Publishing and Consuming Linked Data

Optimizing for the Unknown

Laurens Rietveld

SIKS Dissertation Series No. 2016-04

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

VRIJE UNIVERSITEIT

Publishing and Consuming Linked Data

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. V. Subramaniam,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Exacte Wetenschappen
op woensdag 9 maart 2016 om 15.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Laurens Johannes Rietveld

geboren te Amstelveen

promotor	prof.dr F.A.H. van Harmelen
copromotoren	dr. K.S. Schlobach
	dr. R.J. Hoekstra

To Piet

ACKNOWLEDGEMENTS

It has been an interesting and enjoyable last four years. It is not often that you are allowed to work in such a warm and social environment, surrounded by smart people, and with so much freedom to do fun things.

This great experience can for a large part be contributed to my three supervisors, Frank van Harmelen, Rinke Hoekstra and Stefan Schlobach. I could not wish for a better combination of characters and skills in my supervisors. Thanks to Rinke, for his keen eye and practical advice on topics such as presentations, scientific writing, research in general, and leveraging the goals of a PhD with those of a scientific project. Thanks to Stefan, for the fun discussions, cups of coffee, and enriching my writing skills with some 'youthful arrogance'. And thanks to Frank, for both pushing me to keep my eye on the prize and giving me a lot of freedom in my research at the same time.

Wouter Beek, with whom I enjoyed sharing the office, deserves a special mention as well. I was lucky to work with Wouter, considering that our skills are largely complementary and our collaboration has proved quite productive. I am happy that we will continue collaborating after this thesis.

Regarding my thesis and its defence, I thank the committee (Sören Auer, Victor de Boer, Stefan Decker, Geert-Jan Houben and Peter Mika) for having read and accepted it.

I also thank all the people from the KRR&R and the Web & Media groups. Thanks Chris for the exhausting tennis matches, and sorry for distracting you so much. Thanks Albert for all the short walks outside. Filip, I hope you find the time to continue the nice projects that started in our room. And Archana, sorry for the workspace disruptions. Be gentle in *your* acknowledgements!

CONTENTS

1 INTRODUCTION

1.1 BACKGROUND

The World Wide Web is a huge collection of interlinked information. For us, humans, this information is readily accessible in formats we like, such as news articles rendered as text, audio broadcasts, and videos via services such as YouTube. Where human beings can understand these formats, software agents often cannot. This software has to analyze the text from the news article to extract its meaning, and analyze any audio and video files. Research related to natural language processing and image recognition is promising, but the outcome always comes with a fair margin of error.

A complementary approach to the World Wide Web is that of Linked Data, where information is represented in a machine readable format. Linked Data uses the same underpinnings as the World Wide Web: both use the Hypertext Transfer Protocol (HTTP) to access and retrieve information. In Linked Data, International Resource Identifiers (IRIs[1]) and literals denote 'things', called resources. These resources can be anything, such as geographical locations, documents, abstract concepts like 'Democracy', or numbers and strings. IRIs, literals and blank nodes are used by the Resource Description Framework[2] (RDF) to make statements, –also called triples–, about resources. A set of triples form a graph and a collection of graphs form a dataset. Triples can re-use IRIs from other datasets, creating an interlinked set of datasets called the LOD Cloud. The RDF framework and the re-use IRIs between datasets provide interoperable datasets that are easy to integrate and combine.

RDF is a data model for describing resources, without providing domain semantics. It is up to the Linked Data creators to introduce this domain schema for a given dataset. A domain schema (also known as vocabulary or ontology) defines the relationships (i.e. properties) that connect resources together, the types of resources (i.e. classes), or constraints on how these classes and properties can be combined. The domain independence of RDF provides flexibility: Linked Data creators can create a new vocabulary if it does not yet exist, they can re-use existing vocabularies, and they can generate links (e.g. `owl:sameAs`) with other vocabularies or data sources. A large range of RDF vocabularies are available for re-use, such as

1 See https://www.ietf.org/rfc/rfc3987.txt
2 See http://www.w3.org/RDF/

'Friend of a Friend' (FOAF[3]) or 'Simple Knowledge Organization System' (SKOS[4]).

To summarize, Linked Data and the RDF framework offer:

- a data model to represent semantics in a graph structure;
- a data model that is not bound to a fixed schema, thereby making datasets adaptive and flexible;
- interoperable datasets, by linking to other IRIs and vocabularies;
- distributed but interlinked datasets, called the LOD Cloud
- a platform that is built on the same architecture as the regular World Wide Web.

1.2 MOTIVATION

The Linked Data architecture enables consumption without a-priori knowledge of the schema and content, and enables publishing without knowing how a dataset is going to be used. This unknown (re)use is an intrinsic positive quality of Linked Data, but it presents problems as well. In the rest of this chapter, we discuss these problems in more detail, and show how they relate to the *unknown (re)use* of Linked Data.

Despite the overlap in architecture between Linked Data and the World Wide Web, Linked Data requires a different procedure for publishing and consuming data compared to standard web technology.

Publishing information on websites either involves writing web pages directly in the Hypertext Markup Language (HTML) or it involves using Content Management Systems (CMS) where data publishers write the web pages in free text. Web pages link to each other using anchor tags, allowing consumers to browse the internet. Web pages are served from servers, either hosted by publishers themselves, or via public services such as Twitter, Facebook, DropBox, GitHub or blog services. A simple HTML web page requires little hardware resources for hosting, where more elaborate websites are often backed by databases to store e.g. user information. The typical consumers of these websites are humans, who find websites of interest via search engines, read the contents, and browse between web pages.

This procedure differs from that of Linked Data, where software agents are considered consumers as well and where content is published in the RDF data model. Web Browsers can assume a reader with general human intelligence who can come up with fallback options whenever things go wrong. RDF tools cannot assume a reader with general human intelligence. Where web pages use anchor tags to

3 See `http://xmlns.com/foaf/spec/`
4 See `http://www.w3.org/2004/02/skos/`

link pages, Linked Data uses IRIs to link resources. And where search engines such as Google or Yahoo are used to find information on the web, this is not fully possible for Linked Data. Software agents as data consumers makes data publishing and consumption difficult, in contrast to web pages that are cheap to publish and contain information that most (human) users can read. All these differences require different tooling, which take the possibilities and expressiveness of Linked Data into account.

To illustrate the different procedures of Linked Data publishing and consumption, we identify four stakeholders:

THE LINKED DATA CREATOR, who creates a Linked Dataset. Linked Datasets are often based on data sources such as spreadsheets or relational databases. Linked Data creators first need to create or re-use a domain schema. Using this schema, Linked Data creators can convert the original data to RDF and combine or link the data to other Linked Datasets.

THE DATA PROVIDER, who publishes Linked Data. This can be either Linked *Open* data which is publicly accessible on the internet, or Linked Data for private use (e.g. within the intranet of a company). data.gov.uk –the open data portal of the British government– is an example of Linked *Open* data. The provider has to think about *how* to publish the data as well: as simple files in an RDF format, or via queryable APIs;

THE LINKED DATA DEVELOPER, who takes the data published by a data provider, and develops the end-user interfaces for consumers.

An example of such an application is the BBC website for the 2011 Olympics[5], where several Linked Data sources are combined into a single end-user interface. The developer has to interact with the data, by e.g. executing queries when the data is queryable, or by downloading and processing the dataset file;

THE LINKED DATA SCIENTIST, who analyzes and consumes Linked Data, for instance to study the performance of RDF compression algorithms, study the scalability of SPARQL endpoints, or evaluate a new Linked Data ranking algorithm. Or, the scientist might be interested in Linked Data as an object of study, and wants to analyze typical structural graph properties of Linked Datasets.

A variety of tools and services are available for the Linked Data creator, including: the Protégé [40] ontology editor, the D2R [19] server for converting relational databases to Linked Data, TabLinker [70] for converting CSV files to RDF, and services such as Linked Open Vo-

5 See `http://www.bbc.com/sport/0/olympics/2012/`

cabularies (LOV[6]) to find relevant RDF vocabularies. A similar comprehensive set of tools and methodologies is missing for the Linked Data *provider*, *developer* and *scientist* stakeholders. Therefore we consider these three stakeholders specifically.

1.3 PROBLEMS

In this section we describe typical problems experienced by each of the three stakeholders. These problems are not exclusive to the different stakeholders: Linked Data scientists may encounter the same problems as Linked Data developers. To focus our development and research, we discuss the problems in the context of the stakeholder they most strongly relate to.

1.3.1 *Problems for Linked Data Providers*

Linked Data providers can publish datasets in three ways: First, the simplest method is to host files that are a serialized RDF representation of the Linked Dataset. This method only needs a simple file web-server. However, consuming this data shifts effort to the user: to retrieve a single fact from the file, it needs to be downloaded and processed using an RDF parser. This involves downloading the complete file, processing it in a RDF parser, and extracting the information. And although this method seems straightforward, we show (See chapter 2) that even this more basic publishing method is difficult for many data providers: the majority of published files turns out not to follow standards and best practices. They are served with incorrect HTTP headers, may contain duplicate triples, are published in a corrupt compressed archive, or simply contain serialization errors. These idiosyncrasies are often not known beforehand and may require manual effort from consumers to resolve.

Problem 1 *The number of Linked Datasets that do not follow standards and best practices makes Linked Data re-use impractical.*

The second publishing method is to use dereferenceable IRIs: information about the resources denoted by the IRI is accessible via an HTTP `GET` request, and returned in a machine-readable RDF serialization format (and optionally available in a human-readable form). The content referred to can be hosted as static files on a server, or served dynamically by e.g. fetching information from a SPARQL [47] triple-store.

The third publishing method involves publishing SPARQL endpoints *directly*, as the expressivity and flexibility of the SPARQL query

6 See `http://lov.okfn.org/dataset/lov/`

language can cater to a wider variety of information needs than dereferenceable IRIs. This is particularly useful for Linked Data providers who are in the dark on their consumers' information needs. Where SPARQL can be used as backend for dereferenceable IRIs, using it as backend for other services is not uncommon: tools such as the Linked Data API[7] expose RESTful APIs on top of SPARQL endpoints, and Linked Data front-end browsers such as Pubby[8] or Brwsr[9] retrieve information from a SPARQL backend as well.

The dependency of Linked Data providers on SPARQL –either as backend or directly exposed to the public– comes at a price: the expressivity and flexibility of SPARQL makes SPARQL endpoints relatively expensive to host. Commodity hardware is often not enough for large datasets, as the server requires a fair amount of memory, CPU processing power, and disk space. This is corroborated by the low number of publicly public available SPARQL endpoints [26] (284 published via datahub.io at the time of writing).

Problem 2 *The expressivity of SPARQL demands powerful triple-stores, and makes large Linked Datasets expensive to host.*

1.3.2 *Problems for Linked Data Developers*

Where hosting SPARQL endpoints is difficult for Linked Data providers, access to these endpoints by e.g. Linked Data developers shows to be problematic as well. Access to these SPARQL endpoints is difficult, as the complexity and expressiveness of SPARQL makes it an unforgiving and difficult query language. Take for instance a Linked Data developer who would like to build an application on top of several SPARQL endpoints. This developer will run into questions such as:

- Where can I find SPARQL endpoints?
- Is this query syntactically correct?
- How do I write a particular namespace prefix?
- I am not familiar with a particular vocabulary. What vocabulary IRIs can I use?

These questions should sound familiar to anyone who wrote SPARQL queries. But no SPARQL editor exists that assists the consumer sufficiently, and provides direct feedback for answers such as these. As a result, the Linked Data developer cannot use the same query formulation environment that regular web developers are accustomed to (who use e.g. Integrated Development Environments (IDEs)).

7 See https://github.com/UKGovLD/linked-data-api
8 See http://wifo5-03.informatik.uni-mannheim.de/pubby/
9 See https://github.com/Data2Semantics/brwsr

Problem 3 *The expressivity and complexity of SPARQL makes formulating queries difficult.*

Linking to other datasets is a core principle of Linked Data that enables Linked Data developers to consume and combine data from several data sources. Locating these data sources is not as easy as it seems. For example, the Linked Data developer might be looking for datasets including geographic coordinates, or datasets describing the IRI `http://dbpedia.org/resource/Amsterdam`. Or, the developer is benchmarking a Linked Data application against Linked Datasets with different kinds of structural properties (e.g. an above-average number of links in a dataset). Even under under the assumption that all Linked Datasets are standards compliant (as identified previously in problem 1), this is a difficult task because of the following two issues:

Firstly, the decentralized and distributed nature of Linked Data makes finding datasets difficult. Services such as the DataHub[10] dataset catalog attempt to solve this problem by crowd-sourcing dataset references. But considering that the data sources are entered manually, they are prone to errors, are sometimes outdated, and only cover a limited part of Linked Data. An alternative approach is to crawl Linked Data using e.g. LDspider [58], that collects and follows dereferenceable IRIs. This approach requires considerable effort from the developer, and is limited in scope as well: only a part of Linked Open Data is published via dereferenceable IRIs.

Secondly, selecting Linked Datasets based on their structural properties is impractical, considering that the structure of datasets is often not described. Those datasets that *are* structurally described often lack provenance information about the computational procedure used to generate the meta-data. This makes it difficult to reliably compare structural properties between datasets. A service such as LODStats attempts to provide structural dataset descriptions at a larger scale, but it is based on the DataHub catalog and suffers from its incompleteness.

Problem 4 *The distributed nature of Linked Data and its corresponding meta-data makes locating and accessing Linked Data difficult.*

1.3.3 *Problems for Linked Data Scientists*

The previous problems illustrated how Linked Data providers and developers are working in the dark: data providers are faced with unpredictable information needs of consumers; developers are faced with

10 See `http://datahub.io`

unknown standards conformance, data "obfuscated" behind SPARQL endpoints, and no means for locating datasets. However, the state-of-the-art in Linked Data research methods and resources does not suffice to reduce these unknowns.

Firstly, studies related to the *use* of Linked Data are limited: the number of SPARQL query logs available via workshops such as USEWOD [16] are restricted to only 5 datasets, where we show in chapter 8 that at least 600 SPARQL endpoints are used by Linked Data consumers. In other words, Linked Data scientists are working in the dark as well, and do not have the resources to study the use of Linked Data at large.

Secondly, as we show in chapter 7, typical Linked Data research evaluates and optimizes algorithms for only a handful of datasets such as DBpedia [4], BSBM [75], DBLP [66] and only a few more. This is problematic, considering that we show (in chapters 3 and 7) the relation between algorithmic performance and structural properties of the data. These limited evaluations are not caused by unwillingness of scientists, but by the difficult process of finding and locating datasets (as referred to in problem 4).

As a result, –from both the usage and data perspective–, Linked Data research does not take the true variety into account.

Problem 5 *Linked Data research does not take the true variety of Linked Data into account*

1.4 APPROACH

The key contribution of this thesis is a number of technological advancements for building Linked Data based services. This includes enabling Linked Data publishing, consumption and research, by decreasing the hardware costs and human effort of Linked Data publishing, and to increase the utility and accessibility for Linked Data consumption. We present the contributions of this thesis in more detail below.

1.4.1 *Making Linked Data re-use more practical (Problem 1)*

Existing solutions for clean, standards compliant Linked Data are targeted towards human data creators. But 10 years of Linked Data shows that this solution does not suffice: as we described, many Linked Datasets still do not adhere to standards and best practices. Although the state-of-the-art in standards, guidelines and tools may incrementally improve the state of Linked Data, they do not offer a complete solution. For an *immediate* improvement, we present a single centralized service that re-publishes a clean version of as many

Linked Open Datasets as possible in a canonical format, and presents Linked Data providers with a cleaned and hosted version of their datasets.

