}| - 1$, by creating all the edges between these two levels. Afterwards, we compute the metrics for all the nodes of level $|\mathcal{D}| - 1$, before continuing with the nodes of its' previous level, and so on. As in the case of *DFS* traversal, we visit and we compute the metrics for each node once, however, it is a requirement to create all the possible edges.

5.5.2 Lattice-Based Incremental Algorithms for computing $cmnBest(K, F)$.

Here, we introduce two different ways to compute the lattice measurements incrementally. The main notion is that we exploit the following property: the elements belonging to the intersection of a subset of datasets, is a superset of the elements that belong to the intersection of each of the supersets of this subset, as stated in the following proposition.

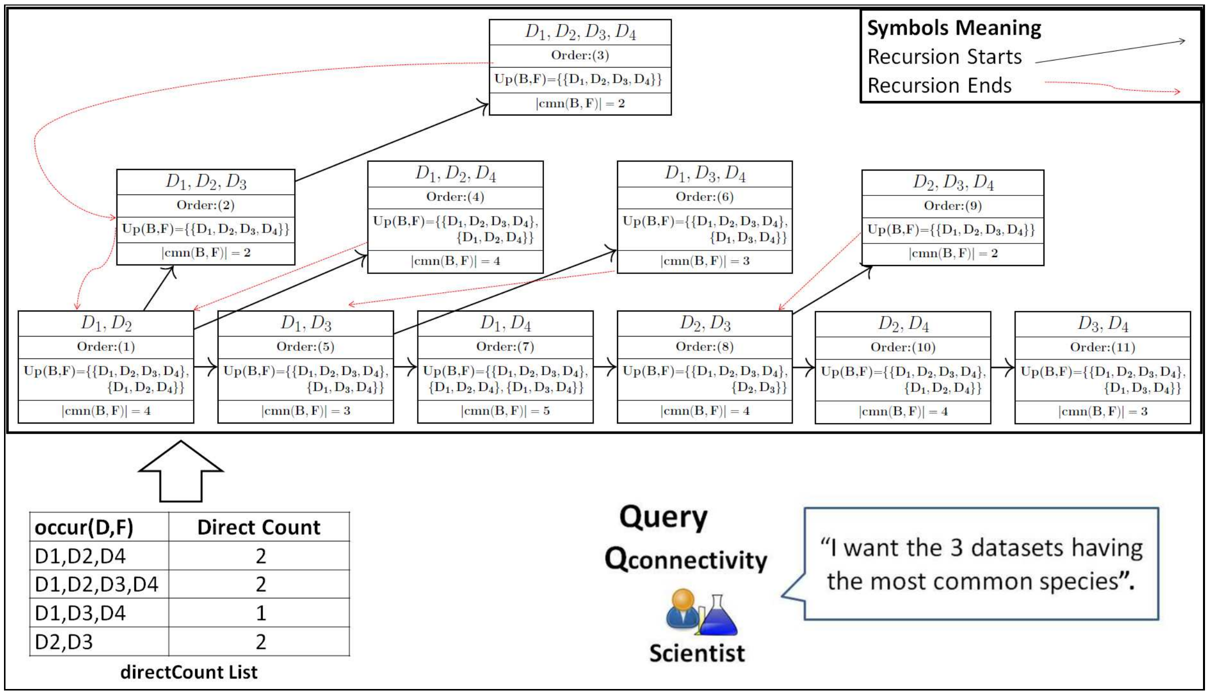
Prop. 6. *Let M and M′ be two families of sets. If M ⊆ M′ then* $\underset{S \in M}{\cap} S \supseteq \underset{S \in M'}{\cap} S$. *(The proof can be found in [126].)*

5.5.2.1 Lattice-Based Depth-First Search Approach for Intersection

Rationale. We show a bottom-up lattice-based (*LB*) incremental algorithm, that can be used for solving $cmnBest(K, F)$ for any measurement type F, for queries such as $Q_{connectivity}$.

Input. The input of Alg. 5 is the set $occur(\mathcal{D}, F)$, the corresponding $directCount(B_i, F)$ score for each $B_i \in occur(\mathcal{D}, F)$, and a parameter L, for stopping the computation of metrics, when level L is reached, e.g., for finding $cmnBest(K, F)$, $L = K$. In Figure 5.8, one can see the $occur(\mathcal{D}, F)$ and the corresponding *directCount* list (which was created by scanning the *Entity-Index* of this example).

Output. It computes at query time the $|cmn(B, F)|$ for all the subsets until a level L, e.g., if $L = |\mathcal{D}|$ it computes the metrics for all the possible subsets $B \in \mathcal{P}(\mathcal{D})$. For finding the


Symbols Meaning
Recursion Starts
Recursion Ends
D1, D2, D3, D4 Order:(3) Up(B,F)={{D1, D2, D3, D4}} |cmn(B, F)| = 2
D1, D2, D3 Order:(2) Up(B,F)={{D1, D2, D3, D4}} |cmn(B, F)| = 2
D1, D2, D4 Order:(4) Up(B,F)={{D1, D2, D3, D4}, {D1, D2, D4}} |cmn(B, F)| = 4
D1, D3, D4 Order:(6) Up(B,F)={{D1, D2, D3, D4}, {D1, D3, D4}} |cmn(B, F)| = 3
D2, D3, D4 Order:(9) Up(B,F)={{D1, D2, D3, D4}} |cmn(B, F)| = 2
D1, D2 Order:(1) Up(B,F)={{D1, D2, D3, D4}, {D1, D2, D4}} |cmn(B, F)| = 4
D1, D3 Order:(5) Up(B,F)={{D1, D2, D3, D4}, {D1, D3, D4}} |cmn(B, F)| = 3
D1, D4 Order:(7) Up(B,F)={{D1, D2, D3, D4}, {D1, D2, D4}, {D1, D3, D4}} |cmn(B, F)| = 5
D2, D3 Order:(8) Up(B,F)={{D1, D2, D3, D4}, {D2, D3}} |cmn(B, F)| = 4
D2, D4 Order:(10) Up(B,F)={{D1, D2, D3, D4}, {D1, D2, D4}} |cmn(B, F)| = 4
D3, D4 Order:(11) Up(B,F)={{D1, D2, D3, D4}, {D1, D3, D4}} |cmn(B, F)| = 3
Query
Qconnectivity
"I want the 3 datasets having the most common species".
Scientist
directCount List

occur(D,F)	Direct Count
D1,D2,D4	2
D1,D2,D3,D4	2
D1,D3,D4	1
D2,D3	2

Figure 5.8: Execution of Bottom-up Lattice-Based Incremental Algorithm for computing *cmnBest*

cmnBest(K, F) it computes the metrics until the level $L = K$, and it keeps in memory the subset having the $arg_B \max\ |cmn(B, F)|$.

The Initial Phase. Alg. 5 starts from the bottom part of the lattice, i.e., from the pairs (lines 1-3), e.g., in Figure 5.8, we start from the pair $\{D_1, D_2\}$. For each pair of datasets, we first find the $Up(B, F)$ by traversing the set $occur(\mathcal{D}, F)$ once (see lines 5-7). Afterwards, the recursive function *cmnLB* is called (line 8), which takes as parameters the subset of datasets that will be visited, the set $Up(B, F)$, and the stop level L.

Computation of Measurements for the Current Node & Exploration of Supersets Alg. 5 computes $|cmn(B, F)|$ by taking the sum of the *directCount* score of $Up(B, F)$, and prints (or stores) the result (see lines 10-11). Afterwards, the target is to visit the supersets of the current node. By following the *dfs* traversal, we visit a superset B' from a subset B (lines 12-19), e.g., see Figure 5.6. Since we know that $Up(B, F) \supseteq Up(B', F)$ (see Prop. 6), we check which entries of $Up(B, F)$ should be transferred to B', i.e., which of them belong also to $Up(B', F)$ (see line 17). In this way, we create the latter set (i.e., $Up(B', F)$) and we visit the superset B'. Thereby, a new recursive call starts, where we perform exactly the same steps. It is obvious that since $Up(B', F) \subseteq Up(B, F)$, the size of the input decreases as we continue upwards. For instance, in Figure 5.8, we first visit the pair $B = \{D_1, D_2\}$, we compute its' $|cmn(B, F)|$ (which is 4), and then we go upwards to the triad $B' = \{D_1, D_2, D_3\}$. From the set

ALGORITHM 5: Computing $|cmn(B,F)|$ incrementally for any possible subset of datasets B

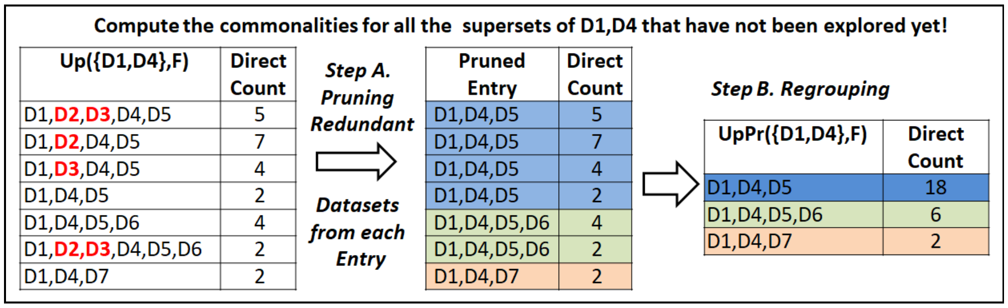
```
Input: The set $occur(\mathcal{D},F)$, the $directCount$ values, the stop Level $L$
Output: The commonalities $|cmn(B,F)|$, for each possible subset $B \in \mathcal{P}(\mathcal{D})$
forall $D_i \in \mathcal{D}$ do
    forall $D_j \in \mathcal{D}, i > j$ do                // Visit each pair of datasets once
        $B \leftarrow \{D_i, D_j\}$
        $Up(B,F) \leftarrow \emptyset$
        forall $B_i \in occur(\mathcal{D},F)$ do           // Traverse the $directCount$ list
            if $B \subseteq B_i$ then
                $Up(B,F) \leftarrow Up(B,F) \cup \{B_i\}$
        $cmnLB(B, Up(B,F), L)$

// The recursive function for computing the metrics incrementally
function $cmnLB(Subset\ B, Up(B,F), level\ L)$                        // Visit $B$
    $|cmn(B,F)| \leftarrow \sum_{B_i \in Up(B,F)} directCount(B_i,F)$
    print $B$, $|cmn(B,F)|$
    if $|B| < L$ then                              // Continue until the stop level
        forall $D_k \in \mathcal{D}$ do         // A strict numerical dfs order is followed
            $B' = B \cup \{D_k\}$          // Add one more dataset to $B$ ($B = prev(B')$)
            $Up(B',F) \leftarrow \emptyset$
            forall $B_i \in Up(B,F)$ do                // Traverse the set $Up(B,F)$
                if $B_i \subseteq B'$ then
                    $Up(B',F) \leftarrow Up(B',F) \cup \{B_i\}$
            $cmnLB(B', Up(B',F), L)$                                // Visit $B'$
```

$Up(B,F)$, only the entry $\{D_1, D_2, D_3, D_4\}$ belongs also to $Up(B',F)$, therefore we transfer it to B'. On the contrary, we do not transfer the entry $\{D_1, D_2, D_4\}$, i.e., it is not a superset of B'.

When the Execution of the Algorithm Finishes. We always continue upwards, until there are no other supersets to explore (e.g., in Step 4 of Figure 5.8 we reached the top of the lattice), or we have reached the desired level L (line 12 in Alg. 5), e.g., for finding the triad having $cmnBest(3,F)$, we should stop at level $L = K = 3$. In any of the previous cases, the recursion ends, thereby, we return to the previous node and we check if there are other supersets that have not been visited, e.g., in Step 6 of Figure 5.8, when we return to the node $\{D_1, D_2\}$, we visit also its superset $\{D_1, D_2, D_4\}$, and we perform again the check of which of the $Up(\{D_1, D_2\}, F)$ belong to the Ups of $\{D_1, D_2, D_4\}$.

LB versus BM. For computing $|cmn(B,F)|$ for a subset of datasets B, we scan the set $Up(B,F)$ instead of $occur(\mathcal{D},F)$ (in BM approach), where $Up(B,F) \subseteq occur(\mathcal{D},F)$.

Time & Space Complexity. Since we follow a depth-first search, we visit each node (subset of datasets) once and we create one edge per node, i.e., $|V| + |V|$, and we need



Compute the commonalities for all the supersets of D1,D4 that have not been explored yet!

Up({D1,D4},F)	Direct Count
D1,D2,D3,D4,D5	5
D1,D2,D4,D5	7
D1,D3,D4,D5	4
D1,D4,D5	2
D1,D4,D5,D6	4
D1,D2,D3,D4,D5,D6	2
D1,D4,D7	2

Step A. Pruning Redundant Datasets from each Entry

Pruned Entry	Direct Count
D1,D4,D5	5
D1,D4,D5	7
D1,D4,D5	4
D1,D4,D5	2
D1,D4,D5,D6	4
D1,D4,D5,D6	2
D1,D4,D7	2

Step B. Regrouping

UpPr({D1,D4},F)	Direct Count
D1,D4,D5	18
D1,D4,D5,D6	6
D1,D4,D7	2

Figure 5.9: Computing the metrics for all the supersets of D_1, D_4

to keep in memory the nodes (subsets of datasets) that exist in the maximum depth of the lattice (which is at most $|D|+1$). For each visited subset B', we iterate over $|Up(B,F)|$ entries (see line 16 of Alg. 5), for finding $|Up(B',F)|$. Therefore, for a given set of subsets BV ($BV = \{B_i, \ldots, B_n\}$), the average number of such iterations per subset B' is: $avgIt_{LB}(BV) = \frac{\sum_{B' \in BV} |Up(B,F)|, prev(B')=B}{|BV|}$. The time complexity is $O(|BV| * avgIt_{LB}(BV))$, which is exponential in number when $|BV| = 2^{|\mathcal{D}|}$. Concerning space complexity, it is $O(|BV_d| * avgIt_{LB}(BV_d))$ where $BV_d = \{B_i, \ldots, B_n\}$ are the subsets that occur in the maximum depth d of lattice (since d is at most $|\mathcal{D}|+1$, then $|BV_d| \leq |\mathcal{D}|+1$).

5.5.2.1.1 Removing Redundant Dataset from $Up(B,F)$ and Regrouping (LB+UPGR).

The target is to further decrease the size of $Up(B,F)$ that is transferred to a superset from a subset in lines 12-19. In particular, we propose an approach, where we remove the redundant datasets from each $B_i \in Up(B,F)$, and we perform a regrouping. First, let $maxID(B)$ be the largest dataset ID in a subset of datasets B, i.e., $maxID(B) = \{k \mid D_k \in B \, and \, \forall D_j \in B, k \geq j\}$ (e.g., $maxID(\{D_1, D_3, D_4\}) = 4$). By using the *dfs* traversal of Figure 5.6, we visit a superset B', where $B' = B \cup \{D_k\}$ and $k = maxID(B')$. For instance, suppose that we visit the subset $B = \{D_1, D_4\}$. Afterwards, all the supersets of B that will be visited, will contain datasets having ID larger than 4, therefore, we will never visit a superset of B containing either D_2 or D_3. In this way, when we visit a dataset B, we define as redundant datasets in a subset $B_i \in Up(B,F)$ the following set of datasets $D_{rdnt}(B_i, B) = \{D_i \in B_i \mid D_i \notin B, i < maxID(B)\}$.

In particular, for a specific B we can replace each $B_i \in Up(B,F)$ with B_i^k, where $B_i^k = B_i \setminus D_{rdnt}(B_i, B)$, and then we regroup the "pruned" entries, i.e., we group the same entries and we sum their *directCount* score. In this way, a new *directCount* list is created in each lattice node. This list contains in its left side the set $UpPr(B,F) = \bigcup_{B_i \in Up(B,F)} B_i \setminus D_{rdnt}(B_i, B)$, and in its right side the *directCount* score of each B_i^k, i.e., $directCountPr(B_i^k, F, B)$ $= \sum_{B_i \in Up(B,F), B_i^k = B_i \setminus D_{rdnt}(B_i,B)} directCount(B_i, F)$.

Example. In Figure 5.9, we can see an example for a subset $B = \{D_1, D_4\}$. In this example, suppose that the set $Up(B,F)$ contains 7 entries. As we go upwards through the *DFS* approach, we will visit supersets of B, which do not contain either D_2 or D_3. Therefore, we can remove these datasets from each entry of $Up(B,F)$. This process results to a new *directCount* list that contains some posting lists that are repeated. For instance, $\{D_1, D_4, D_5\}$ exists 4 times in the new list, with different values. For this reason, we keep each posting list once, however, we sum their *directCount* values, e.g., for $\{D_1, D_4, D_5\}$ the new value is 18. As a result, we will use 3 entries instead of 7 in this example. Thereby, as we continue upwards, we create (and transfer) a new smaller *directCount* list.

LB+UPGR versus LB. It is obvious that $|UpPr(B,F)| \leq |Up(B,F)|$, therefore, we read and transfer smaller number of entries comparing to *LB* in Alg. 5. For each visited subset B', we can iterate over $|UpPr(B,F)|$ entries (see line 16 of Alg. 5), for finding $|UpPr(B',F)|$. Therefore, for a given set of subsets BV, the average number of such iterations per subset B' is: $avgIt_{LB+UPGR}(BV) = \frac{\sum_{B' \in BV} |UpPr(B,F)|, prev(B')=B}{|BV|}$ and the time complexity is $O(|BV| * avgIt_{LB+UPGR}(BV))$ Concerning space complexity, it is $O(|BV_d| * avgIt_{LB+UPGR}(BV_d))$

5.5.2.2 Alternative Approach. Top-Down Breadth-First Search

Rationale. We introduce a top-down lattice-based (*topDownLB*) incremental algorithm (i.e., Alg. 6), which uses a breadth-first search traversal and can be used for solving $cmnBest(K,F)$ for any measurement type F.

Input. Similarly to the case of bottom-up approach, the input is the set $occur(\mathcal{D}, F)$, the corresponding $directCount(B_i, F)$ score for each $B_i \in occur(\mathcal{D}, F)$, and a parameter L, for stopping the computation of metrics, when level L is reached.

Output. The output is the same as in the *bottom-up* approach, i.e., it computes on query time the $|cmn(B,F)|$ for all the subsets until a level L.

The Algorithm. The **top-down** approach starts from the maximum level (i.e., the maximum level equals $|\mathcal{D}|$), as one can see in Figure 5.7 and in line 1 of Alg. 6. It first computes the metrics for all the subsets of datasets of each level (see line 2) before continuing with the previous level. When a node B is visited, we check if $B \in occur(\mathcal{D}, F)$, i.e., if the current subset of datasets B can be found in the *directCount* list (in line 4). In case it holds, we add B to $Up(B,F)$ and then we compute the value $|cmn(B,F)|$.

Afterwards, the list $Up(B,F)$ of the current node is "transferred" to all the subsets B' of the lower level (in lines 8-9), since $B' \subset B$ implies that $Up(B,F) \subseteq Up(B',F)$ (see Prop. 6). For example, for the lattice of Figure 5.8, since $\{D_1, D_2, D_3, D_4\} \in Up(\{D_1, D_2, D_3, D_4\}, F)$ then surely it belongs to the *Up*s of all triads of datasets, e.g., $\{D_1, D_2, D_3, D_4\} \in Up(\{D_1, D_2, D_3\}, F)$, $\{D_1, D_2, D_3, D_4\} \in Up(\{D_1, D_2, D_4\}, F)$, and so on. After finishing with the nodes of the current level, we continue with the nodes of the previous level (i.e., see line 10 and Figure 5.7).

Time & Space Complexity. For this algorithm, we should create in the worst case all the nodes and edges, i.e., $|V| = 2^{|\mathcal{D}|}$ and $|E| = |\mathcal{D}| * 2^{(|\mathcal{D}|-1)}$, therefore its time complexity

ALGORITHM 6: Computing $|cmn(B,F)|$ incrementally for any possible subset of datasets B through a top-down approach

```
Input: The set $occur(\mathcal{D},F)$ the $directCount$ values, the stop Level $L$
Output: The commonalities $|cmn(B,F)|$, for each possible subset $B \in \mathcal{P}(\mathcal{D})$
1  $currentL \leftarrow |\mathcal{D}|$
2  while $currentL \geq L$ do
3      forall $B \in B(currentL)$ do              // $B(currentL) = \{B_i \in P(\mathcal{D}) \mid |B_i| = currentL\}$
4          if $B \in occur(\mathcal{D},F)$ then
5              $Up(B,F) \leftarrow Up(B,F) \cup \{B\}$
6              $|cmn(B,F)| \leftarrow \sum_{B_i \in Up(B,F)} directCount(B_i,F)$
7              print $B$, $|cmn(B,F)|$
8              forall $B' \in down(B)$ do      // $down(B) = \{B_i \in P(\mathcal{D}) \mid B_i \subset B, |B| = |B_i| + 1\}$
9                  $Up(B',F) \leftarrow Up(B',F) \cup Up(B,F)$
10     $currentL \leftarrow currentL - 1$
```

is $O(|V| + |E|)$, The major problem is that since it follows a breadth-first search, we have to keep in memory each subset B (and $Up(B,F)$) of a specific level L (the number of nodes V_k of level L is given by $V_L = \binom{|\mathcal{D}|}{L}$), i.e., its space complexity is $O(|V'_L|)$, where L' is the lattice level containing the highest number of nodes.

5.5.2.3 Comparison of Approaches

For all the lattice-based algorithms (i.e., top-down and bottom-up), we pass from each node once, and we construct the $Up(B,F)$ for each subset B by exploiting set theory properties. On the contrary, by using a baseline method we should read the whole *directCount* list, which is usually quite large. For instance, for the bottom-up approach we should read $|Up(B,F)|$ entries instead of $|occur(\mathcal{D},F)|$, and obviously it holds that $|Up(B,F)| \leq |occur(\mathcal{D},F)|$.

Below, we analyze the differences between the top-down and the bottom-up approaches. The main disadvantage of top-down approach is that we should create all the edges, i.e., the top-down approach requires creating $|\mathcal{D}|/2$ times more edges than bottom-up approach, and to keep all the subsets of a specific level in memory. On the contrary, for the bottom-up approach, there is no need to create all the edges. However, we should check which of the $Up(B,F)$ belong also to $Up(B',F)$, where $B \subset B'$, whereas in the top-down approach we can add $Up(B',F)$ to $Up(B,F)$ ($B \subset B'$) without performing such a check.

A major advantage of bottom-up approach is that we need to keep in memory only d subsets of datasets, where d is at most $|D| + 1$, whereas for the top-down approach, we should keep in memory a high number of datasets (indeed the exact number is given through the binomial coefficient formula). Another advantage of bottom-up approach is that it is more beneficial to be used for computing only the $|cmn(B, F)|$ of subsets that satisfy a specific threshold (e.g., $|cmn(B, F)| \geq 20$). Indeed, fewer nodes will be created since we can exploit Prop. 6 to avoid creating nodes that are impossible to satisfy the threshold, e.g., we know that if $|cmn(B, F)| < 20$, then all for each of the supersets B' of B it holds that $|cmn(B', F)| < 20$. In the past, we have performed a power-law study [165], which has shown that the bottom-up approach is more efficient as the number of dataset grows, which is also confirmed by the experiments that are presented in §5.7.

Concerning the comparison between the two bottom-up approach that follows a *DFS* traversal, by using the *LB+UPGR* approach versus the standard version (without removing the redundant datasets), we read and transfer (from a subset to a superset) a smaller number of entries, since $|UpPr(B, F)| \leq |Up(B, F)|$. However, we should perform some extra operations for pruning and regrouping the entries of $Up(B, F)$, which can be time consuming in some cases, as we will see experimentally in §5.7.

5.5.3 Lattice Based Incremental Algorithm for computing $covBest(K, F)$ (*LB* method).

Here, we show how to use a lattice-based depth-first search traversal and set theory properties for computing coverage.

Rationale. We present a lattice-based (*LB*) incremental algorithm, that can be used for solving $covBest(K, F)$. It reuses measurements between a subset B and a superset B' ($B \subset B'$), by following a depth-first search traversal and by exploiting set theory properties, as it is described below.

How to Reuse the Measurements. For reusing the measurements in an incremental way, we exploit the following set theory property:

Lemma 6. From the set theory, we know that for n sets $\{A_1, A_2, ..., A_n\}$, $|A_1 \cup A_2 \cup \ldots \cup A_n| = |A_1 \cup A_2 \cup \ldots \cup A_{n-1}| + |A_n \setminus \{A_1 \cup A_2 \cup \ldots \cup A_{n-1}\}|$ *(Proof can be found in [126]).*

By exploiting the above property, for a subset B and a superset B' (where $B' = B \cup \{D_k\}$), we can add to $|cov(B', F)|$, the value $|cov(B, F)|$. The only requirement for computing $|cov(B', F)|$, is to find which are the *directCount* entries that contain the newly added dataset D_k, but not any $D_i \in B$, i.e., we should construct the set $occur(\{D_k\}, F)_{\setminus B}$, as it is stated below.

Prop. 7. If $B' = B \cup \{D_k\}$, then,

$$|cov(B', F)| = |cov(B, F)| + \sum_{B_i \in occur(\{D_k\}, F)_{\setminus B}} directCount(B_i, F) \qquad (5.12)$$

Proof.

$$
\begin{aligned}
|cov(B', F)| &\overset{B'=B\cup\{D_m\}}{=} |\{\cup_{D_i \in B} F(D_i)\} \cup F(D_m)| \\
&\overset{(Lemma\ 6)}{=} |\cup_{D_i \in B} F(D_i)| + |F(D_m) \setminus \{\cup_{D_i \in B} F(D_i)\}| \\
&= |cov(B, F)| + |\{k | k \in F(D_m),\ and\ \forall D_i \in B, k \notin F(D_i)\}| \\
&\overset{(Lemma\ 1)}{=} |cov(B, F)| + |\cup_{B'_i \in \mathcal{P}(\mathcal{D})} \{k \in Left(ind_F) | ind_F(k) = B'_i, D_m \in B'_i, B \cap B'_i = \emptyset\}| \\
&\overset{(Def.\ 4)}{=} |cov(B, F)| + |\cup_{B'_i \in occur(\{D_m\}, F)_{\setminus B}} \{k \in Left(ind_F) | ind_F(k) = B'_i\}| \\
&\overset{(Similarly\ to\ Proof\ of\ Prop.3)}{=} |cov(B, F)| + \sum_{B'_i \in occur(\{D_m\}, F)_{\setminus B}} directCount(B'_i, F) \diamond
\end{aligned}
$$

□

Alg. 7 shows the exact steps for computing the coverage incrementally, while Figure 5.10 depicts an example of Alg. 7, for the 5 "lattice nodes" in green color, of Query $Q_{coverage}$ (see Figure 5.3) .

Input. The input is the set $occur(\mathcal{D}, F)$ and the corresponding $directCount(B_i, F)$ score, for each $B_i \in occur(\mathcal{D}, F)$, e.g., see the input in the lower right side of Figure 5.10, and a parameter L, for stopping the computation of metrics, when level L is reached, e.g., for finding $covBest(K, F)$, $L = K$.

Output. It computes at query time the $|cov(B, F)|$ for all the subsets until a level L, e.g., if $L = |\mathcal{D}| + 1$ it computes the metrics for all the possible subsets $B \in \mathcal{P}(\mathcal{D})$. In particular, for finding the $covBest(K, F)$ it computes the metrics until the level $L = K$, and it keeps in memory the subset having the $arg_B \max\ |cov(B, F)|$.

The Initial Phase. We start from the bottom part of the lattice, thereby, we first explore the nodes containing a single dataset (lines 1-4 of Alg. 7), e.g., in Figure 5.10, we start from the dataset D_1. For each single dataset the recursive function *covLB* is called, which takes as parameters the subset of datasets that will be visited, the dataset D_k which was just added, the coverage of the previous visited node, i.e., $|cov(B, F)|$, the set $occur(\mathcal{D}, F)_{\setminus B}$, and the stop level L. For the case of single datasets, we suppose that the previous node is the empty set, therefore, $|cov(\{\emptyset\}, F)| = 0$, while $occur(\mathcal{D}, F)_{\setminus\{\emptyset\}}$ contains all the entries of the *directCount* list, i.e., $occur(\mathcal{D}, F)$ (lines 1-2).

Computation of Measurements for the Current Node. The target is to construct the set $occur(\{D_k\}, F)_{\setminus B}$ (see Lemma 6). For this reason, we read the entries of the input set $occur(\mathcal{D}, F)_{\setminus B}$, and $\forall\ B_i \in occur(\mathcal{D}, F)_{\setminus B}$ we check whether $D_k \in B_i$ (see lines 6-12 of Alg. 7). When $D_k \in B_i$, we add B_i to $occur(\{D_k\}, F)_{\setminus B}$. On the contrary, if $D_k \notin B_i$, we store B_i in another set, i.e., $occur(\mathcal{D}, F)_{\setminus B'}$, since B_i should be scanned again when we reach the supersets of B'. Therefore, we divide the input set $occur(\mathcal{D}, F)_{\setminus B}$ in two disjoint sets, i.e.,

ALGORITHM 7: Computing $|cov(B, F)|$ incrementally for any possible subset of datasets B

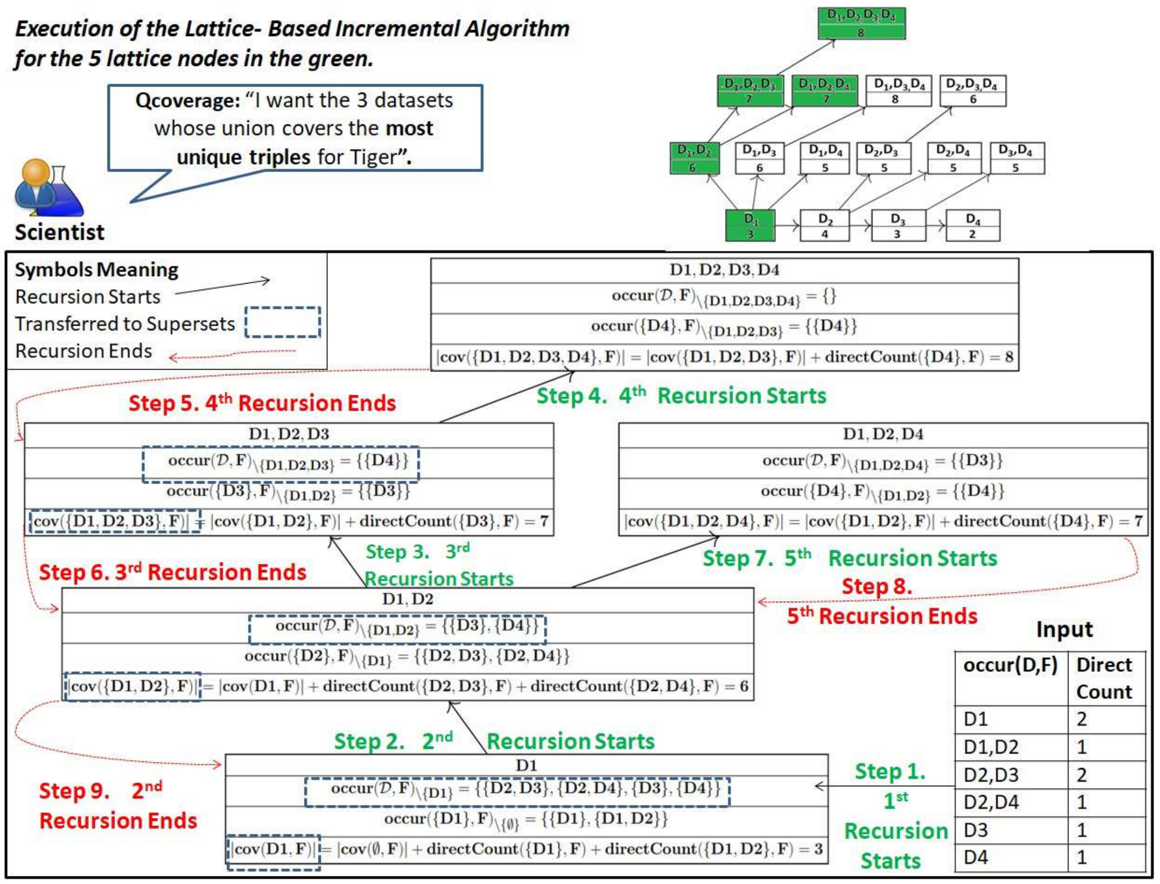
```
Input: The directCount list including the set $occur(\mathcal{D}, F)$ and the directCount values, the
       stop Level $L$
Output: The coverage $|cov(B, F)|$, for each possible subset $B \in \mathcal{P}(\mathcal{D})$
1  $occur(\mathcal{D}, F)_{\setminus\{\emptyset\}} \leftarrow occur(\mathcal{D}, F)$                    // The starting node is the empty set
2  $|cov(\{\emptyset\}, F)| = 0$
3  forall $D_k \in \mathcal{D}$ do                                       // Visit each single dataset once
4      $covLB(\{D_k\}, D_k, |cov(\{\emptyset\}, F)|, occur(\mathcal{D}, F)_{\setminus\{\emptyset\}}, L)$

   // The recursive function for computing the metrics incrementally
5  function covLB(Subset $B'$, Dataset $D_k$, $|cov(B, F)|$, $occur(\mathcal{D}, F)_{\setminus B}$, level $L$)       // Visit $B'$
6      $occur(\{D_k\}, F)_{\setminus B} \leftarrow \emptyset$
7      $occur(\mathcal{D}, F)_{\setminus B'} \leftarrow \emptyset$
8      forall $B_i \in occur(\mathcal{D}, F)_{\setminus B}$ do         // Divide $occur(\mathcal{D}, F)_{\setminus B}$ into two disjoint sets
9          if $D_k \in B_i$ then
10             $occur(\{D_k\}, F)_{\setminus B} \leftarrow occur(\{D_k\}, F)_{\setminus B} \cup \{B_i\}$
11         else if $D_k \notin B_i$ then
12             $occur(\mathcal{D}, F)_{\setminus B'} \leftarrow occur(\mathcal{D}, F)_{\setminus B'} \cup \{B_i\}$
13     $|cov(B', F)| \leftarrow |cov(B, F)| + \sum_{B_i \in occur(\{D_k\}, F)_{\setminus B}} directCount(B_i, F)$
14     print $B'$, $|cov(B', F)|$
15     if $|B'| \leq L$ then                                    // Continue until the stop level
16         forall $D_j \in \mathcal{D}, j > k$ do              // A strict numerical dfs order is followed
17             $B'' = B' \cup \{D_j\}$                    // Add one more dataset to $B'(B' = prev(B''))$
18             $covLB(B'', D_j, |cov(B', F)|, occur(\mathcal{D}, F)_{\setminus B'}, L)$                  // Visit $B''$
```

$occur(\mathcal{D}, F)_{\setminus B} = occur(\{D_k\}, F)_{\setminus B} \cup occur(\mathcal{D}, F)_{\setminus B'}$. Finally, Alg. 7 computes $|cov(B', F)|$ by taking the sum of $|cov(B, F)|$ and the sum of the *directCount* score of $occur(\{D_k\}, F)_{\setminus B}$, and prints (or stores) the result (see lines 13-14).

In Figure 5.10, we first visit the dataset D_1, and we iterate over the set $occur(\mathcal{D}, F)$. Then, we create the sets $occur(\mathcal{D}, F)_{\setminus\{D_1\}} = \{\{D_2, D_3\}, \{D_2, D_4\}, \{D_3\}, \{D_4\}\}$ and $occur(\{D_1\}, F)_{\setminus B} = \{\{D_1\}, \{D_1, D_2\}\}$. Finally, we sum the *directCount* score of $occur(\{D_1\}, F)_{\setminus B}$ entries (which equals 3) and the $|cov(B, F)|$ of the previous subset, which is the empty set ($|cov(\{\emptyset\}, F)| = 0$), for computing $|cov(\{D_1\}, F)| = 3$.

How to explore the Supersets of the Current Node. By following the *dfs* traversal in a recursive way, we visit a superset B'' from a subset B' (lines 16-18), e.g., see Figure 5.10. Indeed, when we visit B'', we add a parameter for "highlighting" the dataset D_j that was last added, we transfer the value $|cov(B', F)|$ and the set $occur(\mathcal{D}, F)_{\setminus B'}$. Thereby, a new recursive call starts, where we perform exactly the same steps. It is obvious that since $occur(\mathcal{D}, F)_{\setminus B'} \subseteq occur(\mathcal{D}, F)_{\setminus B}$, the size of the input (which is scanned in lines 8-12) decreases as we continue upwards.

In Step 2 of Figure 5.10, we "transferred" from D_1 to $\{D_1, D_2\}$ the value $|cov(\{D_1\}, F)|$ and the set $occur(\mathcal{D}, F)_{\setminus\{D_1\}}$. In the second recursion, we iterated over the set $occur(\mathcal{D}, F)_{\setminus\{D_1\}}$


Execution of the Lattice- Based Incremental Algorithm for the 5 lattice nodes in the green.
Qcoverage: "I want the 3 datasets whose union covers the most unique triples for Tiger".
Scientist
D1,D2,D3,D4 8
D1,D2,D3 7
D1,D2,D4 7
D1,D3,D4 8
D2,D3,D4 6
D1,D2 6
D1,D3 6
D1,D4 5
D2,D3 5
D2,D4 5
D3,D4 5
D1 3
D2 4
D3 3
D4 2
Symbols Meaning
Recursion Starts
Transferred to Supersets
Recursion Ends
D1, D2, D3, D4
occur(𝒟, F)\{D1,D2,D3,D4} = {}
occur({D4}, F)\{D1,D2,D3} = {{D4}}
|cov({D1, D2, D3, D4}, F)| = |cov({D1, D2, D3}, F)| + directCount({D4}, F) = 8
Step 4. 4th Recursion Starts
Step 5. 4th Recursion Ends
D1, D2, D3
occur(𝒟, F)\{D1,D2,D3} = {{D4}}
occur({D3}, F)\{D1,D2} = {{D3}}
|cov({D1, D2, D3}, F)| = |cov({D1, D2}, F)| + directCount({D3}, F) = 7
D1, D2, D4
occur(𝒟, F)\{D1,D2,D4} = {{D3}}
occur({D4}, F)\{D1,D2} = {{D4}}
|cov({D1, D2, D4}, F)| = |cov({D1, D2}, F)| + directCount({D4}, F) = 7
Step 3. 3rd Recursion Starts
Step 6. 3rd Recursion Ends
Step 7. 5th Recursion Starts
Step 8. 5th Recursion Ends
D1, D2
occur(𝒟, F)\{D1,D2} = {{D3}, {D4}}
occur({D2}, F)\{D1} = {{D2, D3}, {D2, D4}}
|cov({D1, D2}, F)| = |cov(D1, F)| + directCount({D2, D3}, F) + directCount({D2, D4}, F) = 6
Step 2. 2nd Recursion Starts
D1
occur(𝒟, F)\{D1} = {{D2, D3}, {D2, D4}, {D3}, {D4}}
occur({D1}, F)\{∅} = {{D1}, {D1, D2}}
|cov(D1, F)| = |cov(∅, F)| + directCount({D1}, F) + directCount({D1, D2}, F) = 3
Step 9. 2nd Recursion Ends
Step 1. 1st Recursion Starts
Input

occur(D,F)	Direct Count
D1	2
D1,D2	1
D2,D3	2
D2,D4	1
D3	1
D4	1

Figure 5.10: Execution of Incremental Algorithm for the 5 lattice nodes in green color

once, and we constructed the sets $occur(\{D_2\},F)_{\setminus\{D_1\}} = \{\{D_2,D_3\},\{D_2,D_4\}\}$ and $occur(\mathcal{D},F)_{\setminus\{D_1,D_2\}} = \{\{D_3\},\{D_4\}\}$. Finally, we took the sum of the *directCount* score of $occur(\{D_2\},F)_{\setminus\{D_1\}}$ entries (i.e., equals 3), and of $|cov(\{D_1\},F)|$ (i.e., 3), for computing $|cov(\{D_1,D_2\},F)| = 6$. Finally, the set $occur(\mathcal{D},F)_{\setminus\{D_1,D_2\}}$ will be transferred to the supersets of $\{D_1,D_2\}$, e.g., in Steps 3 and 7 of Figure 5.10.

When the Execution of the Algorithm Finishes. We always continue upwards, until there are no other supersets to explore (e.g., in Step 4 of Figure 5.10 we reached the top of the lattice), or we have reached the desired level L (line 15 in Alg. 7), e.g., for finding the triad having $covBest(3,F)$, we stop at level $L = K = 3$.