Such a centralized point of entry has unique benefits over regular Linked Open Data: all datasets are accessible in one location, are all published in the same canonical serialization format, with a transparent and open-source framework behind it. As a result, consumers that are otherwise in the unknown about the syntactic quality of other people's data, can now browse, download and consume these datasets with minimum effort.

1.4.2 *Making Linked Data querying less costly (Problem 2)*

We present two orthogonal solutions that reduce the hardware costs for hosting queryable Linked Data.

First, we propose an automatic sampling method that reduces the dataset size, and thus reduces the hardware costs for hosting a SPARQL endpoint. Where sampling often implies *random* sampling, we investigate whether it is possible to find the smallest part of the data that entails as many of the answers to typical SPARQL queries as possible. One way of achieving this is what we call *informed* sampling: the Linked Data provider knows what queries the SPARQL endpoint receives, and what kind of queries it can expect. Using these SPARQL queries, the Linked Data provider would be able to select a smaller part of the dataset that answers exactly those queries. However, –as discussed previously–, Linked Data providers are often in the dark on how their SPARQL triple-store is (and will be) used. Therefore we present an *uninformed* approach, where we use the topology of the RDF graph alone to *predict* the relevance of triples.

Our second approach aims at reducing the complexity of the query language, rather than reducing the completeness of a dataset. A query language that only operates on a triple pattern level and which does not support complex operations such as joins, can decrease the hardware costs considerably. We present a deployment that combines existing work on triple pattern querying with our large-scale republications platform.

1.4.3 *Making SPARQL query formulation easier (Problem 3)*

Because the size of Linked Data and the complexity of SPARQL makes querying difficult, Linked Data consumers need a SPARQL editor with a strong focus on usability. Existing clients offer only a small selection of the features that we could offer. We present a browser-based SPARQL editor that has many features known to web developers such as syntax highlighting, syntax checking, and auto-completion functionality.

1.4.4 *Increasing accessibility of Linked (Meta-)Data (Problem 4)*

It remains a challenge to access and locate Linked Open Datasets and their corresponding meta-data: meta-data descriptions are often missing, outdated or incorrect, and finding a dataset according to structural criteria requires significant manual manual effort. Therefore, we present a structural Meta-Dataset that covers a wide range of available Linked Datasets, and includes a IRI and namespace to dataset index, dataset characteristics, and provenance information about the performed processing steps.

1.4.5 *Increasing the variety of Linked Data Research (Problem 5)*

Our approach for increasing the variety of Linked Data in Linked Data research is twofold.

Firstly, we focus on research related to Linked Data *use*. This research area is strongly restricted by the limited number of available query logs. Where these query logs are all collected from SPARQL servers, we propose to track Linked Data usage from the *client*-side using our browser-based SPARQL query editor as a measuring device for interactions with Linked Data. This allows us to monitor Linked Data use at a much wider scale than previously possible.

Secondly, we focus on increasing the variety of Linked Data research from a *data-centric* perspective. As we show in chapter 7, current research only evaluates against a handful of datasets. Therefore, we present a new evaluation paradigm that uses a command-line interface for Linked Datasets to increase the variety of datasets that scientists evaluate against. The result is a fundamentally different evaluation paradigm that enables algorithmic evaluation against hundreds of thousands of datasets.

1.5 CONTENT

The chapters in this thesis are updated versions of peer-reviewed publications. We describe the contents in more detail below.

1.5.1 *Linked Data Provider*

CHAPTER 2

> Wouter Beek, Laurens Rietveld, Hamid R Bazoobandi, Jan Wielemaker, and Stefan Schlobach. LOD Laundromat: A Uniform Way of Publishing Other People's Dirty Data. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2014

In problem 1 we described how Linked Datasets do not follow standards and best practices. Therefore, we present the infrastructure needed to obtain a full (or as-complete-as-possible) copy of available Linked Open Datasets, that are re-published as cleaned, canonical and compressed RDF data files, –called LOD Laundromat *documents*. This infrastructure, –called the LOD Laundromat–, syntactically cleans Linked Open Datasets and re-publishes these in a canonical compressed N-Triples or N-Quads format. The provenance gathered during this cleaning process is published as well. The scope of the LOD Laundromat is as broad as possible: datasets are collected by automatically crawling Linked Data catalogs, by following VoID dataset references, and by processing datasets manually added by users. The supported formats range from RDF/XML documents to web pages with embedded RDFa.

CHAPTER 3

Laurens Rietveld, Rinke Hoekstra, Stefan Schlobach, and Christophe Guéret. Structural Properties as Proxy for Semantic Relevance in RDF Graph Sampling. In *The Semantic Web–ISWC 2014*, pages 81–96. Springer, 2014

The LOD Laundromat shows the large scale –both in terms of document size and document collection size– of published Linked Data. But where the documents published by the LOD Laundromat are cheap to host as simple compressed files, they are expensive to host via a queryable API such as SPARQL triple-stores (Problem 2). In this chapter we present SampLD, an automatic sampling method targeted at maximizing answer coverage for applications that use SPARQL querying. This enables Linked Data providers to decrease their dataset size with minimum loss in recall for their SPARQL queries, and consequently decrease the hardware footprint for hosting SPARQL triple-stores. SampLD includes a number of sampling methods, each using different properties of the graph structure. We empirically show that the relevance of triples for sampling (a semantic notion) can be determined without prior knowledge of the queries, but can be purely based on similar structural graph properties. Experiments show a significantly higher recall of topology based sampling methods over random and naive baseline approaches (e.g. up to 90% for Open-BioMed at a sample size of 6%).

CHAPTER 4

Laurens Rietveld, Ruben Verborgh, Wouter Beek, Miel Vander Sande, and Stefan Schlobach. Linked Data-as-a-Service: The Semantic Web Redeployed. In *Proceedings of the Extended Semantic Web Conference (ESWC)*. Springer, 2015

In this chapter we present an alternative approach that reduces the hosting hardware costs (Problem 2) by reducing query complexity. We do so by extending the LOD Laundromat, and by combining it with a low-cost server-side interface called 'Triple Pattern Fragments' [106]. This, combined with a Triple Pattern Fragment API [105] that provides HTTP access to the HDT files, results in a low cost Linked Data API that consumes a fraction in memory and processing power compared to SPARQL triple-stores. We deploy the Triple Pattern Fragment API on the LOD Laundromat, and in doing so we bridge the gap between the web of downloadable data files and the web of live queryable data. This approach retains the same benefit that SPARQL has: Linked Data providers are able to publish data without knowing how it will be used beforehand. The result is a repeatable, low-cost, open-source data publication process, that decreases the cost for hosting queryable Linked Data.

1.5.2 *Linked Data Developer*

CHAPTER 5

Laurens Rietveld and Rinke Hoekstra. The YASGUI Family of SPARQL Clients. *Semantic Web Journal*, 2015

SPARQL remains the de-facto standard for Linked Data querying, and it provides an alternative for those Linked Data providers that do not want to sacrifice query complexity by using Triple Pattern Fragments. Because the size and complexity of Linked Data and its technology stack makes it difficult to query (Problem 3), access to Linked Data via SPARQL could be greatly facilitated if it were supported by a tool with a strong focus on usability. We improve the state-of-the-art by presenting 'Yet Another SPARQL Graphical User Interface' (YASGUI), a SPARQL query editor that provides Linked Data developers with a robust, feature-rich SPARQL editor, that has seen a large uptake in the community.

CHAPTER 6

Laurens Rietveld, Wouter Beek, Stefan Schlobach, and Rinke Hoekstra. Meta-Data for a lot of LOD. *Semantic Web Journal*, To be published. `http://semantic-web-journal.net/content/meta-data-lot-lod`

Because structural descriptions of datasets are often missing or incomparable between datasets (Problem 4), we extend the LOD Laundromat with a structural Meta-Dataset, that includes a IRI and namespace to dataset index, dataset characteristics, and provenance information about the performed processing steps. This makes it a particularly useful Meta-Dataset for locating datasets based on structural and syntactic characteristics, enabling data comparison, analytics, and global studies of the Web of Data.

1.5.3 *Linked Data Scientist*

CHAPTER 7

> Laurens Rietveld, Wouter Beek, and Stefan Schlobach. LOD Lab: Experiments at LOD Scale. In *Proceedings of the International Semantic Web Conference (ISWC)*. Springer, 2015

and

> Wouter Beek and Laurens Rietveld. Frank: The LOD Cloud at your Fingertips. In *Developers Workshop , Extended Semantic Web Conference (ESWC)*, 2015

There are no resources that facilitate Linked Data research in the large (Problem 5). Therefore, in this chapter we present LOD Lab, a fundamentally different evaluation paradigm that enables algorithmic evaluation against hundreds of thousands of datasets. We show that existing research only evaluates against a handful of datasets, and we present an alternative approach that uses *Frank*, an interface for Linked Data documents that connects to the large collection of files from the LOD Laundromat. We illustrate the viability of the LOD Lab approach by rerunning experiments from three recent Semantic Web research publications. We show that simply rerunning existing experiments within this new evaluation paradigm brings up interesting research questions as to how algorithmic performance relates to (structural) properties of the data.

CHAPTER 8

> Laurens Rietveld and Rinke Hoekstra. YASGUI: Feeling the Pulse of Linked Data. In *Knowledge Engineering and Knowledge Management*, pages 441–452. Springer, 2014

Existing studies related to the use of Linked Data are restricted by the low number of available query logs (Problem 5). In this chapter we propose to track Linked Data usage at the *client* side, using the YASGUI query editor from chapter 5 as a measuring device for interactions with Linked Data. We show how the queries collected by YASGUI enable us to investigate usage patterns that are difficult to measure otherwise. For example, we can determine what part of Linked Open Data is actually used, what part is open or closed, the efficiency and complexity of queries, how they differ from server-side logs, and how these results relate to commonly used dataset statistics.

LINKED DATA PROVIDER

2 THE LOD LAUNDROMAT

In this chapter we describe the infrastructure needed to obtain a full (or as-complete-as-possible) copy of the entire LOD Cloud, that presents Linked Data providers with a cleaned and hosted version of their datasets. We present the LOD Laundromat which crawls and cleans many (649,834) Linked Data documents that are often syntactically incorrect. We republish these documents as compressed RDF data files, creating a uniform and centralized point of entry to a collection of cleaned siblings of existing datasets. Next to the re-*publishing of documents, the LOD Laundromat enables* ***direct*** *publishing as well via a Dropbox plugin.*

This chapter is an updated version of:

Wouter Beek, Laurens Rietveld, Hamid R Bazoobandi, Jan Wielemaker, and Stefan Schlobach. LOD Laundromat: A Uniform Way of Publishing Other People's Dirty Data. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2014

Contributions to this paper:

I co-created the LOD Laundromat with Wouter Beek.

2.1 INTRODUCTION

Uptake of Linked Open Data (LOD) has seen a tremendous growth over the last decade. Due to the inherently heterogeneous nature of interlinked datasets that come from very different sources, LOD is not only a fertile environment for innovative data (re)use, but also for mistakes and incompatibilities [55, 56].

PROBLEM Existing solutions for cleaning Semantic Web (SW) data (standards, guidelines, tools) are targeted towards human data providers. This means they can (and do) choose not to use them. A large part of the Linked Data Cloud does not meet even a core set of data publishing guidelines [55, 56]. Moreover, datasets that are *clean* at creation, can deteriorate over time. Therefore, much of LOD is still difficult to use today, mostly because of mistakes for which solutions exist. This poses an unnecessary impediment to the (re)use of LOD.

Such stains in datasets not only degrade a dataset's own quality, but also the quality of other datasets that link to it (e.g. by using `owl:sameAs`). There is thus an incentive to clean stains in LOD that goes beyond that of the original dataset providers.

CONTRIBUTIONS This chapter presents the LOD Laundromat, which takes action by targeting the data directly. By cleaning dirty LOD without any human intervention, LOD Laundromat is able to make very large amounts of LOD more easily available for further processing. The collection of cleaned documents[1] that LOD Laundromat produces are standards- and guidelines-compliant siblings of existing, idiosyncratic datasets. The service enables Linked Data providers to publish clean datasets via LOD Laundromat and provides Linked Data consumers with a valuable centralized and clean data source.

The data-oriented approach of LOD Laundromat is complementary to existing efforts, since it is preferable that someday the original dataset is cleaned by its own maintainers. However, we believe that until that day, our complementary approach is necessary to make LOD succeed while the momentum is still there. LOD Laundromat is unlike any of the existing initiatives towards realizing standards-compliant LOD in each of the following three ways:

1. The **scale** at which clean data is made available: LOD Laundromat comprises hundreds of thousands of data files, and billions of triples.
2. The **speed** at which data is cleaned and made available: LOD Laundromat cleans about a billion triples a day and makes them immediately available online.
3. The **level of automation**. LOD Laundromat automates the entire data processing pipeline, from dataset collection to serialization in a standards-compliant canonical format that enables easy reuse.

LOD Laundromat implements existing standards in such a way that the resultant data documents are specifically geared towards easy reuse by further tooling. This includes simplifying certain aspects of LOD that often cause problems in practice, such as blank nodes, and significantly reducing the complexity for post-processors to parse the data, e.g., through a syntax that is regular expression-friendly and by producing lexicographically sorted files.

The LOD Laundromat is available at http://lodlaundromat.org. The collection of datasets that it comprises is continuously being extended. Linked Data providers can add new seed points to the LOD Basket by using a Web form, an HTTP GET request or by submitting files via DropBox. The fully automated LOD Washing Machine takes seed points from the LOD Basket and cleans them. Cleaned datasets are disseminated in the LOD Wardrobe. Human data consumers are

1 We distinguish between the term Linked Dataset (a set of triples) and a LOD Laundromat document (a single Linked Data file). One dataset may consist of several documents, depending on the publisher's (often implicit) definition.

able to navigate a large collection of high-quality documents. Machine processors are able to easily load very large amounts of real-world data, by selecting clean data documents through a SPARQL query.

CONTENT This chapter is organized as follows: section 2.2 gives an overview of related work. Section 2.3 specifies the requirements we pose for clean and useful data, and explores alternative approaches towards collecting large amounts of Linked Data. Section 2.4 details the major operationalization decisions that allow the data cleaning process to be fully automated. Section 2.5 elaborates on the way in which LOD Laundromat makes data available for further processing. We conclude in section 2.6.

2.2 RELATED WORK

This section first discusses standards and best practices with respect to Linked Data publishing. Secondly, it discusses existing Linked Data collections and crawlers. Finally, we discuss available Linked Data catalogs and their advantages and disadvantages.

2.2.1 *Standards*

The VoID standard [1] is a vocabulary for formal descriptions of datasets. It supports general metadata such as the homepage of a dataset, access metadata (e.g. which protocols are available), possible links to other datasets, as well as structural metadata: structural metadata includes exemplary resources and statistics such as the number of triples, properties and classes.

Bio2RDF presents a collection of dataset metrics that extends the structural metadata of the VoID description, and provides more detail (e.g. the number of unique objects linked from each predicate).

While such standards are useful from both the data publisher and the data consumer perspective, uptake of VoID is limited.[2] Additionally, from a data consumer perspective, the issue of findability through fully automated means is not resolved.

A number of observations and statistics related to Linked Data publishing best practices are presented in [33], such as the type and number of encountered errors when parsing Linked Open Datasets.

In addition, [56] have analyzed over a billion triples from 4 million crawled RDF/XML documents. [56] shows that on average 15.7% of the RDF nodes in a dataset are blank nodes. Furthermore, it shows that most Linked Data is not fully standards-compliant, corroborating the need for sanitizing Linked Data. However, this study is purely ob-

2 An overview of VoID descriptions that can be found by automated means, is given by the SPARQL Endpoint Status service: http://sparqles.ai.wu.ac.at/

servational, and the accessed data is not made available in a cleaned form.

2.2.2 *Data Crawlers*

Sindice [79] is a Semantic Web indexer. The main question Sindice tries to address, is where and how to find statements about certain resources. It does so by crawling Linked Data resources, including RDF, RDFa and Microformats, although large RDF datasets are imported on a per-instance and manual opt-in basis. Sindice maintains a large cache of this data, and provides access through a user interface and API. Public access to the raw data crawler by Sindice is not available, nor is access through SPARQL, restricting the usefulness of Sindice for Semantic Web and Big Data research. Built on top of Sindice, Sig.ma [104] is an explorative interactive tool, that enables Linked Data discovery. Similar to Sindice, Sig.ma provides an extensive user interface, as well as API access. Even though this service can be quite useful for data exploration, the actual, raw data is not accessible for further processing.

Contrary to Sig.ma and Sindice, data from the Billion Triple Challenge[3] (BTC) 2012 are publicly available and are often used in Big Data research. The BTC dataset is crawled via the DataHub dataset catalog[4], and consists of 1.4 billion triples. It includes large RDF datasets, as well as data in RDFa and Microformats. However, this dataset is not a complete crawl of the Linked Open Data cloud, and includes syntactically invalid files. Additionally, the latest version of this dataset dates back to 2012.

Freebase [22] publishes 1.9 billion triples, taken from manual user input and existing RDF and Microformat datasets. Access to Freebase is possible through an API, through a (non-SPARQL) structured query language, and as a complete dump of N-Triples. However, these dumps include many non-conformant, syntactically incorrect triples. To give a concrete example, the data file that is the dereference of the Freebase concept 'Monkey'[5] visually appears to contain hundreds of triples, but a state-of-the-art standards-conformant parser such as Raptor[6] only extracts 30 triples. Additionally, knowing *which* datasets are included in Freebase, and finding these particular datasets, is not trivial.

LODCache[7], provided by OpenLink, takes a similar crawling approach as Freebase does. However, it does not make the meta-data of the crawling process available, and does not provide clean versions

3 `http://km.aifb.kit.edu/projects/btc-2012/`
4 `http://datahub.io/`
5 `http://rdf.freebase.com/ns/m.08pbxl`
6 Tested with version 2.0.9, `http://librdf.org/raptor/rapper.html`
7 `http://lod.openlinksw.com/`

of the crawled datasets. LODCache does have a SPARQL endpoint, as well as features such as entity IRI and label lookup.

The Open Data Communities service[8] is the UK Department for Communities and Local Government's official Linked Open Data site. Their datasets are published as data dumps, and are accessible through SPARQL and API calls. Although this service supports a broad selection of protocols for accessing the data, the number of datasets is limited and restricted to domain of open government data.

Finally, DyLDO [60] is a long-term experiment to monitor the dynamics of a core set of 80 thousand Linked Data documents on a weekly basis. Each week's crawl is published as an N-Quads file. This work provides interesting insight in how Linked Data evolves over time. However, it is not possible to easily select the triples from a *single* dataset, and not all datasets belonging to the LOD cloud are included. Another form of incompleteness stems from the fact that the crawl is based on IRI dereferences: this does not guarantee that a dataset is included in its entirety (see section 2.3).

2.2.3 *Portals*

Several Linked Data portals exist that try to improve the findability of Linked Datasets. The Datahub lists a large set of RDF datasets and SPARQL endpoints. This catalog is updated manually, and there is no direct connection to the data: all metadata comes from user input. This increases the risk of outdated dataset descriptions[9] and missing or incorrect metadata. vocab.cc [98] builds on top of the BTC dataset. At the time of writing, it provides a list of 422 vocabularies. Access to these vocabularies is possible through SPARQL and an API. This service increases the ease of finding and re-using existing vocabularies. It has the same incompleteness properties that the BTC has, and does not (intend to) include instance data.

2.3 CONTEXT

Due to the points mentioned above, the poor data quality of the LOD cloud requires considerable effort from SW scientists and developers to consume. In practice, this means that LOD is less effectively (re)used than it should and could be. We first enumerate the requirements that we pose on clean datasets in order to be easily (re)usable (section 2.3.1). We then compare three approaches towards collecting LOD, and evaluate each with respect to the completeness of their results (section 2.3.2).

8 http://opendatacommunities.org/
9 For example, DBpedia links to version 3.5.1 instead of 3.9: http://datahub.io/dataset/dbpedia (12 May 2014)

2.3.1 *Requirements*

Besides the obvious requirements of being syntactically correct and standards-compliant, we also pose additional requirements for how SW datasets should be serialized and disseminated. We enumerate these additional requirements, and briefly explain why they result in data that is more useful for Big Data researchers and LOD developers in practice.

EASY GRAMMAR We want LOD to be disseminated in such a way that it is easy to handle by subsequent processors. These subsequent processors are often non-RDF tools, such as Pig [77], grep, sed, and the like. Such easy post-processing is guaranteed by adherence to a uniform data format that can be safely parsed in an unambiguous way, e.g. by being able to extract triples and terms with one simple regular expression.

SPEED We want to allow tools to process LOD in a speedy way. Parsing of data documents may be slow due to the use of inefficient serialization formats (e.g., RDF/XML, RDFa), the occurrence of large numbers of duplicate triples, or the presence of syntax errors that necessitate a parser to come up with fallback options.

QUANTITY We want to make available a large number of data documents (tens of thousands) and triples (billions), to cover a large parts of the LOD cloud.

COMBINE We want to make it easy to combine data documents, e.g., splitting a single document into multiple ones, or appending multiple documents into a single one. This is important for load balancing in large-scale processing, since the distribution of triples across data documents is otherwise very uneven.

STREAMING We want to support streamed processing of triples, in such a way that the streamed processor does not have to perform additional bookkeeping on the processed data, e.g. having to check for statements that were already observed earlier.

COMPLETENESS The data must be a complete representation of the input dataset, to the extent that the original dataset is standards-compliant.

2.3.2 *Dataset completeness*

The first problem that we come across when collecting large amounts of LOD, is that it is difficult to claim completeness. Since there are alternative approaches towards collecting large volumes of LOD, we give an overview of the incompleteness issues that arise for each of those alternatives. At the moment, three options exist for collecting large volumes of LOD:

1. Crawling resources
2. Querying endpoints
3. Downloading datadumps

Resource crawlers try to dereference IRIs to retrieve LOD. This approach has the following unknowns:

1. Datasets that do not contain dereferenceable IRIs are ignored. In [55], 7.2% of the crawled IRIs were not dereferenceable.
2. For IRIs that can be dereferenced, back-links are not included in 50% of the datasets [56]. As a consequence of this, even datasets that contain dereferenceable IRIs exclusively can still have parts that cannot be reached by a crawler.
3. Even for datasets that have only dereferenceable IRIs that include back-links, the crawler can never be certain that the entire dataset has been crawled (e.g. if the dataset contains more than one component).

Querying endpoints is another way to collect large volumes of LOD. The disadvantages of this approach are:

1. Datasets that do not have a query endpoint are ignored. While hundreds of SPARQL endpoints are known to exist today, there are at least thousands of Linked Datasets.
2. Datasets that have a custom API and/or that require an API key in order to send queries, are not generally accessible and require either appropriation to a specific API or the creation of an account in order to receive a custom key.
3. For practical reasons, otherwise standards-compliant SPARQL endpoints put restrictions on either the number of triples that can be retrieved or the number of rows that can be involved in a sort operation that is required for paginated retrieval. For instance, Virtuoso, an often used triple store, by default limits both the result set size and the number of rows within a sorting operation. This makes dataset retrieval incomplete.
4. Existing LOD observatories show that SPARQL endpoints are often inaccessible[10] [26]. This may be a result of the fact that keeping a SPARQL endpoint up and running requires considerably more resources than hosting a Web document.

10 See http://sparqles.okfn.org/

Downloading data dumps is the third approach to collecting large volumes of LOD. Its main disadvantage is that (parts of) datasets that are not available are ignored.
With the LOD Laundromat we want to clean existing datasets, not create a new dataset that is a collection of parts coming from different datasets (like BTC, for instance). For most datasets for which a SPARQL endpoint exists, we are also able to find a datadump version. We therefore believe that downloading datadumps is the best approach for collecting large numbers of data documents for cleaning.

2.4 IMPLEMENTATION: THE LOD WASHING MACHINE

In the previous section we described the requirements that we believe Linked Datasets should fulfill in order to be more useful in practice. We also explained why we have chosen to download datadumps in order to guarantee the best completeness. Here, we will make the aforementioned requirements concrete in such a way that they can be automatically applied to dirty Linked Datasets. The part of the LOD Laundromat that performs automated data cleaning is called the LOD Washing Machine.[11]

STEP A: COLLECT URLS THAT DENOTE DATASET DUMPS Before we start laundrying data, we need some dirty data to fill our LOD Basket. The LOD Washing Machine does not completely automate the search for the initial seed points for collecting LOD. The are four reasons for this. Firstly, catalogs that collect metadata descriptions must be accessed by website-specific APIs. Secondly, standards-compliant metadata descriptions are stored at multiple locations and cannot always be found by Web search operations that can be automated. Thirdly, metadata descriptions of datasets, whether standards-compliant or catalog-specific, are often outdated (e.g., pointing to an old server) or incomplete. Finally, many datasets are not described anywhere and require someone to know the server location where the data is currently stored.

For these reasons, the LOD Washing Machine relies on catalog-specific scripts that collect such seed URLs for washing. This scripts uses e.g. the CKAN API[12], that provides access to the datasets described in the Datahub. This means that URLs that are not included in a LOD catalog or portal are less likely to be washed by the LOD Washing Machine. In addition, we have added several seed points by hand for datasets that we know reside at specific server locations. Anyone can queue washing jobs by adding such seed URLs to the LOD Basket through the LOD Laundromat Website. The LOD Laun-

11 Code available at `https://github.com/LODLaundry/LOD-Washing-Machine`.
12 `http://ckan.org/`

dromat provides DropBox integration well: users can publish Linked Data dumps via their DropBox account. These dumps are then republished as clean data dumps by LOD Laundromat.

Some URL strings – e.g., values for the "URL" property in a catalog – do not parse according to the RFC 3986 grammar.[13] Some URL strings are parsed as IRIs but not as URLs, mostly because of unescaped spaces. Some URL strings parse per RFC 3986, but have no IANA-registered scheme[14], or the `file` scheme which is host-specific and cannot be used for downloading. The LOD Washing Machine uses only URLs that parse per RFC 3986 (after IRI-to-URL conversion) and that have an IANA-registered scheme that is not host-specific.

STEP B: CONNECT TO THE HOSTING SERVER When processing the list of URLs from the previous step, we must be careful with URLs that contain the same authority part, since they are likely to reside at the same server. Since some servers do not accept multiple (near) simultaneous requests from the same IP, we must avoid parallel processing of such URLs. The LOD Washing Machine therefore groups URLs with the same authority and makes sure they get processed in sequence, not in parallel. This is implemented by handling URLs with the same authority in a single thread.

At the level of TCP/IP, not all URL authorities denote a running server or host. Some running servers do not react to requests (neither reject nor accept), and some actively reject establishing a connection. Some connections that are established are broken off during communication.

STEP C: COMMUNICATE WITH THE HOSTING SERVER Once a connection has been established over TCP/IP, the LOD Washing Machine sends an HTTP request, with SSL verification if needed (for secure HTTP), and an accept header that includes a preference for LOD content types. This includes standardized content types and content types that occur in practice.

Some requests are unsuccessful, receiving either a server, existence, or permission error. Some requests are unsuccessful due to redirect loops.

STEP E: UNPACK ARCHIVED DATA Many Linked Datasets are contained in archives. The LOD Washing Machine supports the archive filters and formats that are supported by the libarchive library[15]. The LOD Washing Machine accesses archives in a stream. Since archive entries can themselves be archives, this procedure is nested, resulting in a tree of streams. The root node of the tree is the stream of the orig-

13 http://tools.ietf.org/html/rfc3986
14 http://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml
15 https://code.google.com/p/libarchive/

inal archive file, the leaf nodes are streams of non-archived files, and the other non-leaf nodes are streams of intermediate archived files.

Some archives cannot be read by libarchive, which then throws an exception. We have not been able to open these archives with any of the standard unarchiving tools on Linux. Consequently, the LOD Washing Machine gives up on such archived files, but does report the exception that was thrown.

STEP F: GUESS SERIALIZATION FORMAT In order to parse the contents of the textual data that resides in the leaf nodes of the stream tree, we need to know the grammar of that data. The LOD Washing Machine uses content types to denote the grammar that is used for parsing. There are various ways in which the content type of a streamed file can be assessed. The most reliable way is to parse the whole file using each of the RDF serialization parsers, and take the one that emits the least syntax errors and/or reads the most valid RDF triples. A theoretical example of why one needs to parse the whole file, not just a first segment of it, can be given with respect to the difference between the Turtle and TriG formats. This difference may only become apparent in the last triple that appears in the file, by the occurrence of curly brackets (indicating a named graph).

Unfortunately, parsing every dataset with every parser is inefficient (CPU) and requires either local storage of the whole file (disk space) or multiple downloads of the same file (bandwidth).

In addition, we make the observation that the standardized RDF serialization formats occur in two families: XML-like (RDF/XML, RDFa) and Turtle-like (Turtle, TriG, N-Triples, N-Quads). The distinction between these two families can be reliably made by only looking at an initial segment of the file.

In order to keep the hardware footprint low, the LOD Washing Machine tries to guess the content type of a file based on a parse of only a first chunk of that file, in combination with the extension of the file (if any) and the content type header in the HTTP response message (if any). Using a look-ahead function on the stream, the LOD Washing Machine can use the first bytes on that stream in order to guess its content type, without consuming those bytes so that no redownload is necessary. The number of bytes available in the look-ahead is the same as the stream chunk size that is used for in-memory streaming anyway.

As explained above, this method may result in within-family mistakes, e.g., guessing Turtle for TriG or guessing N-Triples for N-Quads. In order to reduce the number of within-family mistakes, we use the content type and file extension. If these denote serialization formats that belong within the guessed family, we use that format. Otherwise, we use the most generic serialization format within the guessed family.

This approach ensures that the LOD Washing Machine uses a fully streamed pipeline and relatively few hardware resources.

STEP G: SYNTAX ERRORS WHILE PARSING RDF SERIALIZATIONS

The LOD Washing Machine parses the whole file using standards-conforming grammars. For this it uses the parsers from the SemWeb library [110]. This library passes the RDF 1.1 test cases and is actively used in SW research and applications. Using this library, the LOD Washing Machine is able to recognize different kinds of syntax errors and recover from them during parsing. We enumerate some of the most common syntax errors the LOD Laundromat is able to identify:

- Bad encoding sequences (e.g., non-UTF-8).
- Undefined IRI prefixes.
- Missing end-of-statement characters between triples (i.e., 'triples' with more than three terms).
- Non-escaped, illegal characters inside IRIs.
- Multi-line literals in serialization formats that do not support them (e.g., multi-line literals that are only legal in Turtle, also occur in N-Triples and N-Quads).
- Missing or non-matching end tags (e.g., RDF/XML).
- End-of-file occurrence within the last triple (probably indicating a mistake that was made while splitting files).
- IRIs that do not occur in between angular brackets (Turtle-family).