LB versus BM. For computing $|cov(B',F)|$ for a subset B', we reuse $|cov(B,F)|$ and we scan the set $occur(\mathcal{D},F)_{\setminus B}$ instead of $occur(\mathcal{D},F)$ (in *BM* approach), where $occur(\mathcal{D},F)_{\setminus B} \subseteq occur(\mathcal{D},F)$.

Time & Space Complexity. For each visited subset B', we iterate over $|occur(\mathcal{D},F)_{\setminus B}|$ entries (see line 8 of Alg. 7). Therefore, for a given set of subsets BV ($BV = \{B_i,\ldots,B_n\}$), the average number of such iterations per subset B' is: $avgIt_{LB}(BV) = \frac{\sum_{B' \in BV} |occur(\mathcal{D},F)_{\setminus B}|, prev(B')=B}{|BV|}$.

The time complexity is $O(|BV| * avgIt_{LB}(BV))$, which is exponential in number when $|BV| = 2^{|\mathcal{D}|}$. Concerning space complexity, it is $O(|BV_d| * avgIt_{LB}(BV_d))$ where $BV_d = \{B_i, ..., B_n\}$ are the subsets that occur in the maximum depth d of lattice (since d is at most $|\mathcal{D}| + 1$, then $|BV_d| \leq |\mathcal{D}| + 1$).

5.5.3.1 Pruning & Regrouping the *directCount* List for computing $covBest(K, F)$

Here, the target is to decrease the size of the set $occur(\mathcal{D}, F)_{\setminus B}$, which is transferred in line 18 of Alg. 7 from a subset B to a superset B' ($B = prev(B')$), and is scanned in lines 8-12 of Alg. 7 when B' is visited, for solving $covBest(K, F)$ by performing less set operations in the aforementioned lines. For this reason, we propose in §5.5.3.2 a process for pruning the totally redundant entries, and in §5.5.3.3 a process for pruning and regrouping all the entries of the *directCount* list.

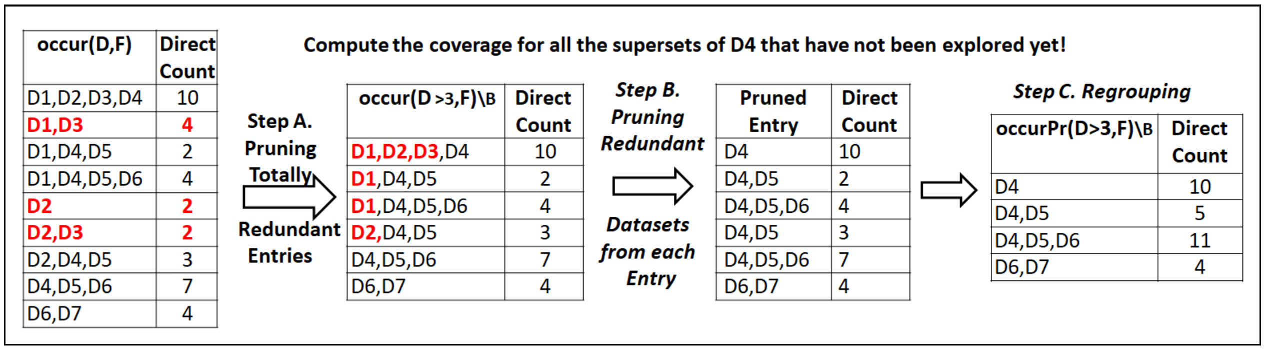
5.5.3.2 Pruning Totally Redundant Entries (LB+TPR)

First, we provide some definitions. Remind that $maxID(B) = \{k \mid D_k \in B \, and \, \forall D_j \in B, k \geq j\}$. Moreover, we define as $D_{\leq k} = \{D_i \in \mathcal{D} \mid i \leq k, D_k \in \mathcal{D}\}$ all datasets whose ID is equal or smaller than the ID of D_k (e.g., in Figure 5.11, $D_{\leq 4} = \{D_1, D_2, D_3, D_4\}$). Finally, let $D_{>k} = \mathcal{D} \setminus D_{\leq k}$ be all datasets whose ID is larger than the ID of D_k (e.g., in Figure 5.11, $D_{>4} = \{D_5, D_6, D_7\}$).

LB+TPR Approach. The target is to remove the redundant entries from the set $occur(\mathcal{D}, F)_{\setminus B'}$, which is created in line 12 of Alg. 7. By using the *dfs* traversal of Figure 5.6 (and the numerical ascending order), we visit a superset B', where $B' = B \cup \{D_k\}$ and $k = maxID(B')$. Due to this traversal, in lines 11-12 of Alg. 7, it is redundant to add B_i to $occur(\mathcal{D}, F)_{\setminus B'}$ if $maxID(B_i) < k$, i.e., in this case, all the IDs of B_i datasets are smaller than k. Indeed, for such a B_i and for any B'' (where $B'' \supset B'$) it holds that $B_i \cap B'' = \emptyset$, which means that B_i will be "transferred" forever, as we go upwards (i.e., it will never pass the check in line 9). Therefore, in line 11 we check if $maxID(B_i) > k$ and only in case it holds, in line 12 we add B_i to the set $occur(D_{>k}, F)_{\setminus B'}$ (instead of $occur(\mathcal{D}, F)_{\setminus B'}$), and then we transfer the set $occur(D_{>k}, F)_{\setminus B'}$ to the supersets of B' (in line 18).

Example. In Figure 5.11, we desire to compute the metrics for all the supersets of D_4 that have not visited yet, i.e., those that do not contain D_1, D_2 or D_3. Due to *dfs* traversal, we have already computed the measurements for any combination containing D_1, D_2 or D_3. Since each time we will add datasets belonging to $D_{>3} = \{D_4, D_5, D_6, D_7\}$, in Step A of Figure 5.11 we prune the totally redundant entries, containing only datasets belonging to $D_{\leq 3}$, but not any $D_i \in D_{>3}$. In this way, we use 6 *directCount* entries instead of 9, i.e., we removed the entries $\{D_1, D_3\}$, $\{D_2\}$ and $\{D_2, D_3\}$.

LB+TPR versus LB. Since $D_{>k} \subseteq \mathcal{D}$, for any subset B it holds that $occur(D_{>k}, F)_{\setminus B} \subseteq occur(\mathcal{D}, F)_{\setminus B}$, therefore, we transfer (line 18 of Alg. 7) and iterate (line 8 of Alg. 7) over


Compute the coverage for all the supersets of D4 that have not been explored yet!
Step A. Pruning Totally Redundant Entries
Step B. Pruning Redundant Datasets from each Entry
Step C. Regrouping

occur(D,F)	Direct Count
D1,D2,D3,D4	10
D1,D3	4
D1,D4,D5	2
D1,D4,D5,D6	4
D2	2
D2,D3	2
D2,D4,D5	3
D4,D5,D6	7
D6,D7	4

occur(D >3,F)\B	Direct Count
D1,D2,D3,D4	10
D1,D4,D5	2
D1,D4,D5,D6	4
D2,D4,D5	3
D4,D5,D6	7
D6,D7	4

Pruned Entry	Direct Count
D4	10
D4,D5	2
D4,D5,D6	4
D4,D5	3
D4,D5,D6	7
D6,D7	4

occurPr(D>3,F)\B	Direct Count
D4	10
D4,D5	5
D4,D5,D6	11
D6,D7	4

Figure 5.11: Pruning Example-Computing the metrics for all the supersets of D_4, that have not been explored yet

smaller (or in the worst case the same) number of entries comparing to *LB* approach of §5.5.3 (see time and space complexity of *LB+TPR* in Table 5.2). However, we should perform an extra check (i.e., $maxID(B_i) > k$).

5.5.3.3 Pruning & Regrouping (LB+PRGR)

The target is to further reduce the size of $occur(D_{>k}, F)_{\setminus B'}$, which is used for computing $|cov(B, F)|$ for the supersets of B'. In this way, we propose an approach, where we remove both the totally redundant *directCount* entries and the redundant datasets from each remaining *directCount* entry, and we perform a regrouping. In particular, for a specific B' (where $maxID(B') = k$), we replace each $B_i \in occur(D_{>k}, F)_{\setminus B'}$ (constructed in lines 8-12 by using *LB+TPR*) with B_i^k, where $B_i^k = B_i \setminus D_{\leq k}$, and then we regroup the "pruned" entries, i.e., we group the same entries and we sum their *directCount* score. In this way, a new *directCount* list is created in each lattice node. This list contains in its left side the set $occurPr(D_{>k}, F)_{\setminus B'} = \bigcup_{B_i \in occur(D_{>k},F)_{\setminus B'}} B_i \setminus D_{\leq k}$ (where $k = maxID(B')$), and in its right side the *directCount* score of each B_i^k, i.e., $directCountPr(B_i^k, F, B') = \sum_{B_i \in occur(D_{>k},F)_{\setminus B'}, B_i^k = B_i \setminus D_{\leq k}} directCount(B_i, F)$.

Example. In Step B of Figure 5.11, we prune in each entry the datasets belonging to $D_{\leq 3} = \{D_1, D_2, D_3\}$, i.e., we have already visited all the nodes containing any of these datasets. Due to pruning, some entries exist multiple times, e.g., $\{D_4, D_5\}$ exists twice, with *directCount* scores 2 and 3. For this reason, in Step C, we regroup the entries, e.g., we keep $\{D_4, D_5\}$ once, with a new *directCount* score, i.e., 5. Due to this process, we use only 4 *directCount* entries, instead of 6 (by using *LB+TPR*). Certainly, as we continue upwards, we create (and transfer) a new smaller *directCount* list.

LB+PRGR versus LB+TPR. It is obvious that $|occurPr(D_{>k}, F)_{\setminus B'}| \leq |occur(D_{>k}, F)_{\setminus B'}|$, therefore, we read and transfer smaller number of entries comparing to *LB+PRGR* in lines 8 and 18 of Alg. 7. However, we should perform additional operations for creating the set $occurPr(B', F)$ and the $directCountPr(B_i^k, F, B')$ of each B_i^k. Table 5.2 presents the time and

Approach	$avgIt$ per B' in line 8 of Alg. 7 ($B = prev(B')$)	Time Complexity	Space Complexity
BM	$\lvert occur(\mathcal{D}, F)\rvert$	$O(\lvert BV\rvert * \lvert occur(\mathcal{D}, F)\rvert)$	$O(\lvert occur(\mathcal{D}, F)\rvert)$
LB	$avgIt_{LB}(BV) = \frac{\sum_{B' \in BV} \lvert occur(\mathcal{D},F)_{\setminus B}\rvert}{\lvert BV\rvert}$	$O(\lvert BV\rvert * avgIt_{LB}(BV))$	$O(\lvert BV_d\rvert * avgIt_{LB}(BV_d))$
$LB{+}TPR$	$avgIt_{LB+TPR}(BV) \frac{\sum_{B' \in BV} \lvert occur(D_{>k},F)_{\setminus B}\rvert}{\lvert BV\rvert}$	$O(\lvert BV\rvert * avgIt_{LB+TPR}(BV))$	$O(\lvert BV_d\rvert * avgIt_{LB+TPR}(BV_d))$
$LB{+}PRGR$	$avgIt_{LB+PRGR}(BV) = \frac{\sum_{B' \in BV} \lvert occurPr(D_{>k},F)_{\setminus B}\rvert}{\lvert BV\rvert}$	$O(\lvert BV\rvert * avgIt_{LB+PRGR}(BV))$	$O(\lvert BV_d\rvert * avgIt_{LB+PRGR}(BV_d))$

Table 5.2: Comparison of different approaches for computing $|cov(B', F)|$, for a set of "visited" subsets BV.

space complexity of *LB+PRGR*.

5.5.3.4 Complexity Summary

Table 5.2 compares the presented approaches in terms of the average iterations per subset B for computing $|cov(B, F)|$, and of time and space complexity. As we have seen in this section, for a specific subset B where $k = maxID(B)$, it holds that $|occurPr(D_{>k}, F)_{\setminus B}| \leq |occur(D_{>k}, F)_{\setminus B}| \leq |occur(\mathcal{D}, F)_{\setminus B}| \leq |occur(\mathcal{D}, F)|$. Therefore, for any given subset B, the number of iterations required for computing $|cov(B, F)|$ (and thus the number of set operations in lines 8-12 of Alg. 7), and the size of the set which is transferred to supersets (line 18 of Alg. 7), is reduced as we go downwards in Table 5.2. In this way, (in lines 8-12) less operations are required for finding the $covBest(K, F)$. However, we should note that for the case of *LB+TPR* we need to perform an extra check comparing to *LB*, i.e., $maxID(B_i) > k$. By using *LB+PRGR*, we perform the previous check, and we also replace each $B_i \in |occur(D_{>k}, F)_{\setminus B}|$, with $B_i^k = B_i \setminus D_{\leq k}$, and to compute the *directCount* score of B_i^k. However, as we shall see experimentally in §5.7, despite these extra operations, the two latter approaches (mainly *LB+PRGR*) are faster comparing to the other ones.

5.5.4 Lattice-Based Incremental Algorithm for computing $enrichBest(K, F, D_m)$

We use a variation of the incremental algorithm (i.e., Alg. 7) for *coverage*. The key difference is that we do not take into account the posting lists and lattice nodes containing D_m.
Input. We give as input the $occur(\mathcal{D}, F)_{\setminus\{D_m\}}$ instead of $occur(\mathcal{D}, F)$, and the lattice level L, e.g., for stopping the computation when $L = K$.
Output. It computes at query time the $|enrich(B, F, D_m)|$ for all the subsets of datasets (that do not contain D_m) until a level L. For finding the $enrichBest(K, F, D_m)$ it computes the metrics until the level $L = K$, and it keeps in memory the subset having the $arg_B \max\ |enrich(B, F, D_m)|$.

Differences with Alg. 7. Below, we analyze the differences comparing to Alg. 7.
• We transfer from a subset B to a superset B' ($B' = B \cup \{D_k\}$) the value $|enrich(B, F, D_m)|$ and the set $occur(\mathcal{D}, F)_{\setminus\{B \cup \{D_m\}\}}$ (instead of $|cov(B, F)|$ and $occur(\mathcal{D}, F)_{\setminus B}$).

• In each recursive call, we read the set $occur(\mathcal{D}, F)_{\setminus\{B\cup\{D_m\}\}}$, and we divide that set into two disjoint sets, i.e., $occur(\{D_k\}, F)_{\setminus\{B\cup\{D_m\}\}}$ and $occur(\mathcal{D}, F)_{\setminus\{B'\cup\{D_m\}\}}$.

• We exploit the set theory property described in Lemma 6, for computing $|enrich(B', F, D_m)|$ as follows, $|enrich(B', F, D_m)| = |enrich(B, F, D_m)| + \sum_{B_i \in occur(\{D_k\},F)_{\setminus\{B\cup\{D_m\}\}}} directCount(B_i, F)$.

Time & Space Complexity. Given a set of subsets BV ($BV = \{B_i, \ldots, B_n\}$), the average number of iterations per $B' \in BV$ is: $avgIt_{enrichLB}(BV) = \frac{\sum_{B' \in BV} |occur(\mathcal{D},F)_{\setminus\{B\cup\{D_m\}\}}|, B = prev(B')}{|BV|}$. The time complexity is $O(|BV| * avgIt_{enrichLB}(BV))$ (where $|BV| \leq 2^{|\mathcal{D}|-1}$), and the space complexity is $O(|BV_d| * avgIt_{enrichLB}(BV_d))$ where $BV_d = \{B_i, \ldots, B_n\}$ ($BV_d \leq |\mathcal{D}|$). For reducing the number of iterations, one can use the pruning and regrouping mechanisms of §5.5.3.1.

5.5.5 Lattice-Based Incremental Algorithm for computing $uniqBest(D_m, F, K)$

We use a variation of the incremental algorithm (i.e., Alg. 7) which is used for *coverage*. The key difference is that we compute the union of the elements of the datasets in B that also occur in D_m.

Input. We give as input the $occur(\{D_m\}, F)$ instead of $occur(\mathcal{D}, F)$, and the level L, i.e., the level where we stop the computation.

Output. It computes at query time the $|uniq(D_m, F, B)|$ for all the subsets of datasets (that do not contain D_m) until a level L. For finding the $uniqBest(D_m, F, K)$ it computes the metrics until the level $L = K$, and it keeps in memory the subset having the $arg_B \max\ |uniq(D_m, F, B)|$.

Differences with Alg. 7. Below, we analyze the differences comparing to Alg. 7.

• We initialize $|uniq(D_m, F, \{\emptyset\})| = |cov(\{D_m\}, F)|$ (all the elements of D_m are unique comparing to the empty set), and we transfer from a subset B to a superset B' ($B' = B \cup \{D_k\}$) the value $|uniq(D_m, F, B)|$ and the set $occur(\{D_m\}, F)_{\setminus B}$ (instead of $|cov(B, F)|$ and $occur(\mathcal{D}, F)_{\setminus B}$).

• In each recursive call, we read $occur(\{D_m\}, F)_{\setminus B}$, and we divide that set into two disjoint sets, i.e., $occur(\{D_m\}, F)_{\setminus B'}$ and $occur(\{D_k\}, F)_{D_m, \setminus B}$, where the set $occur(\{D_k\}, F)_{D_m, \setminus B}$ is defined as $occur(\{D_k\}, F)_{D_m, \setminus B} = occur(\{D_k\}, F)_{D_m} \setminus occur(B, F)_{D_m}$.

• By exploiting the set theory property of Lemma 6, we compute the $|uniq(D_m, F, B')|$ as follows: $|uniq(D_m, F, B')| = |uniq(D_m, F, B)| - \sum_{B_i \in occur(\{D_k\},F)_{D_m, \setminus B}} directCount(B_i, F)$.

Time & Space Complexity. Given a set of subsets BV ($BV = \{B_i, \ldots, B_n\}$), the average number of iterations per $B' \in BV$ is: $avgIt_{uniqLB}(BV) = \frac{\sum_{B' \in BV} |occur(\{D_m\},F)_{\setminus B}|, B = prev(B')}{|BV|}$. The time complexity is $O(|BV| * avgIt_{uniqLB}(BV))$ (where $|BV| \leq 2^{|\mathcal{D}|-1}$), and the space complexity is $O(|BV_d| * avgIt_{uniqLB}(BV_d))$ where $BV_d = \{B_i, \ldots, B_n\}$ ($|BV_d| \leq |\mathcal{D}|$). One can use the pruning and regrouping methods of §5.5.3.1, for reducing the number of iterations.

5.6 Computing Lattice-Based Measurements in Parallel

Here, we show how to parallelize the bottom-up lattice based approach, by splitting the lattice into smaller pieces.

5.6.1 Overview of the Parallelization.

Ideally we would like each machine to compute the measurements for the same (or approximately the same) number of lattice nodes and this is quite challenging. Let LM_i denote the subset of lattice nodes for machine m_i. Ideally we would like $\forall i \in [1, ..., m]$, $|LM_i| = \frac{2^{|\mathcal{D}|}}{m}$. The challenge here is to split the lattice in "slices", where for each "slice" we want to be able to exploit the set-theory properties described earlier in this chapter and to perform the lattice-based measurements of its "slice" incrementally. For this reason, we create a parallelized version of the bottom-up incremental algorithm where we split the lattice in such "slices", where each "slice" of the lattice corresponds to the upper set or a part of the upper set of a specific node.

In particular, let $LP = \{L_1, ..., L_z\}$ denote a partition of the power set of $2^{|\mathcal{D}|}$ nodes, i.e., $L_1 \cup ... \cup L_z = \mathcal{P}(\mathcal{D})$, and if $i \neq j$, $L_i \cap L_j = \emptyset$. Moreover, let θ be a threshold where $\theta = \frac{2^{|\mathcal{D}|}}{t}$, $t \geq 1$ (t can be given by the user). The size of the nodes that each "slice" contains is less or equal θ, i.e., $|L_i| \leq \theta$ and each machine m_i computes the measurements for a subset of lattice "slices" LP, i.e., $LM_i = \{L_j, ..., L_n\}$. Therefore, we split the power set of the lattice in z "slices" where z depends on θ, e.g., by having a big value for θ, fewer "slices" will be created, each of them containing a big number of nodes, whereas by choosing a small value for θ, larger number of "slices" will be created, each of them having fewer number of nodes. As we shall see, each "slice" of the power set corresponds to a unique key-value pair and the "slices" are distributed randomly to the available machines.

5.6.2 Why to parallelize the bottom-up approach.

We decided to generalize the bottom-up approach since (a) it is faster comparing to the top-down as the number of dataset grows [165], (b) it uses depth-first traversal, where there is no need to create all the edges of the lattice and (c) its computation can be divided into the computation of the upper sets of different pairs. The proposed approach can be used for constructing either the whole lattice or a part of it (e.g., measurements until level L). As we have described, the main notion of the *bottom-up* algorithm is that we always start from a subset B containing a pair of sources, we compute the measurements of all the nodes of its upper set (including B) before continuing with the upper set of the next pair and so forth. Therefore, one possible way to traverse in parallel the lattice is to split the lattice in smaller "slices", where each "slice" corresponds to the upper set of each pair. However, the size of the upper set of each pair (or triad, quad and so on) is not the same.

Let $Ups(B)$ be the upper set of a subset B, i.e., a subset of the supersets of B, where for each $B' \in Ups(B)$ there does not exist a dataset D_j in B' whose ID (i.e. j) according to the numerical order is lower than the dataset ID i for any $D_i \in B$. For instance, $\langle D_1, D_2, D_3 \rangle$ is a superset of $\langle D_2, D_3 \rangle$, however, it does not belong to $Ups(D_2, D_3)$ because the ID of dataset D_1, i.e. 1, is lower than the dataset ID for any dataset in $\langle D_2, D_3 \rangle$ (i.e. 1 is lower than 2 and 3). Therefore, $Ups(B)$ is defined as follows: $Ups(B) = \{B' \mid B \subseteq B' \; s.t \nexists j > i, D_j \in B, D_i \in B' \setminus B\}$. The key notion is that the size of each "slice" (i.e., $Ups(B)$) is a power of two (as it is stated in Prop. 8), and for this reason we define (later in this section) a threshold θ which is always a power of two number.

Prop. 8. Let $\mathcal{D} = \{D_1, \ldots, D_n\}$, and $B = \langle D_j, \ldots, D_k \rangle$ where $j < k \leq n$ and $n = |\mathcal{D}|$. The upper set of subset B, i.e. $Ups(B)$, contains 2^{n-k} subsets.

Proof. The upper set of B contains the nodes $Ups(B) = \{B' \mid B \subseteq B' \; s.t \nexists j > i, D_j \in B, D_i \in B' \setminus B\}$. It means that each dataset that belongs to B, belongs also to B', i.e. for each $D_i \in B, D_i \in B'$ and each of the datasets that belongs to B' and not to B, i.e., for each $D_j \in B'$ where $D_j \notin B$, it holds that $j > k$. From the n datasets, $j > k$ holds for $|D'| = n - k$ in number datasets: $D' = \{D_{k+1}, \ldots, D_n\}$ where $(k + 1 < k + 2 < \ldots < n)$. Each $D_j \in D'$ can either belong to a particular subset $B' \in Ups(B)$, i.e., $D_j \in B'$ or not belong to B', i.e., $D_j \notin B'$. It means that there exists two different options for each dataset. Therefore, for $n-k$ datasets, the number of all possible combinations are $2 \times 2 \times \ldots \times 2$ $(n - k \; times) = 2^{n-k}$, i.e., $|Ups(B)| = 2^{n-k}$. □

In our case, we have $|\mathcal{D}|$ datasets where $\mathcal{D} = \{D_1, \ldots, D_n\}$ and for the pair $\langle D_1, D_2 \rangle$, its upper set contains the power set of the set $\mathcal{D}' = \{D_3, \ldots, D_n\}$, i.e, $2^{|\mathcal{D}|-2}$ nodes which corresponds to 1/4 of all the nodes $(\frac{2^{|\mathcal{D}|-2}}{2^{|\mathcal{D}|}})$. Indeed, in such a case one machine will compute at least 1/4 of the nodes regardless of the number of the available machines. For this reason, we propose a distributed version of the bottom-up algorithm where we split the power set in smaller "slices", where the size of each "slice" L_i is less than or equal to a threshold $|L_i| \leq \theta$ where $\theta = 2^{|\mathcal{D}|}/t, t \geq 1$. For instance, if t equals 16, each "slice" of the power set will be equal to or lower than $2^{|\mathcal{D}|}/16$ of the nodes of the power set. In Figure 5.12, we can see an example of 6 datasets and (2^6 nodes). We assume that there are 16 machines, thereby, we can choose $t = 16$ which implies that $\theta = 64/16 = 4$. It means that the size of each slice of the lattice that will be sent to the reducer will contain 4 or less nodes. It is worth mentioning that the size of each "slice" is always a power of two (as it follows by proposition 8), while for each "slice" we just emit a single key-value pair, as we shall see below.

5.6.3 Bottom-up Lattice-Based Algorithm in Parallel

Here, we show how to compute the commonalities by using a parallel version of the algorithm. However, this algorithm can be easily adjusted for computing also the other three metrics, i.e., *coverage, information enrichment* and *uniqueness.*

Map Phase. Let $UpsR(B)$ be all the nodes belonging in the upper set of a subset B that the algorithm have not explored yet (obviously $UpsR(B) \subseteq Ups(B)$). The *Bottom-up* parallel algorithm for commonalities (shown in Alg. 8 and Alg 9) starts in the mapper by computing θ, traverses a subset of pairs, i.e., the set $B_{pairs} = \{B_i \in P(\mathcal{D}) \mid |B_i| = 2\}$, and calls the recursive function *BupMapper*, for each subset B (a pair of datasets) that belongs in B_{pairs} (lines 3-4). The recursive function *BupMapper* computes the size of $Ups(B)$ (lines 6-7) and checks whether its upper set size is less than or equal to θ (line 8). In such a case (lines 9-10), it sends to the reducer a key-value pair having as key the subset B and as value the following two different parts: a) $Up(B, F)$ and b) a value zero which indicates that the reducer will compute the measurements for the whole upper set of subset L. In particular, when the algorithm reaches line 9, the reducer function (see the first function of Alg. 9) will be triggered for computing the measurements for the upper set of B. Otherwise it assigns $|UpsR(B)| = |Ups(B)|$ (line 11) and then it starts to explore its supersets of the next level (lines 12-13).

Each time it checks whether $|UpsR(B)| \leq \theta$ (line 14) and if yes (i.e., the number of the nodes of this upper set is equal to or lower than θ), it sends to the reducer the subset B, $Up(B, F)$ and a node B' indicating the node from which we will start the computation of the measurements of the nodes belonging in $Ups(B)$ (lines 15-16). Specifically, when line 15 is triggered, the aforementioned key-value pair will be given as an input to the Reducer function (see the first function of Alg. 9). Otherwise, it continues upwards (since we follow a Depth-First Search traversal) with a superset B' where, $B' \supset B$, $B' \in UpsR(B)$ and calls the recursive function again (lines 17-22).

Figure 5.12 shows an example where we have selected $\theta = 4$. At first (see step 1 of node $\langle D_1, D_2 \rangle$), $|Ups(D_1, D_2)| > \theta$, therefore we continue with the triad $\langle D_1, D_2, D_3 \rangle$ (i.e., edge 1 is created). In the case of $\langle D_1, D_2, D_3 \rangle$ (see step 2), the size of its upper set is greater than θ, therefore we continue upwards with the quad $\langle D_1, D_2, D_3, D_4 \rangle$ i.e., edge 2 is created. Then, the size of the upper set of the quad $\langle D_1, D_2, D_3, D_4 \rangle$ is equal to θ (see step 3), so we send this "slice" (i.e., nodes in yellow) to a reducer. Therefore, for this "slice", Alg. 8 sends as key the subset $\langle D_1, D_2, D_3, D_4 \rangle$ and a value with two parts: a) $Up(\{D_1, D_2, D_3, D_4\}, F)$ and b) a value '0' indicating that for this "slice" we should perform the measurements for all the nodes belonging in the upper set of $\langle D_1, D_2, D_3, D_4 \rangle$. Afterwards, we return to the previous node (see step 4), i.e., $\langle D_1, D_2, D_3 \rangle$ (see the edge 3 in Figure 5.12), and we check if the remaining nodes of its upper set, i.e., $|UpsR(D_1, D_2, D_3)| = |Ups(D_1, D_2, D_3)| - |Ups(D_1, D_2, D_3, D_4)|$, is less than or equal to θ (see step 5). In particular, $|UpsR(D_1, D_2, D_3)|$ equals θ, therefore we send to the reducer the remaining upper set of the node $\langle D_1, D_2, D_3 \rangle$ in a reducer (nodes in cyan color). Concerning Alg. 8, in this case it emits a key-value pair consisting of subset $\langle D_1, D_2, D_3 \rangle$ as key, and a value with two parts: a) $Up(\{D_1, D_2, D_3\}, F)$ and b) $\langle D_1, D_2, D_3, D_5 \rangle$ (starting node). In this case, we would like to compute the measurements for the upper set of $\langle D_1, D_2, D_3 \rangle$ but we should exclude from the computation the upper set of $\langle D_1, D_2, D_3, D_4 \rangle$ which will

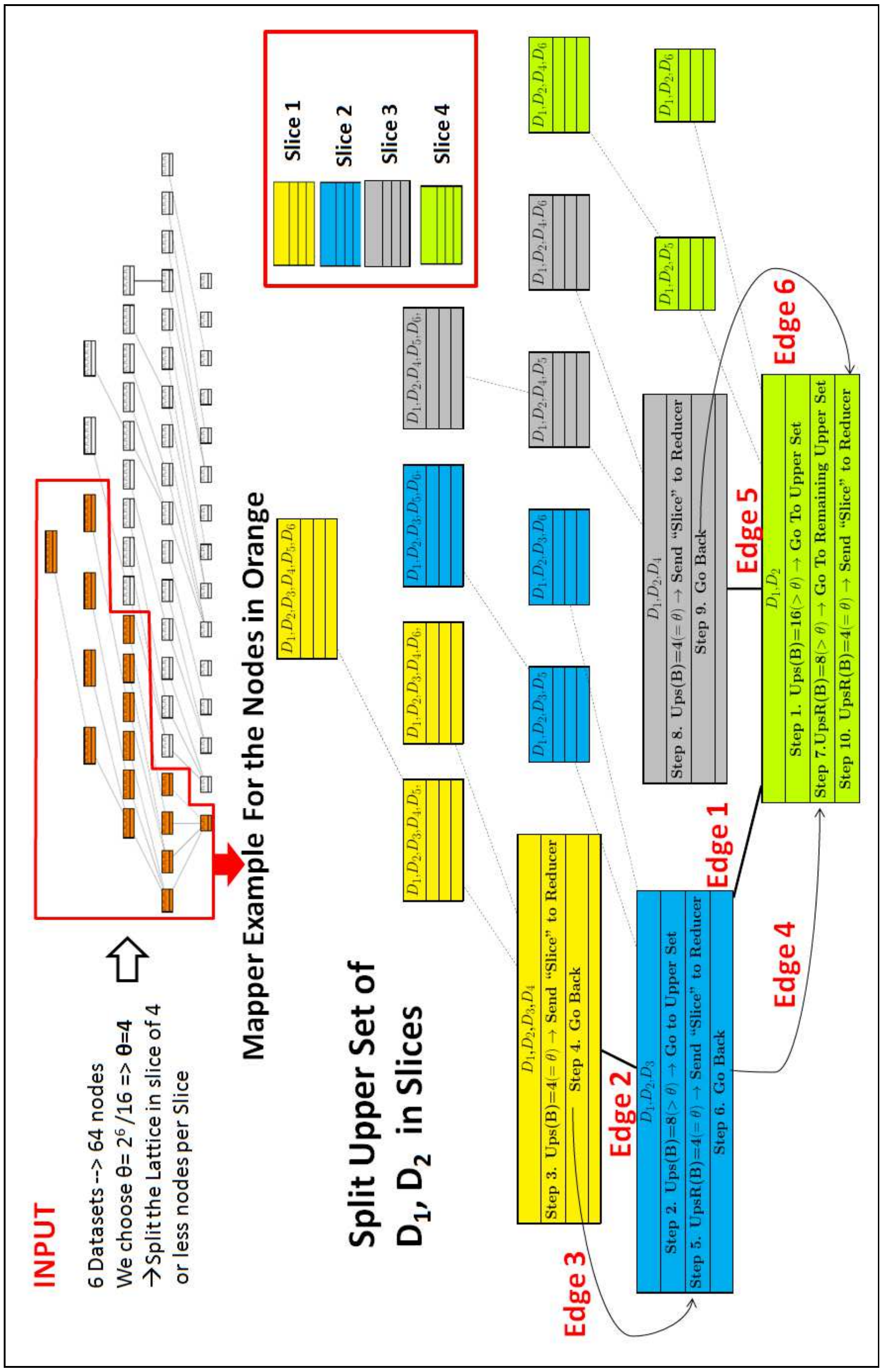

INPUT
6 Datasets --> 64 nodes
We choose θ= 2^6 /16 => θ=4
→Split the Lattice in slice of 4 or less nodes per Slice
Mapper Example For the Nodes in Orange
Slice 1
Slice 2
Slice 3
Slice 4
Split Upper Set of D1, D2 in Slices
D1,D2,D3,D4,D5,D6
D1,D2,D3,D4,D5,
D1,D2,D3,D4,D6,
D1,D2,D3,D5,D6,
D1,D2,D4,D5,D6,
D1,D2,D3,D5
D1,D2,D3,D6
D1,D2,D4,D5
D1,D2,D4,D6
D1,D2,D5
D1,D2,D6
D1,D2,D3,D4
Step 3. Ups(B)=4(= θ) → Send "Slice" to Reducer
Step 4. Go Back
D1,D2,D3
Step 2. Ups(B)=8(> θ) → Go to Upper Set
Step 5. UpsR(B)=4(= θ) → Send "Slice" to Reducer
Step 6. Go Back
D1,D2,D4
Step 8. Ups(B)=4(= θ) → Send "Slice" to Reducer
Step 9. Go Back
D1,D2
Step 1. Ups(B)=16(> θ) → Go To Upper Set
Step 7. UpsR(B)=8(> θ) → Go To Remaining Upper Set
Step 10. UpsR(B)=4(= θ) → Send "Slice" to Reducer
Edge 1
Edge 2
Edge 3
Edge 4
Edge 5
Edge 6

Figure 5.12: Example of splitting the Upper Set of a set of nodes in 4 "Slices", each having 4 nodes

be computed separately, i.e. see "slice" 1 of Figure 5.12. For this reason, we put as a starting node the subset $\langle D_1, D_2, D_3, D_5\rangle$ for starting exploring in the reducer the upper set of $\langle D_1, D_2, D_3\rangle$ from this node and not from $\langle D_1, D_2, D_3, D_4\rangle$. Afterwards, we return to node $\langle D_1, D_2\rangle$ (see step 6 and edge 4) and we check in step 7 whether $|UpsR(D_1, D_2)| < \theta$. Since it is greater than θ, edge 5 is created and we reach the node $\langle D_1, D_2, D_4\rangle$. The size of the upper set of this node equals θ (see step 8), and we send its upper set (nodes in gray) to a reducer. Finally, we return again to node $\langle D_1, D_2\rangle$ (see step 9 and edge 6) and we send its remaining upper set (nodes in green) to a reducer (see step 10) since $|UpsR(D_1, D_2)|$ equals θ. Therefore, each time that we reach a node B whose upper set is equal to or lower than θ, we send a key-value pair to the reducer for computing the measurements for the subsets of $Ups(B)$. Then, we return back to its subset of the previous node and we recheck the size of the "remaining" upper set of the previous node B. Then, we send it to the reducer to compute the "remaining" upper set nodes of B when $|UpsR(B)| \leq \theta$.

We can observe in Figure 5.12, that we finally split the upper set of $\langle D_1, D_2\rangle$ in four different "slices" (and consequently 4 key-value pairs), where the number of the nodes of all of these parts equals 4. Each of these "slices" corresponds to a unique key-value pair, and the "slices" are distributed randomly to the available machines. Therefore, the communication cost (from the mappers to the reducers) of the proposed algorithm is $O(z)$, where z is the number of "slices".

ALGORITHM 8: *Bottom-up* parallel Incremental algorithm Mapper

```
Input: Pairs $B_{pairs}$, $Up$ of pairs and threshold t
Output: Splitting the Lattice in "Slices"
function Mapper ($B_{pairs}$, Up of pairs, t)
    $\theta \leftarrow \frac{2^{|\mathcal{D}|}}{t}$
    forall $B \in B_{pairs}$ do
        BupMapper($B, Up(B,F), \theta$)
function BupMapper (subset $B, Up(B,F), \theta$)
    $D' = \{D_i \in \mathcal{D} | \forall D_k \in B,\ D_i > D_k\}$
    $|Ups(B)| \leftarrow 2^{|D'|}$
    if $|UpsR(B)| \leq \theta$ then
        $emit(B,\ Up(B,F) \mid sNode\ \rightarrow\ 0)$
        $return\ |Ups(B)|$
    $|UpsR(B)| \leftarrow |Ups(B)|$
    $B_{up} \leftarrow \{B_i \mid B_i \supset B, |B_i| = |B| + 1, B_i \in Ups(B)\}$
    forall $B' \in B_{up}$ do
        if $|UpsR(B)| \leq \theta$ then
            $emit(B,\ Up(B,F) \mid sNode\ \rightarrow\ B')$
            return $|Ups(B)|$
        else
            forall $B_i \in Up(B,F)$ do
                if $B_i \subseteq B'$ then
                    $Up(B',F) \leftarrow Up(B',F) \cup \{B_i\}$
            $|Ups(B)|\ \leftarrow\ BupMapper(B', Up(B'), \theta)$
            $|UpsR(B)|\ \leftarrow\ |UpsR(B)| - |Ups(B')|$
    return 0
```

ALGORITHM 9: *Bottom-up* parallel Incremental algorithm Reducer

```
Input: Subset $B$, $Up(B,F)$, starting node $sn$
Output: Computation of $|cmn(B,F)|$
function Reducer ($B, Up(B) \mid sNode\ sn$)
    $|cmn(B,F)| = \sum_{B_i \in Up(B,F)} directCount(B_i,F)$
    BupReducer($B, Up(B,F) \mid sn$)
function BupReducer ($B, Up(B,F) \mid sn$)
    $B_{up} \leftarrow \{B_i \mid B_i \supset B,\ |B_i| = |B| + 1, B_i \in Ups(B)\}$
    forall $B' \in B_{up}, B' \geq sn$ do
        forall $B_i \in Up(B,F)$ do
            if $B_i \subseteq B'$ then
                $Up(B',F) \leftarrow Up(B',F) \cup \{B_i\}$
        $|cmn(B',F)| = \sum_{B_i \in Up(B',F)} directCount(B_i,F)$
        BupReducer($B', Up(B',F), 0$)
```

Reduce Phase. In the reducer (see Alg. 9), the bottom-up algorithm is used for computing the measurements for the nodes of each "slice". In particular, it computes $|cmn(B,F)|$ (line 2 in the right column) and then it starts exploring the upper set of B by calling the recursive function *BupReducer* (line 3). This function computes the measurements for a specific part of $Ups(B)$, i.e., $UpsR(B)$, starting from a specific superset belonging in $UpsR(B)$, i.e., the "starting" node (lines 5-6). Then, for each such superset B', it assigns $Up(B',F)$

Table 5.3: Statistics of indexes for 400 datasets, and for the 25 most popular datasets of each index

Index	Index Size	$\lvert occur(\mathcal{D}, F)\rvert$	Direct Count % of Index Size
Entities ($\lvert\mathcal{D}\rvert = 400$)	368 million	23,357	0.0063%
Literals ($\lvert\mathcal{D}\rvert = 400$)	379 million	336,570	0.0887%
Triples ($\lvert\mathcal{D}\rvert = 400$)	1.8 billion	13,772	0.0007%
Entities ($\lvert\mathcal{D}\rvert = 25$)	303 million	11,139	0.0036%
Literals ($\lvert\mathcal{D}\rvert = 25$)	353 million	64,907	0.0183%
Triples ($\lvert\mathcal{D}\rvert = 25$)	1.6 billion	5,250	0.0003%

where $Up(B', F) \subseteq Up(B, F)$, it finds $|cmn(B', F)|$ for a specific superset and continues upwards (lines 6-11). For instance, for the subset $\langle D_1, D_2\rangle$ of Figure 5.12, Alg. 9 computes its' intersection cardinality (line 2), i.e., $|cmn(\{D_1, D_2\}, F)|$. Afterwards, *BupReducer* function is called (line 3) in order to explore its "remaining" upper set starting from $\langle D_1, D_2, D_5\rangle$. In particular, we ignore $\langle D_1, D_2, D_3\rangle$, $\langle D_1, D_2, D_4\rangle$ since the aforementioned nodes and their upper sets belong to another "slices" and the measurements for them will be computed in another reducers. As we shall see in the experiments, with the parallel version of bottom-up algorithm, we are able to compute the commonalities for the whole lattice even for trillion of nodes (2^{40} subsets of datasets) approximately in 6 hours.