The LOD Washing Machine reports each syntax error it comes across. For data documents that contain syntax errors, there is no formal guarantee that a one-to-one mapping between the original document and a cleaned sibling document exists. This is an inherent characteristic of dirty data and the application of heuristics in order to clean as many stains as possible. In the absence of a formal model describing all the syntactic mistakes that can be made, recovery from arbitrary syntax errors is more of an art than a science. We illustrate this with the following example:

```
ex:a1 ex:a2 $"" ex:b1 ex:b2 ex:b3 .
                ex:c1 ex:c2 ex:c3 .
                ...
                ex:z1 ex:z2 ex:z3 .
            """ .
```

A standards-compliant RDF parser will not be able to parse this piece of syntax and will give a syntax error. A common technique for RDF parsers is to look for the next end-of-triple statement (i.e., the dot at the end of the first line), and resume parsing from there. This results in parsing the collection of triples starting with ⟨`ex:c1`, `ex:c2`, `ex:c3`⟩ and ending with ⟨`ex:z1`, `ex:z2`, `ex:z3`⟩. The three double quotes that occur at the end of the code sample will result in a second syntax error.

However, using other heuristics may produce very different results. For instance, by using minimum error distance, the syntax error can also be recovered by replacing the dollar sign with a double quote sign. This results in a single triple with a unusually long, but standards-compliant, literal term.

STEP H: DE-DUPLICATE RDF STATEMENTS The LOD Washing Machine loads the parsed triples into a memory-based triple store. By loading the triples into a triple store, it performs deduplication of interpreted RDF statements. Deduplication cannot be performed without interpretation, i.e., on the syntax level, because the same RDF statement can be written in different ways. Syntactically, the same triple can look different due to the use of character escaping, the use of extra white spaces, newlines and/or interspersed comments, the use of different/no named prefixes for IRIs, and abbreviation mechanisms in serialization formats that support them (e.g., RDF/XML, Turtle). Another source of the many-to-one mapping between syntax and semantics occurs for RDF datatypes / XML Schema 1.1 datatypes, for which multiple lexical expressions can map onto the same value.[16] For example, the lexical expressions 0.1 and 0.10000000009 map to the same value according to data type `xsd:float`, but to different values according to data type `xsd:decimal`.

While reading RDF statements into the triple store, the contents of different data documents are stored in separate transactions, allowing the concurrent loading of data in multiple threads. Each transaction represents an RDF graph or set of triples, thereby automatically deduplicating triples within the same file.

STEP I: SAVE RDF IN A UNIFORM SERIALIZATION FORMAT Once the triples are parsed using an RDF parser, and the resulting RDF statements are loaded into memory without duplicates, we can use a generator of our choice to serialize the cleaned data. We want our generator to be compliant with existing standards, and we want to support further processing of the data, as discussed in section 2.3.1. The LOD Washing Machine produces data in a canonical format that enforces a one-to-one mapping between data triples and file lines. This means that the end-of-line character can be reliably used in subsequent processing, such as pattern matching (e.g., regular expressions) and parsing. This also means that data documents can be easily split without running the risk of splitting in-between triples. Furthermore, the number of triples in a graph can be easily and reliably deter-

16 http://www.w3.org/TR/xmlschema11-2/

mined by counting the number of lines in a file describing that graph. Secondly, the produced documents are lexicographically sorted. This often improves the compression ratio of these documents, and may allow users to optimize the algorithm via which they consume these documents. Thirdly, the LOD Washing Machine leaves out any header information. This, again, makes it easy to split existing data documents into smaller parts, since the first part of the file is not treated specially due to serialization-specific header declarations (e.g., RDF/XML, RDFa) and namespace definitions (e.g., RDF/XML, Turtle). Fourthly, the LOD Washing Machine replaces all occurrences of blank nodes with well-known IRIs[17], in line with the RDF 1.1 specification[18]. Effectively, this means that blank nodes are interpreted as Skolem constants, not as existentially quantified variables. The Skolem constant is an IRI that is based on the URL that was used to stream the RDF data from, thereby making it a universally unique name at the moment of processing.[19] This makes it easy to append and split data documents, without the need to standardize apart blank nodes that originate from different graphs.

From the existing serialization formats, N-Triples and N-Quads come closest to these requirements. Since the tracking of named graphs is out of scope for our initial version of the LOD laundry (see section 2.6), we use a canonical form of N-Triples and N-Quads that excludes superfluous white space (only one space between the RDF terms in a triple and one space before the end-of-triple character), superfluous newlines (only one newline after the end-of-triple character), and comments (none at all). Newlines that occur in multi-line literals, supported by some serialization formats, are escaped according to the N-Triples 1.1 specification. Also, simple literals are not written, always adding the XML Schema string datatype explicitly.

STEP J: VOID CLOSURE After having stored the data to a canonical format, we make use of the fact that the valid triples are still stored in memory, by performing a quick query on the memory store. In this query we derive any triples that describe Linked Datasets. Specifically, we look for occurrences of predicates in the VoID namespace. We store these triples in a publicly accessible metadata graph that can be queried using SPARQL. For each dataset described in VoID triples, we follow links to datadumps (if present), add them to the LOD Basket, and clean those datadumps by using the LOD Washing Machine as

17 `https://tools.ietf.org/html/rfc5785`

18 `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-skolemization`

19 When a new file is disseminated at the same URL at a later point in time, the same Skolem constant may be used to denote a different blank node. Using skolemization, this becomes an instance of the generic problem that IRIs can denote different things at different times, as the data document is updated.

well. Since a dataset may describe a dataset that describes another dataset, this process is recursive.

STEP K: CONSOLIDATE AND DISSEMINATE DATASETS FOR FURTHER PROCESSING Since we want to incentivise the dataset providers to improve their adherence to guidelines, we keep track of all the mistakes that were discussed in this section. The mistakes (if any) are asserted together with some basic statistics, e.g. number of triples, number of bytes processed, in the publicly queryable metadata graph. For syntax errors we include the line and column number at which the error occurred, relative to the original file. This makes it easy for the dataset maintainers to improve their data and turn out cleaner in a next wash, since the metadata descriptions are automatically updated at future executions of the LOD Washing Machine.

2.5 THE LOD LAUNDROMAT WEB SERVICE

When the LOD Washing Machine has cleaned a data document, it is ironed and folded and made available on a publicly accessible Website that provides additional support for data consumers. We now describe the components that make up this Website and lift out the support features that make LOD Laundromat a good source for finding large volumes of high-quality Linked Data.

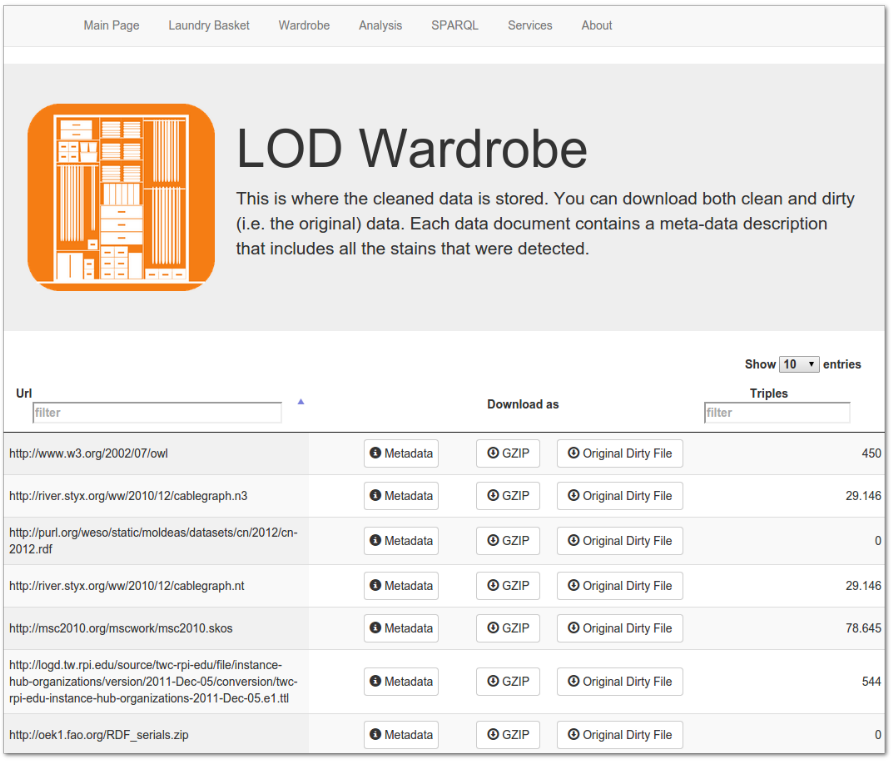
2.5.1 *LOD Wardrobe*

The LOD Wardrobe (Figure 2.1) is where the cleaned datasets are disseminated for human data consumers. The data documents are listed in a table that can be sorted according to various criteria (e.g., cleaning data, number of triples). For every data document, a row in the table includes links to both the old (dirty) and new (cleaned) data files, as well as a button that brings up a pop-up box with all the metadata for that data document. Furthermore, it is easy to filter the table based on a search string and multiple rows from the table can be selected for downloading at the same time.

2.5.2 *SPARQL endpoint*

All the metadata collected during the cleaning process is stored in an RDF graph that is publicly accessible through the SPARQL endpoint `http://lodlaundromat.org/sparql`. For human data consumers we deploy the feature-rich SPARQL editor YASGUI (See chapter 5). For machine consumption the SPARQL endpoint can be queried algorithmically. For instance, a SPARQL query can return URLs for downloading all clean data documents with over one million syntactically

Figure 2.1: The LOD Wardrobe is available at `http://lodlaundromat.org/wardrobe`



LOD Wardrobe

This is where the cleaned data is stored. You can download both clean and dirty (i.e. the original) data. Each data document contains a meta-data description that includes all the stains that were detected.

Show 10 entries

Url	Download as			Triples
http://www.w3.org/2002/07/owl	Metadata	GZIP	Original Dirty File	450
http://river.styx.org/ww/2010/12/cablegraph.n3	Metadata	GZIP	Original Dirty File	29.146
http://purl.org/weso/static/moldeas/datasets/cn/2012/cn-2012.rdf	Metadata	GZIP	Original Dirty File	0
http://river.styx.org/ww/2010/12/cablegraph.nt	Metadata	GZIP	Original Dirty File	29.146
http://msc2010.org/mscwork/msc2010.skos	Metadata	GZIP	Original Dirty File	78.645
http://logd.tw.rpi.edu/source/twc-rpi-edu/file/instance-hub-organizations/version/2011-Dec-05/conversion/twc-rpi-edu-instance-hub-organizations-2011-Dec-05.e1.ttl	Metadata	GZIP	Original Dirty File	544
http://oek1.fao.org/RDF_serials.zip	Metadata	GZIP	Original Dirty File	0

correct triples. In this way LOD Laundromat provides a very simple interface for running Big Data experiments. The metadata stored by the LOD Washing Machine includes information such as the number of triples in a dataset:

- the number of removed duplicates,
- the original serialization format,
- various kinds of syntax errors,
- and more.

The metadata that the LOD Wardrobe publishes is continuously updated whenever new cleaned laundry comes in.

2.5.3 *Visualizations*

Besides access to the datasets, the LOD Laundromat also provides real-time visualizations of the crawled data. These visualizations are small JavaScript widgets that use SPARQL queries on the metadata SPARQL endpoint.

Purely for illustrative purposes, we include a snapshot of such a widget in Figure 2.2. For a collection of 658,171 cleaned documents this widget shows the serialization format that was used to parse the original file. The majority of documents from this collection, 69.2% are formatted as Turtle. RDF/XML and N-Triples amount to 26.3% and 3.2% respectively.

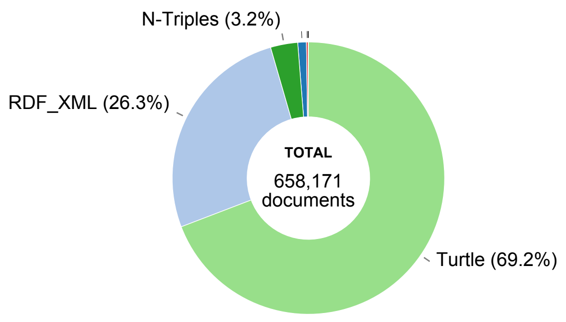


Figure 2.2: RDF serialization formats for a collection of RDF documents. Illustrative example of a visualization widget at `http://lodlaundromat.org/visualizations`.

As another example of the kinds of queries that can be performed on the SPARQL endpoint, we take the HTTP `Content-Length` header (See figure 2.3). Values for this header are often set incorrectly. Ideally, a properly set `Content-Length` header would allow data consumers to retrieve data more efficiently, e.g., by load-balancing data depending on the expected size of the response. However, our results show that around half of the documents return an invalid content length value, thereby showing that in practice it is difficult to reliably make use of this property. This shows that standard compliance for *all* stages of the data publishing pipeline is hard. When taking factors into account such as the number of duplicates, invalid HTTP headers, and parsing or HTTP errors, we notice that more than 50% of the documents fetched from a URL directly, contain errors[20].

2.5.4 *LOD Basket*

In order to extend our collection of datasets over time, users can add seed URLs to the LOD Basket. Seed points can be URLs that either point to VoID descriptions, or to data dumps directly. Seed locations can be added through a Web form, by selecting a file from the user's DropBox account, or through a direct HTTP GET request.

2.6 CONCLUSION

In this chapter we presented the infrastructure needed to obtain a complete-as-possible copy of the entire LOD cloud. Using LOD Laundromat, we publish standards- and guidelines-compliant datasets

20 See SPARQL query `http://yasgui.org/short/NyN9q49D`. This query excludes documents extracted from compressed archives

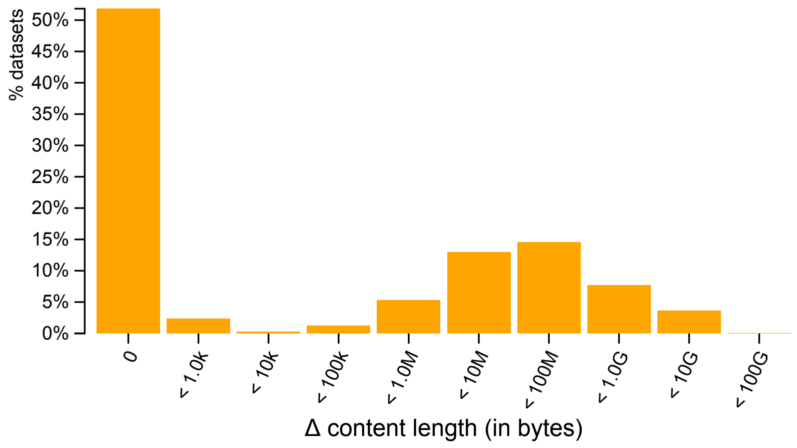


Figure 2.3: Differences between promised HTTP content lengths, and actual content lengths

that are siblings of existing, idiosyncratic datasets, and provide Linked Data providers with a platform to publish their datasets. LOD Laundromat implements a Linked Data cleaner that *continuously* crawls for additional datasets; the amount of data that we publish (over 38 billion triples at the time of writing) already surpasses that of existing data collections, such as the Billion Triple Challenge. In addition, the LOD Laundromat publishes metadata for every cleaned document on a publicly accessible Web site, and through machine-accessible Web services. Because anybody can drop their dirty data in the LOD Basket, the coverage of the LOD Laundromat will increase over time.