5.7 Experimental Evaluation - Efficiency

Here, we report the results concerning measurements that quantify the speedup obtained by the introduced techniques. We exploit the three largest of the constructed semantics-aware indexes, i.e., *Entity*, *Literals* and *Entity-Triples* index, for evaluating the efficiency of the proposed methods by using a single machine. More information about the datasets which were used for constructing the indexes and statistics about the indexes can be found in Chapter 4 (e.g., see the datasets in Table 4.7 and statistics in Table 4.8).

5.7.1 Datasets & Models

Table 5.3 contains statistics for the three indexes that we use in our experiments (see the first 3 rows). For each index, we provide efficiency measurements for the 25 most popular datasets (see the last 3 rows). We observe that in all the cases, the size of $|occur(\mathcal{D}, F)|$ (i.e., the size of each *directCount* list) is extremely smaller than the number of total entries of each index (see the fourth column of Figure 5.3). Indeed, in the worst case (for the *Literals Index*), the size of the *directCount* list is only 0.0887% of the size of the corresponding index.

Models. Concerning commonalities, we compare the efficiency of five different models: a) SPARQL implementations by using *Virtuoso* and *Blazegraph* query engines (see

§5.2), b) a baseline method, i.e., *BM* (see §5.4), c) the lattice-based incremental top-down approach, i.e., *Top-Down (BFS)* (see §5.5.2.2), d) the lattice-based incremental bottom-up approach, i.e., *LB (DFS)* (see §5.5.2.1) and e) the previously mentioned approach by pruning and regrouping the redundant datasets in $Up(B,F)$, i.e., *LB+UPGR* (see §5.5.2).

As regards the rest metrics (coverage, information enrichment and uniqueness) we compare the following four models: i) a baseline approach, i.e., *BM* (see §5.4), ii) the lattice-based incremental bottom-up approach, i.e., *LB*, for all the metrics (see §5.5.3, §5.5.4 and §5.5.5), iii) the lattice-based incremental approach by pruning the totally redundant entries, i.e., *LB+TPR*, (see §5.5.3.2), and iv) the lattice-based incremental approach with pruning and regrouping, i.e., *LB+PRGR* (see §5.5.3.3).

The Selected Performance Metrics. Here, we evaluate the speedup obtained by the proposed methods for computing the metrics for all the possible subsets of datasets. Certainly, for finding the $covBest(K,F)$ (or $cmnBest(K,F)$, etc.) for a given query, e.g., "I desire to find the triad of datasets containing the most triples for entity e", we compute the metrics only for a small part of the lattice, i.e., we stop at level $L = K = 3$. However, the objective is to evaluate the worst case, i.e., how fast are the proposed methods when the metrics for all the possible subsets of datasets should be computed. Indeed, we measure the execution time for $10-25$ datasets (i.e., the 25 most popular datasets in each index) by using preconstructed *directCount* lists, i.e., we compute the metrics for 1,024 (2^{10}) to 33 million (2^{25}) subsets of datasets. Moreover, we compute the average number of entries (i.e., $Up(B,F)$ and $occur(B,F)$) that are transferred from a subset B to its superset B' ($B = prev(B')$) and scanned for computing the metrics of B'.

Plots. For most of the experiments that concern the lattice-based measurements, the y-axis of plots is in log_4 scale and shows the seconds that were needed for computing the measurements. Concerning the x-axis, it shows the number of datasets which were used for each experiment, e.g., if $x = 20$ it means that we used 20 datasets and we computed the metrics for 2^{20} (1,048,576) subsets of datasets.

Hardware Setup & Code. For all the experiments presented in this section (except for the measurements for the parallel version of algorithms), we used a single machine having an *i5* core, 8 GB main memory and 1TB disk space. One can have access to all the datasets and the code (in JAVA programming language), in `http://www.ics.forth.gr/isl/LODsyndesis`.

5.7.2 Efficiency of Commonalities Metrics

Here, we provide results for computing commonalities concerning the efficiency of SPARQL implementations, and of the proposed methods that rely on the semantics-aware indexes (i.e., a baseline one and the lattice-based methods).

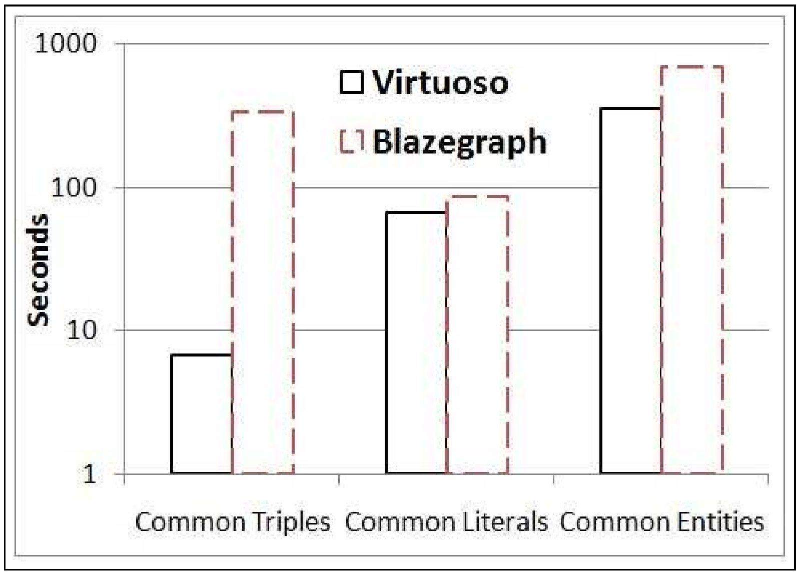


Figure 5.13: Execution Time for finding the commonalities among any subset of 10 datasets (i.e., 45 pairs of datasets), by using Virtuoso and Blazegraph and 2 million triples

5.7.2.1 Comparison of SPARQL implementations - Virtuoso versus Blazegraph

Here, we compare the execution time of two different SPARQL implementations. In particular, we have used a Virtuoso triplestore i.e., *Openlink Virtuoso Version 06.01.3127* (`http://virtuoso.openlinksw.com/`), and a Blazegraph triplestore (`http://www.blazegraph.com`), i.e., *Blazegraph Version 2.1.4*.

Since Blazegraph does not support inference in quads mode, we compare these implementations by using a subset of the semantics-aware triples presented in Chapter 4 (where we have already computed the inference). Indeed, we have selected 10 datasets which contain in total 2 million triples and we have uploaded them to each triplestore. Afterwards, we sent queries for finding the number of common entities, triples and literals between any pair of these 10 datasets, i.e., in total 45 pairs of datasets.

As we can see in Figure 5.13, the SPARQL implementation of *Virtuoso* is faster comparing to *Blazegraph* for all the different cases. *Virtuoso* needed for all the 45 pairs of datasets, 6.7 seconds for finding the common triples, 65 seconds for the case of literals, and 349 seconds for entities, whereas the corresponding execution times of *Blazegraph* are far higher (especially in the case of triples), i.e., 332, 85 and 680 seconds, respectively. Since *Virtuoso* offers faster execution times for the given problems, below we evaluate the performance of *Virtuoso*, by adding even more triples, whereas we evaluate the efficiency of *Virtuoso* by computing the closure on query time.

Table 5.4: Execution time for SPARQL queries for measuring the commonalities between pairs and triads of 10 datasets, by using Virtuoso and 58 million triples.

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

5.7.2.2 Efficiency of Virtuoso SPARQL implementation by Adding more triples

For evaluating Virtuoso triplestore by using much more triples, we selected and uploaded 10 datasets, having 58 million triples and 1 million equivalence relationships. Afterwards, we sent queries for finding the number of common entities, triples and literals of these 10 datasets. In particular, we report the execution time for computing the commonalities between pairs (45 different pairs) and triads (120 different pairs) for this 10 datasets. As we can see in Table 5.4, by using *Virtuoso*, for finding the common real world entities between all the pairs of datasets, we needed approximately 45 minutes (on average 1 minute per pair of datasets). If we do not take into account the `owl:sameAs` relationships and their transitive and symmetric closure (like in the case of §5.7.2.1) 15 minutes were needed for computing the common entities between the pairs of datasets (on average 20 seconds for each pair of datasets). Concerning the triples, we needed even more minutes for finding the common triples, i.e., 50 minutes, since we should compute the closure for both instance and schema elements. On the contrary, by avoiding the computation of inference on query time (i.e., by using the semantics-aware triples) the execution time was 1.5 minute. Regarding the common literals, the execution time was 7 minutes (approximately 10 s per pair on average). Concerning triads of sources, we can see that the execution time increased for each measurement, from 1.84 to 4.82 times.

5.7.2.3 Efficiency of Lattice-based Incremental Algorithms

In comparison to a SPARQL implementation, here we will show that by using even a single machine and the lattice-based algorithms, we can compute the metrics far faster.

Lattice-Based approaches versus a SPARQL implementation. The difference comparing to the plain SPARQL implementations is obvious. Even by having pre-computed the closure and by using much less triples (i.e., 58 million), we need on average 20 seconds for computing the common entities of a specific pair of datasets, while the lattice-based approaches require much less time for performing the measurements even for million subsets of datasets and higher number of triples (i.e., billions of triples), as it can be seen in

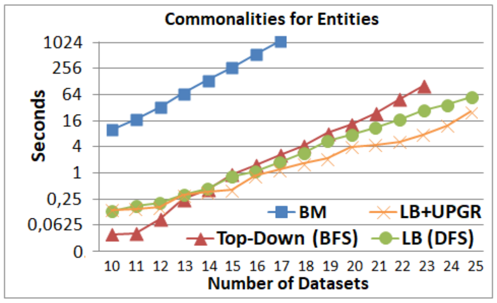


Figure 5.14:
Execution time of $|cmn(B, RWE)|$ of Entities Index among any combination of 10-25 datasets ($2^{10} - 2^{25}$ subsets B)

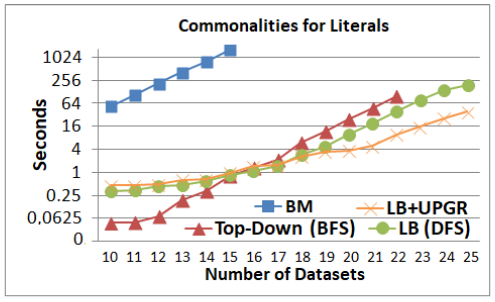


Figure 5.15:
Execution time of $|cmn(B, LIT)|$ of Literals Index among any combination of 10-25 datasets ($2^{10} - 2^{25}$ subsets B)

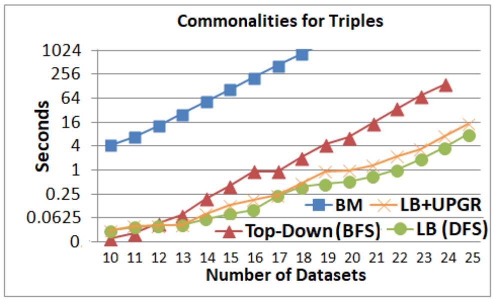


Figure 5.16:
Execution time of $|cmn(B, RWT)|$ of Triples Index among any combination of 10-25 datasets ($2^{10} - 2^{25}$ subsets B)

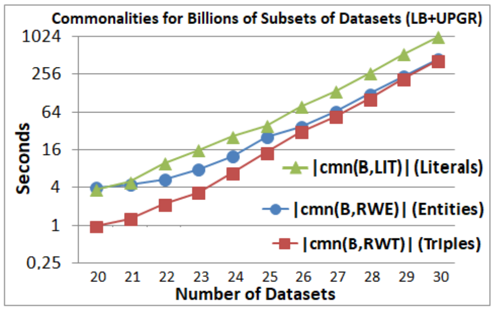


Figure 5.17:
Execution time of commonalities by using *LB+UPGR* among any combination of 20-30 datasets ($2^{20} - 2^{30}$ subsets B)

Figures 5.14, 5.15 and 5.16. In particular, for computing the common entities for 1 million subsets of datasets (2^{20}), we needed a) 13 seconds by using the *Top-Down BFS* approach, b) 7.5 seconds by using the bottom-up *LB (DFS)* approach, and c) 4 seconds by using the bottom-up approach with regrouping. Finally, even the baseline approach is much faster comparing to a SPARQL implementation, i.e., it can compute the metrics for thousands of lattice nodes in 10 seconds.

Baseline method vs Lattice-Based approaches. As we can see in Figures 5.14, 5.15 and 5.16, the incremental algorithms are much more efficient than a baseline approach. In the first three rows of Table 5.6 we show the maximum speedup obtained in the experiments through the incremental approaches versus the *BM* approach. In particular, we achieved even $4,921\times$ speedup by using the *LB+UPGR* approach comparing to a baseline one (it is quite slow as the number of datasets grows). Generally, for all the different models, we can see in Figures 5.14, 5.15 and 5.16, that the execution time increases exponentially as

Table 5.5:
Statistics for measuring $|cmn(B,F)|$ for each index incrementally, for 2^{20} subsets

Category	Top-Down	LB (DFS)	LB+ UPGR
avgIt (Entities)	-	37	6.1
avgIt (Triples)	-	1.4	1.2
avgIt (Literals)	-	53.1	7
Exec. Time (Entities)	13s	7.4s	4s
Exec. Time (Triples)	6.5s	0.4s	0.9s
Exec. Time (Literals)	24s	9.7s	3.6s

Table 5.6:
Max Speedup by using different models for measuring $|cmn(B,F)|$, for each index

Max Speedup of	Entities	Triples	Literals
Top-Down vs BM	684×	446×	3,076×
LB (DFS) vs BM	1,409×	3,555×	4,350×
LB+UPGR vs BM	3,555×	1,785×	4,921×
LB (DFS) vs Top-Down	3.6×	21×	2.46×
LB+UPGR vs Top-Down	12.9×	11.7×	9.7×
LB+UPGR vs LB (DFS)	3.5×	-	5.61×

we add more datasets, which is rational since each time that we add one more dataset, the number of possible subsets of datasets increases exponentially.

Bottom-Up (LB DFS) vs Top-Down (BFS) Approach. Concerning the two incremental approaches, we can clearly see the trade-off in all the experiments. Indeed, for lower number of datasets the top-down approach is faster, since the cost of creating more edges is lower than the cost for checking the $Up(B,F)$ of each subset B. As the number of datasets grows (for ≥ 15 datasets in Figures 5.14, 5.15 and 5.16), the bottom-up approach is faster, since the number of edges (and their cost) increases greatly for the top-down approach. Moreover, in all the experiments for $|\mathcal{D}| > 23$, it was infeasible to use the top-down approach due to memory limitations while with the bottom-up approach we can compute the metrics even for billions of subsets. Finally, as we can see in Table 5.6, we observed even $21\times$ speedup by using the bottom-up *LB (DFS)* approach versus the *top-down* one.

The Gain of Regrouping the $Up(B,F)$ **for the bottom-up approach.** By removing the redundant datasets in each entry of $Up(B,F)$, we achieved a speedup concerning the measurements for entities and literals, i.e., even $3.5\times$ for entities and $5.6\times$ for literals. This difference can be explained by the number of iterations of each approach, i.e., see Table 5.5. In particular, for 2^{20} subsets of datasets, for computing the commonalities of entities the average number of iterations for the *LB (DFS)* is 37, whereas the corresponding number for the *LB+UPGR* is 6.1. Regarding literals, the average number of iterations is 53.1 for *LB (DFS)* and for *LB+UPGR* is 7. However, concerning triples, the *LB* approach was faster than the *LB+UPGR*, because there was not a big gain by the regrouping approach, i.e., in both cases the average number of iterations was very small. Finally, the best execution time that we achieved for 2^{20} subsets of datasets for the three different indexes were the following: a) 4 seconds for computing the commonalities of entities through the *LB+UPGR*, b) 0.4 seconds for computing the commonalities for literals through *LB*, and c) 3.6 seconds for the case of literals through *LB+UPGR*.

Experiments for billions of subsets of datasets In Figure 5.17, we can see experiments

Table 5.7: Execution time for finding the commonalities for subsets of 400 RDF Datasets by using the bottom-up lattice-based algorithm.

Connectivity Measurement	**Number of Subsets Measured**	**Execution Time**
Common RW Entities	18,531,752	51 s
Common RW Triples	1,776,136	3 s
Common Literals	4,979,482 (pairs and triads)	328 s

even for 1 billion of subsets (2^{30}) by using the *LB+UPGR* approach. Specifically, for computing the commonalities for 1 billion subsets, we needed 7.5 minutes for the case of entities, 7 minutes for triples and 17 minutes for literals. For the literals, the input (i.e., the number of posting lists) is quite larger comparing to triples and entities, therefore we expected that the execution time of literals would be longer.

Experiments for all the 400 RDF datasets. In Table 5.7, we introduce the execution time for computing by using the bottom-up lattice-based algorithm the number of common entities, triples and literals for pairs, triads, quads, and quintets i.e., $2 \leq L \leq 5$, of 400 datasets, that share at least one common element (we ignore subsets that do not share elements). For measuring the cardinality of intersection of common entities for 18 million subsets (belonging to pairs, triads and quads and quintets), we needed 51 s, i.e., approximately 363,000 subsets per second. Regarding the computation of common real world triples, the execution time was 3 s for computing the commonalities between 1.7 million subsets, i.e., 592,000 subsets per second, since only a small number of subsets of datasets share common triples. On the contrary, a lot of subsets of datasets share common literals, and for this reason we measured the common literals of datasets only for pairs and triads of sources. The execution time for measuring the common literals for 4.9 million subsets was 328 s, i.e., 15,000 subsets per second. Moreover, the size of the *directCount* list affects the execution time. In particular, the size of the *directCount* list of common literals is 14 times bigger than the *directCount* list of common entities, and 24 times bigger than the corresponding list of triples. As a consequence, in one second we can measure 39 times more subsets containing common triples, and 24 times more subsets having common entities, in comparison to the number of subsets containing common literals.

5.7.3 Efficiency of Coverage

Here, we provide results for computing coverage that concern the efficiency of the proposed methods (i.e., a baseline one and the different lattice-based methods).

Baseline method versus Lattice-based approaches. Figures 5.18, 5.20 and 5.19 show experiments for computing $|cov(B, F)|$, for all the three indexes. It is obvious that the incremental approaches are extremely faster comparing to the baseline *BM* model for all the three cases. More specifically, in the first three rows of Table 5.9 we show the maximum

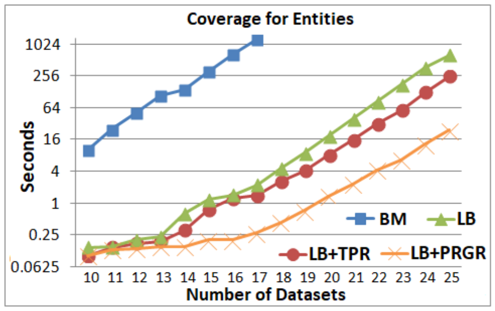


Figure 5.18:
Execution time of $|cov(B, RWE)|$ of Entities Index among any combination of 10-25 datasets ($2^{10} - 2^{25}$ subsets B)

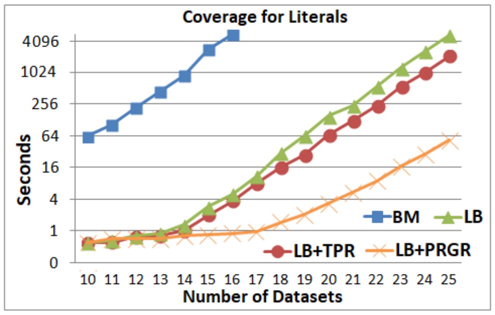


Figure 5.19:
Execution time of $|cov(B, LIT)|$ of Literals Index among any combination of 10-25 datasets ($2^{10} - 2^{25}$ subsets B)

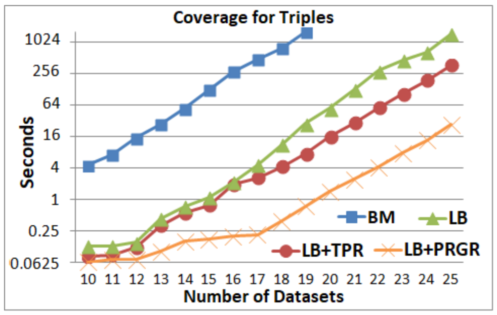


Figure 5.20:
Execution time of $|cov(B, RWT)|$ of Triples Index among any combination of 10-25 datasets ($2^{10} - 2^{25}$ subsets B)

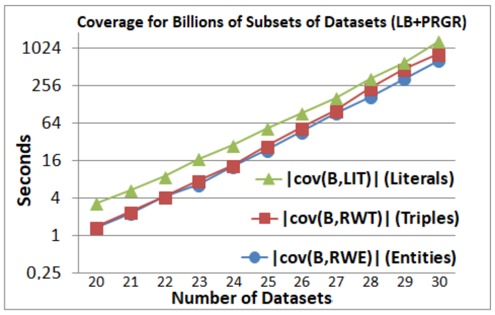


Figure 5.21:
Execution time of coverage by using *LB+PRGR* among any combination of 20-30 datasets ($2^{20} - 2^{30}$ subsets B)

speedup obtained in the experiments of Figures 5.18, 5.20 and 5.19, by using each of the incremental approaches versus the *BM* one. For computing the coverage for literals, the *LB* was even $1,099\times$ faster than *BM*, while the *LB+TPR* and the *LB+PRGR* approaches were up to $1,459\times$ and $6,000\times$ faster, respectively. Indicatively, *BM* needs more than 10 minutes for computing the coverage of entities for 65,536 (2^{16}) subsets of datasets, while by using the best incremental approach (*LB+PRGR*), we computed the coverage, for millions of subsets (i.e., $2^{20} subsets$), for entities in 1.3 seconds, for triples in 1.4 seconds and for literals in 3.2 seconds. Finally, similarly to the case of commonalities, for all the different models, the execution time in many cases increases exponentially.

The Gain of Removing the Totally Redundant Entries. In Figures 5.18, 5.19 and 5.20, we can see a clear speedup by using *LB+TPR* instead of *LB*, especially as the number of datasets grows. In particular, we observed up to $3\times$, $5\times$ and $2.37\times$ speedup comparing

Table 5.8:
Statistics for measuring $|cov(B, F)|$ for each index incrementally, for 2^{20} subsets

Category	LB	LB+TPR	LB+PRGR
avgIt (Entities)	110	44	6.1
avgIt (Triples)	191	87	6.6
avgIt (Literals)	473	263	12.9
Exec. Time (Entities)	18s	9s	1.3s
Exec. Time (Triples)	55s	19.8s	1.4s
Exec. Time (Literals)	151s	64s	3.2s

Table 5.9:
Max Speedup by using different models for measuring $|cov(B, F)|$, for each index

Max Speedup of	Entities	Triples	Literals
LB vs BM	565×	136×	1,099×
LB+TPR vs BM	891×	214×	1,459×
LB+PRGR vs BM	5,773×	2,284×	6,000×
LB+TPR vs LB	3×	5×	2.37×
LB+PRGR vs LB	28×	68×	97×
LB+PRGR vs LB+TPR	10×	14×	33×

to *LB* (see Table 5.9) for entities, triples and literals, respectively. Indicatively, *LB* needs 55 seconds for computing the union of triples for 2^{20} subsets of datasets, while by using *LB+TPR* only 19.8 seconds were needed. This difference is rational according to the analysis of §5.5.3.4, and to the statistics presented in Table 5.8. Indeed, for each different index, we transfer to a superset B' and iterate over smaller *directCount* entries for computing $|cov(B', F)|$, by using *LB+TPR* instead of *LB*. In particular, we performed on average 44 iterations (in line 8 of Alg. 7) by using *LB+TPR* versus 110 by using *LB* for entities, 87 iterations versus 191 for triples, and 263 iterations versus 473 for literals.

The Gain of Pruning and Regrouping. Figures 5.18-5.19 show a clear speedup due to the pruning and regrouping approach (*LB+PRGR*). In particular, we identified even 33× speedup comparing to *LB+TPR* and 97× speedup comparing to *LB* (i.e., see Table 5.9). The difference between all the other models is big, especially as more datasets are added. This difference can be explained by measuring the number of iterations for computing the metric for a superset B' (and by the analysis of §5.5.3.4). For instance, for computing the metrics for 2^{20} subsets of datasets, on average we iterate over only 6.1 *directCount* entries for entities, 6.6 for triples and 12.9 for literals, while even for the second best approach, the corresponding numbers are far higher (see Table 5.8). Indeed, for the case of literals, we performed on average 20× more iterations by using *LB+TPR* instead of *LB+PRGR*.

Measurements for Billions of Subsets. *LB+PRGR* can compute the coverage even for billions of subsets of datasets (2^{30}) as we can see in Figure 5.21. In particular, it needs approximately 12 minutes for entities, 14 minutes for triples and 22 minutes for literals.

5.7.4 Efficiency of Information Enrichment and Uniqueness

For computing $|enrich(B, F, D_m)|$ and $|uniq(D_m, F, B)|$, the algorithms are similar to $|cov(B, F)|$. The main difference is that for the $|enrich(B, F, D_m)|$, we use as input the set $occur(\mathcal{D}, F)_{\setminus\{D_m\}}$, and for the $|uniq(D_m, F, B)|$ the set $occur(\{D_m\}, F)$. Here, we show an indicative experiment for each case by using the *Entity Index*. We have selected a specific dataset D_m whose $|occur(\{D_m\}, RWE)| = 4,031$ and its $|occur(\mathcal{D}, RWE)_{\setminus\{D_m\}}| = 7,108$. We expect that their

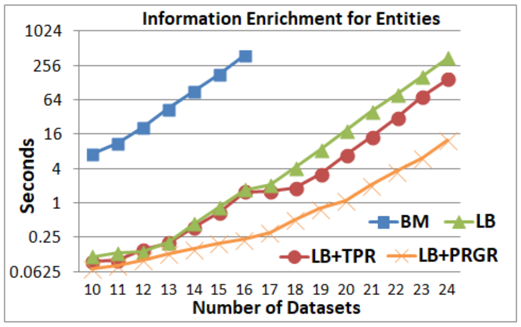


Figure 5.22:
Execution time of $|enrich(B, RWE, D_m)|$ among any combination of 10-24 datasets

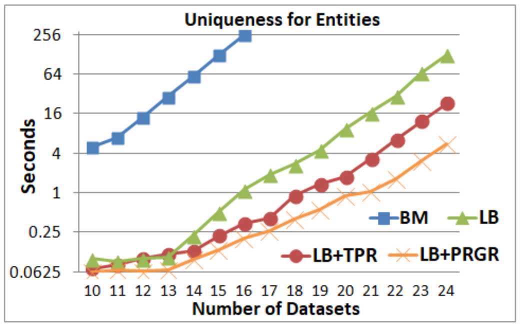


Figure 5.23:
Execution time of $|uniq(D_m, RWE, B)|$ among any combination of 10-24 datasets

execution time will be lower comparing to coverage, since its input ($|occur(\mathcal{D}, RWE)| = 11,139$) is larger comparing to $|occur(\mathcal{D}, RWE)_{\setminus\{D_m\}}|$ and $|occur(\{D_m\}, RWE)|$.

Information Enrichment. We can observe in Figure 5.22, that the incremental approaches are very fast comparing to *BM*, e.g, *LB* is up to $935\times$ faster. Moreover, the *LB+TPR* method offered up to $2.82\times$ speedup versus *LB*, while we obtained up to $11.7\times$ speedup by using *LB+PRGR* instead of *LB+TPR*. Since the input is smaller in this case, the computation of $|enrich(B, RWE, D_m)|$ was up to $1.23\times$ faster comparing to $|cov(B, RWE)|$. Indicatively, by using *LB+PRGR* we needed 1.1 seconds to compute $|enrich(B, RWE, D_m)|$ for over 1 million subsets of datasets (2^{20} subsets).

Uniqueness. We can observe in Figure 5.23, that *LB*, *LB+TPR* and *LB+PRGR* approaches are far faster comparing to *BM*, e.g., *LB* is up to $635\times$ faster than *SFBM*. Moreover, *LB+TPR* was even $5.48\times$ faster than *LB* and *LB+PRGR* was up to $4.1\times$ faster than *LB+TPR*. Comparing to $|cov(B, RWE)|$ and $|enrich(B, RWE, D_m)|$, the computation of $|uniq(D_m, RWE, B)|$ was up to $2.6\times$ and $2.3\times$ faster, respectively, which is rational since the input was much smaller. Indicatively, by using *LB+PRGR* we needed 0.8 seconds to compute $|uniq(D_m, RWE, B)|$ for 2^{20} subsets of datasets.

5.7.5 Efficiency of Parallel Lattice-Based Measurements

Here we compare the execution time of the creation of Lattice by selecting different number of machines for computing $|cmn(B, RWE)|$. For performing the experiments, we use a cluster of 64 virtual machines, each of them having a single core and 1GB main memory.

In particular, we perform experiments for a) the same number of datasets (i.e., 35 datasets), different thresholds and 64 machines (see Table 5.10), b) a specific threshold for a different number of machines (from 1-64 machines) and c) execution time for different size of datasets (from 20-40 datasets), a specific threshold and 64 machines. Regarding a),

Table 5.10: Lattice Measurements for 35 datasets & 2^{35} nodes (34.35 Billions of Nodes)

Size of each "slice"	Number of "slices"	Maximum Nodes /all Nodes from one m_i	Distance from Ideal (Ideal is 1.56%)	Execution Time (Minutes)
≤ 1/4 of all Nodes	595	25.10%	23.54%	185.00
≤ 1/8 of all Nodes	596	18.80%	17.24%	147.00
≤ 1/16 of all Nodes	600	12.60%	11.04%	95.00
≤ 1/32 of all Nodes	611	7.90%	25.1%	59.00
≤ 1/64 of all Nodes	637	6.00%	6.34%	45.00
≤ 1/128 of all Nodes	694	4.10%	2.54%	31.50
≤ 1/256 of all Nodes	814	3.10%	1.54%	25.00
≤ 1/512 of all Nodes	1,061	2.85%	1.29%	22.50
≤ 1/1024 of all Nodes	1,563	2.79%	1.23%	20.20
≤ 1/2048 of all Nodes	2,576	1.64%	0.08%	12.10
≤ 1/4096 of all Nodes	4,612	1.62%	0.06%	12.30
≤ 1/8192 of all Nodes	8,695	1.60%	0.04%	12.50

we can observe in Table 5.10 measurements concerning the lattice of 35 nodes by using 64 machines. In this experiment, we change the threshold each time. In particular, we split the power set in smaller "slices" where the size of each "slice" is less or equal than a specific number of nodes. Since we use 64 machines, the ideal case is each machine to compute the measurements for 1/64 of all the nodes, i.e., 1.56%. We can observe that when we split the lattice in 595 "slices", i.e. a "slice" for each pair of datasets (number of pairs=$\frac{|\mathcal{D}|*|\mathcal{D}-1|}{2}$), where the number of the nodes of each "slice" is less or equal than 1/4 of all the nodes, the execution time of the measurements was over 3 hours and one machine computed the measurements for 25.1% of all the nodes. On the contrary, by choosing the number of each "slice's" nodes to be less or equal than 1/2048 of all the nodes, 2,576 "slices" are created. In that case, the algorithm needed just 12 minutes for performing the measurements, whereas there was not a single machine that computed more than 1.64% of all the nodes (which is close to the ideal case). Moreover, by creating more "slices" where the size of each "slice" is smaller (e.g., each "slice" to be ≤ 1/4096 or ≤ 1/8192 of all the nodes of the power set), we can go "closer" to the ideal case, however, this can lead to an increase in execution time (but no so much as it is shown in the last column of Table 5.10), since the number of key-value pairs, i.e, "slices" are larger, and the communication cost increases. The "slices" are distributed randomly in the machines, therefore, it seems that by choosing to create more "slices", where each "slice" does not contain a large number of nodes, it is not so possible for a single machine to compute a large proportion of nodes comparing to the number of nodes that each of the other machines will compute.

In Figure 5.24, we can observe the scalability which is achieved by using different number of machines. More specifically, we can see that each time that we double the number of machines, the execution time is reduced in half. One machine needs over 11 hours for

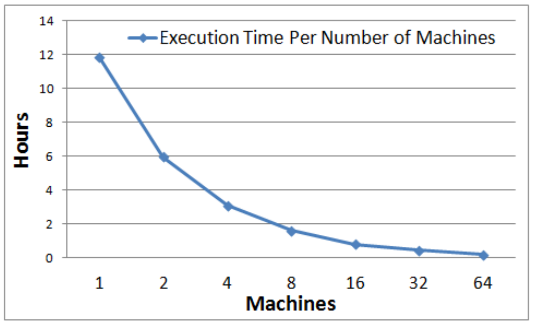


Figure 5.24:
Lattice-based Measurements for Different Number of Machines & 35 datasets

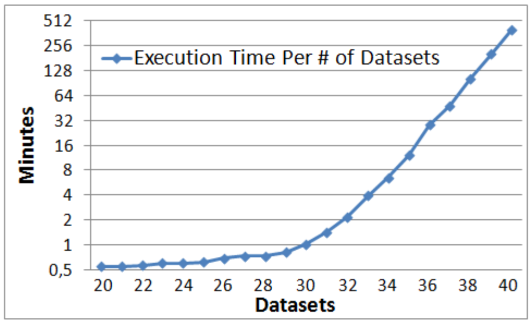


Figure 5.25:
Lattice-based Measurements for Different Number of Datasets and 64 Machines

computing the measurements of 2^{35} subsets while 64 machines need 12 minutes. Concerning the measurements for different number of datasets (by using 64 machines), in Figure 5.25, we can see that for 1 billion or less nodes (e.g. 30 or less datasets), less than 1 minute is needed for computing the lattice of all the nodes. As the number of datasets increase, the execution time is increased exponentially. However, this algorithm was able to compute the measurements for over 1 trillion of nodes, i.e., lattice of 40 datasets, approximately in 6 hours.

5.8 Connectivity Analytics over LOD Cloud Datasets

Here, we show some indicative connectivity measurements for 400 LOD Cloud datasets and 2 billion triples, by using the semantics-aware indexes of Chapter 4 and the content-based metrics that were introduced in this chapter. At first, we show the impact of transitive and symmetric closure. Second, we introduce some general connectivity measurements that indicate the power-law distribution for any category of elements, and afterwards we show subsets of datasets that are highly connected. Moreover, we show indicative measurements by using the proposed union and complement metrics. Finally, we describe some conclusions derived by the experiments.

5.8.1 Datasets Used.

We use the set of 400 real RDF datasets (see Table 4.7) and the 44 million equivalence relationships, which are described in Chapter 4 (i.e., in §4.7).

Table 5.11: Statistics for equivalence relationships.

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

5.8.2 Connectivity Gain from transitive and symmetric closure computation.

In Table 5.11, we report the results of the transitive and symmetric closure for `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass` relationships. By computing the closure, we inferred more than 73 million new `owl:sameAs` relationships, i.e., the increase percentage of `owl:sameAs` triples was 163%. Moreover, for 26 million entities there exists at least two URIs that refer to them (on average 2.6 URIs for the aforementioned entities). Furthermore, we should note that the computation of closure had as a result 2,725 new connected pairs of datasets and 37,445 new connected triads that share at least one common real world entity.

On the contrary, we inferred only 935 `owl:equivalentProperty` relationships (i.e., increase percentage was 11.46%) and 1164 `owl:equivalentClass` triples (i.e., increase percentage was 29%), while there exists 4121 properties and 2041 classes containing having at least two URIs that describe them (on average 2.05 for such properties and 2.11 for such classes).

5.8.3 Statistics Derived by the Indexes.

In Table 5.12, we can see that only a small percentage of each set of elements exists in two or more datasets. In particular, only 0.8% of triples occur in ≥ 2 datasets and 0.24% in ≥ 3 datasets. The corresponding percentages of entities (i.e., 7.73%) and literals (i.e., 11.88%) occurring in ≥ 2 datasets are far higher comparing to triples. However, again most entities and literals occur in 1 dataset. Regarding classes and properties, only a small percentage of them (i.e., less than 1%) occur in ≥ 2 datasets, which means that possibly there is a lack of equivalence relationships between schema elements. For investigating such a case, we created also a different index of triples, where we ignore the property of each triple i.e., we find the common subject-object pairs. For constructing such an index, one can use Algorithm 3; however, one should replace in the mapper (in lines 4 and 6) all the property IDs with a fixed value. As we can see in Table 5.12, if we ignore the property of each

Table 5.12: Elements per number of datasets.

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

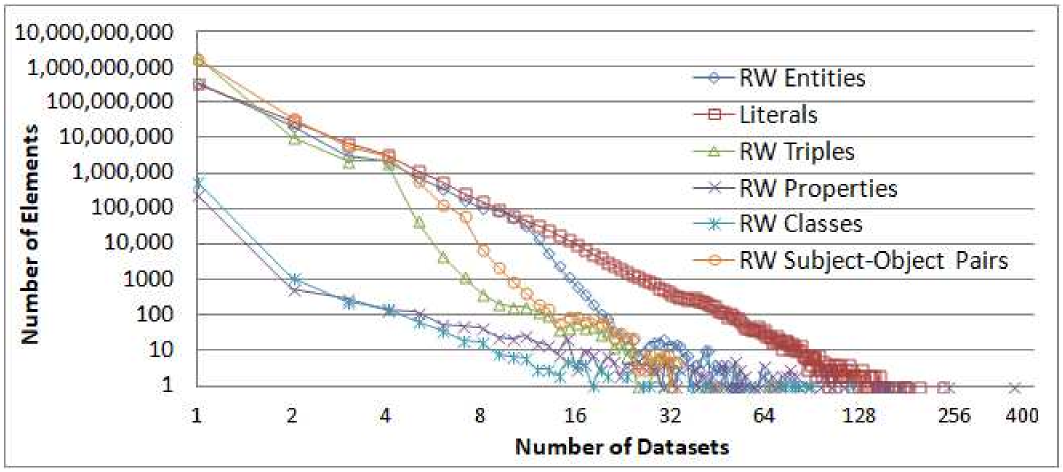


Figure 5.26: Number of datasets where different sets of elements occur.

triple, 2.82% of subject-object pairs occur in two or more datasets, whereas, by taking into account the properties, the corresponding percentage was 0.8%. However, we should mention that it is possible that two or more common subject-object pairs use different properties for describing different facts. For instance, suppose that two different datasets contain the following triples in two datasets D_i and D_j, ⟨di:Aristotle, di:wasBornIn, di:Stagira⟩ and ⟨dj:Aristotle, dj:livedIn, dj:Stagira⟩. These two triples describe different things, however, their subject and objects refer to the same entities.

Moreover, in Figure 5.26, we can observe the distribution of different elements (e.g., triples, entities, etc.), i.e., the number of elements that can be found in a specific number of datasets. We can clearly see a power-law distribution for any category of elements, i.e., there exists a large number of elements (e.g., literals, entities) that occur in a small numbers of datasets, while only a small number of elements can be found in a lot of datasets. It is worth mentioning that there exists over 300,000 entities and over 500,000 of literals that occur in 10 or more datasets; however, less than 1600 real world triples can be found in more than 10 datasets.

Table 5.13: Connected subsets of datasets.

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

5.8.4 Connectivity analytics based on content-based intersection metrics.

In Table 5.13, we show the connectivity among pairs and triads of datasets for different elements, while we also mention the number of datasets that are disconnected, i.e., for a specific measurement type (e.g., number of common literals), these datasets do not have commonalities with other ones. Generally, a big percentage of pairs of datasets share literals, i.e., 78%. It is rational, since each literal can be used for describing several things. For example, the literal “1980” can be the birth year of several people, the year when a movie released, and so forth. Thereby, a lot of datasets can contain this literal, for describing different facts. On the contrary, for the real world entities and triples, only 11.3% of datasets pairs have common entities and 5.59% of pairs of datasets contain same triples. It means that only half of the dataset pairs containing common entities, share also common triples.

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

However, it is also important to check the “degree” of connectivity of the connected pairs of datasets, i.e., how many common elements the aforementioned connected pairs of datasets share. In Figure 5.27, we show the number of datasets’ pairs, whose cardinality of common elements belong to a specific interval of integers, e.g., for the interval $[1, 10)$ we show how many connected pairs have from 1 to 9 common elements (e.g., entities). In particular, most pairs of datasets share less than 10 elements for each set of elements (literals, triples, and entities, properties and classes). In general, we observe a power-law distribution, since many pairs of datasets share a few elements, while only a few pairs of datasets share thousands or millions of elements. Furthermore, we can observe the

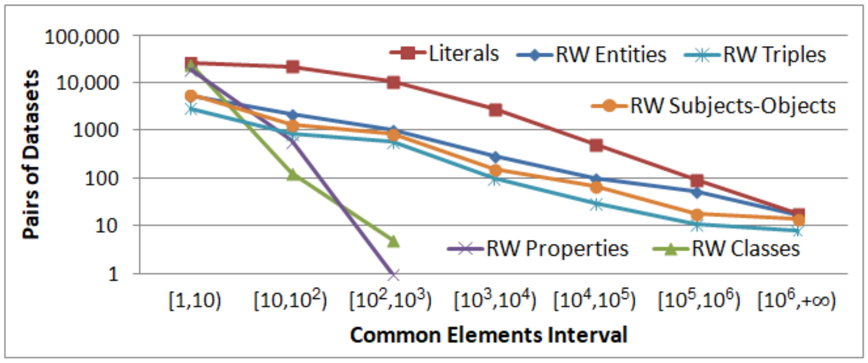


Figure 5.27: Number of connected pairs of datasets per interval for each measurement type F.

difference in the level of connectivity among literals and other elements, e.g., over 10,000 pairs of datasets share at least 100 literals, whereas 1525 pairs of datasets have at least 100 common entities. Concerning common triples, the 84% of connected pairs share less than 100 triples, while as we can see in Figure 5.27, less than 10 pairs share more than one million triples. Regarding entities, most connected pairs (83.2%) have in common 100 or less entities, and only a few pairs share thousands or millions of entities, while for the literals, 77.5% of connected pairs share less than 100 literals. For the remaining categories, most connected pairs sharing properties (96.9% of pairs) and classes (99.4% of pairs) have in common less than 10 properties and classes, respectively.

For the triads of datasets, the percentages are even smaller. In particular, approximately 1% of triads share common elements, classes and triples. However, for the literals the percentage is quite high, i.e., 46.44% of triads share common literals, while 5.38% of triads share a property. In Figure 5.28, we can observe the "degree" of connectivity of triads. Similarly to the case of pairs, we observe a power-law distribution, i.e., most triads of datasets have in common from 1 to 10 elements for each different set of elements. Moreover, it is worth noting that 140 triads of datasets share over 100,000 entities, only 5 triads of datasets contain over 100,000 common triples and 180 triads over 100,000 common literals.

Finally, the last column of Table 5.13 shows the number of datasets that do not have commonalities with other ones for each specific elements. It is worth noting that 87 datasets do not have common entities with other ones, while only 3 datasets do not share common literals. On the contrary, the number of disconnected datasets increases in the cases of common triples and common classes.

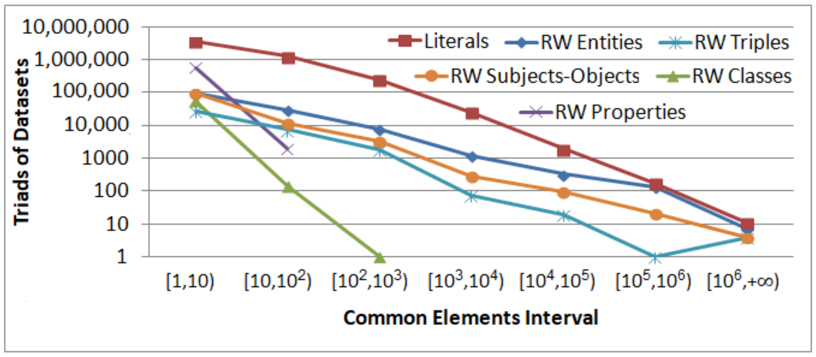


Figure 5.28: Number of connected triads of datasets per interval for each measurement type.

5.8.4.1 Connectivity of each domain

Figure 5.29 shows the unique real world entities and the maximum subset (e.g., subset with the most common entities 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 5 is the cross domain whereas from level 6 to 10 publications domain is the most connected one. In the remaining levels (from 11 to 20) the domain with the most common entities is the social networking domain. Moreover, regarding combinations with datasets from different domains, there are 13 datasets that shares thousands of entities and 20 datasets sharing over a hundred of entities. Most of these real world entities predominantly refer to geographical places and to popular persons. Generally, cross domain datasets take part in the most combinations with datasets from different domains. Finally, in [169], one can find more connectivity analytics for the publications domain, which is one of the most connected domains (as we mentioned above).

5.8.4.2 The most connected subsets of datasets

Table 5.14 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 real world entities). The most connected triad contains three cross domain datasets. Particularly, the subset comprising of the datasets *DBpedia*[4], *Freebase*[5] and *Wikidata*[6] shares over 3.5 million of real world entities, while the quad that contains also *Yago*[7] (apart from these datasets) contains approximately 3 million of common real world entities. Afterwards, triads of datasets

[4]`http://dbpedia.org/`
[5]`http://developers.google.com/freebase/`
[6]`http://wikidata.org`
[7]`http://yago-knowledge.org`

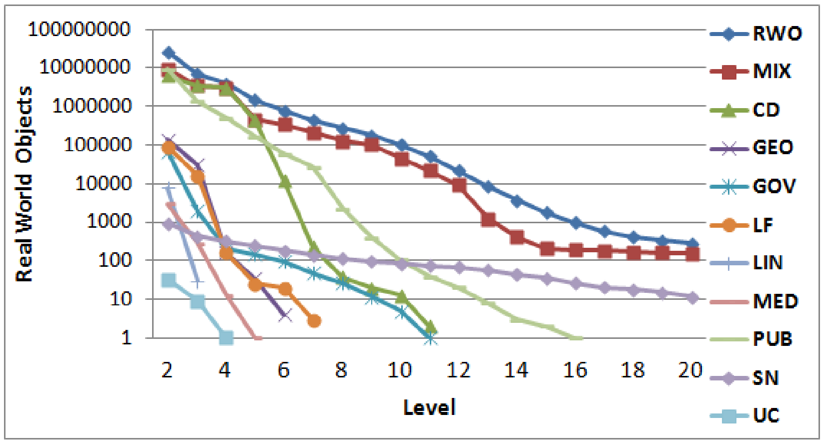


Figure 5.29: Unique(RWE) - Max Subset per Level

Table 5.14: Top-10 Subsets ≥ 3 with the most common RWE

Datasets of subset B	Common Entities
1: {DBpedia,Yago,Wikidata}	3,520,964
2: {DBpedia,Freebase,Wikidata}	3,474,682
3: {DBpedia,Yago,Freebase}	2,995,419
4: {Yago,Freebase,Wikidata}	2,944,415
5: {DBpedia,Yago,Freebase,Wikidata}	2,943,852
6: {DNB,id.loc.gov,VIAF}	1,333,836
7: {bl.uk,id.loc.gov,VIAF}	1,040,862
8: {BNF,id.loc.gov,VIAF}	1,007,312
9: {Wikidata,id.loc.gov,VIAF}	682,296
10: {DNB,VIAF,Wikidata}	642,658

belonging in publication domain follow, such as *VIAF*[8], *DNB*[9] ,*bl.uk*[10], *id.loc.gov*[11] and *BNF*[12]. Finally, *Wikidata* shares a lot of common real world entities with datasets of the publication domain.

In Table 5.15, we can see the top ten subsets containing ≥ 3 datasets that have the most common literals. In particular, the triad *YAGO*, *Wikidata* and *Freebase* share 5.6 million literals, while the quad with the four popular datasets from the cross-domain are again highly connected, i.e., they share 3.4 millions of literals. Moreover, the aforementioned datasets share a large number of literals with VIAF Dataset which belongs to publications domain (see positions 6-10 in Table 5.15).

In Table 5.16, we introduce measurements for the 10 subsets (containing 3 or more

[8]`http://viaf.org/`
[9]`http://www.dnb.de/`
[10]`http://www.bl.uk/`
[11]`http://id.loc.gov/`
[12]`http://www.bnf.fr`

Table 5.15: Top-10 Subsets ≥ 3 with the most common Literals

Datasets of subset B	Common Literals
1: {Yago,Freebase,Wikidata}	5,652,667
2: {DBpedia,Freebase,YAGO}	4,543,208
3: {DBpedia,YAGO,Wikidata}	4,397,132
4: {DBpedia,Freebase,Wikidata}	3,858,660
5: {DBpedia,Yago,Freebase,Wikidata}	3,423,764
6: {DBpedia,VIAF,YAGO}	1,720,303
7: {Wikidata,VIAF,YAGO}	1,528,431
8: {Freebase,VIAF,YAGO}	1,400,834
9: {DBpedia,Freebase,VIAF}	1,396,376
10: {DBpedia,Freebase,VIAF,YAGO}	1,229,087

Table 5.16: Top-10 subsets with ≥ 3 datasets having the most common real world triples.

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

datasets) having the most common triples. As we can see, the combinations of the four popular cross-domain datasets (*DBpedia* [4], *Yago* [26], *Freebase* [8] and *Wikidata* [25]) share a lot of triples, Concerning other subsets of datasets, some datasets from publications domain (i.e., *bl.uk* [21], *BNF* [1] and *VIAF* [22]), and one dataset from government domain (i.e., *JRC-names* [12]) share many triples with the aforementioned cross-domain ones. Regarding the connectivity for other sets of elements, the most connected triad of datasets, concerning classes, contains the following set of datasets (*DBpedia*, *Opencyc* [17], *ImageSnippets* [10]) with 188 common classes, while for the properties, the most connected triad includes (*VIVO Wustl* [24], *FAO* [7], *VIVO scripps* [23]) with 68 common properties. All the measurements for pairs, triads, and quads of subsets of datasets (in total 11,689,103 million subsets) for each different measurement type are accessible through *LODsyndesis* and *datahub.io* (`http://datahub.io/dataset/connectivity-of-lod-datasets`), in CSV and RDF format, by using VoID-WH ontology [163], which is an extension of VoID ontology [132] (in total we have created 99,221,766 million triples).

Table 5.17: Top-10 datasets with the most entities, triples and literals existing at least in 3 datasets.

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

5.8.4.3 The most popular datasets

Table 5.17 shows the top ten datasets that contain the most entities, literals and triples that can be found in three or more datasets. As we can see, *Wikidata* contains the most entities and triples that can be found in three or more datasets, while *Yago* is the dataset having the most literals that occur at least in three datasets. In all categories, the four datasets from the cross-domain are the most popular, while we can observe that *VIAF* dataset (from publications domain) occurs in the fifth position in all cases. Concerning the remaining positions, most datasets belong to publications domain (e.g., *id.loc.gov* [13], *DNB* [5], *Radatana* [18], and others). Moreover, there exists also a dataset from the geographical domain, i.e., *GeoNames* [9], which contains a lot of entities and literals that occur in three or more datasets, and a dataset from media domain, i.e., *LMDB* [14], which contains several triples that occur in more than two datasets.

5.8.5 Indicative Union and Complement Measurements

Here, we show indicative measurements, by exploiting the union and complement metrics. For each of the measurements below, the results retrieved approximately in 1 second by using *LB+PRGR*.

5.8.5.1 Coverage Measurements.

Tables 5.18 and 5.19 show examples for selecting the most relevant datasets for a specific task. Regarding Table 5.18, suppose that we desire to find K datasets having the most triples for the "Seafood red list species", i.e., a set of 55 fishes from unsustainable fisheries, for 5 different lattice levels. In total, there are 14,660 triples for these fishes in 16

Table 5.18:
Best subset of datasets of each level L having the most triples for Seafood red list species

L	$covBest(K, F)$	$\|cov(B, F)\|$	$covPer$	$covGain$
1	Fishbase	3,785	25.8%	-
2	Fishbase, Freebase	7,036	47.9%	85.0%
3	Fishbase, Freebase, DBpedia	9,745	66.4%	38.5%
4	Fishbase, Freebase, DBpedia, WoRMS	11,032	75.2%	13.2%
5	Fishbase, Freebase, DBpedia, WoRMS, Wikidata	11,507	78.4%	4.3%

Table 5.19:
Top-5 triads of datasets that cover the most triples for EDGE species

Triad of Datasets	$\|cov(B, F)\|$	$covPer$
DBpedia, Freebase, Wikidata	12,139	62.3%
DBpedia, Freebase, Taxonconcept	11,771	60.4%
DBpedia, Freebase,Geospecies	11,750	60.3%
Freebase, Taxonconcept, Wikidata	11,531	59.1%
Freebase, Geospecies, Wikidata	11,507	59.0%

Table 5.20:
Top-5 dataset pairs providing complementary triples for the entities of DBpedia

Pair of Datasets	$\|enrich(B, F, D_m)\|$	$enrichPer$
Freebase, YAGO	179,582,545	115.0%
Freebase, Wikidata	176,053,104	112.8%
Wikidata, YAGO	153,894,606	98.6%
Freebase, GeoNames	126,031,309	80.7%
Freebase, VIAF	112,948,225	72.3%

Table 5.21:
$|uniq|$ of Wikidata (WD) & YAGO (YG), vs DBpedia (DB) and Freebase (FR)

$\|uniq\|$ **of**	**Versus**	$uniqPer$
Wikidata Triples	DB,FR,YG	92.5%
Wikidata Entities	DB,FR,YG	77.5%
Wikidata Literals	DB,FR,YG	68.3%
YAGO Triples	DB,FR,WD	93.6%
YAGO Entities	DB,FR,WD	73.3%
YAGO Literals	DB,FR,WD	68.5%

(out of 400) datasets. Table 5.18 shows for each level L, the subset of datasets B with the $arg_B \max\ |cov(B, F)|$, the $covPer(B, F)$, and the $covGain(B', F, B)$ as we go from a subset B to a superset B'. The best single dataset is *FishBase*, while as we add more datasets, the *covGain* decreases, e.g., if we add *Wikidata* to the quad of datasets {Fishbase,Freebase,DBpedia,WoRMS}, the *covGain* is only 4.3%. Concerning Table 5.19, our target is to find the top-5 triads of datasets whose union contains the most triples for all the 115 entities belonging to EDGE ("Evolutionarily Distinct and Globally Endangered") species, and their corresponding $covPer(B, F)$. For these species, there exists 19,485 unique triples in 13 datasets (out of 400). The triad having the $arg_B \max\ |cov(B, F)|$ is {Wikidata, Freebase, DBpedia}, i.e., their union contains 12,139 triples for EDGE species (i.e, 62.3% of all the available triples).

5.8.5.2 Information Enrichment Measurements.

Table 5.20 shows the top five pairs of datasets that contain the maximum number of complementary triples for the entities of DBpedia, and their corresponding $|enrich(B, F, D_m)|$ and *enrichPer*. In particular, the pair {Freebase, YAGO} enriches the content for DBpedia entities with over 179 millions of new triples, while all the top five pairs of datasets offer over 112 millions of new triples, i.e., triples that are not included in DBpedia.

5.8.5.3 Uniqueness Measurements.

Table 5.21 shows the percentage of the unique entities, triples and literals of Wikidata (WK) and YAGO (YG), comparing to the other popular Knowledge Bases (KBs), i.e., i) Wikidata versus Freebase (FR), DBpedia (DB) and YAGO, and (ii) YAGO versus the three remaining ones. We can see that both KBs have a high percentage (over 90%) of unique triples comparing to the other KBs. Concerning entities, the percentage is 77.5% for Wikidata and 73.3% for YAGO, meaning that 22.5% of Wikidata entities and 26.7% of YAGO entities, occur at least in one of the three other popular KBs. As regards literals, we can see that the percentages are decreased, however, both KBs contain a high percentage of unique literals (i.e., over 68%).

5.8.6 Conclusions about the Connectivity at LOD Scale

The measurements revealed the sparsity of LOD Cloud. In general, we observed a power-law distribution, i.e., a large percentage of elements (entities, classes, etc.) occur in one dataset, while most connected datasets contain a small number of common elements, which means that a lot of publishers do not connect their entities with other datasets. Most subsets of datasets share literals, while only a few pairs and triads of datasets share triples. Moreover, most datasets share some properties from popular ontologies (such as rdf, rdfs, etc.): however, it seems that there is a lack of connections for schema elements. Consequently, it is hard to find common triples among the datasets, even between datasets sharing a lot of common entities and common literals. Indeed, by ignoring the property of each triple, we identified that there exists 3,465 pairs of datasets having common subject-object pairs, but not common triples. Concerning the most connected datasets, they are the four datasets belonging to cross-domain (i.e., *DBpedia*, *Yago*, *Freebase* and *Wikidata*), while there are also combinations containing datasets from cross-domain and publication domain that are highly connected. Moreover, the most popular datasets (containing elements that can be found in three or more datasets) are predominantly the four popular cross-domain datasets, while in this list one can find also datasets from publications and geographical domains.

5.9 Epilogue

We proposed content-based intersection, union and complement metrics among any subset of RDF datasets, that are applicable for several real world tasks. For the computation of intersection union and complement metrics, it is a requirement to solve four maximization problems, which are prohibitively expensive by using a baseline model or an implementation of SPARQL query language. For solving such maximization problems and for making feasible their computation at large scale, we proposed lattice-based incremental algorithms that exploit a set of pre-constructed semantics-aware indexes. In particular, we proposed two different traversals for computing the cardinality of intersection (a bottom-up depth first search and a top-down breadth first search), whereas for the rest metrics, we proposed bottom-up depth-first search algorithms that rely on set-theory properties and pruning and regrouping methods. Furthermore, we introduced methods for computing the metrics in parallel.

Concerning the evaluation results, for all the metrics, the lattice-based incremental approaches were extremely faster than a SPARQL implementation, which needed on average even 1 minute for computing the metrics for one pair of datasets, while lattice-based approaches can compute the metrics for million subsets of datasets in a few seconds. Moreover, they were very fast comparing to a baseline model which does not exploit set theory properties (even more than 5000× faster).

As regards intersection metrics, we were able to compute the cardinality of intersection for millions of subsets of datasets for entities, triples, and literals in a few seconds (4, 0.9 and 3.6 seconds, respectively), whereas we were able to perform the measurements even for billions of combinations of datasets in less than 10 minutes. Concerning the different methods, we achieved even a $21\times$ speedup by using a bottom-up approach instead of a top-down and a $5.61\times$ speedup by combining the bottom-up approach with a regrouping method. Regarding the rest three metrics, by combining the bottom-up incremental approach with pruning and regrouping methods, we observed up to 97× speedup, and we managed to compute the cardinality of the union for millions of subsets of datasets, for entities, triples, and literals in 1.3, 1.4 and 3.2 seconds, respectively. Moreover, we computed the cardinality of the absolute and relative complement of entities for a single dataset, with respect to millions of subsets of datasets, in 1.1 and 0.8 seconds. Furthermore, by using the parallel version of the bottom-up algorithm for intersection, and a cluster of 64 machines we were able to compute the lattice of billions of nodes in less than 1 minute.

Finally, we exploited the semantics-aware indexes and the content-based metrics for making various measurements over the entire LOD. A few indicative follow. Regarding `owl:sameAs` relationships, the transitive and symmetric closure of the `owl:sameAs` relationships of all datasets yielded more than 73 million new `owl:sameAs` relationships, and this increases the connectivity of the datasets: over 30% of the 9,075 connected pairs of datasets that share entities (URIs) are due to these new relationships. The measurements also reveal the "sparsity" of the current LOD cloud and make evident the need for better connectivity. Generally, only a small percentage of entities, triples, and schema elements exist in two or more datasets, indicatively, only 1.9% of real world entities and 0.33% of triples are part of three or more datasets. On the contrary, a high percentage of pairs (i.e., 78%) and triads (i.e., 46.44%) of datasets share literals. Concerning the most popular datasets, i.e., they are highly connected with other ones, they belong predominantly to cross-domain (e.g., *DBpedia*) and publications domain (e.g., *VIAF*).

Chapter 6
The `LODsyndesis` Suite of Services

In this chapter, we introduce methods and services that exploit the semantics-aware indexes and the content-based measurements, for offering global scale services for any entity. Moreover, we use the indexes for offering *Data Enrichment*, i.e., for creating features and word embeddings for Machine-Learning tasks. Indeed, we provide services for making it feasible to answer all the queries of Table 3.1. Furthermore, we describe in more details two data enrichment tools, called $\texttt{LODsyndesis}_{\mathcal{ML}}$ and `LODVec`, that can be exploited for Machine-Learning based tasks, and finally an ongoing research prototype, called `LODQA`, for *Question Answering* over hundreds of linked datasets. Concerning the contributions of this chapter:

- we provide an overview of the `LODsyndesis` services for each of the tasks A-E (introduced in Chapter 1.2), and we describe how one can use these services (e.g., through a REST API),
- we describe $\texttt{LODsyndesis}_{\mathcal{ML}}$, which is a research prototype that exploits `LODsyndesis` services for creating features for machine learning tasks,
- we introduce `LODVec`, which is a tool that can create URI embeddings by exploiting all the 400 datasets that are indexes through `LODsyndesis`,
- we describe in brief `LODQA` which is an ongoing work that exploits several `LODsyndesis` services for offering *Question Answering* over hundreds of linked datasets.

The rest of this chapter is organized as follows. In §6.1 we introduce all the services of `LODsyndesis` for all the tasks A-E. In §6.2 and §6.3, we describe the tools $\texttt{LODsyndesis}_{\mathcal{ML}}$ and `LODVec`, which can be exploited for Machine-Learning based tasks, whereas §6.4 introduces the Question Answering System `LODQA`. Finally, §6.5 concludes this chapter.

Publications related to this chapter. The work presented in this chapter has been published predominantly in [166, 169, 172], and secondarily in [73, 167, 170, 189].

6.1 LODsyndesis Services for Tasks A-E

Here, we introduce all the services for the tasks A-E (see Chapter 3) that are offered by the webpage of LODsyndesis (`http://www.ics.forth.gr/isl/LODsyndesis`) and from the REST [206] API of LODsyndesis. First, in §6.1.1, we show how to find the URI of one or more keywords (e.g., the URI of Aristotle). Afterwards, in §6.1.2 we introduce services for *Object Coreference and All Facts for an entity* (i.e., task A), whereas in §6.1.3, we show services related to task B (i.e., *Connectivity Analytics and Visualization*). Moreover, in §6.1.4, we introduce several advanced *Dataset Search, Discovery and Selection* services (i.e., task C), which are based on the intersection, union and complement metrics presented in Chapter 5. In §6.1.5, we show how to exploit LODsyndesis, for offering *Data Enrichment* services (i.e., task D), by focusing on tools that can be exploited for Machine-Learning based datasets, while in §6.1.6 we introduce services that can be used for improving *Data Quality* (i.e., task E).

6.1.1 How to find the URI of an Entity

For most of the services offered by LODsyndesis the input is a URI, e.g., `http://dbpedia.org/resource/Aristotle`. For this reason, we offer a keyword to entity service, which can be used in order to find the URI for one or more keywords (e.g., Aristotle).

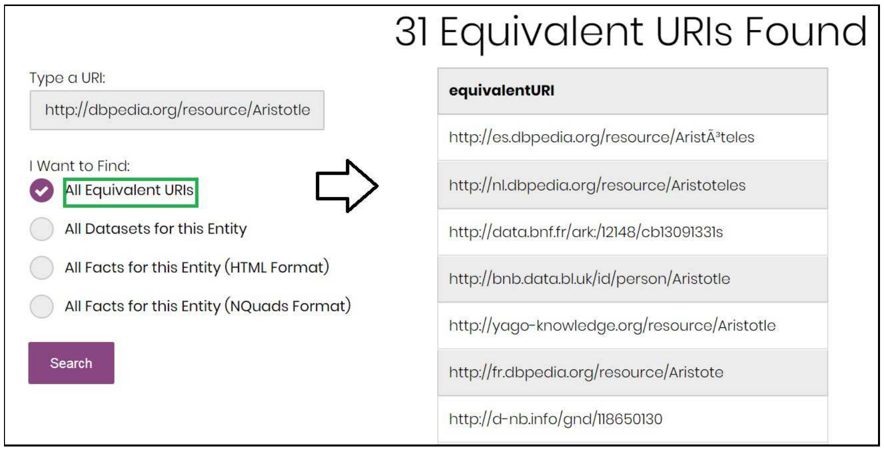
How to use it: For the services that are offered through the HTML page, we offer an auto-complete mechanism, i.e., the users can type a keyword and the webpage automatically shows to the users a list of URIs, containing that keyword. For instance, by typing "Aristotle", it will automatically show to the users the URI `http://dbpedia.org/resource/Aristotle`. Moreover, one can use our REST API (see the first service in Table 6.1) to find the corresponding URIs for a specific keyword, e.g., one can send the following GET request: `LODsyndesis/rest-api/keywordEntity?keyword=Aristotle`, to find the URI of Aristotle. The REST API provides the output in CSV [3], JSON [11] or XML [6] format.

6.1.2 Task A. Object Coreference and All Facts for an Entity Service

6.1.2.1 Object Coreference Service

We offer an object coreference service for answering queries $Q_{objectCor}$ and Q_{prov}, i.e., for retrieving for a given URI u all its equivalent URIs or/and all the datasets where that entity occurs. We offer this service for 412 million URIs, for both instance and schema elements (properties and classes).

Which indexes are used for offering this service. For providing such functionality, we need to exploit the equivalence catalogs, i.e., `EntEqCat`,`PropEqCat` and `ClEqCat`, for retrieving the equivalent URIs for a given URI, and the *Entity-Index, Property-Index* and *Class Index* for retrieving the provenance of each real world entity, property and class, respectively.



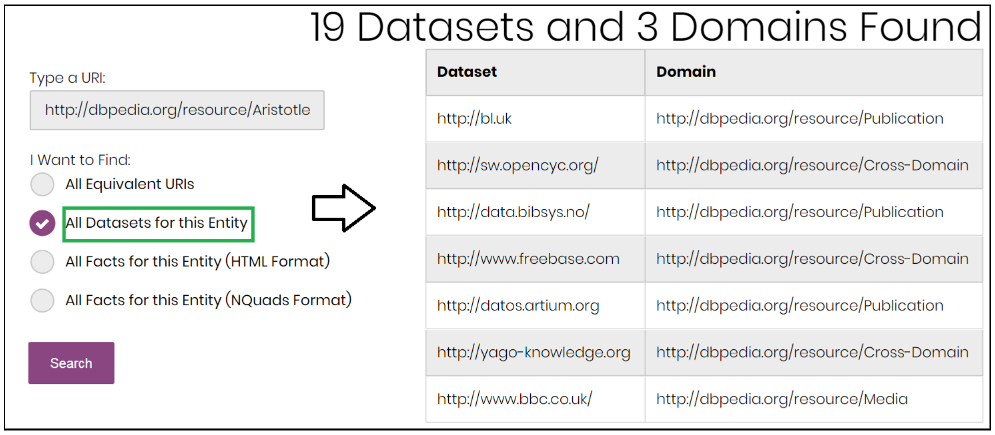
31 Equivalent URIs Found

Type a URI:
http://dbpedia.org/resource/Aristotle

I Want to Find:
[✓ All Equivalent URIs]
All Datasets for this Entity
All Facts for this Entity (HTML Format)
All Facts for this Entity (NQuads Format)

equivalentURI
http://es.dbpedia.org/resource/AristÃ³teles
http://nl.dbpedia.org/resource/Aristoteles
http://data.bnf.fr/ark:/12148/cb13091331s
http://bnb.data.bl.uk/id/person/Aristotle
http://yago-knowledge.org/resource/Aristotle
http://fr.dbpedia.org/resource/Aristote
http://d-nb.info/gnd/118650130

Figure 6.1: Object Coreference Service. Find all the equivalent URIs to “Aristotle”



19 Datasets and 3 Domains Found

Type a URI:
http://dbpedia.org/resource/Aristotle

I Want to Find:
All Equivalent URIs
[✓ All Datasets for this Entity]
All Facts for this Entity (HTML Format)
All Facts for this Entity (NQuads Format)

Dataset	Domain
http://bl.uk	http://dbpedia.org/resource/Publication
http://sw.opencyc.org/	http://dbpedia.org/resource/Cross-Domain
http://data.bibsys.no/	http://dbpedia.org/resource/Publication
http://www.freebase.com	http://dbpedia.org/resource/Cross-Domain
http://datos.artium.org	http://dbpedia.org/resource/Publication
http://yago-knowledge.org	http://dbpedia.org/resource/Cross-Domain
http://www.bbc.co.uk/	http://dbpedia.org/resource/Media

Figure 6.2: Provenance Service. Find all the datasets where the entity “Aristotle” occurs

How to use it: We offer an HTML page[1] where a user can type a URI and select whether they desire to find equivalent URIs or/and datasets where that entity occurs. In Figure 6.1, one can see an example for finding all the equivalent URIs of “Aristotle” (i.e., we have found 31 equivalent URIs) and in Figure 6.2 we show an example for retrieving all the datasets where that entity occurs (i.e., 19 datasets from 3 different domains provide information for “Aristotle”). Moreover, one can use our REST API (see the second service in Table 6.1), to exploit these services programmatically. For instance, all the equivalent URIs for Aristotle can be found by sending to `LODsyndesis` the following GET request: `LODsyndesis/rest-api/objectCoreference?uri=http://dbpedia.org/resource/Aristotle`, while for finding all the datasets containing information about Aristotle, one should use the following request `LODsyndesis/rest-api/objectCoreference?uri=http://dbpedia.org/resource/`

[1]`https://demos.isl.ics.forth.gr/lodsyndesis/Config?type=objectCoreference`

Aristotle&provenance=true. The results are offered in N-Triples [20], JSON or XML format. This service needs on average less than one second for retrieving the equivalent URIs and the provenance of an entity (e.g., 0.1 s to retrieve in JSON format the equivalent URIs of Aristotle).

6.1.2.2 Service for Finding all the Facts for an Entity

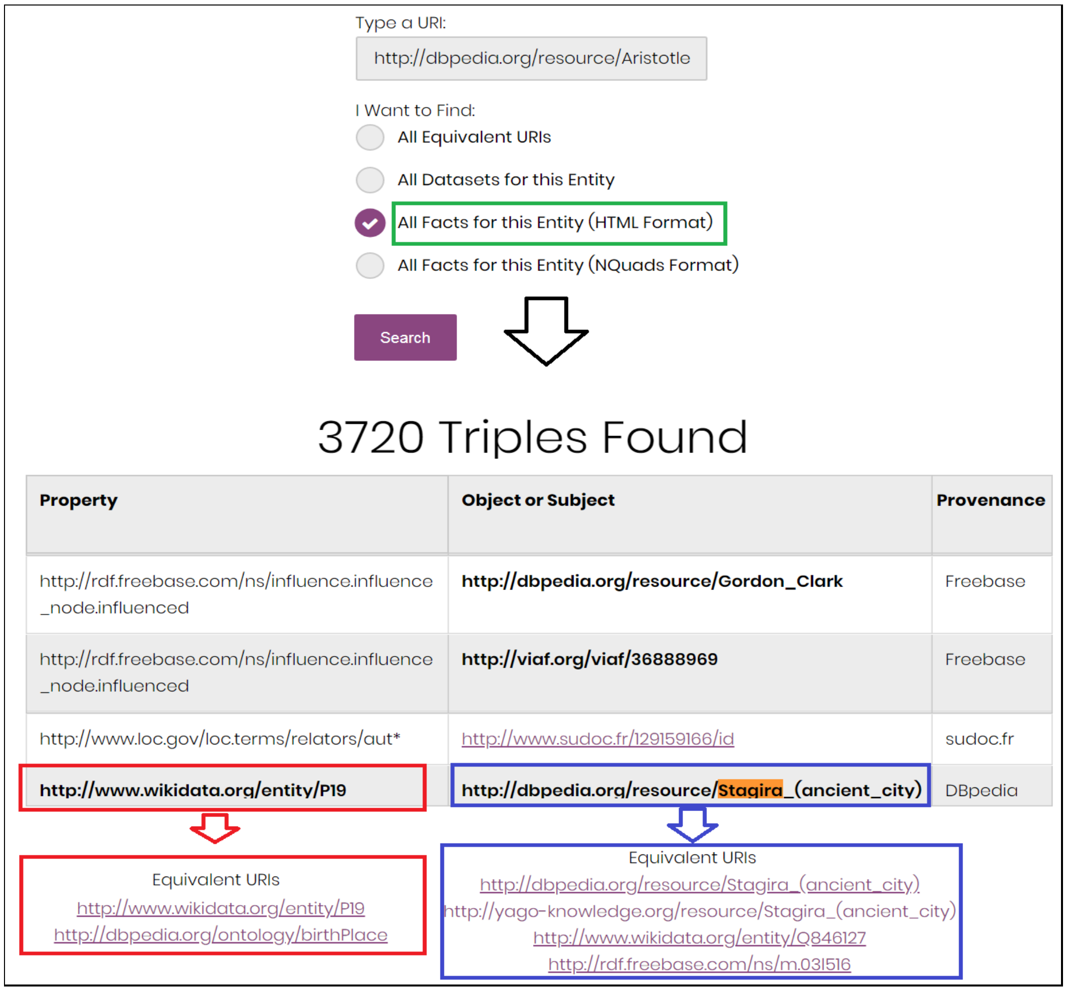
We offer a service for retrieving all the triples for any entity for 412 million URIs and two billion triples, i.e., for answering the query $Q_{allFacts}$ for all these URIs. By using such a service, one can browse or export all the available triples for an entity, and the provenance of each triple. These data could be used for various purposes, e.g., creating a mediator or a semantic warehouse for a set of entities, finding complementary information for one or more entities, comparing the values for an entity from several datasets and others.

Which indexes are used for offering this service. We exploit the *Entity-Triples* index, where we have already stored in the same place all the triples (and their provenance) for a specific real world entity. Moreover, we use the equivalence catalogs, i.e., EntEqCat, PropEqCat and ClEqCat, as it is described below.

How to use it: In the same HTML page as in the case of *Object Coreference*, by typing a URI, one can select to see (or to export) all the available triples for a specific entity. For instance, in Figure 6.3, we show an example for retrieving all the triples and their provenance for the URI http://dbpedia.org/resource/Aristotle, i.e., we retrieved 3,720 triples. In our indexes, we have replaced each URI with a unique identifier. For not returning to the users such identifiers (since they are not human-readable), we use the equivalence catalogs, i.e., EntEqCat,PropEqCat and ClEqCat, and we replace each such identifier with a single URI (which of course corresponds to that identifier). Moreover, by clicking to that URI, one can also see all its equivalent URIs. For example, in Figure 6.3 we replaced the identifier of the entity "Stagira_(Ancient_City)" with its' URI in DBpedia, and by clicking that URI, one can see all its' equivalent URIs (i.e., four unique URIs for that entity). Moreover, through our REST API (see the third service in Table 6.1), one can retrieve all the triples for this entity by sending the following GET request LODsyndesis/rest-api/allFacts?uri=http://dbpedia.org/resource/Aristotle. The results of this service can be downloaded in N-Quads [19], JSON or XML format. This service needs on average less than 10 seconds to retrieve all the facts for an entity (e.g., 3.5 s for collecting all the triples for Aristotle in JSON format).

6.1.2.3 Global Namespace Service

In many cases, one would like to find fast all the datasets that contain a specific namespace (or prefix), i.e., remind that a namespace is the first part of the URI, and it usually indicates the provenance of a URI, e.g., for the URI http://dbpedia.org/resource/Aristotle, the


Type a URI:

http://dbpedia.org/resource/Aristotle

I Want to Find:

- All Equivalent URIs
- All Datasets for this Entity
- [✓ All Facts for this Entity (HTML Format)]
- All Facts for this Entity (NQuads Format)

3720 Triples Found

Property	Object or Subject	Provenance
http://rdf.freebase.com/ns/influence.influence_node.influenced	**http://dbpedia.org/resource/Gordon_Clark**	Freebase
http://rdf.freebase.com/ns/influence.influence_node.influenced	**http://viaf.org/viaf/36888969**	Freebase
http://www.loc.gov/loc.terms/relators/aut*	http://www.sudoc.fr/129159166/id	sudoc.fr
http://www.wikidata.org/entity/P19	**http://dbpedia.org/resource/Stagira_(ancient_city)**	DBpedia

Equivalent URIs
http://www.wikidata.org/entity/P19
http://dbpedia.org/ontology/birthPlace

Equivalent URIs
http://dbpedia.org/resource/Stagira_(ancient_city)
http://yago-knowledge.org/resource/Stagira_(ancient_city)
http://www.wikidata.org/entity/Q846127
http://rdf.freebase.com/ns/m.031516

Figure 6.3: All Facts Service. Find all the triples of the real world entity “Aristotle”

namespace is `http://dbpedia.org`. Moreover, one can also find the datasets which contain terms from a specific ontology (e.g., `http://www.cidoc-crm.org`). We offer such a service for approximately 1 million namespaces.

Which index is used for offering this service. For this service, we use the `PrefixIndex`, which stores all the datasets where a namespace occurs.

How to use it: We offer an HTML page, where a user can type a namespace, and it returns back all the datasets that contain it, and the number of distinct URIs, where the given prefix occurs. Alternatively, through our REST API (see the last service in Table 6.1) one can send such a GET request to `LODsyndesis` `LODsyndesis/rest-api/namespaceLookup?namespace=http://www.cidoc-crm.org`. The output can be an HTML page, or N-Triples, XML and JSON (through the REST API). It needs less than 1 s for deriving all the datasets for a namespace (for the previous example, 0.1 s was needed).

6.1.3 Services for Task B. Connectivity Analytics & Visualization

6.1.3.1 Connectivity Analytics.

We provide measurement concerning the number of common real world entities, triples, classes, properties and literals among any pair, triad, and quad of 400 LOD Cloud Datasets, which can be exploited for offering connectivity analytics (see §5.8), i.e., for answering queries such as $Q_{connectivity}$.

Which measurements are used for offering this service. We use the connectivity measurements described in Chapter 5, which were produced through the bottom-up lattice-based incremental approach for intersection.

How to download the results of these measurements. The results of the connectivity metrics have been published in CSV and in N-Triples format (through VoID-WH ontology [162]) in datahub (`http://old.datahub.io/dataset/connectivity-of-lod-datasets`). In total, we have created 99 million triples for publishing the connectivity measurements.

6.1.3.2 3D Visualizations of LOD Cloud Datasets

We exploit the connectivity metrics for offering 3D visualizations of hundreds of LOD Cloud datasets, as it is described in [189]. An example can be seen in Figure 6.4. In particular, in the left side we can see the classical visualization of the LOD cloud, where each dataset is as a circle (whose size indicates the size of the dataset in triples), and the commonalities between two datasets (in terms of common URIs) are made evident by an edge that connects the dataset's circles. On the contrary, through a 3D visualization which is shown in the right side of Figure 6.4, we adopt a quite familiar metaphor, specifically that of an urban area, where each dataset is visualized as a building. By using such a visualization, we can use all the dimensions of a building for representing different statistics about each dataset, e.g., the height of a building can indicate the size of a dataset in terms of triples. Concerning the commonalities, if two datasets share common entities, then a line segment is created, which can be represented either as a road or as a bridge, that connects the corresponding buildings (see these two different representations in Figure 6.5).

Which measurements are used for offering 3D visualizations. First, for constructing the buildings, we have used some major statistics of each dataset, such as the number of triples, URIs, and others. Concerning the commonalities, we mainly exploit the measurements between pairs of datasets. In particular, the width of these bridges/roads, indicates the strength of the connection that the correlated datasets have, and it is calculated by the division of the number of common entities between two datasets D_i and D_j with the number of common entities of the most connected pair (i.e., $cmnBest(2, RWE)$).

How to use this 3D Visualization. The 3D visualization is offered through the following link `https://www.ics.forth.gr/isl/3DLod/` (all the details are given in [189]).

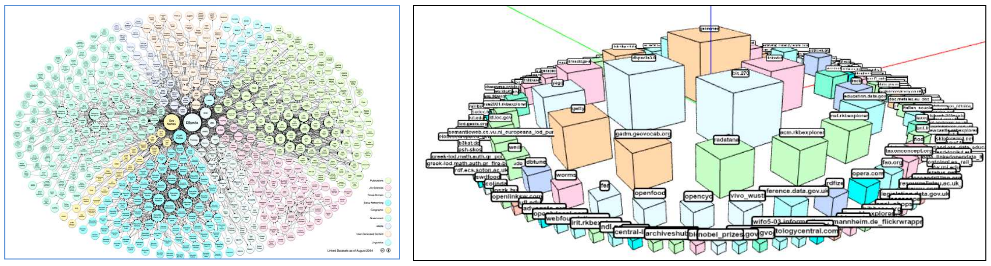


Figure 6.4: *Left*: The LOD cloud diagram. *Right*: One perspective of the introduced interactive LOD 3D model.