The LOD Laundromat is our solution to problem 1 from the introduction of this thesis, where we observed that Linked Datasets do not follow standards and best practices, and are difficult to consume. Our solution presents Linked Data providers with a cleaned and hosted version of their datasets, and provides feedback on any syntactic errors they might have introduced. At the same time, this infrastructure benefits Linked Data developers and researchers as well, as it provides a centralized service for accessing clean Linked Open Datasets that are published in a very simple canonical form of N-Triples or N-Quads. By using the LOD Laundromat, data consumers do not have to worry about different serialization formats, syntax errors, encoding issues, or triple duplicates. As a result, LOD Laundromat can act as an enabler for Big Data and SW research, as well as a provider of data for Web-scale applications.

.

3 REDUCING SIZE BY RDF GRAPH SAMPLING

The previous section showed the large scale of published Linked Data –both in terms of document size and document collection size. But these LOD Laundromat documents are often not directly accessible via a SPARQL triple-store –something that requires investments from Linked Data providers in powerful hardware–. Random sampling of these datasets would reduce this hardware footprint, but it would reduce the answer coverage as well. In this chapter, we propose an automatic sampling method that uses the graph topology, and is targeted at maximizing the answer coverage for applications using SPARQL querying. The approach we present is novel: no similar RDF sampling approach exist. Additionally, the concept of creating a sample aimed at maximizing SPARQL answer coverage, is unique. We empirically show that the relevance of triples for sampling (a semantic notion) is influenced by the topology of the graph (purely structural), and can be determined without prior knowledge of the queries. Experiments show a significantly higher recall of topology based sampling methods over random and naive baseline approaches (e.g. up to 90% for Open-BioMed at a sample size of 6%).

This chapter is an updated version of:

Laurens Rietveld, Rinke Hoekstra, Stefan Schlobach, and Christophe Guéret. Structural Properties as Proxy for Semantic Relevance in RDF Graph Sampling. In *The Semantic Web–ISWC 2014*, pages 81–96. Springer, 2014

Contributions to this paper:

I am the creator of SampLD and main author of the paper.

3.1 INTRODUCTION

The Linked Data cloud grows every year [6, 26] and has turned into a knowledge base of unprecedented size and complexity. As we have seen in the previous chapter, the LOD Laundromat already re-publishes 649,834 documents with over 38,606,408,433 triples. And with datasets such as DBpedia (459M triples) and Linked Geo Data (289M triples) that are central to many Linked Data applications, this poses problems with respect to the scalability of our current infrastructure and tools.

PROBLEM As discussed in problem 2, providers have a difficult time serving a stable public SPARQL endpoint [26], and publishing large datasets requires investments in powerful hardware, where cloud-based hosting is not free either. These costs are avoidable if we know which part of the dataset is needed for our application, i.e. if only we could a priori pick the data that is actually being *used*. Under several circumstances, e.g. for prototyping, demoing or testing, developers and users are content with relevant subsets of the data, and accept the possibility of incomplete results that comes with it.

CONTRIBUTIONS The approach discussed in this chapter is the first of two orthogonal solutions (see chapter 4 for the other solution). Our analysis of five large datasets (larger than 50 million triples) shows that for a realistic set of queries, at most 2% of the dataset is actually used (see the 'coverage' column in Table 3.1): a clear opportunity for pruning RDF datasets to more manageable sizes. Unfortunately, this set of queries is not always known: queries are not logged or logs are not available because of privacy or property rights issues. And even if a query set is available, it may not be representative or suitable, e.g. it contains queries that return the entire dataset.

We define *relevant sampling* as the task of finding those parts of an RDF graph that maximize a task-specific relevance function while minimizing size. For our use case, this relevance function relies on semantics: we try to find the smallest part of the data that entails as many of the original answers to typical SPARQL queries as possible. This chapter investigates whether we can use structural properties of RDF graphs to predict the relevance of triples for typical queries.

To evaluate the approach, we represent "typical use" by means of a large number of SPARQL queries fired against datasets of various size and domain: *DBpedia 3.9* [4], *Linked Geo Data* [7], *MetaLex* [53], *OpenBioMed*[1], *Bio2RDF* [14] and *Semantic Web Dog Food* [73] (see Table 3.1). The queries were obtained from server logs of the triple stores hosting the datasets and range between 800 and 5000 queries for each dataset.

Given these datasets and query logs, we then

1. rewrite RDF graphs into directed unlabeled graphs;
2. analyze the topology of these graphs using standard network analysis methods –much like the network analysis measures presented in chapter 6–;
3. assign the derived weights to triples;
4. generate samples for every percentile of the size of the original graph.

These steps were implemented as a scalable sampling pipeline, called *SampLD*[2].

1 See http://www.open-biomed.org.uk/
2 See https://github.com/Data2Semantics/GraphSampling/

Our results show that the topology of the graph alone helps to predict the relevance of triples for typical use in SPARQL queries. In other words, we show that without prior knowledge of the queries to be answered, we can determine to a surprisingly high degree which triples in the dataset can safely be ignored and which cannot. As a result, we are able to achieve a recall of up to .96 with a sample size as small as 6%, using *only* the structural properties of the graph. This means that we can use purely *structural* properties of a knowledge base as proxy for a *semantic* notion of relevance, allowing Linked Data providers to reduce their hardware footprint significantly, with little decreased recall.

CONTENT The rest of the chapter is organized as follows. We first discuss related work in Section 3.2, followed by the problem definition (Section 3.3) and a description of our approach (Section 3.4). Section 3.5 discusses the experiment setup and evaluation, after which we present the results (Section 3.6). Finally we discuss our conclusions in Section 3.7.

3.2 RELATED WORK

Except for random sampling [99], extracting relevant parts of Linked Data graphs has not been done before. However, there are a number of related approaches that deserve mentioning: relevance ranking for Linked Data, generating SPARQL benchmark queries, graph rewriting techniques, and non-deterministic network sampling techniques.

NETWORK SAMPLING [65] evaluates several non-deterministic methods for sampling networks: random node selection, random edge selection, and exploration techniques such as random walk. Quality of the samples is measured as the structural similarity of the sample with respect to the original network. This differs from our notion of quality, as we do not strive at creating a structurally representative sample, but rather optimize for the ability to answer the same queries. Nevertheless, the sampling methods discussed by [65] are interesting baselines for our approach; we use the random edge sampling method in our evaluation (See Section 3.5.1).

RELEVANCE RANKING Existing work on Linked Data relevance ranking focuses on determining the relevance of individual triples for answering a single query [3, 10, 36, 54, 63]. Graph summaries, such as in [29], are collections of important RDF *resources* that may be presented to users to assist them in formulating SPARQL queries, e.g. by providing context-dependent auto completion services. However, summarization does not produce a list of triples ordered by relevance.

TRank [103] ranks RDF entity types by exploiting type hierarchies. However, this algorithm still ranks entities and not triples. In contrast, TripleRank [36] uses 3d tensor decomposition to model and rank triples in RDF graphs. It takes knowledge about different link types into account, and can be seen as a multi-model counterpart to web authority ranking with HITS. TripleRank uses the rankings to drive a faceted browser. Because of the expressiveness of a tensor decomposition, TripleRank does not scale very well, and [36] only evaluate small graphs of maximally 160K triples. Lastly, TripleRank prunes predicates that dominate the dataset, an understandable design decision when developing a user facing application, but it has an adverse effect on the quality of samples as prominent predicates in the data are likely to be used in queries as well.

ObjectRank [10] is an adaptation of PageRank that implements a form of link semantics where every type of edge is represented by a particular weight. This approach cannot be applied in cases where these weights are not known beforehand. SemRank [3] ranks relations and paths based on search results. This approach can filter results based on earlier results, but it is not applicable to *a-priori* estimation of the relevancy of the triples in a dataset. Finally, stream-based approaches such as [42] derive the schema. This approach is not suitable for retrieving the most relevant *factual* data either, regarding a set of queries.

Concluding, existing approaches on ranking RDF data either require prior knowledge such as query sets, a-priori assignments of weights, produce samples that may miss important triples, or focus on resources rather than triples.

SYNTHETIC QUERIES SPLODGE [41] is a benchmark query generator for arbitrary, real-world RDF datasets. Queries are generated based on features of the RDF dataset. SPLODGE-based queries would allow us to run the sampling pipeline on many more datasets, because we would not be restricted by the requirement of having a dataset *plus* corresponding query logs. However, *benchmark* queries do not necessarily resemble *actual* queries, since they are meant to test the performance of Linked Data storage systems [2]. Furthermore, SPLODGE introduces a dependency between the dataset features and the queries it generates, that may not exist for user queries.[3]

RDF GRAPH REWRITING RDF graphs can be turned into networks that are built around a particular property, e.g. the social aspects of co-authorship, by extracting that information from the RDF data [108].

3 In an earlier stage of this work, we ran experiments against a synthetic data and query set generated by SP^{2}Bench [94]. The results were different from any of the datasets we review here, as the structural properties of the dataset were quite different, and the SPARQL queries (tailored to benchmarking triple-stores) are incomparable to regular queries as well.

Edge labels can sometimes be ignored when they are not directly needed, e.g. to determine the context of a resource [54], or when searching for paths connecting two resources [45].

The networks generated by these rewriting approaches leave out *contextual* information that may be critical to assess the relevance of triples. The triples $\langle :\mathtt{bob}, :\mathtt{hasAge}, \text{“50"}\rangle$ and $\langle :\mathtt{anna}, :\mathtt{hasWeight}, \text{“50"}\rangle$ share the same literal ("50"), but it respectively denotes an *age* and a *weight*. Finally, predicates play an important role in our notion of *relevance*. They are crucial for answering SPARQL queries, which suggests that they should carry as much weight as subjects and objects in our selection methodology. Therefore, our approach uses different strategies to remove the edge labels, while still keeping the context of the triples.

In [51], RDF graphs are rewritten to a bipartite network consisting of subjects, objects and predicates, with a separate statement node connecting the three. This method preserves the role of predicates, but increases the number of edges and nodes up to a threefold, making it difficult to scale. Additionally, the resulting graph is no longer *directed*, disqualifying analysis techniques that take this into account.

3.3 CONTEXT

In the previous section, we presented related work on RDF ranking and network sampling, and showed that sampling for RDF graphs has not been done before. This section introduces a very generic framework for such RDF sampling. We elaborate on different RDF sampling scenarios, and present the particular sampling scenario addressed in this chapter.

3.3.1 *Definitions*

A '*sample*' is just an arbitrary subset of a graph, so we introduce a notion of *relevance* to determine whether a sample is a suitable replacement of the original graph. Relevance is determined by a relevance function that varies from application to application. The following definitions use the same SPARQL definitions as presented in [80].

Theorem 1 *An RDF graph $\mathcal{G}$ is a set of triples. A* sample $\mathcal{G}'$ *of $\mathcal{G}$ is a proper subset of $\mathcal{G}$. A sample is* relevant *w.r.t. a relevance function* $F(\mathcal{G}', \mathcal{G})$ *if it maximizes* F *while minimizing its size.*

Finding a relevant sample is a multi-objective optimization problem: selecting a sample small enough, while still achieving a recall which is high enough. Moreover, there is no sample that fits all tasks and problems, and for each application scenario a specific relevance function has to be defined. In this chapter, relevance is defined in terms of the *coverage of answers* with respect to SPARQL queries.

Theorem 2 *The* relevance function for SPARQL querying $F_s(\mathcal{G}', \mathcal{G})$ *is the probability that the solution* μ *to an arbitrary SPARQL query* Q *is also a solution to* Q *w.r.t.* $\mathcal{G}'$*.*

As usual, all elements $T \in \mathcal{G}$ are triples of the form $(s, p, o) \in I \times I \times (I \cup L)$, where s is called the subject, p the predicate, and o the object of T. I denotes all IRIs, where L denotes all literals. For this chapter, we ignore blank nodes.

In other words, a *relevant sample* of a RDF graph is a smallest subset of the graph, on the basis of which the largest possible number of answers that can be found with respect to the original RDF graph. As common in multi-objective optimization problems, the solution cannot be expected to be a single *best* sample, but a set of samples of increasing size.

3.3.2 *Problem Description*

The next step is to determine the method by which a sample is made. As briefly discussed in the introduction, this method is restricted mostly by the pre-existence of a suitable set of queries. To what extent can these queries be used to *inform* the sampling procedure? As we have seen in the related work, sampling without prior knowledge – *uninformed* sampling – is currently an unsolved problem. And in fact, even if we *do* have prior knowledge, a method that does not rely on prior knowledge is still useful.

With *informed* sampling, there is a complete picture of what queries to expect: we know exactly which queries we want to have answered, and we consequently know which part of the dataset is required to answer these queries. Given the size of Linked Data sets, this part can still be too large to handle. This indicates a need for heuristics or uninformed methods to reduce the size even more.

If we have an incomplete notion of what queries to expect, e.g. we only know part of the queries or only know their structure or features, we could still use this information to create a *semi-informed* selection of the data. This requires a deeper understanding of what features of queries determine relevance, how these relate to the dataset, and what sampling method is the best fit.

We focus on a comparison of methods for uninformed sampling, the results of which can be used to augment scenarios where more information is available. For instance, a comparison of query features as studied in [81, 85], combined with performance of our uninformed sampling methods, could form the basis of a system for semi-informed sampling.

To reiterate, our hypothesis is that we can use standard network metrics on RDF graphs as useful proxies for the relevance function. To test this hypothesis, we implemented a scalable sampling pipeline, SampLD, that can run several network metrics to select the top

Dataset	**#Tripl.**	**Avg. Deg.**	**Tripl. w/ literals**	**#Q**	**Coverage**	**Q w/ literals**	**#Triple patt. per query** (avg / stdev)
DBpedia 3.9	459M	5.78	25.44%	1640	0.003%	61.6%	1.07 / 0.40
LGD	289M	4.06	46.35%	891	1.917%	70.7%	1.07 / 0.27
MetaLex	204M	4.24	12.40%	4933	0.016%	1.1%	2.02 / 0.21
Open-BioMed	79M	3.66	45.37%	931	0.011%	3.1%	1.44 / 3.72
Bio2RDF/KEGG	50M	6.20	35.06%	1297	2.013%	99.8%	1.00 / 0.00
SWDF	240K	5.19	34.87%	193	39.438%	62.4%	1.80 / 1.50

Table 3.1: Data and query set statistics

ranked triples from RDF datasets and evaluate the quality of those samples by their capability to answer real SPARQL queries.

3.3.3 *Datasets*

We evaluate the quality of our sampling methods for the six datasets listed in the introduction: DBPedia, Linked Geo Data (LGD), MetaLex, Open-BioMed (OBM), Bio2RDF[4] and Semantic Web Dog Food (SWDF). These datasets were chosen based on the availability of SPARQL queries. These datasets were the only ones with an available corresponding large query set. The MetaLex query logs were made available by the maintainers of the dataset. In the other five cases, we used server query logs made available by the USEWOD workshop series [16].

Table 3.1 shows that the size of these dataset ranges from 240K triples to 459M triples. The number of available queries per dataset ranges between 193 and 4933. Interestingly, for all but one of our datasets, less than 2% of the triples is actually *needed* to answer the queries. This indicates that only a very small portion of the datasets is relevant, which corroborates our intuition that the costs for using Linked Data sets can be significantly reduced by selecting only the relevant part of a dataset. However, these low numbers make finding this small set of relevant triples more difficult as well. Other relevant dataset properties shown in this table are the average degree of the subjects and objects, the percentage of triples where the object is a literal, the percentage of queries of which at least one binding uses a literal, and the average and standard deviation of the number of triple patterns per query.

4 For Bio2RDF, we use the KEGG dataset [62], as this is the only Bio2RDF dataset for which USEWOD provides query logs. KEGG includes biological systems information, genomic information, and chemical information.