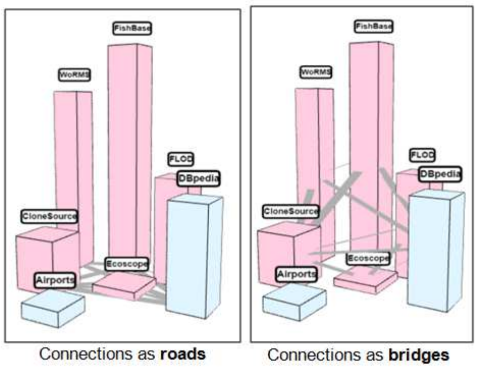


Figure 6.5: Two ways to visualize the connections

6.1.4 Services for Task C. Dataset Search, Discovery and Selection

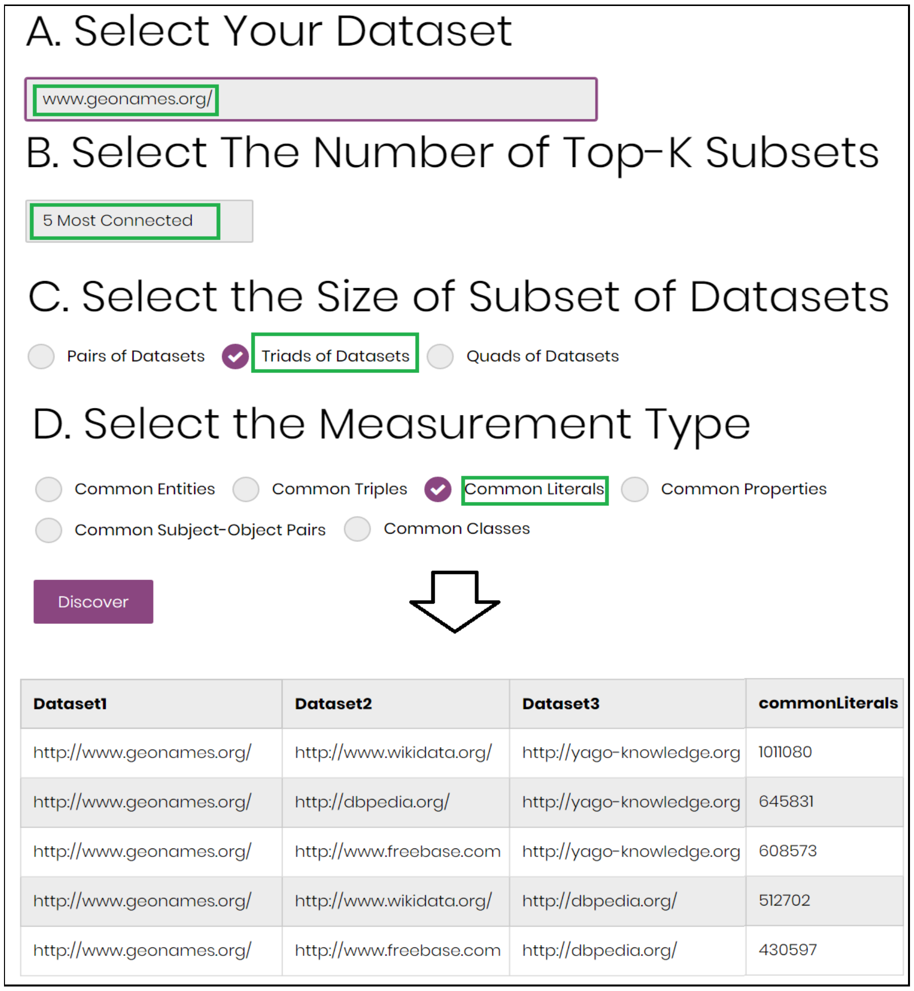
The proposed intersection, union and complement measurements can be directly used for dataset search discovery and selection services. We offer three different types of services, i.e., *Dataset-Based* services, where the focus is a specific dataset, i.e., *Entity-Based* Search, where the focus is one or more entities and *hybrid* services, where the focus can be both a dataset and one or more entities.

6.1.4.1 Dataset-Based Connectivity Service

We offer a connectivity service for finding the top-K connected pairs, triads or quads of datasets to a dataset D_i, for several measurement types, e.g., entities, literals, triples and others. Therefore, the objective of this service is to answer the query $Q_{datConnectivity}$.

Which methods are used for offering this service. We exploit the bottom-up lattice-based incremental for intersection, which was described in Chapter 5.

How to use this service: We offer an HTML page, where a user can select a dataset D_i from a list of 400 datasets, for finding the most connected datasets to D_i for a specific mea-



A. Select Your Dataset

www.geonames.org/

B. Select The Number of Top-K Subsets

5 Most Connected

C. Select the Size of Subset of Datasets

Pairs of Datasets [✓ Triads of Datasets] Quads of Datasets

D. Select the Measurement Type

Common Entities, Common Triples, [✓ Common Literals], Common Properties, Common Subject-Object Pairs, Common Classes

Dataset1	Dataset2	Dataset3	commonLiterals
http://www.geonames.org/	http://www.wikidata.org/	http://yago-knowledge.org	1011080
http://www.geonames.org/	http://dbpedia.org/	http://yago-knowledge.org	645831
http://www.geonames.org/	http://www.freebase.com	http://yago-knowledge.org	608573
http://www.geonames.org/	http://www.wikidata.org/	http://dbpedia.org/	512702
http://www.geonames.org/	http://www.freebase.com	http://dbpedia.org/	430597

Figure 6.6: Dataset-Based Connectivity Service. Find the top-5 most connected datasets to GeoNames according to the number of common literals

surement type. For instance, in Figure 6.6, we selected to find the top-5 most connected triads of datasets containing *GeoNames* dataset according to the number of common literals. As we can see, the triad *GeoNames*, *YAGO* and *Wikidata* share over 1 million literals. Moreover, one can use the REST API (see the fifth service in Table 6.1) for retrieving the most connected datasets to a given one, programmatically. For instance, for discovering the five most connected triads of datasets (w.r.t. the number of common entities) that contain the dataset of British Museum, one should send the following GET request: `LODsyndesis/rest-api/datasetDiscovery?dataset=http://collection.britishmuseum.org/\&connections\_number=5\&subset\_size=triads\&measurement\_type=Entities`. One can see the output of this service in N-Triples, CSV, JSON and XML format. This service needs on average less than 5 s for returning the most connected datasets for a given one, e.g., for retrieving the most connected triads containing British Museum, the execution time was 1.5 s.

6.1.4.2 Entity-Based Connectivity and Coverage Services

For this kind of services, the user should always select a set of entities, i.e., the target is to answer the queries $Q_{coverage}$, and $Q_{connectivity}$ for a given set of entities E'. The user can provide either i) a list of URIs or ii) the URI of a specific class. In the second case, `LODsyndesis` retrieves automatically all the URIs of the given class. These services focus on the triples E', therefore $F = RWT_{E'}$. In particular, one can find the K single, pairs, triads, quads or quintets of datasets that a) maximize the number of triples for E' (i.e., coverage), or b) having the most common triples for E' (i.e.,commonalities).

Which methods are used for offering this service. We exploit the bottom-up lattice-based incremental approach for intersection, and the corresponding method for union (with pruning and regrouping), which were described in Chapter 5.

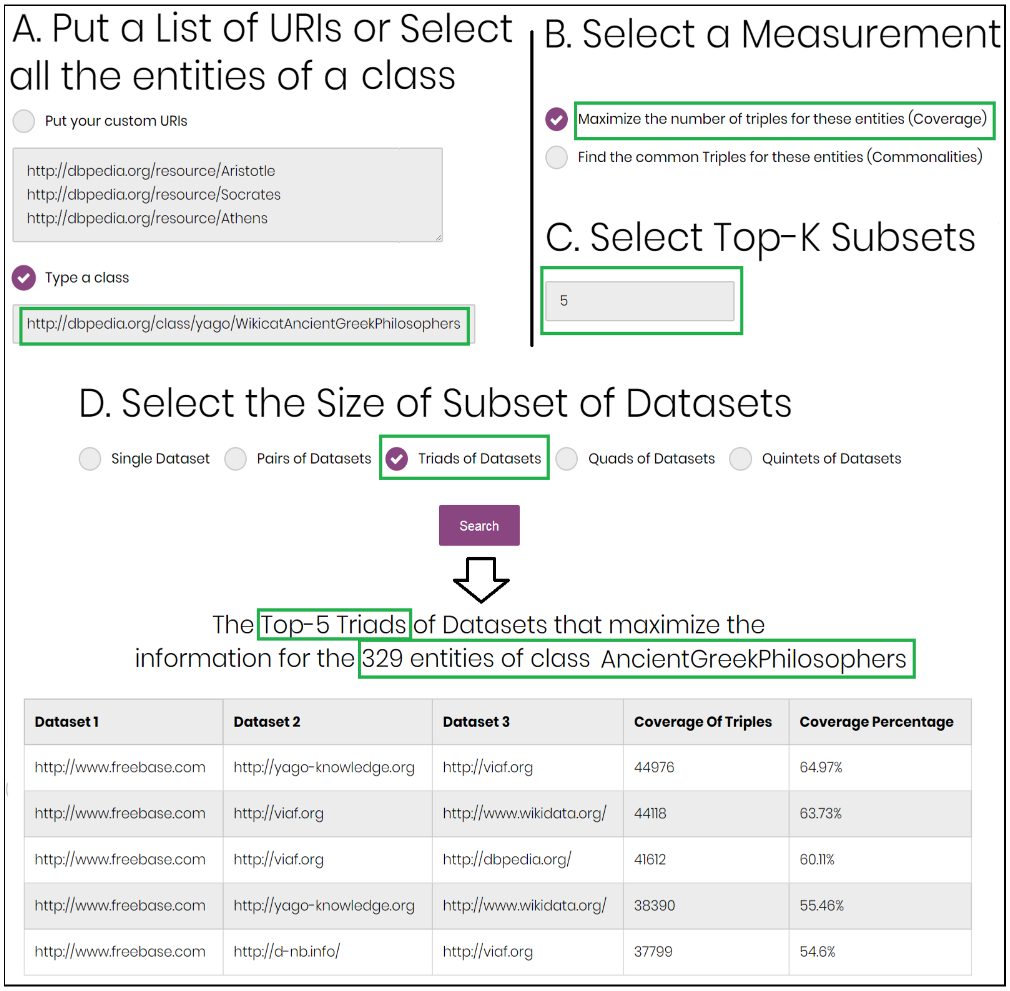
How to use this service: We offer an HTML page, where a user can type a URI of a class or a set of URIs, the desired measurement type, the size of subsets of datasets and a value for selecting the top-K results. The result is a table containing a ranked list of subsets of datasets. For instance, in Figure 6.7, we selected to find the top-10 triads of datasets that maximize the available triples for all the entities belonging to "Ancient Philosophers" class. As we can see, the union of *YAGO*, *VIAF* and *Freebase* contain the maximum number of triples for these 329 entities, i.e., they contain 44,976 triples for them, which corresponds to the 64.97% of all the triples for that entities.

6.1.4.3 Hybrid Services

For the hybrid services, the user should select both a dataset D_i and a set of entities E'. For the selected dataset D_i and set of entities E', we offer services based on information enrichment and uniqueness metrics, i.e., the is to answer the questions $Q_{enrichment}$ and $Q_{uniqueness}$. Concerning E', the user can select to give i) a list of URIs (manually), ii) a URI of a given RDF class, or iii) all the entities of dataset D_i. Concerning information enrichment, one can find the top-K single, pairs, triads and quads of datasets with the the most complementary triples to D_i for the set of entities E'. Regarding uniqueness, one can check the number of unique triples of dataset D_i versus single, pairs, triads and quads of datasets, given the entities E'.

Which methods are used for offering this service. We exploit the bottom-up lattice-based incremental methods (with pruning and regrouping) for complement metrics (information enrichment and uniqueness), which were described in Chapter 5.

How to use this service: We offer an HTML page, where a user can select a dataset D_i and a set of entities, the desired measurement type, the size of subsets of datasets and the



A. Put a List of URIs or Select all the entities of a class

Put your custom URIs

```
http://dbpedia.org/resource/Aristotle
http://dbpedia.org/resource/Socrates
http://dbpedia.org/resource/Athens
```

[✓ Type a class]

http://dbpedia.org/class/yago/WikicatAncientGreekPhilosophers

B. Select a Measurement

[✓ Maximize the number of triples for these entities (Coverage)]

Find the common Triples for these entities (Commonalities)

C. Select Top-K Subsets

5

D. Select the Size of Subset of Datasets

Single Dataset | Pairs of Datasets | [✓ Triads of Datasets] | Quads of Datasets | Quintets of Datasets

The Top-5 Triads of Datasets that maximize the information for the 329 entities of class AncientGreekPhilosophers

Dataset 1	Dataset 2	Dataset 3	Coverage Of Triples	Coverage Percentage
http://www.freebase.com	http://yago-knowledge.org	http://viaf.org	44976	64.97%
http://www.freebase.com	http://viaf.org	http://www.wikidata.org/	44118	63.73%
http://www.freebase.com	http://viaf.org	http://dbpedia.org/	41612	60.11%
http://www.freebase.com	http://yago-knowledge.org	http://www.wikidata.org/	38390	55.46%
http://www.freebase.com	http://d-nb.info/	http://viaf.org	37799	54.6%

Figure 6.7: Entity-Based Services. Find the top-5 triads of datasets that maximize the number of triples for Ancient Philosophers

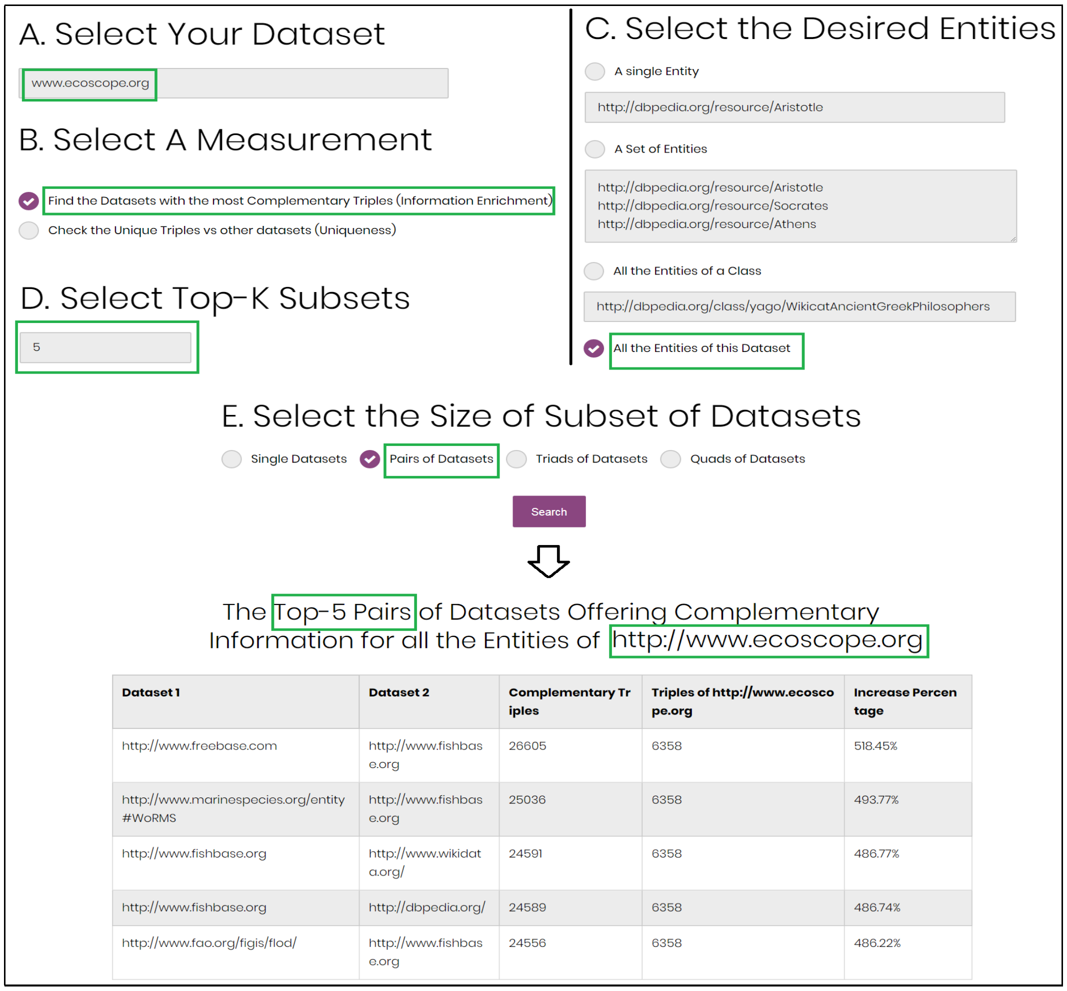
value *K*, and it returns to the user a ranked list of subsets of datasets. For instance, in Figure 6.8, we selected to find the top-5 pairs of datasets that offers complementary triples for all the entities of *Ecoscope* dataset. As we can see, the pair of datasets including *Freebase* and *Fishbase* offer the maximum number of complementary triples for the entities of *Ecoscope*, i.e., they offer over 26 thousand complementary triples for *Ecoscope* entities and the enrichment percentage was over 500%.

6.1.5 Services for Task D. Data Enrichment

First, one can use the information enrichment service described in §6.1.4.3, for finding the datasets containing complementary information for the entities of a given dataset D_i, i.e.,

Table 6.1: `LODsyndesis` REST API - GET Requests.

ID	Service URL	Description	Parameters	Response Types
1	LODsyndesis/rest-api/ keywordEntity	Finds all the URIs, containing one or more keywords.	**keyword**: Put one or more keywords.	text/csv, application/json, application/xml
2	LODsyndesis/rest-api/ objectCoreference	Finds all the equivalent entities of a given URI or the datasets where it occurs.	**uri**: Put any URI (Entity or Schema Element). **provenance**: It is an optional parameter. Put true for showing the datasets where the selected entity occurs.	application/n-triples, application/json, application/xml
3	LODsyndesis/rest-api/ allFacts	Finds all the facts (and their provenance) for a given URI (or an equivalent one).	**uri**: Put a URI that represents an entity.	application/n-quads, application/json, application/xml
4	LODsyndesis/rest-api/ factChecking	Checks a specific fact for a given entity.	**uri**:Put a URI that represents a single entity. **fact**: Put a fact, separate words by using space. **threshold:** Ratio of how many words of the fact should exist in the triple.	application/n-triples, application/json, application/xml
5	LODsyndesis/rest-api/ datasetDiscovery	Finds the most connected datasets to a given one for several measurement types.	**dataset**: Put a URI of an RDF Dataset. **connections_number**:It is optional. It can be any integer greater than zero, i.e., for showing the top-k connected datasets. **subset_size**: It can be any of the following: [pairs, triads, quads] (e.g., select pairs for finding the most connected pairs of datasets). **measurement_type**: It can be any of the following: [Entities, Literals, Properties, Triples, Classes, SubjectObject].	application/n-triples, application/json, application/xml
6	LODsyndesis/rest-api/ namespaceLookup	Finds all the datasets where a namespace occurs.	**namespace**: Put any namespace.	text/csv, application/json, application/xml



A. Select Your Dataset

www.ecoscope.org

B. Select A Measurement

[✓ Find the Datasets with the most Complementary Triples (Information Enrichment)]
Check the Unique Triples vs other datasets (Uniqueness)

D. Select Top-K Subsets

5

C. Select the Desired Entities

A single Entity
http://dbpedia.org/resource/Aristotle

A Set of Entities
http://dbpedia.org/resource/Aristotle
http://dbpedia.org/resource/Socrates
http://dbpedia.org/resource/Athens

All the Entities of a Class
http://dbpedia.org/class/yago/WikicatAncientGreekPhilosophers

[✓ All the Entities of this Dataset]

E. Select the Size of Subset of Datasets

Single Datasets [✓ Pairs of Datasets] Triads of Datasets Quads of Datasets

Search

The Top-5 Pairs of Datasets Offering Complementary Information for all the Entities of http://www.ecoscope.org

Dataset 1	Dataset 2	Complementary Triples	Triples of http://www.ecoscope.org	Increase Percentage
http://www.freebase.com	http://www.fishbase.org	26605	6358	518.45%
http://www.marinespecies.org/entity#WoRMS	http://www.fishbase.org	25036	6358	493.77%
http://www.fishbase.org	http://www.wikidata.org/	24591	6358	486.77%
http://www.fishbase.org	http://dbpedia.org/	24589	6358	486.74%
http://www.fao.org/figis/flod/	http://www.fishbase.org	24556	6358	486.22%

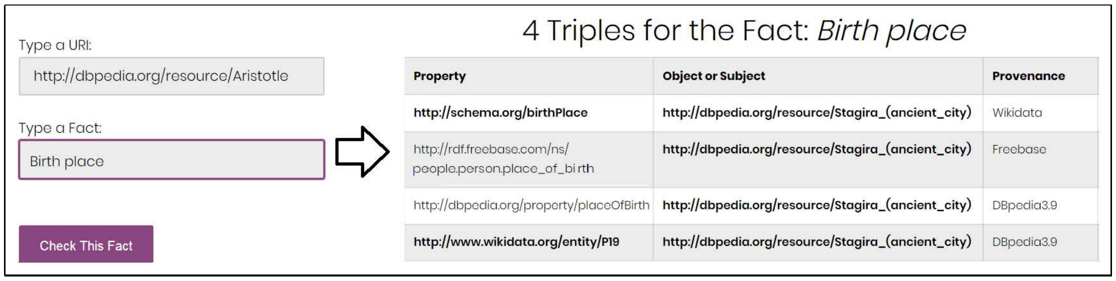
Figure 6.8: Hybrid Services. Find the top-5 pairs of datasets that offer the maximum number of complementary triples for the entities of Ecoscope Dataset

for answering $Q_{enrichment}$. Moreover, we propose two Data Enrichment tools that exploit the indexes of `LODsyndesis` i.e., `LODsyndesis`$_{\mathcal{ML}}$ [166] and `LODVec` [172], for improving the execution of several Machine-Learning based tasks. These tools are described in §6.2 and §6.3, respectively.

6.1.6 Services for Task E. Quality Assessment

6.1.6.1 Services for Data Veracity

By collecting all the available information for an entity, one can easily search whether a specific fact is verified from one or more datasets for a given entity (i.e., to verify the cor-


Type a URI:

http://dbpedia.org/resource/Aristotle

Type a Fact:

Birth place

4 Triples for the Fact: *Birth place*

Property	Object or Subject	Provenance
http://schema.org/birthPlace	http://dbpedia.org/resource/Stagira_(ancient_city)	Wikidata
http://rdf.freebase.com/ns/people.person.place_of_birth	http://dbpedia.org/resource/Stagira_(ancient_city)	Freebase
http://dbpedia.org/property/placeOfBirth	http://dbpedia.org/resource/Stagira_(ancient_city)	DBpedia3.9
http://www.wikidata.org/entity/P19	http://dbpedia.org/resource/Stagira_(ancient_city)	DBpedia3.9

Figure 6.9: Fact Checking Service. Compare the values for the birth place of "Aristotle"

rectness and veracity of information), e.g., "Had Aristotle lived in Athens?". One can also compare all the values for a specific fact (i.e., for answering queries such as $Q_{veracity}$), e.g., "I want to find all the values for the birth date of Aristotle, and their provenance". For the first type of questions, we can see which datasets verify that fact, while for the second type of questions, two or more datasets can provide conflicting answers, therefore, we can compare them for deciding which is the correct one. Moreover, one can submit comparative questions, e.g., "Which was the relationship between Socrates and Aristotle?". For comparing the values for all these questions, we offer a fact checking service, that contains over 2 billions of facts.

Which indexes are used for offering this service. We exploit the *Entity-Triples* index, where we have already stored in the same place all the values (and their provenance) for a specific real world fact.

How to exploit this service. We offer an HTML page where a user can type a URI and a set of words representing a fact, e.g., in Figure 6.9 we show an example for comparing the values for the birth place of "Aristotle". Moreover, one can use our REST API (see the fourth service in Table 6.1) to check for a fact programmatically. For instance, in order to find which is the birth date of Aristotle, one can send the following GET request: `LODsyndesis/rest-api/factChecking?uri=http://dbpedia.org/resource/Aristotle\&fact=birthdate`. The output of this service can be seen as tables in HTML page, while the REST API offers the output in N-Quads, JSON or XML format. This service needs on average less than 10 s to check a fact, e.g., to find the birth date of Aristotle we needed 3.5 s.

6.1.6.2 Services for Assessing Dataset Quality

For assessing the quality of a dataset, one can use the dataset-based connectivity services, described in §6.1.4.1, and the hybrid services, introduced in §6.1.4.3. By using such services, it is feasible to answer queries $Q_{uniqueness}$, $Q_{datConnectivity}$, $Q_{datQuality}$. In particular, one can assess the quality of a datasets in terms of connectivity, information enrichment and uniqueness, i.e., to evaluate the level of connectivity of a dataset, whether a dataset contains unique information comparing to other datasets, if the dataset offers complementary to other datasets, and so forth. Through all these measurements, we make it also feasible to answer query $Q_{datReuse}$, e.g., a dataset sharing a high number of common entities with other datasets, and offers complementary information to them is more valuable to be reused, comparing to a dataset that is either disconnected or contains redundant information.

6.2 LODsyndesisML. How Linked Data can aid Machine Learning-based tasks

Here, we show how the wealth of Linked Data and the ML machinery can be jointly exploited for improving the quality of automated methods for various time consuming and/or tedious tasks, which are important also in the area of digital libraries, like automatic semantic annotation or classification, completion of missing values, clustering, or computing recommendations. Specifically, we focus on exploiting Linked Data for discovering and creating features for a set of entities. We introduce a tool (research prototype) LODsyndesis$_{\mathcal{ML}}$ that we have designed and implemented, that (i) discovers datasets and URIs containing information for a set of entities by exploiting LODsyndesis [165], (ii) provides the user with a large number of possible features that can be created for these entities (including features for direct and indirect related entities of any path) and (iii) produces automatically an enriched dataset for the features selected by the user. For testing whether this enriched dataset can improve ML tasks, we report experimental results over two datasets (from [209]) for predicting the popularity of a set of movies and books. Figure 6.10 illustrates the running example, where we create features for classifying whether a book is *Popular* or *Non-Popular*, containing data discovered from *DBpedia* [142] and *British National Library* [196]. We evaluate this approach by performing a 5-fold cross validation for estimating the performance of different models for the produced datasets. The evaluation showed that the additional features did improve the accuracy of prediction.

The rest of this section is organized as follows: §6.2.1 discusses related approaches, while §6.2.2 introduces the placement of LODsyndesis$_{\mathcal{ML}}$ in the Data Integration Landscape. §6.2.3 states the problem and describes the functionality of the proposed tool (research prototype), §6.2.4 discusses the steps of the process, §6.2.5 reports the results of the evaluation and discusses the effectiveness of the proposed features.

6.2.1 Related Work.

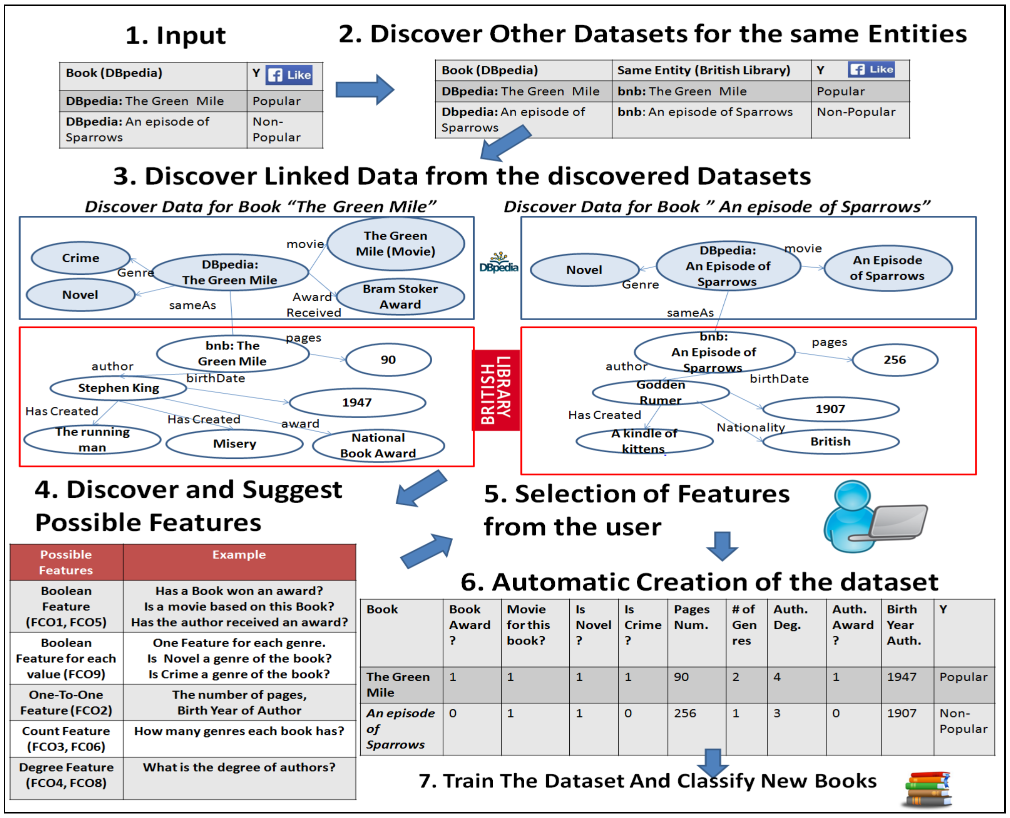
There are several proposals for using Linked Data for generating features. LiDDM [176] is a tool that retrieves data from Linked Data cloud by sending queries. For finding possible features the users can either construct their own queries or use an automatic SPARQL query builder that shows to the users all the possible predicates that can be used (from a specific SPARQL endpoint). It offers also operators for integrating and filtering data from two or more sources. The authors in [58] presented a modular framework for constructing semantic features from Linked Data, where the user specifies the SPARQL queries that should be used for generating the features. Another work that uses SPARQL queries is described in [175], where the user can submit queries which are combined with SPARQL aggregates (e.g., count). Comparing to our approach, the previous tools presuppose that the user is familiar with SPARQL, and they do not assist the user in discovering automatically datasets containing information for the same entities. The closest tool to our approach is FeGeLOD [193] which combines data from several datasets by traversing `owl:sameAs` paths and generates automatically six different categories of features. *RapidMiner Semantic Web Extension* tool [208] (which is the extension of FeGeLOD) supports the same features while it integrates the data that are derived from multiple sources. Instead we show the provenance of the data without integrating them, i.e., if a feature is provided by two or more sources, the user can decide which source to select for creating this feature. Moreover, we also discover datasets containing the same entities by exploiting `LODsyndesis` where the class of equivalence for each entity has already been pre-computed for 400 datasets, whereas the aforementioned tool finds relevant data by traversing links on-the-fly. Finally, we also provide other kinds of features, such as degree of an entity, boolean features for each value of a predicate, as well as features for "sub-entities", i.e., entities correlated with the entities that one wants to classify (e.g., actors of a movie).

6.2.2 Placement of $\texttt{LODsyndesis}_{\mathcal{ML}}$ in the Data Integration Landscape

$\texttt{LODsyndesis}_{\mathcal{ML}}$ can be considered as special data integration scenario that involves several basic services, e.g., dataset discovery, query answering, which however should be related to one particular dataset (the dataset to be enriched). With respect to the data integration landscape (see §2.3), $\texttt{LODsyndesis}_{\mathcal{ML}}$, we could say that the process that is followed includes several dimensions. In particular, it includes *Dataset Discovery* and *Dataset Selection*, i.e., for identifying and selecting the datasets that will be used, *Fetching and Transformation*, i.e., for offering a common data representation, *Instance Matching* since it exploits the cross-identity reasoning among several datasets, and *Publishing*, because it produces a new enriched dataset. Finally, its' *Basic Service to Deliver* is *Question Answering* for machine-learning based tasks.

6.2.3 Linked Data-based Feature Creation Operators

Let E be the set of entities for which we want to generate features. Below we will show how we can derive a set of features $(f_1, ..., f_k)$ where each f_i is a feature and $f_i(e)$ denotes the value of that feature for an entity $e \in E$. Each $f_i(e)$ is actually derived by the data that are related to e. Specifically we have identified the following nine (9) frequently occurring


1. Input
2. Discover Other Datasets for the same Entities
3. Discover Linked Data from the discovered Datasets
Discover Data for Book "The Green Mile"
Discover Data for Book " An episode of Sparrows"
Crime
Novel
Genre
DBpedia: The Green Mile
movie
The Green Mile (Movie)
Bram Stoker Award
Award Received
sameAs
DBpedia
Novel
Genre
DBpedia: An Episode of Sparrows
movie
An Episode of Sparrows
sameAs
bnb: The Green Mile
pages
90
author
Stephen King
birthDate
1947
Has Created
The running man
Has Created
Misery
award
National Book Award
BRITISH LIBRARY
bnb: An Episode of Sparrows
pages
256
author
Godden Rumer
birthDate
1907
Has Created
A kindle of kittens
Nationality
British
4. Discover and Suggest Possible Features
5. Selection of Features from the user
6. Automatic Creation of the dataset
7. Train The Dataset And Classify New Books

Book (DBpedia)	Y Like
DBpedia: The Green Mile	Popular
DBpedia: An episode of Sparrows	Non-Popular

Book (DBpedia)	Same Entity (British Library)	Y Like
DBpedia: The Green Mile	**bnb:** The Green Mile	Popular
Dbpedia: An episode of Sparrows	**bnb:** An episode of Sparrows	Non-Popular

Possible Features	Example
Boolean Feature (FCO1, FCO5)	Has a Book won an award? Is a movie based on this Book? Has the author received an award?
Boolean Feature for each value (FCO9)	One Feature for each genre. Is Novel a genre of the book? Is Crime a genre of the book?
One-To-One Feature (FCO2)	The number of pages, Birth Year of Author
Count Feature (FCO3, FC06)	How many genres each book has?
Degree Feature (FCO4, FCO8)	What is the degree of authors?

Book	Book Award ?	Movie for this book?	Is Novel ?	Is Crime ?	Pages Num.	# of Genres	Auth. Deg.	Auth. Award ?	Birth Year Auth.	Y
The Green Mile	1	1	1	1	90	2	4	1	1947	Popular
An episode of Sparrows	0	1	1	0	256	1	3	0	1907	Non-Popular

Figure 6.10: Running Example of LODsyndesis$_{\mathcal{ML}}$

Linked Data-based Feature Creation Operators, for short *FCO*s. In their definition, shown in Table 6.2, P denotes the set of properties, p, p_1 and p_2 are properties and $\mathcal{T}$ denotes the triples for the entities that are indexed by LODsyndesis.

In our running example of Figure 6.10, FCO1 can be used for representing whether a book has been nominated for winning an award or not. FCO2 suits to properties that are functional (one-to-one), e.g. person's birth country, number of pages of the book, and its value can be numerical or categorical. FCO3 counts the values of a property, e.g. the number of genres of a book. FCO4 measures the number of distinct triples that involve e, in our running example the degree of the author of "The Green Mile" book is 3, while the degree of the author of "An episode of Sparrows" is 2. FCO5-FCO9 correspond to features related to "sub-entities" or "related" entities to e. Specifically, FCO5 corresponds to one characteristic of a "sub(related)-entity" of e, e.g. whether at least one actor of a movie has won an award in the past or not. FCO6 counts the distinct values of one characteristic of the "sub-entities", e.g. the total number of movies where the actors of a movie have played. FCO7 finds the most frequently occurring characteristic of these entities, e.g. the country where most of the actors of a movie were born. FCO8 measures the average number of

Table 6.2: Feature Creation Operators

id	Operator defining f_i	Type	$f_i(e)$
1	p.exists	boolean	$f_i(e) = 1$ if (e,p,o) or $(o,p,e) \in \mathcal{T}$, otherwise $f_i(e) = 0$
2	p.value	num/categ	$f_i(e) = \{ v \mid (e,p,v) \in \mathcal{T} \}$
3	p.valuesCard	int	$f_i(e) = \lvert\{ v \mid (e,p,v) \in \mathcal{T} \}\rvert$
4	degree	double	$f_i(e) = \lvert\{(s,p,o) \in \mathcal{T} \mid s = e \text{ or } o = e\}\rvert$
5	p1.p2.exists	boolean	$f_i(e) = 1$ if $\exists$ $o2$ s.t. $\{(e,p1,o1),(o1,p2,o2)\} \subseteq \mathcal{T}$
6	p1.p2.count	int	$f_i(e) = \lvert\{ o2 \mid (e,p1,o1),(o1,p2,o2) \in \mathcal{T}\}\rvert$
7	p1.p2.value.maxFreq	num/categ	$f_i(e)$ = most frequent $o2$ in $\{ o2 \mid (e,p1,o1),(o1,p2,o2) \in \mathcal{T}\}$
8	average degree	double	$f_i(e) = \frac{\lvert triples(C)\rvert}{\lvert C\rvert}$ s.t. $C = \{ c \mid (e,p,c) \in \mathcal{T}\}$ and $triples(C) = \{(s,p,o) \in \mathcal{T} \mid s \in C \text{ or } o \in C\}$
9	p.values.AsFeatures	boolean	for each $v \in \{ v \mid (e,p,v) \in \mathcal{T}\}$ we get the feature $f_{iv}(e) = 1$ if (e,p,v) or $(v,p,e) \in \mathcal{T}$, otherwise $f_{iv}(e) = 0$

distinct triples for a set of "sub-entities", e.g., the average number of triples for the actors of a movie. The last one, FCO9, does not create one feature but a set of features, e.g. one boolean feature for each genre that a book can possibly belong to. In our running example, we take all genres of both books and for each genre (e.g., novel) we create a distinct boolean feature (both books belong to the genre Novel, but only "The Green Mile" book belongs to the genre Crime). Generally, the operators FCO1-FCO4 and FCO9 concern a single entity (e.g., a book, a person, a country, etc.) while operators FCO5-FCO8 a set of entities (e.g., all actors of a movie). Consequently, for the "sub-entities" that are connected through a functional property (one-to-one) with the entities that we want to classify, operators FCO1-FCO4 and FCO9 are used instead of operators FCO5-FCO8. The user can explore direct or indirect "sub-entities", e.g., authors of a book, countries of authors of a book and so forth, for any formulated path, while the list of operators can be easily extended by adding more operators.

Additional Functionality of `LODsyndesis`$_{\mathcal{ML}}$.

Here we introduce some useful (for the user) metadata and restrictions for feature selection .

"Completeness" of a Property for a given set of entities. We compute the percentage of instances for which a given property exists, e.g., the percentage of books for which we have information about the number of their pages. If E'_p is the set of entities being subject

Table 6.3: Restrictions of features with respect to the characteristics of a property

Feature Operators	**Can be Applied for**
Boolean (FCO1, FCO5)	All properties having $\frac{\lvert E'_p \rvert}{\lvert E \rvert} < 1$
Boolean for each Value (FCO9)	All properties $p \in P_{Many}$, $range(p) \neq Numeric$
One-to-one Relationship (FCO2)	All properties $p \in P_{1-1}$
Count (FCO3, FCO6)	All properties $p \in P_{Many}$
Degree (FCO4, FCO8)	All properties having $range(p) = \mathcal{U}$

or object of triples with predicate p, i.e. $E'_p = \{e \in E \mid (e\ p\ o) \text{ or } (o\ p\ e) \in \mathcal{T}\}$, then the percentage is given by $\frac{\lvert E'_p \rvert}{\lvert E \rvert}$.

Multiplicity & Range of a Property. Here we find the multiplicity of a specific property, i.e., whether it is a one-to-one or one-to-many relation. We define the set of one-to-one properties as $P_{1-1} = \{p \mid (e\ p\ o_i) \in \mathcal{T} \text{ and } \nexists\ (e\ p\ o_{ii}) \in \mathcal{T},\ o_i \neq o_{ii}, \forall\ e \in E\}$. The rest properties, i.e., one-to-many, are defined as $P_{Many} = P \setminus P_{1-1}$, while we denote as $range(p) \in \{String, Numeric, \mathcal{U}\}$ a property's range, i.e., whether it is a set of Strings, Numeric Values or URIs.

Restrictions derived from metadata. Table 6.3 shows the restrictions which are derived by taking into account the "completeness", the multiplicity and the range of a property. It is worth mentioning that the "completeness" of a property can also be exploited for discovering missing values for the entities. In addition, the users can define their own restrictions, e.g., they can exclude properties that belong to popular ontologies such as *rdf*, *rdfs*, *foaf* and *owl*.

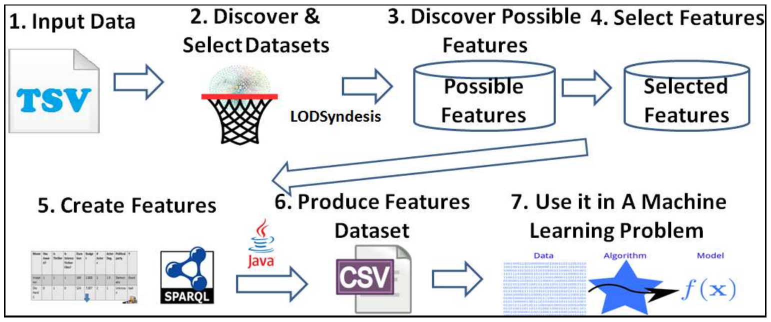


Figure 6.11: Process of $\text{LODsyndesis}_{\mathcal{ML}}$

6.2.4 The Steps of the Proposed Approach

Here we describe the tool (research prototype) $\text{LODsyndesis}_{\mathcal{ML}}$ that we have designed and implemented. It is worth noting that $\text{LODsyndesis}_{\mathcal{ML}}$ discovers and creates features by exploiting Linked Data for any domain. Even a user that is not familiar with Semantic Web technologies and SPARQL can use it for creating features for feeding a Machine Learning problem. The process is shown in Figure 6.11 and is described in brief below. First, it takes as input a file containing a set of URIs that refer to particular entities, i.e., movies, books and so forth. In case of knowing the entities but not their URIs, one can exploit an entity identification tool like DBpedia Spotlight [151] and XLink [92] for detecting automatically a URI for a specific entity. Then, it connects to `LODsyndesis` for discovering automatically datasets containing information for the same entities and shows to the user the available datasets. Afterwards, it discovers and shows to the user possible features that can characterize the entities (or related "sub-entities") of the dataset and the user selects which features to create. The next step is to create the features and to produce the output dataset to be used in any ML problem. Below, we describe in more detail the whole process, while additional information and a demo can be found in `http://www.ics.forth.gr/isl/LODsyndesis`.

1. Input: The input of `LODsyndesis`$_{\mathcal{ML}}$ is a file in tab separated value (tsv) format containing URIs describing entities and possibly their class, e.g., URIs for a book and if each book is Popular or Non-Popular.

2. Discover Data by using LODsyndesis: `LODsyndesis`$_{\mathcal{ML}}$ reads the tsv file and connects to `LODsyndesis` in order to discover (a) datasets containing information for the same entities and (b) the URIs for these entities for each dataset (the indexes of `LODsyndesis` have already pre-computed the closure of *owl:sameAs* relationships for 400 datasets). Then, the user selects the desired datasets. Concerning the running example of Figure 6.10, we observe that we found two different datasets containing information for the books of that example .

3. Discover Possible Features: `LODsyndesis`$_{\mathcal{ML}}$ sends SPARQL [200] queries for a sample of the aforementioned entities to the SPARQL endpoints of the selected datasets. Afterwards, a number of possible features and their provenance are discovered and returned to the user. Therefore, in this step we do not create any feature, we just discover possible features and we apply the restrictions described in §6.2.3. The result is a table where each row corresponds to a possible feature derived from a specific source while each column consists of a checkbox for a specific feature category. The order that the features appear in the rows is descending with respect to the "completeness" of each property. Particularly, when a property occurs for all the entities, it is placed first in the list, while those with the smallest number of occurrences are placed at the end of the list. Moreover, the user can view the metadata described in §6.2.3. Afterwards, the user can select the desired features (by taking into account their provenance) and can also explore features for (direct or indirect) "sub-entities" of any formulated path and create more features.

4. Feature Selection and 5. Feature Creation: The user selects the desired features and clicks on a button for initiating the dataset creation. Then, the tool sends SPARQL queries

for creating the features. For each feature operators category, it sends $|E|$ in number SPARQL queries (one query per entity e for each operator). It is worth noting that for values that are neither numeric nor boolean, it performs a mapping for converting them to numeric. Concerning missing values, we just put a unique constant value. However, for improving datasets' quality, several transformations could be applied after this step, like those proposed in [41] for removing erroneous and inconsistent data or filling missing values. Here, we do not focus on this task and the data used in the experiments have not been transformed or cleaned by using such techniques.
6. Production of Features' Dataset and 7. Exploitation of the Produced Dataset in a ML problem: The user is informed that the process is completed and that two csv files have been produced: one for the categorical and one for the continuous features. Then, the produced datasets can be given as an input for a ML problem (e.g., classification of books).

6.2.5 Evaluation

The datasets, which are used in our experiments (derived from [209]), contain the URIs of movies and books from *DBpedia* [142] and the corresponding classification value, i.e., *Popular* or *Non-Popular* according to the number of Facebook users' likes. We use 1,570 entities for Movies Dataset and 1,076 entities for Books Dataset. The initial datasets are loaded and then more data are discovered by using `LODsyndesis`$_{\mathcal{ML}}$ from the following sources: *British National Library* [196], *Wikidata* [251] and *DBpedia* [142]. In particular, we exploit `LODsyndesis`$_{\mathcal{ML}}$ for discovering, selecting and creating a number of different features for predicting the class of these entities. Afterwards, *MATLAB* [15] is used for performing a) a 5-fold cross validation for model selection and b) a comparison of a number of different models for measuring accuracy, which is defined as: $accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$ [255]. For each dataset, we repeat the 5-fold cross validation process 15 times for different sizes of the test set, i.e. 10%, 20%, & 30%. Each time a chi-square test of independence [263] is performed (for excluding variables that are independent of the class variable) for 4 different values of significance level (or threshold) a: 0.01, 0.05, 0.1, 1. For each value of threshold a we test 10 different models: (a) 2 *Naive Bayes* models (Empirical & Uniform), (b) 3 *Random Forest* models with 50 trees and different min leaf sizes: 1, 3 & 5, (c) 3 *K-Nearest Neighbours* models with K: 3, 5 &15, (d) a *linear SVM model* and (e) the *trivial* model. In each iteration the best model is obtained for the training set (by using cross validation). Finally, the accuracy of the best model is estimated on the test set.

Creation of Features. In Figure 6.12 we can observe how the number of possible features increases when a) more datasets are added and b) features of "sub-entities" are created, i.e., approximately the possible features are doubled when we explore a "sub-entity" (e.g. the authors of a book). Moreover, Figure 6.13 shows the time for generating a feature

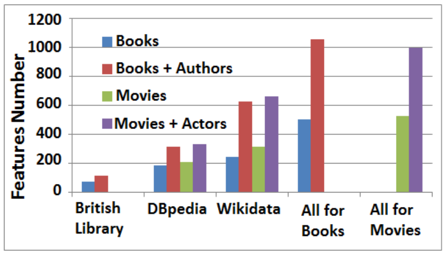


Figure 6.12:
Features Number Per Dataset For Books & Movies

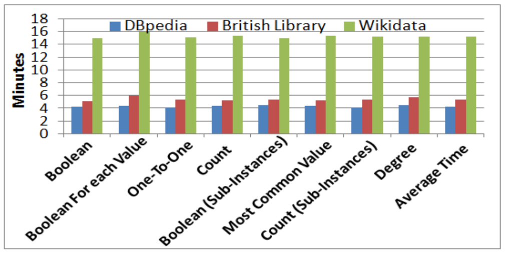
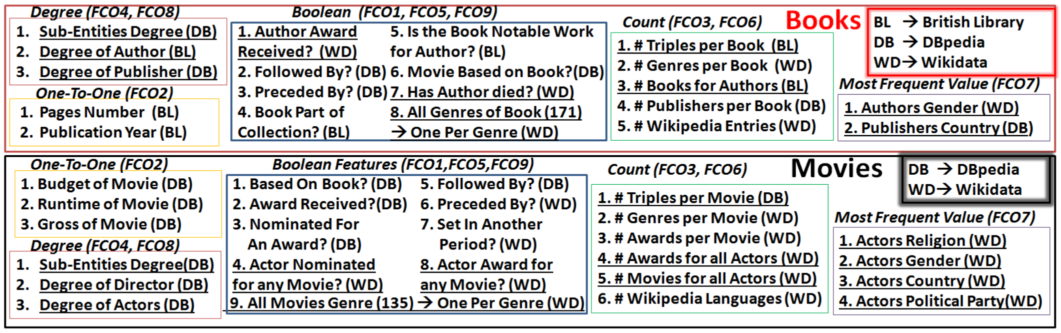


Figure 6.13:
Generation Time for each Feature Operators Category and Dataset


Books

BL → British Library
DB → DBpedia
WD→ Wikidata

Degree (FCO4, FCO8)
1. Sub-Entities Degree (DB)
2. Degree of Author (BL)
3. Degree of Publisher (DB)

One-To-One (FCO2)
1. Pages Number (BL)
2. Publication Year (BL)

Boolean (FCO1, FCO5, FCO9)
1. Author Award Received? (WD)
2. Followed By? (DB)
3. Preceded By? (DB)
4. Book Part of Collection? (BL)
5. Is the Book Notable Work for Author? (BL)
6. Movie Based on Book?(DB)
7. Has Author died? (WD)
8. All Genres of Book (171) → One Per Genre (WD)

Count (FCO3, FCO6)
1. # Triples per Book (BL)
2. # Genres per Book (WD)
3. # Books for Authors (BL)
4. # Publishers per Book (DB)
5. # Wikipedia Entries (WD)

Most Frequent Value (FCO7)
1. Authors Gender (WD)
2. Publishers Country (DB)

Movies

DB → DBpedia
WD→ Wikidata

One-To-One (FCO2)
1. Budget of Movie (DB)
2. Runtime of Movie (DB)
3. Gross of Movie (DB)

Degree (FCO4, FCO8)
1. Sub-Entities Degree(DB)
2. Degree of Director (DB)
3. Degree of Actors (DB)

Boolean Features (FCO1,FCO5,FCO9)
1. Based On Book? (DB)
2. Award Received?(DB)
3. Nominated For An Award? (DB)
4. Actor Nominated for any Movie? (WD)
5. Followed By? (DB)
6. Preceded By? (WD)
7. Set In Another Period? (WD)
8. Actor Award for any Movie? (WD)
9. All Movies Genre (135) → One Per Genre (WD)

Count (FCO3, FCO6)
1. # Triples per Movie (DB)
2. # Genres per Movie (WD)
3. # Awards per Movie (WD)
4. # Awards for all Actors (WD)
5. # Movies for all Actors (WD)
6. # Wikipedia Languages (WD)

Most Frequent Value (FCO7)
1. Actors Religion (WD)
2. Actors Gender (WD)
3. Actors Country (WD)
4. Actors Political Party(WD)

Figure 6.14: Selected Features for Books & Movies with their Provenance

Table 6.4: Accuracy for each Feature Operators Category (Movies & Books test size 0.2)

Feature Operators Category	**Average (Movies)**	**Max (Movies)**	**Average (Books)**	**Max (Books)**
All Features (FCO1-FCO9)	0.871	0.906	0.730	0.762
Continuous Features	0.861	0.896	0.709	0.739
New Features (FCO4-FCO9)	0.835	0.865	0.650	0.675
Existing Features (FCO1-FCO3)	0.827	0.855	0.694	0.716
Count (FCO3,FCO6)	0.830	0.862	0.706	0.709
1-1 Relationship (FCO2)	0.791	0.808	0.570	0.607
Categorical Features	0.760	0.818	0.673	0.694
Boolean (FCO1, FCO5, FCO9)	0.750	0.774	0.634	0.656
Degree (FCO4,FCO8)	0.741	0.780	0.608	0.627
Most Frequent Value (FCO7)	0.698	0.758	0.560	0.595
Trivial Case	0.495	0.532	0.508	0.551

for each different category (and each dataset) for 1,076 books. As we can see, the generation time depends highly on the dataset to which we send SPARQL queries, e.g., DBpedia's response time is much shorter than Wikidata's. Concerning the generation time of a specific feature operators category, the degree operators (FCO4, FCO8) and the boolean for each value of a predicate (FCO9) need more time to be generated while the remaining ones need approximately the same time on average for being generated. Finally, the execution time for retrieving the similar entities from `LODsyndesis` was 105 seconds.

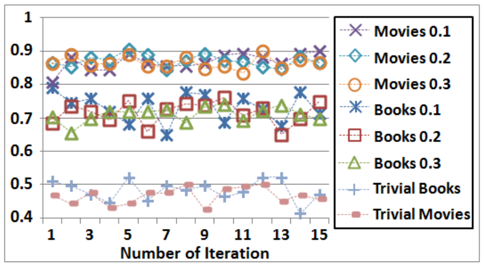


Figure 6.15:
Accuracy in Each Iteration & Test Size for Dataset Books and Movies

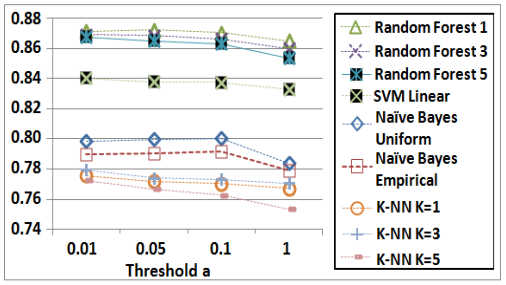


Figure 6.16:
Average Accuracy of Models in Cross Validation for Movies with Test Size 0.2

Results for Movies Dataset. Figure 6.14 shows the selected features (and their category) for the dataset of movies (features belonging in the additional categories that we propose are underlined). In total we sent 39,250 queries and we created 159 features (147 categorical and 12 continuous). In Figure 6.15 we can see a plot with the accuracy of each test size (using the best model selected by the cross validation process) and we can observe that the accuracy is much higher comparing to a trivial case while the highest variation occurred for test size equal to 0.1. In all iterations, the best model was a *Random Forest* (with different parameters in many cases). Figure 6.16 shows the average accuracy for each model (and each threshold a) in the cross validation process for test size 0.2. We observe that *Random Forest* models achieved higher accuracy (mainly when min leaf size equals 1, i.e., Random Forest 1) comparing to the other models. The next ones with the highest accuracy is the *linear SVM*, followed by the two *Naive Bayes* models and finally the *K-NN* ones. However, all these models are better comparing to the trivial one whose accuracy is approximately 0.5. Table 6.4 shows the average and maximum accuracy for each different features' category in descending order with respect to their average accuracy. The continuous ones (mainly the count features, i.e., FCO3 and FCO6) seem to be the most predictive while all the categories achieved high accuracy comparing to a trivial case. Moreover, the average accuracy of features that other approaches also support (i.e., FCO1-FCO3) was 0.827 while for the additional features that we propose (e, FCO4-FCO9 in Table 6.4) the average accuracy was 0.835. By combining all the categories of features, the average accuracy was 0.871, which means that the additional features improved the accuracy in this particular problem.

Results for Books Dataset. Figure 6.14 shows the selected features (and their categories) for the books dataset. In total we sent 21,520 queries and we created 190 features (180 categorical and 10 continuous). In Figure 6.15 we can see a plot with the accuracy of each test size and we observe that the accuracy is much higher comparing to the trivial case while the highest variation occurred for test size equal to 0.1. In 42 iterations, the best model was a *Random Forest* (with different parameters in many cases) while in 3 cases the best model was a *linear SVM* while the variations of *K-NN* algorithms were more effective than the *Naive Bayes* ones. As we can observe in Table 6.4, the combination of all features gave the maximum accuracy, while the continuous features, and especially the count features (FCO3, FCO6), were more predictive comparing to the remaining ones. Moreover, for the feature operators FCO1-FCO3 the average accuracy was 0.694 while for the feature operators FCO4-FCO9 was 0.65. However, by combining both types of features the average accuracy improved, i.e., 0.73, therefore the additional features improved the accuracy for books' dataset, too.

6.2.6 Synopsis.

We have shown how we can exploit the wealth of Linked Data and the ML machinery for improving the quality of automatic classification. We presented a tool, called `LODsyndesis`$_{\mathcal{ML}}$, which exploits Linked Data (and the related technologies) for discovering automatically features for any set of entities. We categorized the features and we detailed the process for producing them. For evaluating the benefits of our approach, we used two datasets and the results showed that the additional features did improve the accuracy of predictions, while the most effective model for both datasets was a *Random Forest* one.

6.3 LODVec. Knowledge Graph Embeddings over Hundreds of Linked Datasets

There is an increasing trend of exploiting LOD (Linked Open Data) for creating embeddings for URIs (Uniform Resource Identifiers), which can be exploitable for a number of tasks. Indicatively, they can be exploited i) for machine learning-based tasks [211], such as classification, regression, etc., ii) for similarity-based tasks [72], e.g. "Give me the top-K related entities to a given one", iii) for link prediction purposes [177], and others. There have been proposed several novel methods [211] taking as input RDF (Resource Description Frameworks) knowledge graphs and producing URI sequences for a set of given entities, i.e., sequences starting from a focused entity (or URI) that contains a path of URIs which is reachable from that entity. These sequences are given as input for producing URI embeddings which can be exploited in the aforementioned tasks.

However, current approaches exploit usually a single dataset for creating URI embeddings for one or more URIs (or entities). Moreover, many approaches are difficult to be configured by non-experts, since they do not provide an interactive service. Our objective is to make it feasible to exploit hundreds of RDF datasets simultaneously, for creating URI sequences and URI embeddings for any given entity (i.e., a URI). Concerning our research hypothesis, we assume that it is better to exploit multiple datasets for creating URI sequences and embeddings, instead of using one or few datasets, whereas we assume that there is not a single knowledge graph that can outperform all the others for any possible task.

Generally, it is not easy to combine all the available information for a given entity as we have seen in Chapter 2. For example, each of the three datasets of Figure 6.17 uses a different URI for representing the movie "Inception" and the schema element "actor", while these datasets are located in different places. Indeed, for combining information from several datasets, one should collect the desired data by performing cross-dataset reasoning.

For tackling the above difficulties, we exploit `LODsyndesis`, since its' indexes offer direct and fast access to all the available information for any entity (see Chapter 4). In this way, it is feasible to create URI sequences for the same entity from multiple datasets, therefore, the number of possible URI sequences that can be created is highly increased (as we shall see in §6.3.4). For example, suppose that the required task is to predict the exact rating of a movie, e.g., "Inception" (see Figure6.17), and for this reason we plan to create embeddings from URI sequences for using them in such a machine learning task. In Figure6.17, by using only one dataset, say *DBpedia* (`http://dbpedia.org/`), we can find data such as the genres of this movie. However, it does not contain information about the awards won by this movie, e.g., in Figure 6.17 such data occur in *Wikidata* (`http://wikidata.org/`). On the contrary, by using `LODsyndesis` (see the lower side of Figure 6.17), it is feasible to create URI sequences by using all the three datasets of Figure 6.17.

We introduce a research prototype, i.e., `LODVec`, that (i) takes as input one or more entities (as URIs or in plain text), (ii) it offers several configurable options for creating URI sequences and embeddings for these entities from hundreds of datasets through `LODsyndesis` and (iii) it produces URI sequences based on user's selections. Moreover, `LODVec` (iv) converts the produced sequences into vector representations (i.e., embeddings) by exploiting *word2vec* approach [153, 154] through *dl4j* API (`https://deeplearning4j.org/`). Finally, it can (v) exploit the produced vectors for several purposes, e.g., for finding the most common words for a given one through *dl4j* library or/and for performing classification and regression tasks by using *WEKA* API [255]. For testing the proposed approach, we report experimental results for machine learning classification and regression tasks by using two datasets containing movies and music albums, i.e., the target of classification task is to classify whether a movie or a music album has a high or low rating, whereas the target of regression task is to predict their exact rating. We introduce experiments showing the impact of using multiple datasets and cross-dataset reasoning in terms of effectiveness, whereas, we compare the performance of different configurations.

The rest of this section is organized as follows: §6.3.1 introduces the background and related work, while §6.3.2 discusses the placement of `LODVec` in the Data Integration Landscape. §6.3.3 provides the problem statement, the algorithm for creating URI sequences

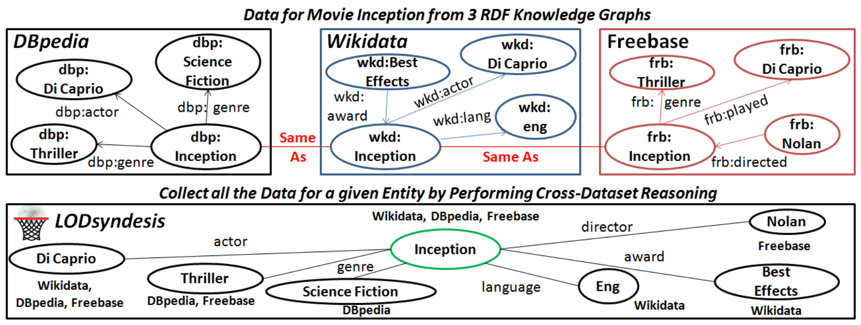


Figure 6.17: Running example containing 3 knowledge graphs and LODsyndesis

and all the steps and functionalities that our approach supports, whereas §6.3.4 includes the experimental evaluation about the effectiveness of our approach.

6.3.1 Background & Related Work

6.3.1.1 Background

Word2Vec. It is a shallow two-layer neural network model for producing word embeddings [153, 154]. It takes as input a text, and it produces a vector with several (usually hundreds of) dimensions for each unique word appearing in the text. The target of *word2Vec* [153, 154] is to group the vectors of similar words closely in the vector space. Here, we will exploit this model for creating vectors for entities, by using the skip-gram model, which is a method that uses a specific word for predicting a target context, since "it produces more accurate results for large datasets" [2]. Our target is to use this model for placing similar entities (e.g., similar movies) to a close position in the vector space.

6.3.1.2 Related work

Knowledge graph embedding approaches. *RDF2Vec* [211] is an approach that takes as input an RDF knowledge graph, produces URI sequences based on several strategies, such as random graph walks, and uses *word2vec* for creating vectors. They have also proposed strategies for performing biased graph walks [61], which are based on a number of metrics and statistics, such as the frequency of properties, objects, pagerank and others. They have tested these strategies for multiple tasks, such as classification and regression, by using two datasets *Wikidata* and *DBpedia*, whereas they have used the *GloVe* model [195] for creating RDF embeddings by exploiting global patterns. The authors in [111] used several bibliographic RDF datasets and *word2vec* for enriching the data of scientific publications with information from multiple data sources, while in [123] the authors exploited enriched ontology structures for producing RDF embeddings which were used for the task of Entity

[2]https://deeplearning4j.org/docs/latest/deeplearning4j-nlp-word2vec

Linking. In [177], the authors combined embeddings from *DBpedia* and social network datasets for performing link prediction, whereas in [72] Wikipedia knowledge graph was exploited for finding the most similar entities to a given one for a specific time period. Concerning other graph-based models, such as TransH [253] and TransR [146], they use algorithms for creating entity and relation graph embeddings, i.e., the relationships between two entities are represented as translations in the embedding space.

Feature extraction approaches combining data from several datasets. In [166], the authors proposed a tool that can send SPARQL queries in several endpoints for creating features. However, it does not produce embeddings and it cannot collect all the data for a given entity (i.e., cross-dataset reasoning is required). Moreover, RapidMiner Semantic Web Extension [208] creates features by integrating data from a lot of datasets. However, it performs the integration task by traversing owl:sameAs paths on-the-fly (through SPARQL queries, which can be time-consuming), and not by exploiting pre-constructed indexes.

Novelty & Comparison with other approaches. To the best of our knowledge, this is the first work providing an interactive tool which can easily create URI embeddings for any set of entities by combining data from hundreds of datasets simultaneously. Since current approaches do not take into account the equivalences in schema and instance level, they have been mainly tested on a single dataset. At this point our objective is a) to offer a simple way for creating URI sequences for multiple datasets, and b) to investigate whether the creation of even simple URI sequences and embeddings from different datasets can improve the effectiveness of several tasks (e.g., machine-learning tasks). Concerning the limitations of our approach, for the time being we do not support methods for creating longer URI sequences (e.g., through random walks [60]), and algorithms that have been successfully applied to knowledge graphs (e.g., [146, 253]).

6.3.2 Placement of LODVec in the Data Integration Landscape

LODVec can be also considered as special data integration scenario. Similarly to LODsyndesis$_{\mathcal{ML}}$, it includes the steps of *Dataset Discovery* and *Dataset Selection*, i.e., for selecting the most appropriate datasets. Moreover, it performs *Fetching and Transformation*, i.e., for converting the data to embeddings, *Instance & Schema Matching* i.e., by exploiting the cross-dataset identity reasoning, and *Publishing*, i.e., it produces a new dataset containing embeddings. Finally, its' *Basic Service to Deliver* is *Question Answering* for machine-learning based tasks.

6.3.3 LODVec: The Proposed Approach

§6.3.3.1 introduces the problem statement, §6.3.3.2 shows some useful notations and metadata, and §6.3.3.3 describes all the steps of LODVec.

6.3.3.1 Problem Statement

Creating URI Sequences. The input is a set of entities $E' \subseteq E$, and the first target for each $e \in E'$ is to create a finite set of URI sequences $Seq_U(e)$. Each URI sequence $s \in Seq_U(e)$ is of the form $\langle e, p, o\rangle \in \mathcal{T}$ where $p \in P$ and $o \in (C \cup E)$ (remind that C is the set of classes). Here, we exploit the neighborhood of an entity e by following both direct and inverse edges, therefore each URI sequence corresponds to a single triple containing e either as a subject or as an object. Our target is to collect all the produced $Seq_U(e)$ for each $e \in E'$, i.e., we construct the set $Seq_U E' = \bigcup_{e \in E'} Seq_U(e)$, where $Seq_U E' \subseteq T'$.
From URI Sequences to URI Embeddings. The target is to map each entity e to a vector space $v(e)$ by using the set $Seq_U E'$, and *word2vec* algorithm (by using skip-gram model). We expect that if two entities e and e' are similar, then their produced vectors $v(e)$ and $v(e')$ will be close to the vector space.
Output Exploitation. Our target is the output vectors to be directly exploitable for machine learning classification and regression tasks, and for finding the top-K similar entities to a given entity e. Concerning the first case, one should also provide as input the corresponding categorical or continuous variable $Y(e)$ for a given entity e. On the second case, there is no need for additional input.

6.3.3.2 Metadata & Notations

First, remind that $D = \{D_i, ..., D_n\}$ is a set of datasets, and as $triples(D_i)$ the set of triples for a given dataset D_i (e.g., DBpedia). Table 6.5 represents a set of notations and metadata that can aid users to select what type of URI sequences will be created. The first one corresponds to the datasets containing a triple t, while the second one indicates which datasets contain at least one triple for an entity e. The third one denotes the datasets that contain at least one triple, for one or more entities $e \in E'$. The fourth is used for showing how many of the entities in E' can be found in a single dataset D_i (i.e., a number in range $[0, |E'|]$), whereas the fifth and the sixth formulas denote all the properties of an entity e and of all the entities E', respectively. The seventh formula shows the number of entities, for whom there is at least one triple that contain a property p. Formula 8 shows all the objects of a triple whose subject is e and predicate is p (e.g., all the actors of a movie). The last formula shows the frequency (popularity) of a URI in the whole graph, i.e., how many triples contain a URI u.

6.3.3.3 The Steps & Algorithm for Creating URI Sequences

Here, we describe the functionality and all the steps of `LODVec` (see Figure 6.18).

Steps 1-4. Input & Configuration. The first step (see Figure 6.18) is to receive the input entities E' (we support over 412 million URIs), which can be given a) as a list of URIs (e.g.,

ID	Notation or Metadata	Formula
1	Provenance of Triple t	$prov(t) = \{D_i \in D \mid t = \langle s,p,o\rangle, t \in triples(D_i)\}$
2	Provenance of Entity e	$dsets(e) = \{D_i \in D \mid \exists\langle e,p,o\rangle \in triples(D_i)\}$
3	Provenance of Entities E'	$dsets_{E'} = \bigcup_{e\in E'} dsets(e)$
4	Coverage of a Dataset given E'	$covD(D_i,E') = \lvert\{e \in E' \mid D_i \in dsets(e)\}\rvert$
5	Coverage of Properties given E'	$covP(p,E') = \lvert\{e \in E', \mid \exists\langle e,p,o\rangle \in T'\}\rvert$
6	Properties of an Entity e	$Prop(e) = \{p \in P \mid \langle e,p,o\rangle \in T'\}$
7	Properties of Entities E'	$Prop_{E'} = \bigcup_{e\in E'} Prop(e)$
8	Objects of an entity-prop. pair	$Obj(e,p) = \{o \in U \mid \langle e,p,o\rangle \in T'\}$
9	Frequency of a given URI u	$freq(u) = \lvert\{t \in T' \mid t = \langle s,p,o\rangle, s = u \text{ or } o = u\}\rvert$

Table 6.5: Notations-Metadata

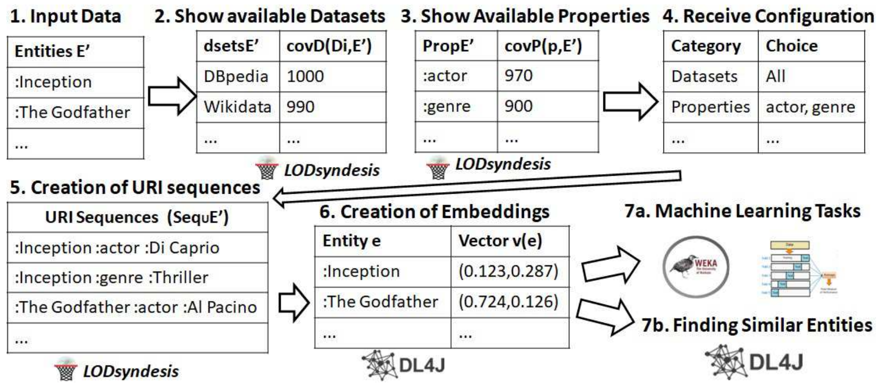

1. Input Data
Entities E'
:Inception
:The Godfather
...
2. Show available Datasets
dsetsE' | covD(Di,E')
DBpedia | 1000
Wikidata | 990
3. Show Available Properties
PropE' | covP(p,E')
:actor | 970
:genre | 900
4. Receive Configuration
Category | Choice
Datasets | All
Properties | actor, genre
LODsyndesis
5. Creation of URI sequences
URI Sequences (SequE')
:Inception :actor :Di Caprio
:Inception :genre :Thriller
:The Godfather :actor :Al Pacino
6. Creation of Embeddings
Entity e | Vector v(e)
:Inception | (0.123,0.287)
:The Godfather | (0.724,0.126)
7a. Machine Learning Tasks
WEKA
7b. Finding Similar Entities
DL4J

Figure 6.18: The steps of LODVec approach.

dbp:Inception), b) as a list of entities in plain text (e.g., "Inception"), or c) just a URI that represents an RDF class or a category (e.g., dbc:Films_about_dreams)! In the latter case, LODVec retrieves automatically the desired URIs. By exploiting LODsyndesis, LODVec can optionally show to the user (i) the datasets containing the input entities (i.e., $dsets_{E'}$), in descending order according to their $covD(D_i,E')$, and (ii) the list of available properties for the given entities, i.e., $Prop_{E'}$, in descending order according to their $covP(p,E')$ (i.e., Steps 2 and 3). In Step 4 of Figure 6.18, the user selects which configuration will be used, i.e., the input of Alg. 10. In particular, one can select the datasets D_{sel} that will be used ($D_{sel} \subseteq dsets_{E'}$), the desired properties $Prop_{sel}$ ($Prop_{sel} \subseteq Prop_{E'}$) and whether all the possible sequences or the top-K ones will be created. For creating the top-K URI sequences, one should give as input the exact number K (i.e., a positive integer), and the method to sort the triples according to the frequency of each triple's object (i.e., in ascending, descending or random order).

Step 5. Algorithm for Creating URI Sequences. Alg. 10 creates URI sequences for a set of entities E'. First, it initializes the desired output (line 1). Then, it iterates over all the input entities and for each entity e (lines 2-3), it initializes its corresponding set of URI sequences and a frequency map (it is used for creating only the top-K sequences). The next step is to traverse each direct or inverse property p of entity e (i.e., $Prop(e)$) that be-

ALGORITHM 10: Creating *URI sequences* for a set of entities E'

```
Input: Entities $E'$, properties $Prop_{sel}$, datasets $D_{sel}$, cardinality $crd$ ({"All" | "Top-K"}),
       Integer $K$, sortType ({"asc"| "desc"| "rand"})
Output: URI Sequences $Seq_U E'$ for all the entities $E'$
1  $Seq_U E' \leftarrow \emptyset$
2  forall $e \in E'$ do
3      $Seq_U(e) \leftarrow \emptyset, freqMap(e) \leftarrow \emptyset$
4      forall $p \in (Prop(e) \cap Prop_{sel})$ do
5          forall $o \in Obj(e,p), prov(\langle e,p,o \rangle) \cap D_{sel} \neq \emptyset$ do
6              if $crd \equiv$ "All" then
7                  $Seq_U(e) \leftarrow Seq_U(e) \cup \{\langle e,p,o \rangle\}$
8              else if $crd \equiv$ "Top-K" then
9                  $freqMap(e) \leftarrow freqMap(e) \cup \{\langle e,p,o \rangle, freq(o)\}$
10     if $crd \equiv$ "Top-K" then
11         $sortSeq(e) \leftarrow mergeSortByValue(freqMap(e))_{sortType}$
12         forall $\langle e,p,o \rangle \in Left(sortSeq(e))$ do
13             $Seq_U(e) \leftarrow Seq_U(e) \cup \{\langle e,p,o \rangle\}$
14             if $|Seq_U(e)| \equiv K$ then
15                 break
16     $Seq_U E' \leftarrow Seq_U E' \cup Seq_U(e)$
17 Return $Seq_U E'$
```

long also to the desired properties given by the user (i.e., $Prop_{sel}$). Remind that the indexes of `LODsyndesis` store in the same place all the objects for each entity-property pair (e.g., movie-actor). Therefore, for each p we traverse all the possible objects, and we check if at least one dataset containing that triple, belongs to the desired datasets D_{sel} (lines 4-5). In such a case, we check the cardinality variable, i.e., if the user desires to find all the possible URI sequences, we add the URI sequence (i.e., triple) to $Seq_u(e)$ (lines 6-7). Otherwise, we add the triple and its object's frequency, i.e., $freq(o)$ (retrieved by sending a request to `LODsyndesis` REST services [169]), to the frequency map (lines 8-9). After traversing all the desired triples for the entity e, if the user wants to create the top-K URI sequences, we sort the values according to the given sortType (lines 10-11), e.g., if sortType equals "desc", we will sort the K triples, which are found in the left side of $freqMap(e)$, in descending order with respect to their object's frequency. We iterate over the sorted map (i.e., $sortSeq(e)$), that contains in its left side (i.e., $Left$) a URI sequence and in its right side the $freq(o)$, and we add to $Seq_u(e)$ the top-K results (lines 12-15). Finally, we add the $Seq_U(e)$ to $Seq_U E'$, whereas after all the iterations, Alg. 10 returns all the URI sequences.

The time complexity of Alg. 10 is $O(|triples_{E'}|)$, since in the worst case we read all the triples for the entities E'. The space complexity is $O(Seq_U E')$, since we keep in memory all the produced sequences. We do not load in memory the triples of each entity, since we use random access methods for reading the desired part of the indexes. Finally, for creating the top-K sequences for each entity e we also use a sorting algorithm whose time complexity is $O(n * log_n)$.

Steps 6-7. Creation of Embeddings & Exploitation. The next step (Step 6 of Figure 6.18) is to exploit the produced set $Seq_U E'$ for creating one vector per entity e. Indeed, we use the *word2vec* implementation of *dl4j* library, which produces as output a vector $v(e)$ for each $e \in E'$. These vectors can be exploited for (i) machine-learning and (ii) similarity tasks. Regarding task (i), the vectors can be downloaded in ".arff" format for performing classification and regression through *WEKA* API (Step 7a of Figure 6.18), while `LODVec` offers a service for producing automatically such results. Concerning task (ii), they can be downloaded in ".txt" format, which is directly accessible from *dl4j* API. Finally, `LODVec` offers a service for finding the top-K related entities to a given one (Step 7b of Figure 6.18).

6.3.4 Experimental Evaluation

In §6.3.4.1, we evaluate the impact of cross-dataset reasoning and of using multiple datasets, for four machine-learning tasks, while §6.3.4.2 shows a small example for the task of finding similar entities. All the experiments were performed on a single machine with an *i5 core*, *8GB RAM*, and *1 TB disc space*.

6.3.4.1 Machine Learning Tasks - The Gain of Using Multiple Datasets

Movies & Music Albums Datasets. We use the Metacritic Movies and Music Albums datasets (derived from [209]) containing the DBpedia URIs of 2,000 movies and of 1,600 music albums. Both datasets contain an average rating of all time reviews for each movie and music album. Concerning movies, 1,000 of them have high rating (> 60), and the remaining 1,000 ones have low rating (< 40). Regarding music albums, 800 of them have high rating (> 79), and the other 800 ones have low rating (< 63). The goal of (binary) classification is to predict whether a movie or a music album has a high or a low rating, whereas the target of regression is to find the exact rating of each movie and music album.

Word2Vec Parameters. For building our *word2vec* model, we use the skip-gram model of *dl4j* library, we exclude URIs existing < 5 times in the produced sequences ($minWordFrequency = 5$), we use 10 iterations and we select the window size parameter to be 2 ($windowSize = 2$). For each entity e, we produce a single vector $v(e)$ with 100 dimensions ($layerSize = 100$), and we expect that movies and albums with similar rating will be placed closely in the vector space.

Machine Learning Models & Metrics. The vectors produced by `LODVec` are given as input in WEKA API [255], for performing classification and regression, by using a 10-fold cross validation [255]. Regarding classification (CF), we use the default implementation of *Naive Bayes* (NB) and *Support Vector Machine* (SMO) of WEKA. For each model, we mea-

Datasets	Prop.	*crd*	$\|Seq_U(e)\|$ Average	Creation Time	NB (CF)	SVM (CF)	LR (Reg)	SMOreg (Reg)
Freebase (FR)	All	All	112.0	4.3 min	79.71%	82.02%	16.37	16.56
DBpedia (DB)	All	All	23.8	4.0 min	68.42%	71.14%	20.02	20.71
Wikidata (WK)	All	All	22.5	4.2 min	66.88%	67.66%	20.81	21.40
DB,WK	All	All	38.3	4.4 min	68.62%	74.92%	19.18	19.90
DB,FR	All	All	132.0	4.5 min	79.75%	82.51%	16.11	16.35
FR,WK	All	All	129.0	4.6 min	81.20%	83.32%	16.25	16.55
DB,FR,WK	All	All	144.7	4.7 min	82.11%	84.10%	16.01	16.31
All 14 $dsets(E')$	All	All	**170.1**	5.7 min	**82.41%**	**84.70%**	**15.57**	**15.65**
FR	type	All	5.5	3.5 min	67.49%	71.40%	19.18	19.59
All 14 $dsets(E')$	type	All	18.3	4.0 min	67.53%	72.92%	18.90	19.42
All 14 $dsets(E')$	ct+tp	All	30.9	4.7 min	73.13%	74.90%	18.80	19.04
All 14 $dsets(E')$	All	30 asc	30.0	120 min	66.95%	72.50%	19.58	20.05
All 14 $dsets(E')$	All	30 rand	30.0	6.0 min	69.24%	73.10%	19.86	20.26
All 14 $dsets(E')$	All	30 desc	30.0	120 min	71.43%	75.86%	19.07	19.46

Table 6.6: Classification and regression experiments on Movies dataset

sure the accuracy percentage (percentage of correct predictions), i.e., the goal is to maximize that percentage. Concerning regression (Reg), we use the default implementation of *Linear Regression* (LR) and *Support Vector Machine for Regression* (SMOreg) of WEKA, and we measure the root mean squared error (RMSE), i.e., the target is to minimize the RMSE value. Finally, we measure the accuracy and the RMSE for the trivial *Vote* method, that selects randomly a class and a rating for each entity.

Results for both Tasks and Datasets. Tables 6.6 and 6.7 show several statistics and experiments for movies and music albums, respectively. In each Table, the first column shows the used RDF datasets, the second the properties, the third one the cardinality, the fourth one the average URI sequences per entity, and the fifth one the total creation time of all URI sequences. The columns 6 and 7 show the accuracy of the classification task (SF) and the last two columns the RMSE value for the regression task (Reg), by using different models.

Results for Movies. In rows 2-9 of Table 6.6, we show experiments by using all the possible properties and by creating all the possible URI sequences, however, in each case we use a different subset of datasets. We can see how the average number of URI sequences increases as we add more datasets (see the fourth column of Table 6.6), e.g., by using *DBpedia* we created 23.8 URI sequences per movie, whereas with all the datasets we created on average 170.1 URI sequences. Moreover, as we add more datasets, the total time for creating all the URI sequences did not increase so much, i.e., by using only *DBpedia* we needed 4 minutes (i.e., 0.12 seconds per entity), whereas by using all datasets the corresponding time was 5.7 minutes (i.e., 0.17 seconds per entity).