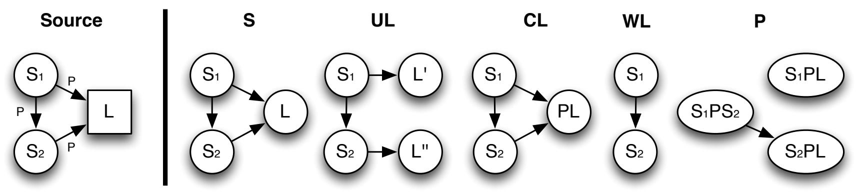


Figure 3.1: Rewrite methods

3.4 SAMPLING PIPELINE

The SampLD pipeline calculates and evaluates the quality of samples across different uninformed sampling methods for multiple large datasets.[5] The procedure consists of the following four phases:

1. *rewrite* an RDF graph to a directed unlabeled graph,
2. *analyze* the rewritten graph using standard network analysis algorithms,
3. *assign* the node weights to triples, creating a ranked list of triples,
4. *generate* samples from the ranked list of triples.

We briefly discuss each of the four phases here.

STEP 1: GRAPH REWRITING Standard network analysis methods, are not readily suited for *labeled* graphs, nor do they take into account that a data model may be reflected in the verbatim RDF graph structure in many ways [43]. Since the edge labels (predicates) play an important role in RDF, simply ignoring them may negatively impact the quality of samples (see the related work). For this reason, the SampLD pipeline can evaluate pairwise combinations of network analysis techniques and alternative representations of the RDF graph, and compare their performance across sample sizes.

SampLD implements five rewriting methods: a *simple* (S), *unique literals* (UL), *without literals* (WL), *context literals* (CL), and *path* (P). As can be seen in Figure 3.1, the first four methods convert every triple (s, p, o) to a directed edge $s \rightarrow o$. If several triples result in the same edge (e.g. when only the predicate differs), we do not assert that edge more than once. These first four methods differ primarily with respect to their treatment of literal values (i.e. non-IRI nodes) in the graph. It is important to note that for all approaches, any removed literals are re-added to the RDF graph during the round-trip phase detailed below. Since literals do not have outgoing edges they have a

5 The SampLD pipeline and evaluation procedure are available online at https://github.com/Data2Semantics/GraphSampling/

different effect on network metrics than IRIs, e.g. by acting as a 'sink' for PageRank.

The Simple (S) method retains the exact structure of the original RDF graph. It treats every *syntactically* unique literal as a single node, taking the data type and language tag of the literal into account. Two occurrences of the literal "`50"^^xsd:Integer` result in a single node. The Unique literals (UL) method converts every occurrence of a literal as a separate node. The average degree drops, and the graph becomes larger but will be less connected. The Context literals (CL) approach preserves the context of literals, it groups literals that share the same predicate together in a single node. This allows us to make sure that the network metrics distinguish e.g. between integers that express weight, and those that express age. This also results in fewer connections and a lower average degree since predicate-literal pairs will be less frequent than literals. The Without literals (WL) method simply ignores all occurrences of literals. As a result the graph becomes smaller.

The fifth method, Path (P), is triple, rather than resource oriented. It represents every triple (s, p, o) as a single node, and two nodes are connected when they form a path of length 2, i.e. the first triple's object should be the second triple's subject. Asserting edges between triples that share *any* resource discards the direction of the triple, and produces a highly verbose graph, cf. [51], as a resource shared between n triples would generate $\frac{n(n-1)}{2}$ edges. Also, occurrences of triples with `rdf:type` predicates would result in an extremely large number of connections. The path method has the advantage that it results in a smaller graph with low connectedness, and that maintains directedness, where we can assign weights directly to *triples* rather than resources (as with the other methods).

STEP 2: NETWORK ANALYSIS In the second step, SampLD applies three common network analysis metrics to the rewritten RDF graph: *PageRank*, *in degree* and *out degree*. These are applied "as is" on the rewritten graphs.

STEP 3: ASSIGN TRIPLE WEIGHTS Once we have obtained the network analysis metrics for all nodes in the rewritten graph, the (aggregated) values are assigned as *weights* on the triples in the original graph. For method P, we simply assign the value of the node that corresponds to the triple. For the S, UL, WL and CL methods, we retrieve the values for the subject and object of each triple, and assign whichever is *highest* as weight to that triple[6]. When the object of a triple is a literal, it has no corresponding node in the graph produced through the WL method: in such cases, the value of the *subject*

6 One can also use the minimum or average node weight. We found that the maximum value performs better

will function as weight for the triple as a whole. The result of this assignment phase is a ranked list of triples, ordered by weight in descending order. The distribution of triple weights typically follows a 'long tail' distribution, where large numbers of triples may share the same weight. To prevent potential bias when determining a sample, these triples with equal weights are added to the ranked list in random order.

STEP 4: GENERATING SAMPLES Given the ranked list of triples, generating the sample is a matter of selecting the desired top-*k* percent of the triples, and removing the weights. The 'best' *k* value can differ per use-case, and depends on both the minimum required quality of the sample, and the maximum desired sample size. For our current purposes SampLD produces samples for each accumulative percentile of the total number of triples, resulting in 100 samples *each* for every combination of dataset, rewrite method and analysis algorithm.

IMPLEMENTATION Because the datasets we use are quite large, ranging up to 459 Million triples for DBPedia, each of these steps was implemented using libraries for scalable distributed computing (Pig [38] and Giraph [8]). Scale also means that we are restricted in the types of network metrics we could evaluate. For instance, Betweenness Centrality is difficult to paralellize because of the need for shared memory [101]. However many of the tasks *are* parallelizable, e.g. we use Pig to fetch the weights of all triples.

Given the large number of samples we evaluate (over 15.000, considering all sample sizes, datasets and sampling methods), SampLD uses a novel scalable evaluation method that avoids the expensive procedure (in terms of hardware and time) of loading each sample in triple-stores to calculate the recall.

3.5 EXPERIMENT SETUP AND EVALUATION

The quality of a sample is measured by its ability to return answers on a set of queries: we are interested in the *average recall* taken over all queries. Typically, these queries are taken from publicly available server query logs, discarding those that are designed to return all triples in a dataset, and focusing on `SELECT` queries, as these are the most dominant. A naive approach would be to execute each query on both the original dataset and the sample, and compare the results. This is virtually impossible, given the large number of samples we are dealing with: 6 datasets means 15.600 samples (one, for every combination of dataset (6), sampling method (15) and baseline (1+10), and percentile(100)), or $1.4 \cdot 10^{12}$ triples in total.

Instead, SampLD (a.) executes the queries once on the original dataset and analyzes which triples are used to answer the query, (b.) uses a cluster to check which weight these triples have. It then (c.) checks whether these triples *would have been* included in a sample, and calculates recall. This avoids the need to load and query each sample. Below, we give a detailed description of this procedure.

TERMINOLOGY For each graph G we have a set of `SELECT` queries $\mathcal{Q}$, acting as our relevance measure. Each $Q \in \mathcal{Q}$ contains a set of variables $\mathcal{V}$, of which some may be projection variables V_p (i.e. variables for which results are returned). Executing a query Q on G returns a result set R_q^g, containing a set of query solutions $\mathcal{S}$. Each query solution $S \in \mathcal{S}$ contains a set of bindings $\mathcal{B}$. Each bindings $B \in \mathcal{B}$ is a mapping between projection variable $V_p \in \mathcal{V}$ and a value from our our graph: $\mu : V_p \rightarrow (I \cup L)$

REQUIRED TRIPLES Rewriting a `SELECT` query into a `CONSTRUCT` query returns a bag of *all* triples needed to answer the `SELECT` query. However, there is no way to determine what role individual triples play in answering the query: some triples may be essential in answering all query solutions, others just circumstantial. Therefore, SampLD extracts triples from the query on a *query solution* level. It instantiates each triple pattern, by replacing each variable used in the query triple patterns with the corresponding value from the query solution. As a result, the query contains triple patterns without variables, and only IRIs and literals. These instantiated triple patterns ('query triples') show us which triples are *required* to produce this specific query solution. This procedure is not trivial.

First, because not all variables used in a query are also projection variables, and blank nodes are inaccessible as well, we rewrite every `SELECT` query to the '`SELECT DISTINCT *`' form and replace blank nodes with unique variable names. This ensures that all nodes and edges in the matching part of the original graph G are available to us for identifying the query triples. However, queries that already expected DISTINCT results need to be treated with a bit more care. Suppose we have the following query and dataset:

```
SELECT DISTINCT ?city  WHERE {          <university1> :inCity <London> .
  ?university :inCity ?city ;           <university1> :rating <high> .
              :rating :high .           <university2> :inCity <London> .
}                                       <university2> :rating <high> .
```

Rewriting the query to '`SELECT DISTINCT *`' results in two query solutions, using all four dataset triples. However, we only either need at least the first two triples, or the last two, but not all four. SampLD therefore tracks each distinct combination of bindings for the projection variables V_p.

Secondly, when the clauses of a UNION contain the same variable, but only one clause matches with the original graph G, the other clause should not be instantiated. We instantiate each clause following the normal procedure, but use an `ASK` query to check wether the instantiated clause exists in G. If it does not exist, we discard the query triples belonging to that clause.

Thirdly, we ignore the GROUP and ORDER solution modifiers, because they do not tell us anything about the actual triples required to answer a query. The LIMIT modifier is a bit different as it indicates that the user requests a specific number of results, but not exactly *which* results. The limit is used in recall calculation as a cap on the maximum number of results to be expected for a query. In other words, for these queries we don't check whether *every* query solution for G is present for the sample but only look at the proportion of query solutions.

Finally, we currently ignore negations in SPARQL queries since they are very scarce in our query sets. Negations may *increase* the result set for smaller sample sizes, giving us a means to measure precision, but the effect would be negligible.

CALCULATE RECALL For all query triples discovered in the previous step, and for each combination of rewrite method and network analysis algorithm, we find the weight of this triple in the ranked list of triples. The result provides us with information on the required triples of our query solutions, and their weights given by our sampling methods. SampLD can now determine whether a query solution would be returned for any given sample, given the weight of its triples, and given a k cutoff percentage. If a triple from a query solution is not included in the sample, we mark that solution as *unanswered*, otherwise it is *answered*.

Recall is then determined as follows. Remember that query solutions are grouped under distinct combinations of projection variables. For every such combination, we check each query solution, and whenever one of the grouped query solutions is marked as *answered*, the combination is marked answered as well. For queries with OPTIONAL clauses, we do not penalize valid query solutions that do not have a binding for the optional variable, even though the binding may be present for the original dataset.[7] If present, the value for the LIMIT modifier is used to cap the maximum number of answered query solutions.

The recall for the *query* is the number of answered projection variable combinations, divided by the total number of distinct projection variable combinations. For each *sample*, SampLD uses the average recall, or arithmetic mean over the query recall scores as our measure of relevance.

7 Queries with only OPTIONAL clauses are ignored in our evaluation

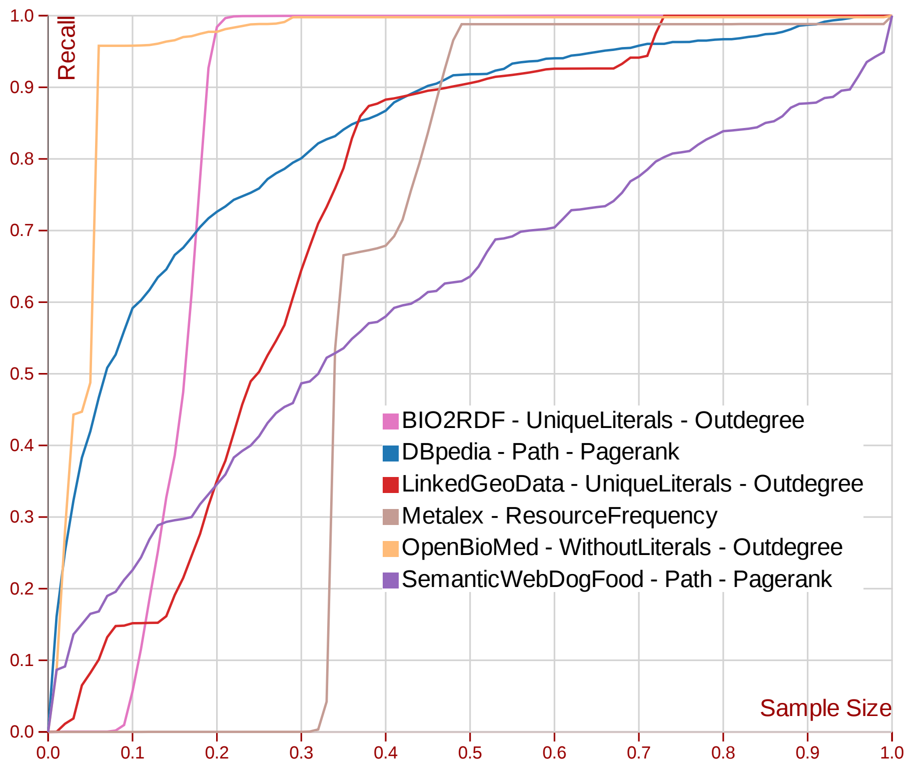


Figure 3.2: Best performing sample methods per dataset

3.5.1 *Baselines*

In our evaluation we use two baselines: a *random selection* (rand) and using *resource frequency* (freq). Random selection is based on 10 random samples for each of our datasets. Then, for each corresponding query we calculate recall using the 10 sampled graphs, and average the recall over each query. The resource frequency baseline counts the number of occurrences of every subject, predicate and object present in a dataset. Each triple is then assigned a weight equal to the sum of the frequencies of its subject, predicate and object.

3.6 RESULTS

This section discusses the results we obtained for each of the datasets. An interactive overview of these results, including recall plots, significance tests, and degree distributions for *every* dataset, is available online[8]. Figure 3.2 shows the best performing sample method for each of our datasets. An ideal situation would show a maximum recall for a low sample size.[9] For *all* the datasets, these best performing sampling methods outperform the random sample, with often a large

8 See `http://data2semantics.github.io/GraphSampling/`

9 Note that all the presented plots are clickeable, and point to the online interactive version

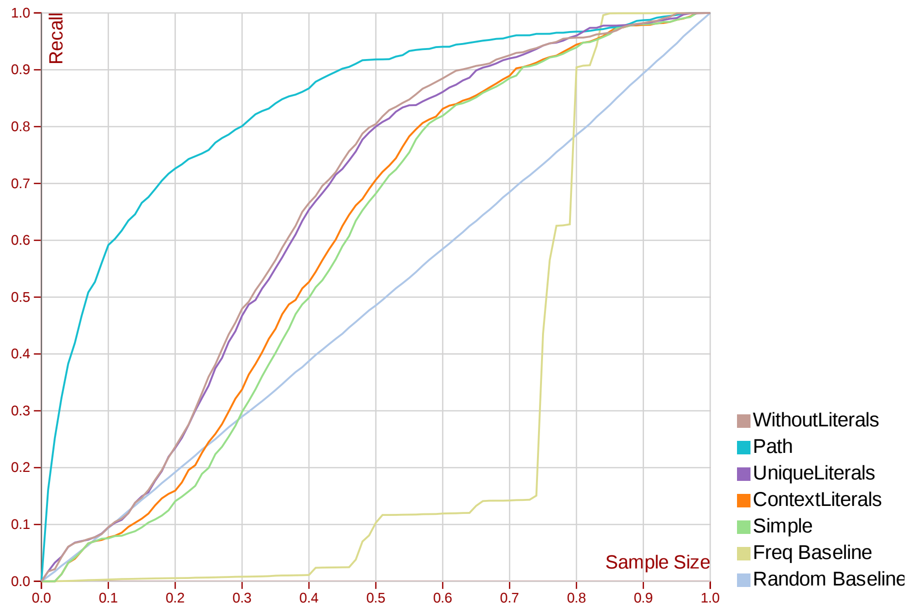


Figure 3.3: PageRank Sampling methods on DBPedia

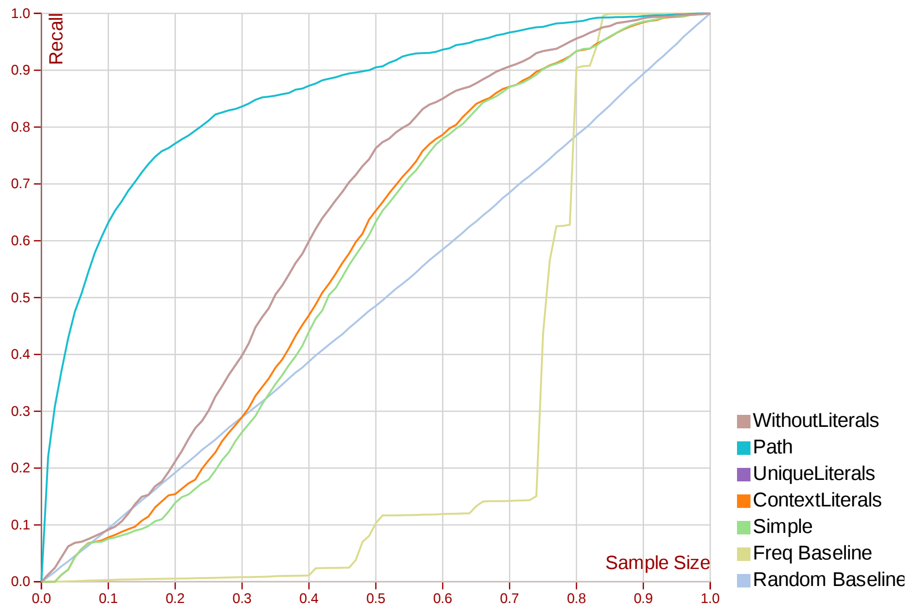


Figure 3.4: In Degree Sampling methods on DBPedia

difference in recall between both. The *P* rewrite method (see Figure 3.1) combined with a PageRank analysis performs best for Semantic Web Dog Food and DBpedia (see also Figure 3.2). The *UL* method combined with an out degree analysis performs best for Bio2RDF and Linked Geo Data. For Open-BioMed the *WL* and out degree performs best, where the naive resource frequency method performs best for MetaLex. For each dataset, the random baseline follows an (almost) linear line from recall 0 to 1; a stark difference with the sampling methods.

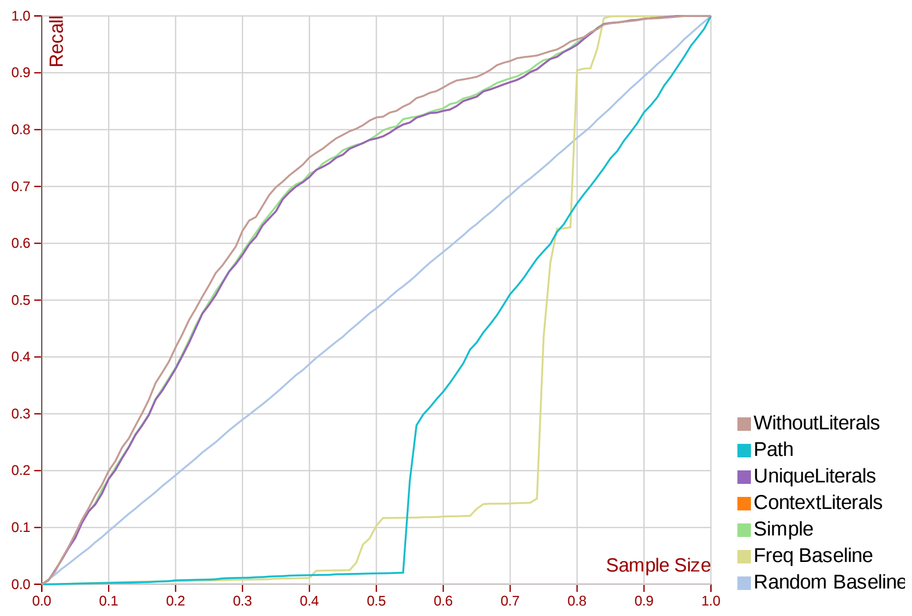


Figure 3.5: Out Degree Sampling methods on DBPedia

Figure 3.2 also shows that both the sample *quality* and *method* differs between datasets. Zooming in on sample sizes 10%, 25% and 50%, the majority of the best performing sampling methods have significantly better average recall ($\alpha = 0.05$) than the random sample. Exceptions are LGD and Bio2RDF for sample size 10%, and MetaLex for sample size 10% and 25%.

The dataset properties listed in Table 3.1 help explain results for some sampling methods. The resource frequency baseline performs extremely bad for OBM[10]: for all possible sample sizes, the recall is almost zero. Of all objects in Open-BioMed triples, 45.37% are literals. In combination with 32% duplicate literals, this results in high rankings for triples that contain literals for this particular baseline. However, *all* of the queries use at least one triple pattern consisting *only* of IRIs. As most dataset triples contain a literal, and as these triples are ranked high, the performance of this specific baseline is extremely bad.

Another observation is the presences of 'plateaus' in Figure 3.2, and for some sample sizes a steep increase in recall. This is because some triples are required for answering a large number of queries. Only once that triple is included in a sample, the recall can suddenly rise to a much higher level. For the Bio2RDF sample created using PageRank and the path rewrite method (viewable online), the difference in recall between a sample size of 1% and 40% is extremely small. In other words, choosing a sample size of 1% will result in more or less the same sample quality as a sample size of 40%.

10 See `http://data2semantics.github.io/GraphSampling/`

Figures 3.3, 3.4 and 3.5 show the performance of the sampling methods for DBpedia. The P method combined with either PageRank or in degree performs best on DBpedia, where both baselines are amongst the worst performing sampling methods. A sample size of 7% based on the P and PageRank sampling method already results in an average recall of 0.5. Notably, this same rewrite method (P) performs worst on DBpedia when applied with out degree. This difference is caused by triples with literals acting as sink for the path rewrite method: because a literal can never occur in a subject position, that triple can never link to any other triple. This causes triples with literals to *always* receive an out degree of zero for the P rewrite method. Because 2/3 of the DBpedia queries require at least one literal, the average recall is extremely low. This 'sink' effect of P is stronger compared to other rewrite methods: the triple weight of these other methods is based on the weight of the subject and object (see section 3.4). For triples containing literals, the object will have an out degree of zero. However, the subject may have a larger out degree. As the subject and object weights are aggregated, these triples will often receive a non-zero triple weight, contrary to the P rewrite method.

Although our plots show striking differences between the datasets, there are similarities as well. First, the out degree combined with the UL, CL and S methods performs very similar across all datasets and sample sizes[11]. The reason for this similarity is that these rewrite methods *only* differ in how they treat literals: as-is, unique, or concatenated with the predicate. These are exactly those nodes which *always* have an out degree of zero, as literals only have incoming edges. Therefore, this combination of rewrite methods and network analysis algorithms performs consistently the same. Second, the in degree of the S and CL rewrite methods are similar as well for all datasets with only a slight deviation in information loss for DBpedia. The main idea behind the CL is appending the predicate to the literal to provide context. The similarity for the in degree of both rewrite methods might indicate only a small difference between the literals in both rewrite methods regarding incoming links: adding context to these literals has no added value. DBpedia is the only dataset with a difference between both rewrite methods. This makes sense, as this dataset has many distinct predicates (53.000), which increases the chances of a single literal being used in multiple contexts, something the CL rewrite method is designed for.

What do these similarities buy us? They provide rules of thumb for applying SampLD to new datasets, as it shows which combinations of rewrite methods and network analysis algorithms you can safely ignore, restricting the number of samples to create and analyze for each dataset.

11 See our online results at `http://data2semantics.github.io/GraphSampling/`

3.7 CONCLUSION

In this chapter we introduced SampLD, a sampling methods suitable for Linked Data publishers who are willing to sacrifice completeness for more manageable data sizes.

We tested the hypothesis as to whether we can use uninformed, network topology based methods to estimate semantic relevance of triples in an RDF graph. We introduced a pipeline that uses network analysis techniques for scalable calculation and selection of ranked triples, *SampLD*. It can use five ways to rewrite labeled, directed graphs (RDF) to unlabeled directed graphs, and runs a parallelized network analysis (indegree, outdegree and PageRank). We furthermore implemented a method for determining the recall of queries against our samples that does not require us to load every sample in a triple store (a major bottleneck). As a result, SampLD allows us to evaluate 15.600 different combinations of datasets, rewritten graphs, network analysis and sample sizes.

RDF graph topology, query type and structure, sample size; each of these can influence the quality of samples produced by a combination of graph rewriting and network analysis. We do not offer a definitive answer as to which combination is the best fit, because we cannot predict the best performing sampling method given a data and query set. Applying machine learning methods on the results for several independent data and query sets could provide more information here, but although SampLD provides the technical means, the number of publicly available query sets is currently too limited to learn significant correlations (6 query sets in total for USEWOD 2013, only 3 in 2014)[12]. This limitation makes a large scale sampling evaluation on e.g. the LOD Laundromat impossible. Without the proper relevance measures (i.e. query sets), we cannot automatically determine which sampling method performs well for a given dataset.

Our results, all of which are available online[13], indicate that the topology of RDF graphs *can* be used to determine good samples that, in many cases, significantly outperform our baselines. Indeed, this shows that we can mimic *semantic* relevance through *structural* properties of RDF graphs, without an a-priori notion of relevance. As a result, publishers are able to decrease their hardware footprint (See problem 2 from the introduction) for hosting queryable endpoints with a minimal loss in completeness. The applicability of SampLD can go beyond that of hosting: Linked Data publishers who want to archive datasets but are limited by hardware constraints, can use SampLD to select the most relevant subset of the data.

12 See `http://data.semanticweb.org/usewod/2013/challenge.html` and `http://usewod.org/reader/release-of-the-2014-usewod-log-data-set.html`, respectively

13 Interactive plots of our results are available at `http://data2semantics.github.io/GraphSampling/`.

SampLD does come with a reduction in completeness of query results. For those publishers who are unwilling to sacrifice completeness, we present our second solution for decreasing the hardware footprint in the next chapter.

4 REDUCING QUERY COMPLEXITY

In the previous chapter we reduced the hosting hardware costs of queryable endpoints by reducing the dataset size. In this chapter we present Linked Data providers with an alternative approach where we reduce the complexity of the query language itself. We do so by extending the LOD Laundromat, and by combining it with a low-cost server-side interface called 'Triple Pattern Fragments' [106]. In doing so, we bridge the gap between the web of downloadable data files from the LOD Laundromat, and the web of live queryable data.
The result is a repeatable, low-cost, open-source data publication process, that decreases the cost for hosting queryable Linked Data. We demonstrate its viability by republishing all 649,834 LOD Laundromat documents as queryable APIs.

This chapter is an updated version of:

Laurens Rietveld, Ruben Verborgh, Wouter Beek, Miel Vander Sande, and Stefan Schlobach. Linked Data-as-a-Service: The Semantic Web Redeployed. In *Proceedings of the Extended Semantic Web Conference (ESWC)*. Springer, 2015

Contributions to this paper:

This was joined work with 4 main authors and I am responsible for integrating Triple Pattern Fragments into the LOD Laundromat.

4.1 INTRODUCTION

In 2001 the Semantic Web promised to provide a distributed and heterogeneous data space, like the traditional Web, that could at the same time be used as a machine-readable Web Services platform [17]. Data providers would open up their knowledge for potentially unanticipated reuse by data consumers. Intelligent agents would navigate this worldwide and heterogeneous data space in order to perform intelligent tasks. In 2015 this promise remains largely unmet.

PROBLEM When we look at empirical data about the rudimentary infrastructure of the Semantic Web today, we see multiple problems: Millions of data documents exist that potentially contain information that is relevant for intelligent agents. However, only a tiny percentage of these data documents can be straightforwardly used by software clients. Typically, online data sources cannot be consistently queried over a prolonged period of time, so that no commercial Web Service would dare to depend on general query endpoint availability and consistency. In practice, Semantic Web applications run locally on self-deployed and centralized triple-stores housing data that has been integrated and cleaned for a specific application or purpose. Meanwhile, the universally accessible and automatically navigable online Linked Open Data Cloud remains structurally disjointed, unreliable, and — as a result — largely unused for building the next generation of large-scale Web solutions.

Those Linked Data providers willing to sacrifice completeness for hardware footprint can choose to use sampling methods such as SampLD (chapter 3). Those publishers who do not choose to do so are left no option than to increase their hardware footprint. The problem here is *sustainability*. While it is technically possible to publish data in a standards-compliant way, many data publishers are unable to do so. While it is technically possible to pose structured live queries against a large dataset, this is prohibitively expensive in terms of both engineering effort and hardware support.

Take for instance the concept of federation, in which a query is evaluated against multiple datasets at the same time. According to the original promise of the Semantic Web, federation is crucial, since it allows an automated agent to make intelligent decisions based on an array of knowledge sources that are both distributed and heterogeneous. In practice, however, federation is extremely difficult [71] since most datasets do not have a live query endpoint; the few query endpoints that do exist often have low availability; the few available live query endpoints sometimes implement constrained APIs which makes it difficult to guarantee that queries are answered in a consistent way (as discussed in chapter 2).

CONTRIBUTIONS This chapter presents a redeployment of the LOD Cloud that makes the Semantic Web queryable on an unprecedented scale, while retaining its originally defined properties of openness and heterogeneity. We provide an architecture plus working implementation which allows queries that span a large number of heterogeneous datasets to be performed. The working implementation consists of a full-scale and continuously updating copy of the LOD Cloud as it exists today. This complementary copy can be queried by intelligent agents, while guaranteeing that an answer will be established consistently and reliably. We call this complementary copy *Linked Data-as-a-Service* (LDaaS).

LDaaS was created by tightly combining two existing state-of-the-art approaches: the LOD Laundromat (see chapter 2) and Linked Data Fragments [106]. While the integration itself is straightforward, we show that its consistent execution delivers a system that is able to meet a wide-spanning array of requirements that have not been met before in both width and depth. This demonstrates the scalability and viability of the LDaaS publication process, and it retrains the same benefits of SPARQL: Linked Data providers are able to publish data without knowing how it will be used a-priori.

CONTENT This chapter is organized as follows: Section 4.2 gives an overview of the core concepts and related work. Section 4.3 details the motivation behind LDaaS. Section 4.4 specifies the architecture and design of LDaaS, which we evaluate in section 4.5. We conclude in section 4.6.

4.2 CORE CONCEPTS & RELATED WORK

4.2.1 *Web Interfaces to RDF Data*

In order to characterize the many possibilities for hosting Linked Datasets on the Web, *Linked Data Fragments* (LDF) [107] was introduced as a uniform view on all possible Web APIs to Linked Data. The common characteristic of all interfaces is that, in one way or another, they offer specific parts of a dataset. Consequently, by analyzing the parts offered by an interface, we can analyze the interface itself. Each such part is called a *Linked Data Fragment*, consisting of:

- **data:** the triples of the dataset that match an interface-specific *selector*;
- **metadata:** triples that describe the fragment;
- **controls:** hyperlinks and/or hypermedia forms that lead to other fragments.