Classification and Regression Results. First, for the trivial *Vote* method, we obtained 50% accuracy, whereas the *RMSE* value was 23.1. The single dataset with the highest accuracy and the lowest RMSE was *FreeBase* (`http://freebase.com`), i.e., we obtained 82% accuracy through *SVM* model, whereas its RMSE value was 16.37 (through *LR* model). The corresponding percentages for *DBpedia* and *Wikidata* were much smaller. However, by taking each pair of these 3 datasets, the accuracy increased, and the RMSE value decreased in all cases (versus using only one dataset from each pair). Certainly, by using only *FreeBase*, we achieved better results comparing to use both *DBpedia* and *Wikidata*, which seems rational, since from *Freebase* we created a large number of sequences. However, by combining *Freebase* with either *DBpedia* or *Wikidata*, or by using all 3 datasets (these combinations are feasible due to cross-dataset reasoning), we obtained better results. By using all the 14 available datasets (out of 400 datasets) having data about these movies, we achieved the highest accuracy (84.7%) and the lowest RMSE (15.57). However, we should note that 8 of these 14 datasets offered very few URI sequences for this task. Finally, *SVM* outperformed *NB* in all classification cases, while *LR* was more effective in regression tasks.

Other Configurations. In rows 10-15 of Table 6.6, we show indicatively some experiments with different configurations. By creating URI sequences containing only the property *rdf:type*, we obtained better results by using all datasets in comparison to use only *Freebase*, while by creating sequences that contain both *rdf:type* and *dcterms:subject* (ct+tp) the results improved. Regarding experiments containing the top-*K* sequences, by creating only the top-30 URI sequences according to objects frequency for each movie in descending order (i.e., triples with the 30 most popular objects per movie), we achieved the highest accuracy and the lowest RMSE. On the contrary, a random order was more effective versus an ascending one. Concerning the creation time of *desc* and *asc*, is it slower versus the other configurations (i.e., 3.6 seconds per entity), since we send several requests to LODsyndesis [169] for retrieving the frequency of objects.

Results for Music Albums. Table 6.7 shows experiments by creating all the URI sequences, each time by using a different subset of datasets. By using only *DBpedia* (see the fourth column), we created on average 16.3 URI sequences per album, whereas by using all the datasets, we created 35.9. Concerning the creation time of URI sequences, it is again low (i.e., 0.11 seconds per entity).

Classification and Regression Results. For the trivial *Vote* method, we obtained 50% accuracy, whereas the *RMSE value* was 13.95. The single dataset having the best performance was *DBpedia* (see Table 6.7), whereas *Freebase* was not so accurate for this task. Therefore, even by selecting to use exactly one dataset for both movies and music albums (i.e., the same dataset in both cases), we will not be able to obtain good results for both tasks. Similarly to movies, as we add more datasets, the results are better for both regression and classification, except for the pairs containing the dataset *Wikidata*. By including all the 5 available datasets for music albums, we obtain the highest accuracy (i.e., 71.31%) and the lowest *RMSE* value (i.e., 12.41). Finally, similarly to the case of movies, the best

Datasets	Prop.	*crd*	$\|Seq_U(e)\|$ Average	Creation Time	NB (CF)	SVM (CF)	LR (Reg)	SMOreg (Reg)
Freebase (FR)	All	All	9.2	2.1 min	52.75%	56.57%	13.74	14.08
DBpedia (DB)	All	All	16.3	2.3 min	64.01%	68.21%	12.75	13.07
Wikidata (WK)	All	All	6.7	1.9 min	60.38%	61.37%	13.89	14.68
DB,WK	All	All	20.5	2.3 min	63.42%	67.61%	12.85	13.10
DB,FR	All	All	25.5	2.3 min	64.30%	68.64%	12.60	13.03
FR,WK	All	All	16.0	2.2 min	54.10%	57.65%	14.00	14.50
DB,FR,WK	All	All	29.8	2.4 min	64.73%	69.02%	12.55	13.01
All 5 $dsets(E')$	All	All	**35.9**	2.5 min	**67.21%**	**71.31%**	**12.41**	**12.72**

Table 6.7: Classification and regression experiments on Music Albums dataset

Datasets	Position 1	Position 2	Position 3	Position 4	Position 5
DB	Pink Panther 2	**Ratatouille**	Shrek 2	Rain Mant	**Space Chimps**
DB,FR	**Finding Nemo**	**Toy Story 2**	**Incredibles**	**Toy Story**	Princess and the Frog
All	**Finding Nemo**	**Ratatouille**	**Incredibles**	**Toy Story 3**	**Toy Story**

Table 6.8: Top-5 related movies to "Wall-E" movie by using different datasets

model was the *SVM* for classification and the *LR* for regression.

6.3.4.2 Finding Similar Entities

`LODVec` can produce the top-*K* similar entities for a given one, since it uses *word2vec* from *dl4j* API. Table 6.8 shows an indicative example, i.e., finding the 5 most similar movies (from the set of 2,000 movies) to the animated movie "WALL-E", by creating all the URI sequences and by using a) only *DBpedia*, b) *DBpedia* and *Freebase* and c) all datasets. For evaluating the results, we typed in *Google Search Engine* the keywords "WALL-E related movies", and we retrieved a list with related movies that "People also search for WALL-E". By using only *DBpedia*, 2 of 5 movies were in the list of related movies (the bold ones in Table 6.8), whereas by using both *Freebase* and *DBpedia*, 4 of 5 movies were at that list. Finally, by exploiting all the datasets, all the 5 movies were part of the list.

6.3.5 Epilogue.

We introduced a prototype called `LODVec` that exploits the semantically enriched indexes of `LODsyndesis` knowledge graph, and offers configurable options for creating URI sequences and embeddings through *word2vec* algorithm, for over 400 million entities from 400 RDF datasets. The produced embeddings can be exploited in several tasks. In our case, we evaluated the gain of using multiple datasets (and cross-dataset reasoning) for four machine-learning tasks, e.g., for classifying whether a movie has a high or low rating. Indicatively, by using data from *DBpedia* we created 23.8 URI sequences per movie, while by combining information from 14 datasets, the corresponding number was 170.1. We identified even 13% increase in the accuracy of predicting if a movie is highly rated by using multiple datasets, instead of using only *DBpedia*.

6.4 LODQA. Question Answering over large number of datasets

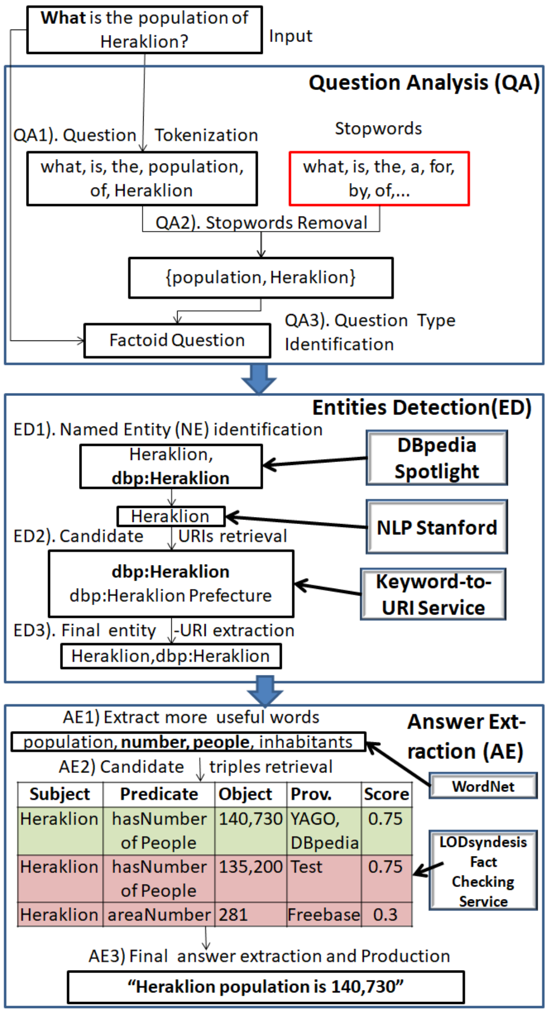
Open domain (as opposed to closed domain) Question Answering (QA) is a challenging task, since it requires to tackle a number of issues: (i) the issue of data distribution, i.e., several datasets, that are usually distributed in different places, should be exploited for being able to support open domain question answering, (ii) the difficulty of word sense disambiguation, because the associated vocabulary is not restricted to a single domain, and (iii) the difficulty (or inability) to apply computationally expensive techniques, such as deep NLP analysis, due to the huge size of the underlying sources.

For tackling these challenges, recently we have focused on Open Domain Question Answering over Linked Data. In particular, we have created LODQA [73], which is an ongoing research prototype, that is able to answer factoid, confirmation, and definition questions. In particular, it is a Linked Data-based Question Answering system that exploits LODsyndesis since it offers two distinctive features for the QA process, which are not supported by a single source: (a) it is feasible to verify an answer to a given question from several sources, and (b) the number of questions that can be answered is highly increased, because datasets usually contain complementary information for the same topics and entities (i.e., see Chapter 4).

Essentially, we try to find the best triple(s) for answering the incoming question; we do not carry out any other information integration techniques (like those surveyed in [173]). Regarding (a), suppose that the given question is "What is the population of Heraklion?", and the system retrieves two candidate triples, i.e., (Heraklion, population, 140,730), (Heraklion, population, 135,200). The two triples contain a different value for the population of that city, however, suppose that the first triple can be verified from four datasets (say D_1, D_2, D_3, D_4), whereas the second one only from a single dataset (say D5). In this example, LODQA will return as correct answer the first triple, because it can be verified from a larger number of datasets, thereby, we have more evidence about its correctness. Regarding (b), suppose that LODQA receives the two following questions for an other domain (say marine domain), "Is Yellowfin Tuna a predator of Atlantic pomfret?" and "Which is the genus of Yellowfin Tuna". These two questions are addressed to the same entity i.e. "Yellowfin Tuna", however there is not a single dataset where we can find the desired information for answering both questions. Indeed, LODQA is able to answer the first question by using data from Ecoscope dataset, whereas the second question is answerable by a triple that occur in DBpedia knowledge base.

6.4.1 The process offered by `LODQA`

`LODQA` is an information extraction approach that consists of three main phases *(i) Question Analysis (QA)*, *(ii) Entities Detection (ED)* and *(iii) Answer Extraction (AE)*, where each of them includes multiple steps. To make these steps more clear, Figure 6.19 shows a running example i.e., the steps for answering the factoid question "What is the population of Heraklion?".


What is the population of Heraklion?
Input
Question Analysis (QA)
QA1). Question Tokenization
Stopwords
what, is, the, population, of, Heraklion
what, is, the, a, for, by, of,...
QA2). Stopwords Removal
{population, Heraklion}
QA3). Question Type Identification
Factoid Question
Entities Detection(ED)
ED1). Named Entity (NE) identification
Heraklion, dbp:Heraklion
DBpedia Spotlight
Heraklion
NLP Stanford
ED2). Candidate URIs retrieval
dbp:Heraklion
dbp:Heraklion Prefecture
Keyword-to-URI Service
ED3). Final entity -URI extraction
Heraklion,dbp:Heraklion
AE1) Extract more useful words
population, number, people, inhabitants
Answer Ext-raction (AE)
WordNet
AE2) Candidate triples retrieval
LODsyndesis Fact Checking Service
AE3) Final answer extraction and Production
"Heraklion population is 140,730"

Subject	Predicate	Object	Prov.	Score
Heraklion	hasNumber of People	140,730	YAGO, DBpedia	0.75
Heraklion	hasNumber of People	135,200	Test	0.75
Heraklion	areaNumber	281	Freebase	0.3

Figure 6.19: The QA Process over the running example

Concerning the *Question Analysis* process, it performs question cleaning (e.g., removal of stopwords) and it identifies the question type (e.g., factoid) by exploiting heuristics, e.g., in Figure 6.19 it identified that the question is factoid and it removed the stopwords {which,is,the,of}. Regarding the *Entities Detection* step, it recognizes the entities of the question and it performs linking and disambiguation, by using both pure NLP methods and Linked Data-based methods, specifically Stanford CoreNLP [98, 149] and DBpedia

Spotlight [151], whereas it also exploits the *keyword-to-URI* service of LODsyndesis (see §6.1.1) for identifying the URI for a given keyword. In the example of Figure 6.19, LODQA found that the entity of the input question is "Heraklion".

Concerning the *Answer Extraction* step, it uses *WordNet* [155] for tackling the possible lexical gap between a given question and the answer which can be found in the underlying sources (e.g., it extracts synonyms for expanding the keywords for the given question). Then, it exploits the *Fact Checking* service of LODsyndesis (see §6.1.6.1), for retrieving candidate triples for the identified entity and a set of keywords. Afterwards, it scores each candidate triple for producing the final answer (by taking also into account the provenance), e.g., in Figure 6.19, it produced the answer "the population of Heraklion is 140,730", since we were able to verify that answer from two datasets.

More details for all the aforementioned steps can be found in [73].

6.4.2 Evaluation

We have performed a comparative evaluation over the SimpleQuestions(v2) collection [48], for evaluating the whole process by selecting randomly a set 1,000 factoid questions, where each question contained on average 7 words. The subset of the collection that was used in the experiments, is accessible online[3].

6.4.2.1 Effectiveness and Efficiency

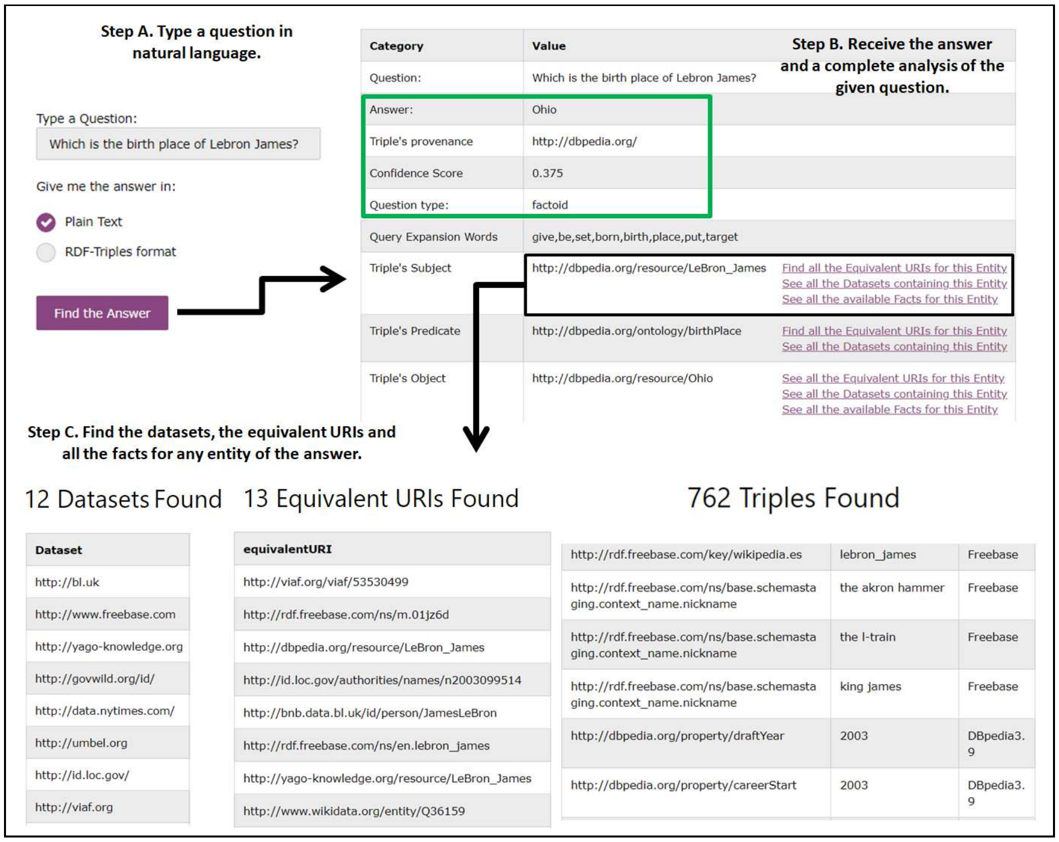
As regards effectiveness, by using all the question words expansion steps, we achieved the highest accuracy (i.e., 49%), whereas by taking into account only the questions that passes the Entities Detection Step (i.e., all the questions that we detected the correct entities), the accuracy increases (i.e., 64%). Concerning efficiency, it needs on average 5.33 seconds to answer a question. The most time consuming steps are to retrieve the candidate triples (57.2% of the required time) and to detect the entities of the question (30% of the required time). Furthermore, it is worth noting that the minimum time for answering a question was 1.6 seconds and the maximum one was 37.46. Finally, half of the questions (i.e., median value) were answered in less than 3.7 seconds. More details about the experiments can be found in [73].

6.4.2.2 The Benefits of using Multiple Datasets through LODsyndesis

LODsyndesis contains information for over 28 million entities from at least two datasets, whereas for 6.9 million entities we can retrieve information from at least three datasets. It is worth noting that there exists 28.4 million of possible questions that can be verified by more than one dataset, i.e., corresponds to simple questions answerable from at least two

[3]http://islcatalog.ics.forth.gr/tr/dataset/simplequestions-v2-1000-questions

datasets. Therefore, it is evident that by using multiple datasets, we increase the probability of answering a given question, whereas the number of verifiable questions is increased, too. Moreover, for the entities existing in at least two datasets, if we use only a single dataset (even the dataset containing the most triples for each entity), the average number of triples per entity is 17.3. On the contrary, due to the cross-dataset reasoning (i.e., computation of transitive and symmetric closure of equivalent relationships), LODsyndesis collects all the available information for each entity from all the datasets. Due to this process, the average number of triples of each of these entities highly increases (i.e., it becomes 29.3).


Step A. Type a question in natural language.
Type a Question:
Which is the birth place of Lebron James?
Give me the answer in:
Plain Text
RDF-Triples format
Find the Answer
Step B. Receive the answer and a complete analysis of the given question.
Step C. Find the datasets, the equivalent URIs and all the facts for any entity of the answer.

Category	Value	
Question:	Which is the birth place of Lebron James?	
Answer:	Ohio	
Triple's provenance	http://dbpedia.org/	
Confidence Score	0.375	
Question type:	factoid	
Query Expansion Words	give,be,set,born,birth,place,put,target	
Triple's Subject	http://dbpedia.org/resource/LeBron_James	Find all the Equivalent URIs for this Entity See all the Datasets containing this Entity See all the available Facts for this Entity
Triple's Predicate	http://dbpedia.org/ontology/birthPlace	Find all the Equivalent URIs for this Entity See all the Datasets containing this Entity
Triple's Object	http://dbpedia.org/resource/Ohio	See all the Equivalent URIs for this Entity See all the Datasets containing this Entity See all the available Facts for this Entity

12 Datasets Found

Dataset
http://bl.uk
http://www.freebase.com
http://yago-knowledge.org
http://govwild.org/id/
http://data.nytimes.com/
http://umbel.org
http://id.loc.gov/
http://viaf.org

13 Equivalent URIs Found

equivalentURI
http://viaf.org/viaf/53530499
http://rdf.freebase.com/ns/m.01jz6d
http://dbpedia.org/resource/LeBron_James
http://id.loc.gov/authorities/names/n2003099514
http://bnb.data.bl.uk/id/person/JamesLeBron
http://rdf.freebase.com/ns/en.lebron_james
http://yago-knowledge.org/resource/LeBron_James
http://www.wikidata.org/entity/Q36159

762 Triples Found

http://rdf.freebase.com/key/wikipedia.es	lebron_james	Freebase
http://rdf.freebase.com/ns/base.schemastaging.context_name.nickname	the akron hammer	Freebase
http://rdf.freebase.com/ns/base.schemastaging.context_name.nickname	the l-train	Freebase
http://rdf.freebase.com/ns/base.schemastaging.context_name.nickname	king james	Freebase
http://dbpedia.org/property/draftYear	2003	DBpedia3.9
http://dbpedia.org/property/careerStart	2003	DBpedia3.9

Figure 6.20: An example of the LODQA demo

6.4.3 Web Demo and Related Links

LODQA is currently hosted and runs in a single machine with an *i5* core, 8 GB main memory and 60 GB disk space. Although the hosting machine has a low computational power, the interaction is real time, i.e., few seconds are needed to answer a question.

In the *website* https://demos.isl.ics.forth.gr/LODQA/DemoQuestions, we offer a list of demo factoid, confirmation and definition questions, enabling the user to run questions for each of these categories. Three indicative question-answer pairs, one for each question type, are in order: (Which was the birth place of Socrates?, Athens), (Is Nintendo located in Kyoto?, Yes!), (What is Parthenon?, the parthenon, a temple built in honor of athena...).

As we can see in Figure 6.20, for the question "Which is the birth place of Lebron James?", LODQA returns the answer (i.e., "Ohio" in this example), and also a complete analysis for the question (see the right part of Figure 6.20). Specifically, except for the short answer, one can find more information about its *provenance*, its *type* and its *confidence score*. Moreover, it returns the triple (in RDF format) where we found the question, whereas one can explore more information for each entity which is part of the triple, by using the services of LODsyndesis which were introduced in this chapter. Finally, a *tutorial video* is accessible in `https://youtu.be/bSbKLlQBukk`.

6.5 Epilogue

In this chapter, we presented LODsyndesis which is a suite of services over the datasets of the entire Linked Open Data Cloud. LODsyndesis exploits the methods, algorithms and semantics-aware indexes described in this dissertation for offering advanced services for tasks A-E. In particular, we described how to exploit these services either through our HTML webpage or by using the provided REST API, for over 400 million entities and 2 billion triples from 400 real RDF datasets. As a product, to the best of our knowledge, the indexes of LODsyndesis (which is the suite of services that exploits the aforementioned indexes and measurements) constitute the biggest knowledge graph of LOD that is complete with respect to the inferable equivalence relations.

Moreover, we introduced two *Data Enrichment* tools that can be exploited for several Machine-Learning tasks, i.e., the tools LODsyndesis$_{\mathcal{ML}}$ and LODVec. Concerning LODsyndesis$_{\mathcal{ML}}$, it exploits several Linked Datasets and provides a configurable tool for creating automatically features for any set of entities by sending SPARQL queries. Regarding LODVec, it exploits the indexes of LODsyndesis, for offering configurable options for creating URI sequences and embeddings through *word2vec* algorithm from hundreds of datasets. By using these two tools which exploit several datasets simultaneously, we identified an increase in the accuracy of predictions for several Machine-Learning tasks (particularly for classification and regression tasks).

Finally, we presented in brief the ongoing work LODQA, which is an approach that exploit hundreds of datasets simultaneously through LODsyndesis, and follows a variety of methods, for answering a question expressed in natural language.

Chapter 7
Conclusion

7.1 Synopsis of Contributions

In this thesis, we focused on proposing novel methods, supported by special indexes and algorithms, for covering the needs of several tasks that are related to the connectivity and semantic integration among large number of Linked Datasets. In particular, the objective was to provide services for fulfilling the requirements and needs of scientists of any scientific field (or even simple users) for the following tasks: (a) for obtaining complete information about one particular entity, (b) for assessing the connectivity between any subset of datasets and monitoring their evolution over time, (c) for discovering the most relevant datasets for a given task or for a given dataset, (d) for enriching the content of a given datasets from several ones, and (e) for improving and estimating the quality of data.

At first, we identified the major challenges for achieving the above targets. In particular, we identified that even seemingly simple tasks, such as finding all the available information for an entity, presupposes knowledge of all datasets, and it is a requirement to perform cross-dataset identity reasoning, i.e., to compute the symmetric and transitive closure of the equivalence relationships that exist among entities and schemas. Moreover, we identified that Dataset Discovery is also a big challenge, since current approaches exploit only the metadata of datasets, without taking into consideration their contents.

For analyzing the work that has been done in the area in the past decade, we provided a survey about Large Scale Semantic Integration of Linked Data. In particular, we described the Linked Data Ecosystem, by identifying the main actors and use cases and the main difficulties of the data integration process. Since the integration landscape is wide and complex, we factorized the process according to five dimensions, we discussed the spectrum of binding types that are used for achieving integration, as well as the kinds of evidence that are usually used for creating such bindings. Subsequently we used these dimensions for describing the work that has been done in the area, by giving emphasis on approaches for large scale semantic integration. We identified and discussed 18 in number tools for linked data integration and 15 services that are available for large in number RDF datasets, including `LODsyndesis`, which is the suite of services that has been developed in this dissertation. By taking into account the above analysis, we described the placement of this dissertation in the Data Integration Landscape, we identified the major research gaps and the research directions by introducing several motivating scenarios for each task, and we described the novelty of our approach, i.e., this is the first work providing cross-dataset identity reasoning for both schema and instance elements at large scale and content-based Dataset Discovery metrics among any subset of datasets.

For tackling the challenges, we introduced algorithms for performing cross-dataset identity reasoning among different datasets, i.e., computing the transitive and symmetric closure of equivalence relationships, either by using a single machine or a cluster of machines. Moreover, we introduced parallel MapReduce methods [69] for creating five semantics-aware indexes (for entities, triples, classes, properties and literals), for enabling the fast access to all the available information of any entity. Concerning efficiency, we needed only 45 seconds to compute the transitive and symmetric closure for 13 million `owl:sameAs` relationships by using an algorithm for a single machine. Since that algorithm was not scalable, we used a variation of Hash-to-Min algorithm [203] for computing the closure for 44 millions of `owl:sameAs` relationships in less than 10 minutes by using 96 virtual machines. Moreover, by using the aforementioned cluster of machines, we constructed the five semantics-aware indexes for 2 billion triples (derived from 400 RDF datasets) in 81 minutes, whereas we reported experiments which revealed the scalability of the parallel algorithms.

For offering content-based Dataset Discovery among any subset of datasets, we proposed several content-based metrics that are based on intersection, union and complement. We showed that for computing these metrics, it is a prerequisite to take into account the whole contents of datasets (and not only metadata) and to solve maximization problems. First, we showed that it is prohibitively expensive to computet such metrics by using either a baseline method or a SPARQL implementation. For making feasible the computation of these metrics at large scale we proposed lattice-based incremental algorithms that exploit the posting lists of the semantics-aware indexes and set-theory properties. In particular, we proposed two different traversals for computing the cardinality of intersection, i.e., a bottom-up depth first search and a top-down breadth first search traversal. On the contrary, for computing the cardinality of union, absolute complement and relative complement, we proposed bottom-up depth-first search algorithms that are based on set-theory properties and pruning and regrouping methods. Furthermore, we introduced methods for computing the metrics in parallel by splitting the lattice in small pieces.

Concerning efficiency, for all the metrics, the lattice-based incremental approaches were even more than 5000× faster than a SPARQL implementation and a baseline method which does not exploit set theory properties. Indicatively, a SPARQL implementation needs even on average one minute to perform the metrics for a pair of datasets, whereas the proposed incremental approaches can compute the metrics for millions of subsets even in one second. As regards intersection metrics, by using a bottom-up lattice-based approach, we computed the cardinality of intersection for millions of subsets of datasets for entities, triples, and literals in a few seconds (4, 0.9 and 3.6 seconds, respectively), whereas it was feasible to perform the measurements even for billions of combinations of datasets in less than 10 minutes. Concerning the different methods, we achieved even a $21\times$ speedup by using a bottom-up approach instead of a top-down and a $5.61\times$ speedup by combining the bottom-up approach with a pruning method.

Regarding union and complement metrics, by combining the bottom-up lattice-based incremental approach with pruning and regrouping methods, we observed up to 97× speedup, and we managed to compute the cardinality of the union for millions of subsets of datasets, for entities, triples, and literals in 1.3, 1.4 and 3.2 seconds, respectively. Moreover, we computed the cardinality of the absolute and relative complement of entities for a single dataset, with respect to millions of subsets of datasets, in 1.1 and 0.8 seconds. By using the parallel version of the bottom-up lattice-based algorithm for intersection, we managed to split the lattice in small slices in a very efficient way, i.e., in the experiments, where we used a cluster of 64 machines, we managed each machine to compute the metrics for almost the same number of lattice nodes. With this algorithm, we computed the cardinality of intersection for 1 billion nodes in less than 1 minute and for one trillion nodes in approximately 6 hours.

Moreover, we reported connectivity analytics by exploiting the semantics-aware indexes and the content-based metrics over 400 Linked Datasets. Regarding `owl:sameAs` relationships, the computation of cross-identity reasoning yielded more than 73 million new `owl:sameAs` relationships, and this increases the connectivity of the datasets: over 30% of the 9,025 of connected pairs of datasets that share entities are due to these new relationships. However, the measurements also reveal the "sparsity" of the current LOD cloud and make evident the need for better connectivity. Only a small percentage of entities, triples and literals exist in two or more datasets, i.e., 7.73%, 11.88% and 0.7% respectively, whereas the percentages are even worse for the schema elements (i.e., less than 1% of properties and classes are used in at least two datasets). Moreover, only 1.9% of real world entities are part of three or more datasets. Concerning the connectivity among different datasets, there exists 11.3% of pairs and 1.24% of triad of datasets that share at least one entity. On the contrary, a high percentage of pairs (i.e., 78%) and triads (i.e., 46.44%) of datasets share literals, but only a small percentage of pairs (i.e., 5.59%) and triads (i.e., 0.33%) have triples in common. Concerning the most popular datasets, which are highly connected with other datasets, they belong predominantly to cross-domain (i.e., DBpedia, Freebase, YAGO and Wikidata), and secondarily to the publications domain. It is worth noting that these four cross-domain datasets share over 2.9 million entities and 3.4 million literals.

For exploiting the semantics-aware indexes and the content-based measurements, we presented the LODsyndesis suite of services, which includes services for all the tasks presented in this dissertation. Indeed, we described how one can use these services for over 400 million entities and 2 billion triples by accessing our webpage (which also offers a REST API). Moreover, we presented two tools which use the semantics-aware indexes and LODsyndesis services for improving the execution of Machine-Learning tasks, i.e., LODsyndesis$_{\mathcal{ML}}$ and LODVec. By using these tools, we created features and word embeddings by using several datasets simultaneously, and we managed to improve the accuracy of predictions for several Machine-Learning tasks, (e.g., predicting whether a movie is popular or not, according to user ratings). Furthermore, we introduced an ongoing work (i.e., LODQA) for enabling question answering over hundreds of Linked Datasets.

Finally, we should notice that the proposed methods indexes and algorithms are applicable for any given RDF dataset of any domain, whereas as a product, to the best of our knowledge, LODsyndesis constitutes the biggest knowledge graph of LOD that is complete with respect to the inferable equivalence relationships. All the services, offered by LODsyndesis, can be directly exploited as core services, for various higher level Data Integration services at large scale.

7.2 Directions for Future Work and Research

There are several aspects that are worth further work and research.

Content-Based Metrics for Complex Queries

The proposed lattice-based algorithms can compute the lattice of measurements for millions of subsets of datasets in a few seconds. However, in the worst case where more datasets (i.e., even hundreds of datasets) should be pre-selected for a given query (e.g., for entities that occur in a large number of datasets), it is prohibitively expensive to compute the metrics even with the proposed methods. One potential solution is to pre-select the top-K datasets according to some heuristics (such as their cardinality value $|cov(D_i, F)|$, their domain, etc.) and to apply the algorithms for these K datasets, however, it is a problem that requires further research. Moreover, we would like to be able to propose methods for answering complex queries that combine two or more metrics, e.g., queries such as "Give me the K datasets that contain at least 1,000 common entities with my dataset, and these datasets increase the degree of the entities of my dataset at least 2 times". Such a query requires the computation of both intersection and complement metrics.

Providing LOD Scale Analytics for a Dataset On-The-Fly

The propose Dataset Discovery metrics require that a dataset D_i should be indexed, for providing measurements for it. However, we plan to propose methods for receiving as input any RDF dataset D_i on-the-fly, and to offer as an output LOD Scale analytics about it at real time. Such a process could require to update the indexes at real time and to perform content-based measurements by using the updated posting lists that contain the dataset D_i. By offering such a service, a publisher would be able to identify whether his/her dataset a) is connected to others, b) offers unique information comparing to the other datasets, and c) contains common facts with other datasets. Finally, one research direction which is related to this task, is to create indexes and provide such LOD scale analytics by using as input streaming data.

Quality of Equivalence Relationships
We plan to work on automatic ways for evaluating/improving the quality of equivalence relationships among different datasets (which are required for creating the semantics-aware indexes and for computing the metrics), since it is time-consuming to evaluate them in a semi-manually way.

Exploitation of Indexes for Several Tasks
One possible future direction could be to exploit the constructed indexes for several tasks, such as Keyword Search, Instance and Schema Matching, Entity Recognition and others.

Embeddings over Large Number of Datasets
As a future work for `LODVec` we plan (a) to create longer URI sequences, (b) to create vectors through other models, such as *GloVe* [195], and (c) to apply graph-based techniques, such as [61, 253]. Moreover, our goal is to train the whole knowledge graph for retrieving the top-K similar entities for any given entity quickly.

Evaluation Collections and Reproducible Results
There is a lack of large collections and challenges for evaluating the quality of automated methods for data integration and consequently for reporting reproducible and comparative results. Such collections could be useful for having a systematic method for testing the quality of the *overall* integration processes, and measuring the human effort that is required. That would allow the community to investigate and evaluate novel ideas.

Novel Processes for Large Scale Data Integration
It would be interesting to propose novel automatic processes, by supporting both bottom-up and top-down schema mapping, automatic contextualization of resources, and to combine inductive and deductive methods, which could include machine-learning based methods, that will be applicable to thousands of datasets.

Comparing the Efficiency of different frameworks
It would be interesting to execute the algorithms by using not only *MapReduce* [69] , but also other popular big data frameworks, such as *SPARK* [259], and to perform experiments for comparing the efficiency of these frameworks for each of the proposed algorithms.

SPARQL Extension for Computing Content-Based Metrics Among any Subset of Datasets
One future direction could be a SPARQL extension, which will be applicable for computing content-based metrics among any subset of datasets. Such an implementation could require the construction of dedicated indexes and algorithms for the given problem.

Question Answering over Large Number of Datasets
We plan to improve `LODQA` system for making it capable of returning responses to more complex questions, i.e., list questions or questions that require combining paths of triples, by exploiting the indexes of `LODsyndesis`.

Bibliography

[1] Bibliotheque nationale de France. http://www.bnf.fr. Accessed: 2020-01-10.

[2] Blazegraph database. http://www.blazegraph.com/. Accessed: 2020-01-10.

[3] Common format and MIME type for Comma-Separated Values (CSV) Files. http://tools.ietf.org/html/rfc4180. Accessed: 2020-01-10.

[4] DBpedia. http://dbpedia.org. Accessed: 2020-01-10.

[5] Deutschen National Bibliothek. http://www.dnb.de. Accessed: 2020-01-10.

[6] Extensible markup language (XML). http://www.w3.org/XML/, note = Accessed: 2020-01-10.

[7] Food and Agriculture Organization of the United Nations. http://www.fao.org/. Accessed: 2020-01-10.

[8] Freebase. http://developers.google.com/freebase/. Accessed: 2020-01-10.

[9] GeoNames geographical database. http://www.geonames.org/. Accessed: 2020-01-10.

[10] ImageSnippets. http://www.imagesnippets.com/. Accessed: 2020-01-10.

[11] The javascript object notation (JSON) data interchange format. http://buildbot.tools.ietf.org/html/rfc7158. Accessed: 2020-01-10.

[12] JRC-Names. http://ec.europa.eu/jrc/en/language-technologies/jrc-names. Accessed: 2020-01-10.

[13] Library of Congress Linked Data Service. http://id.loc.gov/. Accessed: 2020-01-10.

[14] Linked Movie Data Base (LMDB). http://linkedmdb.org/. Accessed: 2020-01-10.

[15] MATLAB - MathWorks. https://www.mathworks.com/products/matlab.html.

[16] Okeanos cloud computing service. http://okeanos.grnet.gr. Accessed: 2020-01-10.

[17] OpenCyc. `http://www.cyc.com/opencyc/`. Accessed: 2020-01-10.

[18] Radatana. `http://data.bibsys.no/`. Accessed: 2020-01-10.

[19] RDF 1.1 N-Quads. `http://www.w3.org/TR/n-quads/`, note = Accessed: 2020-01-10.

[20] RDF 1.1 N-Triples. `http://www.w3.org/TR/n-triples/`, note = Accessed: 2020-01-10.

[21] The British Library. `http://bl.uk`. Accessed: 2020-01-10.

[22] The Virtual International Authority File. `http://viaf.org`. Accessed: 2020-01-10.

[23] VIVO Scripps. `http://vivo.scripps.edu/`. Accessed: 2020-01-10.

[24] VIVO Wustl. `http://old.datahub.io/dataset/vivo-wustl`. Accessed: 2020-01-10.

[25] Wikidata. `http://www.wikidata.org`. Accessed: 2020-01-10.

[26] YAGO. `http://yago-knowledge.org`. Accessed: 2020-01-10.

[27] Alberto Abello, Oscar Romero, Torben Bach Pedersen, Rafael Berlanga, Victoria Nebot, Maria Jose Aramburu, and Alkis Simitsis. Using semantic web technologies for exploratory OLAP: a survey. *IEEE TKDE*, 27(2):571–588, 2015.

[28] Maribel Acosta, Elena Simperl, Fabian Flöck, and Maria-Esther Vidal. Enhancing answer completeness of SPARQL queries via crowdsourcing. *Web Semantics: Science, Services and Agents on the World Wide Web*, 45:41–62, 2017.

[29] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. ANAPSID: an adaptive query processing engine for SPARQL endpoints. In *International Semantic Web Conference*, pages 18–34. Springer, 2011.

[30] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Fabian Flöck, and Jens Lehmann. Detecting Linked Data quality issues via crowdsourcing: A DBpedia study. *Semantic Web*, (Preprint):1–33, 2016.

[31] 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.

[32] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer, 2nd Edition*. The MIT Press, 2008.

[33] Ciro Baron Neto, Kay Müller, Martin Brümmer, Dimitris Kontokostas, and Sebastian Hellmann. LODVader: An Interface to LOD Visualization, Analytics and DiscovERy in Real-time. In *Proceedings of WWW*, pages 163–166, 2016.

[34] M. Ben Ellefi, Z. Bellahsene, J. G. Breslin, E. Demidova, S. Dietze, J. Szymański, and K. Todorov. RDF dataset profiling–a survey of features, methods, vocabularies and applications. *Semantic Web*, (Preprint):1–29, 2018.

[35] Sonia Bergamaschi, Silvana Castano, and Maurizio Vincini. Semantic integration of semistructured and structured data sources. *ACM Sigmod Record*, 28(1):54–59, 1999.

[36] Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. DIANE Publishing Company, 2001.

[37] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[38] Abraham Bernstein, James Hendler, and Natalya Noy. A new look at the semantic web. *Commun. ACM*, 59(9):35–37, 2016.

[39] 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.

[40] Nikos Bikakis, Chrisa Tsinaraki, Nektarios Gioldasis, Ioannis Stavrakantonakis, and Stavros Christodoulakis. The XML and semantic web worlds: technologies, interoperability and integration: a survey of the state of the art. In *Semantic Hyper/Multimedia Adaptation*, pages 319–360. Springer, 2013.

[41] Stefan Bischof, Christoph Martin, Axel Polleres, and Patrik Schneider. Collecting, integrating, enriching and republishing open city data as Linked Data. In *International Semantic Web Conference*, pages 57–75. Springer, 2015.

[42] 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.

[43] Christian Bizer and Andreas Schultz. The R2R framework: Publishing and discovering mappings on the web. In *Proceedings of the First International Conference on Consuming Linked Data*, pages 97–108, 2010.

[44] Christian Bizer, Andreas Schultz, David Ruiz, and Carlos R Rivero. Benchmarking the performance of Linked Data translation systems. In *Linked Data on the Web (LDOW)*, 2012.

[45] Christoph Böhm, Gerard de Melo, Felix Naumann, and Gerhard Weikum. LINDA: distributed web-of-data-scale entity matching. In *Proceedings of the 21st ACM CIKM Conference*, pages 2104–2108, 2012.

[46] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD Conference*, pages 1247–1250, 2008.

[47] Angela Bonifati, Fabiano Cattaneo, Stefano Ceri, Alfonso Fuggetta, Stefano Paraboschi, et al. Designing data marts for data warehouses. *ACM transactions on software engineering and methodology*, 10(4):452–483, 2001.

[48] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075, 2015.

[49] Qingqing Cai and Alexander Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *ACL*, pages 423–433, 2013.

[50] Alison Callahan, José Cruz-Toledo, Peter Ansell, and Michel Dumontier. Bio2RDF release 2: improved coverage, interoperability and provenance of life science Linked Data. In *Extended Semantic Web Conference*, pages 200–212. Springer, 2013.

[51] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering sparql queries over relational databases. *Semantic Web*, 8(3):471–487, 2017.

[52] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information integration: Conceptual modeling and reasoning support. In *Proceedings of 3rd IFCIS Conference*, pages 280–289, 1998.

[53] Diego Calvanese, Martin Giese, Dag Hovland, and Martin Rezk. Ontology-based integration of cross-linked datasets. In *International Semantic Web Conference*, pages 199–216. Springer, 2015.

[54] Diego Calvanese, Domenico Lembo, and Maurizio Lenzerini. Survey on methods for query rewriting and query answering using views. *Integrazione, Warehousing e Mining di sorgenti eterogenee*, 25, 2001.

[55] Silvana Castano, Alfio Ferrara, Stefano Montanelli, and Gaia Varese. Ontology and instance matching. In *Knowledge-driven multimedia information extraction and ontology evolution*, pages 167–195. Springer, 2011.

[56] Adriane Chapman, Elena Simperl, Laura Koesten, George Konstantinidis, Luis-Daniel Ibáñez, Emilia Kacprzak, and Paul Groth. Dataset search: a survey. *The VLDB Journal*, pages 1–22, 2019.

[57] Michelle Cheatham and Pascal Hitzler. String similarity metrics for ontology alignment. In *ISWC*, pages 294–309. Springer, 2013.

[58] Weiwei Cheng, Gjergji Kasneci, Thore Graepel, David Stern, and Ralf Herbrich. Automated feature generation from structured knowledge. In *CIKM*, pages 1395–1404. ACM, 2011.

[59] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. Entity resolution in the web of data. *Synthesis Lectures on the Semantic Web*, 5(3):1–122, 2015.

[60] Michael Cochez, Petar Ristoski, Simone Paolo Ponzetto, and Heiko Paulheim. Biased graph walks for RDF graph embeddings. In *WIMS*, page 21. ACM, 2017.

[61] Michael Cochez, Petar Ristoski, Simone Paolo Ponzetto, and Heiko Paulheim. Global RDF vector space embeddings. In *ISWC*, pages 190–207. Springer, 2017.

[62] Diego Collarana, Mikhail Galkin, Ignacio Traverso-Ribón, Christoph Lange, Maria-Esther Vidal, and Sören Auer. Semantic data integration for knowledge graph construction at query time. In *ICSC*, pages 109–116. IEEE, 2017.

[63] Diego Collarana, Mikhail Galkin, Ignacio Traverso-Ribón, Maria-Esther Vidal, Christoph Lange, and Sören Auer. MINTE: semantically integrating RDF graphs. In *Proceedings of WIMS Conference*, page 22. ACM, 2017.

[64] Gianluca Correndo, Manuel Salvadores, Ian Millard, Hugh Glaser, and Nigel Shadbolt. SPARQL query rewriting for implementing data integration over Linked Data. In *Proceedings of the 2010 EDBT/ICDT Workshops*, pages 1–11. ACM, 2010.

[65] Isabel F Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. AgreementMaker: efficient matching for large real-world schemas and ontologies. *Proceedings of the VLDB Endowment*, 2(2):1586–1589, 2009.

[66] Aba-Sah Dadzie and Matthew Rowe. Approaches to visualising Linked Data: A survey. *Semantic Web*, 2(2):89–124, 2011.

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

[68] Evangelia Daskalaki, Giorgos Flouris, Irini Fundulaki, and Tzanina Saveta. Instance matching benchmarks in the era of Linked Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 39:1–14, 2016.

[69] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[70] Jeremy Debattista, Sören Auer, and Christoph Lange. Luzzu - a methodology and framework for Linked Data quality assessment. *Journal of Data and Information Quality (JDIQ)*, 8(1):4, 2016.

[71] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the WWW*, pages 469–478. ACM, 2012.

[72] Stefan Dietze, Nilamadhaba Mohapatra, Vasileios Iosifidis, Asif Ekbal, and Pavlos Fafalios. Time-aware and corpus-specific entity relatedness. pages 33–39, 2018.

[73] Eleftherios Dimitrakis, Konstantinos Sgontzos, Michalis Mountantonakis, and Yannis Tzitzikas. Enabling efficient question answering over hundreds of linked datasets. In *International Workshop on Information Search, Integration, and Personalization*, pages 3–17. Springer, 2019.

[74] 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 CIKM*, pages 652–659. ACM, 2004.

[75] Li Ding, Vassilios Peristeras, and Michael Hausenblas. Linked open government data. *IEEE Intelligent systems*, 27(3):11–15, 2012.

[76] Renata Queiroz Dividino, Thomas Gottron, Ansgar Scherp, and Gerd Gröner. From changes to dynamics: Dynamics analysis of linked open data sources. In *PROFILES@ ESWC*, 2014.

[77] Warith Eddine Djeddi and Mohamed Tarek Khadir. A novel approach using context-based measure for matching large scale ontologies. In *International Conference on DaWaK*, pages 320–331. Springer, 2014.

[78] Martin Doerr. The CIDOC conceptual reference module: an ontological approach to semantic interoperability of metadata. *AI magazine*, 24(3):75, 2003.

[79] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of SIGKDD*, pages 601–610. ACM, 2014.

[80] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Data fusion: resolving conflicts from multiple sources. In *Handbook of Data Quality*, pages 293–318. Springer, 2013.

[81] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. *Proceedings of the VLDB Endowment*, 7(10):881–892, 2014.

[82] Xin Luna Dong and Felix Naumann. Data fusion: resolving data conflicts for integration. *Proceedings of the VLDB Endowment*, 2(2):1654–1655, 2009.

[83] Xin Luna Dong, Barna Saha, and Divesh Srivastava. Less is more: Selecting sources wisely for integration. In *Proceedings of the VLDB Endowment*, volume 6, pages 37–48. VLDB Endowment, 2012.

[84] Xin Luna Dong and Divesh Srivastava. Big data integration. *Synthesis Lectures on Data Management*, 7(1):1–198, 2015.

[85] Vasilis Efthymiou, Kostas Stefanidis, and Vassilis Christophides. Minoan ER: Progressive entity resolution in the web of data. In *19th International Conference on Extending Database Technology*, 2016.

[86] Khadija M Elbedweihy, Stuart N Wrigley, Paul Clough, and Fabio Ciravegna. An overview of semantic search evaluation initiatives. *Web Semantics: Science, Services and Agents on the World Wide Web*, 30:82–105, 2015.

[87] Mohamed Ben Ellefi, Zohra Bellahsene, Stefan Dietze, and Konstantin Todorov. Dataset recommendation for data linking: an intensional approach. In *International Semantic Web Conference*, pages 36–51. Springer, 2016.

[88] Kemele M Endris, Mikhail Galkin, Ioanna Lytra, Mohamed Nadjib Mami, Maria-Esther Vidal, and Sören Auer. MULDER: Querying the Linked Data web by bridging RDF molecule templates. In *DEXA*, pages 3–18. Springer, 2017.

[89] Orri Erling and Ivan Mikhailov. Virtuoso: RDF support in a native RDBMS. In *Semantic Web Information Management*, pages 501–519. Springer, 2010.

[90] Ivan Ermilov, Sören Auer, and Claus Stadler. Csv2rdf: User-driven csv to RDF mass conversion framework. In *ISEM*, volume 13, pages 4–6, 2013.

[91] Ivan Ermilov, Jens Lehmann, Michael Martin, and Sören Auer. LODStats: The data web census dataset. In *International Semantic Web Conference*, pages 38–46. Springer, 2016.

[92] Pavlos Fafalios, Manolis Baritakis, and Yannis Tzitzikas. Configuring named entity extraction through real-time exploitation of Linked Data. In *WIMS14*, page 10. ACM, 2014.

[93] Pavlos Fafalios, Thanos Yannakis, and Yannis Tzitzikas. Querying the web of data with SPARQL-LD. In *TPDL*, pages 175–187. Springer, 2016.

[94] Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked Data quality of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO. *Semantic Web*, (Preprint):1–53, 2016.

[95] Michael Färber, Basil Ell, Carsten Menne, and Achim Rettinger. A comparative survey of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO. *Semantic Web Journal*, 2015.

[96] Javier D Fernández, Wouter Beek, Miguel A Martínez-Prieto, and Mario Arias. LOD-a-lot. In *ISWC*, pages 75–83. Springer, 2017.

[97] Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary RDF representation for publication and exchange (HDT). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:22–41, 2013.

[98] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[99] Valeria Fionda, Giuseppe Pirrò, and Claudio Gutierrez. NautiLOD: A formal language for the web of data graph. *ACM Transactions on the Web (TWEB)*, 9(1):5, 2015.

[100] Annika Flemming. Quality characteristics of Linked Data publishing datasources. *Master's thesis, Humboldt-Universität of Berlin*, 2010.

[101] Giorgos Flouris, Dimitris Manakanatas, Haridimos Kondylakis, Dimitris Plexousakis, and Grigoris Antoniou. Ontology change: Classification and survey. *The Knowledge Engineering Review*, 23(2):117–152, 2008.

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

[103] Daniel Gerber, Diego Esteves, Jens Lehmann, Lorenz Bühmann, Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, and René Speck. DeFacto-Temporal and multilingual deep fact validation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 35:85–101, 2015.

[104] José M Giménez-Garcıa, Harsh Thakkar, and Antoine Zimmermann. Assessing trust with pagerank in the web of data. In *3rd International Workshop on Dataset PROFIling and fEderated Search for Linked Data*, 2016.

[105] Hugh Glaser, Afraz Jaffri, and Ian Millard. Managing co-reference on the semantic web. 2009.

[106] Olaf Görlitz and Steffen Staab. Splendid: SPARQL endpoint federation exploiting VoID descriptions. In *Proceedings of COLD*, pages 13–24, 2011.

[107] Paul Groth, Antonis Loizou, Alasdair JG Gray, Carole Goble, Lee Harland, and Steve Pettifer. Api-centric Linked Data integration: The open phacts discovery platform case study. *Web Semantics: Science, Services and Agents on the World Wide Web*, 29:12–18, 2014.

[108] Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken. Challenges of source selection in the WoD. In *International Semantic Web Conference*, pages 313–328. Springer, 2017.

[109] 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.

[110] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):158–182, 2005.

[111] Arben Hajra and Klaus Tochtermann. Linking science: approaches for linking scientific publications across different LOD repositories. *IJMSO*, 12(2-3):124–141, 2017.

[112] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: the teenage years. In *Proceedings of VLDB*, pages 9–16, 2006.

[113] Andreas Harth, Craig A Knoblock, Steffen Stadtmüller, Rudi Studer, and Pedro Szekely. On-the-fly integration of static and dynamic Linked Data. In *Proceedings of the COLD Workshop*, pages 1613–0073. Citeseer, 2013.

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

[115] Olaf Hartig. An overview on execution strategies for Linked Data queries. *Datenbank-Spektrum*, 13(2):89–99, 2013.

[116] Olaf Hartig. SQUIN: a traversal based query execution system for the web of Linked Data. In *SIGMOD*, pages 1081–1084. ACM, 2013.

[117] Olaf Hartig and Jun Zhao. Using web data provenance for quality assessment. In *Proceedings of SWPM*, pages 29–34, 2009.

[118] 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.

[119] 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.

[120] 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.

[121] Katja Hose, Ralf Schenkel, Martin Theobald, and Gerhard Weikum. Database foundations for scalable RDF processing. In *Proceedings of the 7th international conference on Reasoning web: semantic technologies for the web of data*, pages 202–249. Springer-Verlag, 2011.

[122] Filip Ilievski, Wouter Beek, Marieke van Erp, Laurens Rietveld, and Stefan Schlobach. LOTUS: Adaptive text search for big Linked Data. In *International Semantic Web Conference*, pages 470–485. Springer, 2016.

[123] Emrah Inan and Oguz Dikenelli. Effect of enriched ontology structures on RDF embedding-based entity linking. In *MTSR Conference*, pages 15–24. Springer, 2017.

[124] Antoine Isaac and Bernhard Haslhofer. Europeana linked open data–data. europeana. eu. *Semantic Web*, 4(3):291–297, 2013.

[125] Krzysztof Janowicz, Pascal Hitzler, Benjamin Adams, Dave Kolas, II Vardeman, et al. Five stars of Linked Data vocabulary use. *Semantic Web*, 5(3):173–176, 2014.

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

[127] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. Logmap: Logic-based and scalable ontology matching. In *International Semantic Web Conference*, pages 273–288. Springer, 2011.

[128] Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O' Byrne, and Aidan Hogan. Observing Linked Data dynamics. In *Extended Semantic Web Conference*, pages 213–227. Springer, 2013.

[129] Maulik R Kamdar and Mark A Musen. PhLeGrA: Graph analytics in pharmacology over the web of life sciences linked open data. In *Proceedings of World Wide Web Conference*, pages 321–329, 2017.

[130] U Kang, Charalampos E Tsourakakis, and Christos Faloutsos. PEGASUS: mining peta-scale graphs. *Knowledge and information systems*, 27(2):303–325, 2011.

[131] Zoi Kaoudi and Ioana Manolescu. RDF in the clouds: a survey. *The VLDB Journal*, 24(1):67–91, 2015.

[132] Michael Hausenblas Keith Alexander, Richard Cyganiak and Jun Zhao. Describing Linked Datasets with the VoID vocabulary, W3C interest group note, 2011.

[133] Shahan Khatchadourian and Mariano P Consens. Exploring RDF usage and interlinking in the linked open data cloud using ExpLOD. In *LDOW*, 2010.

[134] Michel Klein. Combining and relating ontologies: an analysis of problems and solutions. In *OIS@ IJCAI*, 2001.

[135] 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 *WISE*, pages 815–816. Springer, 2012.

[136] Craig A Knoblock and Pedro Szekely. Exploiting semantics for big data integration. *AI Magazine*, 36(1), 2015.

[137] Haridimos Kondylakis and Dimitris Plexousakis. Ontology evolution without tears. *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:42–58, 2013.

[138] 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 WWW*, pages 747–758. ACM, 2014.

[139] Dimitris Kontokostas, Amrapali Zaveri, Sören Auer, and Jens Lehmann. Triplecheckmate: A tool for crowdsourcing the quality assessment of Linked Data. pages 265–272. Springer, 2013.

[140] Christina Lantzaki, Panagiotis Papadakos, Anastasia Analyti, and Yannis Tzitzikas. Radius-aware approximate blank node matching using signatures. *Knowledge and Information Systems*, pages 1–38, 2016.

[141] Wangchao Le, Songyun Duan, Anastasios Kementsietsidis, Feifei Li, and Min Wang. Rewriting queries on SPARQL views. In *Proceedings of the 20th international conference on World wide web*, pages 655–664. ACM, 2011.

[142] Jens Lehmann, Robert Isele, et al. DBpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[143] Luiz André P Paes Leme, Giseli Rabello Lopes, Bernardo Pereira Nunes, Marco Antonio Casanova, and Stefan Dietze. Identifying candidate datasets for data interlinking. In *ICWE*, pages 354–366. Springer, 2013.

[144] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[145] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. Truth finding on the deep web: Is the problem solved? In *Proceedings of the VLDB Endowment*, volume 6, pages 97–108. VLDB Endowment, 2012.

[146] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI conference*, 2015.

[147] Vanessa Lopez, Christina Unger, Philipp Cimiano, and Enrico Motta. Evaluating question answering over Linked Data. *Web Semantics Science Services And Agents On The World Wide Web*, 21:3–13, 2013.

[148] Konstantinos Makris, Nikos Bikakis, Nektarios Gioldasis, and Stavros Christodoulakis. SPARQL-RW: transparent query access over mapped RDF data sources. In *Proceedings of the 15th EDBT Conference*, pages 610–613. ACM, 2012.

[149] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60, 2014.

[150] John P McCrae and Philipp Cimiano. Linghub: a Linked Data based portal supporting the discovery of language resources. In *SEMANTiCS (Posters & Demos)*, volume 1481, pages 88–91. 2015.

[151] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. DBpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems*, pages 1–8. ACM, 2011.

[152] 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.

[153] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[154] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[155] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[156] Nikos Minadakis, Yannis Marketakis, Haridimos Kondylakis, Giorgos Flouris, Maria Theodoridou, Martin Doerr, and Gerald de Jong. X3ML framework: An effective suite for supporting data mappings. In *Workshop for Extending, Mapping and Focusing the CRMb•"co-located with TPDL*, 2015.

[157] Paolo Missier, Khalid Belhajjame, and James Cheney. The W3C PROV family of specifications for modelling provenance metadata. In *Proceedings of the 16th International EDBT Conference*, pages 773–776. ACM, 2013.

[158] Gabriela Montoya. *Answering SPARQL Queries using Views.* PhD thesis, Université de Nantes, 2016.

[159] Gabriela Montoya, Luis-Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Semlav: Local-as-view mediation for SPARQL queries. In *TLDKS XIII*, pages 33–58. Springer, 2014.

[160] Camilo Morales, Diego Collarana, Maria-Esther Vidal, and Sören Auer. MateTee: a semantic similarity metric based on translation embeddings for knowledge graphs. In *ICWE*, pages 246–263. Springer, 2017.

[161] Luc Moreau, Bertram Ludäscher, Ilkay Altintas, Roger S Barga, Shawn Bowers, Steven Callahan, George Chin Jr, Ben Clifford, Shirley Cohen, Sarah Cohen-Boulakia, et al. The first provenance challenge. *Concurrency and computation: practice and experience*, 20(5):409–418, 2008.

[162] 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.

[163] Michalis Mountantonakis, Nikos Minadakis, Yannis Marketakis, Pavlos Fafalios, and Yannis Tzitzikas. Quantifying the connectivity of a semantic warehouse and understanding its evolution over time. *IJSWIS*, 12(3):27–78, 2016.

[164] Michalis Mountantonakis, Nikos Minadakis, Yannis Marketakis, Pavlos Fafalios, and Yannis Tzitzikas. Connectivity, value, and evolution of a semantic warehouse. In *Innovations, Developments, and Applications of Semantic Web and Information Systems*, pages 1–31. IGI Global, 2018.

[165] Michalis Mountantonakis and Yannis Tzitzikas. On measuring the lattice of commonalities among several Linked Datasets. *Proceedings of the VLDB Endowment*, 9(12):1101–1112, 2016.

[166] Michalis Mountantonakis and Yannis Tzitzikas. How Linked Data can aid machine learning-based tasks. In *International Conference on Theory and Practice of Digital Libraries*, pages 155–168. Springer, 2017.

[167] Michalis Mountantonakis and Yannis Tzitzikas. Services for large scale semantic integration of data. *ERCIM NEWS*, 25:57–58, 2017.

[168] Michalis Mountantonakis and Yannis Tzitzikas. High performance methods for linked open data connectivity analytics. *Information*, 9(6), 2018.

[169] Michalis Mountantonakis and Yannis Tzitzikas. LODsyndesis: Global scale knowledge services. *Heritage*, 1(2):335–348, 2018.

[170] Michalis Mountantonakis and Yannis Tzitzikas. LODsyndesis: the biggest knowledge graph of the linked open data cloud that includes all inferred equivalence relationships. *ERCIM NEWS*, (114):43–44, 2018.

[171] Michalis Mountantonakis and Yannis Tzitzikas. Scalable methods for measuring the connectivity and quality of large numbers of Linked Datasets. *Journal of Data and Information Quality (JDIQ)*, 9(3):15, 2018.

[172] Michalis Mountantonakis and Yannis Tzitzikas. Knowledge graph embeddings over hundreds of linked datasets. In *Research Conference on Metadata and Semantics Research*, pages 150–162. Springer, 2019.

[173] Michalis Mountantonakis and Yannis Tzitzikas. Large-scale semantic integration of Linked Data: A survey. *ACM Computing Surveys (CSUR)*, 52(5):103, 2019.

[174] Michalis Mountantonakis and Yannis Tzitzikas. Content-based union and complement metrics for dataset search over RDF knowledge graphs (accepted for publication). *JDIQ*, 2020.

[175] Jindrich Mynarz and Vojtech Svátek. Towards a benchmark for LOD-enhanced knowledge discovery from structured data. In *KNOW@ LOD*, pages 41–48, 2013.

[176] Venkata Narasimha, Pavan Kappara, Ryutaro Ichise, and OP Vyas. LiDDM: a data mining system for Linked Data. In *Workshop on LDOW*, volume 813, 2011.

[177] Yaroslav Nechaev, Francesco Corcoglioniti, and Claudio Giuliano. Type prediction combining linked open data and social media. In *CIKM*, pages 1033–1042. ACM, 2018.

[178] Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm. A survey of current link discovery frameworks. *Semantic Web*, (Preprint):1–18, 2015.

[179] 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.

[180] DuyHoa Ngo, Zohra Bellahsene, and R Coletta. Yam++-a combination of graph matching and machine learning approach to ontology alignment task. *Journal of Web Semantics*, 16, 2012.

[181] Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes-a time-efficient approach for large-scale link discovery on the web of data. In *IJCAI*, pages 2312–2317, 2011.

[182] Andriy Nikolov, Mathieu d' Aquin, and Enrico Motta. What should i link to? identifying relevant sources and classes for data linking. In *Joint International Semantic Technology Conference*, pages 284–299. Springer, 2011.

[183] Natalya F Noy. Semantic integration: a survey of ontology-based approaches. *ACM Sigmod Record*, 33(4):65–70, 2004.

[184] Natasha Noy, Matthew Burgess, and Dan Brickley. Google dataset search: Building a search engine for datasets in an open web ecosystem. 2019.

[185] Peter Ochieng and Swaib Kyanda. A statistically-based ontology matching tool. *Distributed and Parallel Databases*, pages 1–23, 2017.

[186] Damla Oguz, Belgin Ergenc, Shaoyi Yin, Oguz Dikenelli, and Abdelkader Hameurlain. Federated query processing on Linked Data: a qualitative survey and open challenges. *The Knowledge Engineering Review*, 30(5):545–563, 2015.

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

[188] Lorena Otero-Cerdeira, Francisco J Rodríguez-Martínez, and Alma Gómez-Rodríguez. Ontology matching: A literature review. *Expert Systems with Applications*, 42(2):949–971, 2015.

[189] Maria-Evangelia Papadaki, Panagiotis Papadakos, Michalis Mountantonakis, and Yannis Tzitzikas. An interactive 3D visualization for the LOD Cloud. In *International Workshop on Big Data Visual Exploration and Analytics (BigVis'2018 at EDBT/ICDT 2018)*, 2018.

[190] Jeff Pasternack and Dan Roth. Latent credibility analysis. In *Proceedings of WWW*, pages 1009–1020. ACM, 2013.

[191] Norman W Paton, Klitos Christodoulou, Alvaro AA Fernandes, Bijan Parsia, and Cornelia Hedeler. Pay-as-you-go data integration for Linked Data: opportunities, challenges and architectures. In *Proceedings of the 4th International Workshop on Semantic Web Information Management*, page 3. ACM, 2012.

[192] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.

[193] Heiko Paulheim and Johannes Fümkranz. Unsupervised generation of data mining features from linked open data. In *Proceedings of WIMS 2012*, page 31. ACM, 2012.

[194] Peng Peng, Lei Zou, M Tamer Özsu, Lei Chen, and Dongyan Zhao. Processing SPARQL queries over distributed RDF graphs. *The VLDB Journal*, 25(2):243–268, 2016.

[195] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of EMNLP conference*, pages 1532–1543, 2014.

[196] Maureen Pennock and Michael Day. Managing and preserving digital collections at the british library. *Managing Digital Cultural Objects: Analysis, discovery and retrieval*, page 111, 2016.

[197] Petar Petrovski, Volha Bryl, and Christian Bizer. Integrating product data from websites offering microdata markup. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 1299–1304. ACM, 2014.

[198] Emmanuel Pietriga, Hande Gözükan, Caroline Appert, Marie Destandau, Šejla Čebirić, François Goasdoué, and Ioana Manolescu. Browsing Linked Data catalogs with LODAtlas. In *ISWC*, pages 137–153. Springer, 2018.

[199] Axel Polleres, Maulik R Kamdar, Javier D Fernandez, Tania Tudorache, and Mark A Musen. A more decentralized vision for Linked Data. In *DeSemWeb2018 Workshop*, 2018.

[200] Eric Prud' Hommeaux, Andy Seaborne, et al. SPARQL query language for RDF. *W3C recommendation*, 2008.

[201] Bastian Quilitz and Ulf Leser. Querying distributed RDF data sources with SPARQL. In *ESWC*, pages 524–538. Springer, 2008.

[202] Erhard Rahm. The case for holistic data integration. In *East European Conference on ADBIS*, pages 11–27. Springer, 2016.

[203] Vibhor Rastogi, Ashwin Machanavajjhala, Laukik Chitnis, and Anish Das Sarma. Finding connected components in map-reduce in logarithmic rounds. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 50–61. IEEE, 2013.

[204] Thomas Rebele, Fabian Suchanek, Johannes Hoffart, Joanna Biega, Erdal Kuzey, and Gerhard Weikum. Yago: a multilingual knowledge base from wikipedia, wordnet, and geonames. In *ISWC*, pages 177–185. Springer, 2016.

[205] Theodoros Rekatsinas, Xin Luna Dong, Lise Getoor, and Divesh Srivastava. Finding quality in quantity: The challenge of discovering valuable sources for integration. In *CIDR*, 2015.

[206] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, Inc., 2008.

[207] Laurens Rietveld, Wouter Beek, and Stefan Schlobach. LOD lab: Experiments at lod scale. In *ISWC*, pages 339–355. Springer, 2015.

[208] Petar Ristoski, Christian Bizer, and Heiko Paulheim. Mining the web of Linked Data with rapidminer. *Web Semantics: Science, Services and Agents on the World Wide Web*, 35:142–151, 2015.

[209] Petar Ristoski, Gerben Klaas Dirk de Vries, and Heiko Paulheim. A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In *International Semantic Web Conference*, pages 186–194. Springer, 2016.

[210] Petar Ristoski and Heiko Paulheim. Rdf2vec: RDF graph embeddings for data mining. In *ISWC*, pages 498–514. Springer, 2016.

[211] Petar Ristoski, Jessica Rosati, Tommaso Di Noia, Renato De Leone, and Heiko Paulheim. RDF2Vec: RDF graph embeddings and their applications. *Semantic Web*, 10(4):721–752, 2019.

[212] Muhammad Saleem, Ali Hasnain, and Axel-Cyrille Ngonga Ngomo. Largerdfbench: a billion triples benchmark for SPARQL endpoint federation. *Journal of Web Semantics*, 2018.

[213] Muhammad Saleem, Yasar Khan, Ali Hasnain, Ivan Ermilov, and Axel-Cyrille Ngonga Ngomo. A fine-grained evaluation of SPARQL endpoint federation systems. *Semantic Web*, (Preprint):1–26, 2015.

[214] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation. In *European Semantic Web Conference*, pages 176–191. Springer, 2014.

[215] Muhammad Saleem, Axel-Cyrille Ngonga Ngomo, Josiane Xavier Parreira, Helena F Deus, and Manfred Hauswirth. Daw: Duplicate-aware federated query processing over the web of data. In *ISWC*, pages 574–590. Springer, 2013.

[216] Muhammad Saleem, Shanmukha S Padmanabhuni, Axel-Cyrille Ngonga Ngomo, Aftab Iqbal, Jonas S Almeida, Stefan Decker, and Helena F Deus. TopFed: TCGA tailored federated query processing and linking to lod. *J. Biomedical Semantics*, 5(1):1, 2014.

[217] Cristina Sarasua, Elena Simperl, and Natalya F Noy. Crowdmap: Crowdsourcing ontology alignment with microtasks. In *International Semantic Web Conference*, pages 525–541. Springer, 2012.

[218] Cristina Sarasua, Steffen Staab, and Matthias Thimm. Methods for intrinsic evaluation of links in the web of data. In *European Semantic Web Conference*, pages 68–84. Springer, 2017.

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

[220] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. Fedbench: A benchmark suite for federated semantic data query processing. *The Semantic Web–ISWC 2011*, pages 585–600, 2011.

[221] Falk Scholer, Diane Kelly, and Ben Carterette. Information retrieval evaluation using test collections. *Information Retrieval Journal*, 19(3):225–229, 2016.

[222] Andreas Schultz, Andrea Matteini, Robert Isele, Pablo N Mendes, Christian Bizer, and Christian Becker. LDIF-a framework for large-scale Linked Data integration. In *21st International WWW Conference, Developers Track*, 2012.

[223] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on Linked Data. In *International Semantic Web Conference*, pages 601–616. Springer, 2011.

[224] Juan F Sequeda, Marcelo Arenas, and Daniel P Miranker. On directly mapping relational databases to RDF and OWL. In *Proceedings of the 21st international conference on World Wide Web*, pages 649–658. ACM, 2012.

[225] Juan F Sequeda, Syed Hamid Tirmizi, Oscar Corcho, and Daniel P Miranker. Survey of directly mapping SQL databases to the semantic web. *The Knowledge Engineering Review*, 26(4):445–486, 2011.

[226] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE intelligent systems*, 21(3):96–101, 2006.

[227] Pavel Shvaiko and Jérôme Euzenat. Ontology matching: state of the art and future challenges. *IEEE TKDE*, 25(1):158–176, 2013.

[228] Behjat Siddiquie, Shiv N Vitaladevuni, and Larry S Davis. Combining multiple kernels for efficient image classification. In *2009 Workshop on Applications of Computer Vision (WACV)*, pages 1–8. IEEE, 2009.

[229] Karen Smith-Yoshimura. Analysis of 2018 international Linked Data survey for implementers. *Code{4}lib Journal*, (42), 2018.

[230] Stefano Spaccapietra and Christine Parent. View integration: A step forward in solving structural conflicts. *IEEE transactions on Knowledge and data Engineering*, 6(2):258–274, 1994.

[231] Stefano Spaccapietra, Christine Parent, and Yann Dupont. Model independent assertions for integration of heterogeneous schemas. *The VLDB Journal*, 1(1):81–126, 1992.

[232] Fabian M Suchanek, Serge Abiteboul, and Pierre Senellart. Paris: Probabilistic alignment of relations, instances, and schema. *Proceedings of the VLDB Endowment*, 5(3):157–168, 2011.

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

[234] Ignacio Traverso, Maria-Esther Vidal, Benedikt Kämpgen, and York Sure-Vetter. GADES: A graph-based semantic similarity measure. In *Proceedings of the 12th International Conference on Semantic Systems*, pages 101–104. ACM, 2016.

[235] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference*, pages 210–218. Springer, 2017.

[236] Yannis Tzitzikas et al. Integrating heterogeneous and distributed information about marine species through a top level ontology. In *MTSR*, pages 289–301. Springer, 2013.

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

[238] Yannis Tzitzikas, Christina Lantzaki, and Dimitris Zeginis. Blank node matching and RDF/s comparison functions. In *International Semantic Web Conference*, pages 591–607. Springer, 2012.

[239] Yannis Tzitzikas, Nikos Manolis, and Panagiotis Papadakos. Faceted exploration of RDF/s datasets: a survey. *JIIS*, pages 1–36, 2016.

[240] Yannis Tzitzikas, Yannis Marketakis, et al. Towards a global record of stocks and fisheries. In *8th International Conference on Information and Communication Technologies in Agriculture, Food and Environment (HAICTA 2017), Chania, Greece*, pages 328–340, 2017.

[241] Yannis Tzitzikas and Carlo Meghini. Ostensive automatic schema mapping for taxonomy-based peer-to-peer systems. In *CIA*, pages 78–92. Springer, 2003.

[242] Yannis Tzitzikas, Nikos Minadakis, Yannis Marketakis, Pavlos Fafalios, Carlo Allocca, and Michalis Mountantonakis. Quantifying the connectivity of a semantic warehouse. In *EDBT/ICDT Workshops*, pages 249–256, 2014.

[243] Yannis Tzitzikas, Nikos Minadakis, Yannis Marketakis, Pavlos Fafalios, Carlo Allocca, Michalis Mountantonakis, and Ioanna Zidianaki. Matware: Constructing and exploiting domain specific warehouses by aggregating semantic data. In *ESWC*, pages 721–736. Springer, 2014.

[244] Yannis Tzitzikas, Nicolas Spyratos, and Panos Constantopoulos. Mediators over taxonomy-based information sources. *The VLDB Journal*, 14(1):112–136, 2005.

[245] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank Van Harmelen, and Henri Bal. WEBPIE: A web-scale parallel inference engine using MapReduce. *Web Semantics: Science, Services and Agents on the World Wide Web*, 10:59–75, 2012.

[246] Andre Valdestilhas, Tommaso Soru, Markus Nentwig, Edgard Marx, Muhammad Saleem, and Axel-Cyrille Ngonga Ngomo. Where is my URI? In *ESWC*, pages 671–681. Springer, 2018.

[247] Pierre-Yves Vandenbussche, Ghislain A Atemezing, María Poveda-Villalón, and Bernard Vatant. Linked open vocabularies (lov): a gateway to reusable semantic vocabularies on the web. *Semantic Web*, 8(3):437–452, 2017.

[248] Pierre-Yves Vandenbussche, Jürgen Umbrich, Luca Matteis, Aidan Hogan, and Carlos Buil-Aranda. SPARQLES: monitoring public SPARQL endpoints. *Semantic Web*, (Preprint):1–17, 2016.

[249] Panos Vassiliadis and Timos Sellis. A survey of logical models for OLAP databases. *ACM Sigmod Record*, 28(4):64–69, 1999.

[250] 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.

[251] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[252] Andreas Wagner, Peter Haase, Achim Rettinger, and Holger Lamm. Entity-based data source contextualization for searching the web of data. In *European Semantic Web Conference*, pages 25–41. Springer, 2014.

[253] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI conference on artificial intelligence*, 2014.

[254] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data*, 3, 2016.

[255] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[256] Guohui Xiao, Dag Hovland, Dimitris Bilidas, Martin Rezk, Martin Giese, and Diego Calvanese. Efficient ontology-based data integration with canonical iris. In *European Semantic Web Conference*, pages 697–713. Springer, 2018.

[257] Ramana Yerneni, Chen Li, Jeffrey Ullman, and Hector Garcia-Molina. Optimizing large join queries in mediation systems. *Database Theory*, pages 348–364, 1999.

[258] Semih Yumusak, Erdogan Dogdu, Halife Kodaz, Andreas Kamilaris, and Pierre-Yves Vandenbussche. SpEnD: Linked Data SPARQL endpoints discovery using search engines. *IEICE TRANSACTIONS on Information and Systems*, 100(4):758–767, 2017.

[259] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[260] 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 SEMANTiCS*, pages 97–104. ACM, 2013.

[261] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for Linked Data: A survey. *Semantic Web*, 7(1):63–93, 2015.

[262] Linhong Zhu, Majid Ghasemi-Gol, Pedro Szekely, Aram Galstyan, and Craig A Knoblock. Unsupervised entity resolution on multi-type graphs. In *International Semantic Web Conference*, pages 649–667. Springer, 2016.

[263] Minhaz Fahim Zibran. Chi-squared test of independence. *Department of Computer Science, University of Calgary, Alberta, Canada*, 2007.

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

Appendix A
Publications and Systems

Publications

The research activity related to this thesis has so far produced the following publications (ordered by publication date):

(1) M. Mountantonakis and Y. Tzitzikas, *On Measuring the Lattice of Commonalities Among Several Linked Datasets*, Proceedings of the VLDB Endowment (PVLDB), 2016

(2) M. Mountantonakis and Y. Tzitzikas, *How Linked Data can aid Machine Learning-based Tasks*, 21st International Conference on Theory and Practice of Digital Libraries (TPDL), (pp. 155-168), September 2017, Thessaloniki, Greece

(3) M. Mountantonakis and Y. Tzitzikas, *Scalable Methods for Measuring the Connectivity and Quality of Large Numbers of Linked Datasets*, ACM Journal of Data and Information Quality (JDIQ), 9(3), 15, 2018

(4) M. Mountantonakis and Y. Tzitzikas, *High Performance Methods for Linked Open Data Connectivity Analytics*, Information 2018, 9, 134.(Special Issue Semantics for Big Data Integration)

(5) M. Mountantonakis and Y. Tzitzikas, *LODsyndesis: Global Scale Knowledge Services*, Heritage. Open Access Journal (ISSN 2571-9408), 1(2), 335-348, MDPI.(Special Issue: On Provenance of Knowledge and Documentation: Select Papers from CIDOC 2018), 2018.

(6) M. Mountantonakis and Y. Tzitzikas, *Large Scale Semantic Integration of Linked Data: A survey*, ACM Computing Surveys, 52(5), Sept. 2019

(7) M. Mountantonakis and Y. Tzitzikas, *Knowledge Graph Embeddings over Hundreds of Linked Datasets*, 13th International Conference on Metadata and Semantics Research, Rome, Italy, October 2019

(8) M. Mountantonakis and Y. Tzitzikas, *Content-based Union and Complement Metrics for Dataset Search over RDF Knowledge Graphs*, ACM Journal of Data and Information Quality (JDIQ), 2020

In more details and with regard to the contributions of this thesis:

- In (1) we proposed algorithms for computing the cross-dataset identity closure, for constructing the semantics-aware indexes for URIs, and for computing the cardinality of intersection (through lattice-based incremental methods) among any subset of datasets by using a single machine.
- In (2) we introduced the tool $\texttt{LODsyndesis}_{\mathcal{ML}}$, which enables the creation of features for Machine-Learning based tasks by using multiple datasets.
- In (3) and (4), we introduced algorithms a) for computing in parallel the transitive and symmetric closure of equivalence relationships, b) for creating in parallel semantics-aware indexes for the whole contents of datasets, and c) for computing in parallel lattice-based measurements. Moreover, we reported connectivity analytics for a big subset of the entire LOD Cloud.
- In (5) we described the suite of services that are available through `LODsyndesis`.
- In (6) we presented the survey that includes the work that has been done in the area of Linked Data integration in the last decade.
- In (7) we described how to compute union and complement metrics by using the semantics-aware indexes and special lattice-based incremental algorithms.
- In (8) we introduced `LODVec`, which enables the creation of URI embeddings by using hundreds of datasets, simultaneously.

Other Publications

Here, we show some other related publications to this thesis (ordered by publication date):

(9) M. Mountantonakis and Y. Tzitzikas, *Services for Large Scale Semantic Integration of Data*, ERCIM News 2017 (111), October 2017

(10) M. Mountantonakis and Y. Tzitzikas, *LODsyndesis: The Biggest Knowledge Graph of the Linked Open Data Cloud that Includes all Inferred Equivalence Relationships*, ERCIM News 2018 (114), July 2018

(11) M. Mountantonakis, N. Minadakis, Y. Marketakis, P. Fafalios, Y. Tzitzikas, Connectivity, Value, and Evolution of a Semantic Warehouse. In Innovations, Developments, and Applications of Semantic Web and Information Systems (pp. 1-31). IGI Global, 2018

(12) ME Papadaki, P Papadakos, M Mountantonakis and Y Tzitzikas, *An Interactive 3D Visualization for the LOD Cloud.*, EDBT/ICDT Workshops, 100-103, 2018

(13) E. Dimitrakis, K. Sgontzos, M. Mountantonakis, and Y. Tzitzikas, *Enabling efficient question answering over hundreds of linked datasets*, International Workshop on Information Search, Integration, and Personalization. Springer, Cham, 2019.

In more details (and with regard to the contributions of this thesis):

- In (9) and (10) we introduced in brief the `LODsyndesis` services.
- In (11) we described in brief the connectivity metrics over large number of datasets
- In (12) the connectivity metrics used for offering more informative 3D visualization

for the LOD Cloud.
- In (13) we exploited several LODsyndesis services for offering Question Answering over hundreds of Linked datasets.

Systems

In the context of this thesis, the following systems were developed:

Web pages of Systems.

- LODsyndesis: http://www.ics.forth.gr/isl/LODsyndesis
- LODsyndesis$_{\mathcal{ML}}$: https://demos.isl.ics.forth.gr/lodsyndesis/LODsyndesisML
- LODVec: https://demos.isl.ics.forth.gr/lodvec/
- LODQA: https://demos.isl.ics.forth.gr/LODQA

Videos of Systems.

- LODsyndesis: https://youtu.be/UdQDgod6XME
- LODsyndesis$_{\mathcal{ML}}$: https://youtu.be/S_ILRTZarjA
- LODVec: https://youtu.be/qR9RFZVs4TY
- LODQA: https://youtu.be/bSbKLlQBukk

Appendix B
Acronyms

BFS Breadth-First Search

CC Connected Component

DFS Depth-First Search

FCO Feature Creation Operator

IRIS Internationalized Resource Identifiers

LB Lattice-Based

LOD Linked Open Data

ML Machine Learning

QA Question Answering

RDF Resource Description Framework

RWC Real World Classes

RWE Real World Entities

RWP Real World Properties

RWT Real World Triples

URI Uniform Resource Identifiers