The choices made for each of those elements influence the functional and non-functional properties of an interface. This includes the effort of a server to generate fragments, the cacheability of those fragments, the availability and performance of query execution, and the party responsible for executing those queries.

Using this conceptual framework, we will now discuss several interfaces.

DATA DUMPS File-based datasets are conceptually the most simple APIs: the *data* consists of all triples of the dataset. They are possibly combined into a compressed archive and published at a single URL. Sometimes the archive contains *metadata*, but *controls*—with the possible exception of HTTP IRIs in RDF triples—are not present. Query execution on these file-based datasets is entirely the responsibility of the client; obtaining up-to-date query results requires re-downloading the entire dataset periodically or upon change.

LINKED DATA DOCUMENTS By organizing triples by subject, Linked Data Documents allow to *dereference* the URL of entities.

A document's *data* consists of triples related to the entity (usually triples where the subject or object is that entity). It might contain *metadata* triples about the document (e.g. creator, date) and its *controls* are the URLs of other entities, which can be dereferenced in turn. Linked Data Documents provide a fast way to collect the authoritative information about a particular entity and they are cache-friendly, but predicate- or object-based queries are practically infeasible.

SPARQL ENDPOINTS The SPARQL query language allows to express very precise selections of triples in RDF datasets. A SPARQL endpoint allows the execution of SPARQL queries on a dataset through HTTP. A fragment's *data* consists of triples matching the query (assuming the `CONSTRUCT` form); the *metadata* and *control* sets are empty. Query execution is performed entirely by the server, and because each client can ask highly individualized requests, the cacheability of SPARQL fragments is quite low. This, combined with complexity of SPARQL query execution, likely contributes to the low availability of public SPARQL endpoints [5, 26]. To mitigate this, many endpoints restrict usage, by reducing the allowed query execution time, limiting the number of rows that can be returned or sorted, or not supporting more expensive SPARQL features [26].

TRIPLE PATTERN FRAGMENTS The Triple Pattern Fragments (TPF) API [105] has been designed to minimize server processing, while at the same time enabling efficient live querying on the client side. A fragment's *data* consists of all triples that match a specific triple pattern, and can possibly be paged. Each fragment (page) contains the estimated total number of matches, to allow for query planning, and contains hypermedia controls to find all other Triple Pattern Fragments of the same dataset. The controls ensure each fragment is *self-describing*: just like regular webpages do for humans, fragments describes in a machine-interpretable way what the possible actions are and how clients can perform them. Consequently, clients can use the interface without needing the specification. Complex SPARQL queries are decomposed by clients into Triple Pattern Fragments. Since requests are less granular, fragments are more likely to be reused across clients, improving the benefits of caching [105]. Because of the decreased complexity, the server does not necessarily require a triple-store to generate its fragments.

OTHER SPECIFIC APIS Several APIs with custom fragments types have been proposed, including the Linked Data Platform [97], the SPARQL Graph Store Protocol [76], and other HTTP interfaces such as the Linked Data API[1] and Restpark [69]. In contrast to Triple Pattern Fragments, the fragments offered by these APIs are

1 See `https://code.google.com/p/linked-data-api/`

not self-describing: clients require an implementation of the corresponding specification in order to use the API, unlike the typically self-explanatory resources on the human Web. Furthermore, no query engines for these interfaces have been implemented to date.

4.2.2 *Existing Approaches to Linked Data-as-a-Service*

Sindice and LODCache, both discussed in chapter 2, provide access to Linked Datasets via a centralized interface. Sindice does not support downloads of raw data versions, and access is granted through a customized API. The LODCache SPARQL endpoint suffers from issues such as low availability, presumably related to its enormous size of more than 50 billion triples. There is no transparent procedure to include data manually or automatically. Given the focus on size, its main purpose is likely to showcase the scalability of the Virtuoso triple-store, rather than providing a sustainable model for Linked Data consumption on the Web.

Other initiatives, such as Europeana [50], aggregate data from specific content domains, and allow queries through customized APIs.

DyLDO [60] is a long-term experiment to monitor the dynamics of a core set of 80 thousand Linked Data documents on a weekly basis. Each week's crawl is published as an N-Quads file. This work provides interesting insight in how Linked Data evolves over time. It is not possible to easily select triples from a *single* dataset, and not all datasets belonging to the Linked Data Cloud are included. Another form of incompleteness stems from the fact that the crawl is based on IRI de-referencing, not guaranteeing datasets are included in their entirety.

Finally, Dydra[2] is a cloud-based RDF graph database, which allows users without hosting capabilities to publish RDF graphs on the Web. Via their Web interface, Dydra provides a SPARQL endpoint, the option to configure permissions, and other graph management features. However, access to Dydra is limited: free access is severely restricted, and there are no public pay plans for paid services.

4.3 MOTIVATION

In this section, we motivate why there is a need for an alternative deployment of the Semantic Web, and why we opt for a Linked Data-as-a-Service approach.

2 See `http://dydra.com`

4.3.1 *Canonical form*

One of the biggest hurdles towards Web-scale live querying is that — at the moment — Semantic Web datasets cannot all be queried in the same, uniform way (Problem 1).

Problem 1 *In practice, there is no single, uniform way in which the LOD Cloud can be queried today.*

First of all, most Semantic Web datasets that are available online are data dumps [33, 56], which implies that they cannot be queried live. In order to perform structured queries on such datasets, one has to download the data dumps and deploy them locally. Secondly, –as discussed in chapter 2– many data dumps that are available online are not fully standards-compliant. This makes the aforementioned local deployment relatively difficult, since it requires the use of tools that can cope with archive errors, HTTP errors, multiple syntax formats, syntax errors, etc. Thirdly, not all datasets that *can* be queried live use a standardized query language (such as SPARQL). Indeed, some require a data consumer to formulate a query in a dedicated query language or to use a custom API. Fourthly, most custom APIs are not self-describing, making it relatively difficult for a machine processor to create such queries on the fly. Fifthly, most online datasets that can be queried live and that are using standardized query languages such as SPARQL are imposing restrictions on queries that can be expressed and results that can be returned [5, 26]. Finally, different SPARQL endpoints impose *different* restrictions [26]. This makes it difficult for a data consumer to predict whether, and if so how, a query will be answered. The latter point is especially relevant in the case of federated querying (see Section 4.3.4), where sub-queries are evaluated against multiple endpoints with potentially heterogeneous implementations.

For the last decade or so, Problem 1 has been approached by creating standards, formulating guidelines, and building tools. In addition, Semantic Web evangelists have tried to educate and convince data producers to follow those guidelines and use those tools. This may still be the long-term solution. However, we observe that this approach has been taken for over a decade, yet leading to the heterogeneous deployment described above. We therefore introduce the complementary Solution 1 that allows all Semantic Web data to be queried live in a uniform way and machine-accessible way.

Solution 1 *Allow all Semantic Web documents to be queried through a uniform interface that is standards-compatible and self-descriptive.*

4.3.2 *Scalability & availability*

After the first 14 years of Semantic Web deployment there are at least millions of data documents [30,39] but only 260 live query endpoints [26]. Even though the number of endpoints is growing over time [5, 26], at the current growth rate, the gap between data dumps and live queryable data will only increase (Problem 2). The number of query endpoints remains relatively low compared to the number of datasets, and many of the endpoints that do exist suffer from limited availability [26].

Problem 2 *Existing deployment techniques do not suffice to close the gap between the Web of downloadable data documents and the Web of live queryable data.*

Several causes contribute to Problem 2 –a specific instantiation of problem 2 from chapter 1. Firstly, it is difficult to deploy Semantic Web data, since this currently requires a complicated stack of software products. SampLD –presented in the previous chapter– reduces the hardware footprint for these deployments, but comes at the cost of completeness. Secondly, existing query endpoints perform most calculations on the server-side, resulting in a relatively high cost and thus a negative incentive for the data publisher. Thirdly, in the presence of dedicated query languages, custom APIs, and restricted SPARQL endpoints, some have advocated to avoid SPARQL endpoints altogether, recommending the more flexible data dumps instead, thereby giving up on live querying. Solution 2 addresses these causes.

Solution 2 *Strike a balance between server- and client-side processing, and automatically deploy all Semantic Web data as live query endpoints. If clients desire more flexibility, they can download the full data dumps as well.*

4.3.3 *Linked Data-as-a-Service*

Even though software solutions exist to facilitate an easy deployment of various Web-related services such as email, chat, file sharing, etc., in practice users gravitate towards centralized online deployments (e.g., Google and Microsoft mail, Facebook chat, Dropbox file sharing). We observe similar effects in the (lack of) popularization of Semantic Web technologies (Problem 3). Even though multiple software solutions exist for creating, storing, and deploying Semantic Web services (e.g., RDF parsers, triple-stores, SPARQL endpoints), empirical observations indicate that the deployment of such services with existing solutions has been problematic [56]. As a consequence, live querying of Semantic Web data has not yet taken off in the same way as other Web-related tasks have.

Problem 3 *Even though a technology stack for publishing Semantic Web data exists today, there is currently no simplified Web Service that does the same thing on a Web-scale.*

While technologies exist that make it possible to publish a live query endpoint over Semantic Web data, there is currently no simplified Web *Service* that allows data to be deployed on a very large scale. Under the assumption that take-up of traditional Web Services is an indicator of future take-up of Semantic Web Services (an assumption that cannot be proven, only argued for), it follows that many data publishers may prefer a simplified Web Service to at least perform some of the data publishing tasks (Solution 3).

Solution 3 *Provide a service to take care of the tasks that have proven to be problematic for data publishers, having an effective cost model for servicing a high number of data consumers.*

4.3.4 *Federation*

In a federated query, sub-queries are evaluated by different query endpoints. For example, one may be interested in who happens to know a given person by querying a collection of HTML files that contain FOAF profiles in RDFa. At present, querying multiple endpoints is problematic (Problem 4), because of the cumulating unavailability of individual endpoints, as well as the heterogeneity of interfaces to Linked Data.

Problem 4 *On the current deployment of the Semantic Web it is difficult to query across multiple datasets.*

Given the heterogeneous nature of today's Semantic Web deployment (Section 4.3.1), there are no LOD Cloud-wide guarantees as to whether, and if so how, sub-queries will be evaluated by different endpoints. In addition, properties of datasets (i.e., metadata descriptions) may be relevant for deciding algorithmically which datasets to query in a federated context. Several initiatives exist that seek to describe datasets in terms of Linked Data (e.g., VoID, VoID-ext, Bio2RDF metrics, etc.). However, such metadata descriptions are often not available, and oftentimes do not contain enough metadata in order to make efficient query federation possible.

Solution 4 *Allow federated queries to be evaluated across multiple datasets. Allow metadata descriptions to be used in order to determine which datasets to query.*

4.4 WORKFLOW & ARCHITECTURAL DESIGN

The scale of the LOD Cloud requires a low-cost data publishing workflow. Therefore, the LOD Laundromat service is designed as a (re)publishing platform for data dumps, i.e. data files. As detailed in Section 4.2.1, data dumps are the most simple API that can be offered. To allow structured live querying, while still maintaining technical and economical scalability, we have integrated the low-cost TPF API.

We first discuss the publishing workflow supported by the combination of the LOD Laundromat and Triple Pattern Fragments. We then elaborate on the architectural design of their integration, and how we improved both approaches to keep LDaaS scalable.

4.4.1 *Re-publishing Workflow*

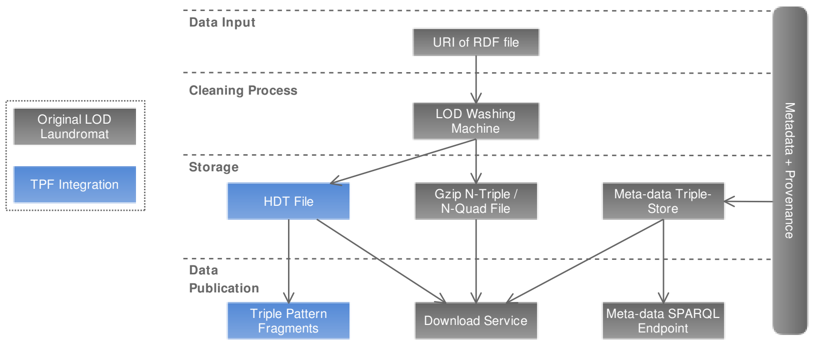


Figure 4.1: LOD Laundromat (re)-Publishing Workflow

Figure 4.1 shows the re-publishing workflow of the LOD Laundromat, extended with the Triple Pattern Fragment API. Here, we see how the LOD Laundromat (1) takes a reference to an online RDF document as input, (2) cleans the data in the LOD Washing Machine, (3) stores several representations of the data, and publishes the data through several APIs. The original workflow of LOD Laundromat was already presented in chapter 2. There, we discuss the workflow steps related to the LDaaS extension specifically.

STORAGE Next to compressed LOD Laundromat Gzip files, the datasets are stored as 'Header, Dictionary Triples' (HDT) files as well. HDT files are compressed, indexed files, in a binary serialization format. HDT files are suitable for browsing and querying RDF data without requiring to decompression and/or ingestion into a triple-store [35].

DATA PUBLICATION The LOD Wardrobe[3] module provides several APIs to access the data generated by the LOD Laundromat.

The first API supports complete control over the data: the HDT and cleaned compressed N-Triples/N-Quads files are available for download. An RDF dump of the LOD Laundromat metadata is available for download as well. The HDT files allows users to down-

3 See http://lodlaundromat.org/wardrobe/

load datasets and either query them directly on the command-line (via triple-pattern queries), or to publish these via a Triple Pattern Fragment API. The low-level access to the compressed N-Triples/N-Quads files allow bulk processing of such files, particularly considering the advantages that come with this canonical format: streamed processing of a sorted set of statements.

The TPF API provides access via triple pattern queries, and uses HDT files as storage type. This low-cost API, discussed in Section 4.2, enables structured querying on the crawled datasets.

Finally, the third API is a SPARQL endpoint, which provides SPARQL access to the metadata triple-store.

4.4.2 *LDaaS Architectural Design*

The architecture for crawling cleaning Linked Data is described in chapter 2, where the architecture of TPF is described in [105]. Below we discuss the measures we took to combine both systems, and the improvements we made to the scalability of LDaaS.

TPF HORIZONTAL SCALABILITY

The HDT library can read and query HDT files that are larger than main memory by loading them as *memory-mapped files*. This means the file is mapped byte-by-byte to pages of virtual memory of the application, and regions of the file are swapped in and out by the operating system as needed. This is not horizontally scalable though: although this approach works for 30–40 large datasets, in practice, processes with hundreds or thousands of memory-mapped files tend to become unstable.

Therefore, we extended the TPF architecture with an alternative strategy in which only very large HDT files ($\geqslant 10$ GB) are mapped to memory. In order to query the remaining majority of smaller files, an out-of-process approach is used. When an HTTP request arrives, the server spawns an external process that briefly loads the corresponding HDT file, queries the requested triple pattern, and closes it again. While this involves a larger delay than if the file were mapped to the server process, the overhead is limited for such smaller files because of the efficient HDT index format, and it guarantees the server process' stability. The few large files are still memory-mapped, because spawning new processes for them would result in a noticeably longer delay.

HDT FILE GENERATION

The original LOD Laundromat architecture creates and serves clean compressed Gzipped N-Quads and N-Triples files. To support the use of a TPF API, we extended this implementation by generating HDT files as well. The HDT files are automatically generated based on the clean compressed Gzipped N-Quads and N-Triples files. Because the latest implementation of HDT does not support named graphs, the N-Quads files are processed as regular triples, without the specified graph.

ADDING TPF DATASETS EFFICIENTLY Datasets crawled by the LOD Laundromat should become available via the TPF API in a timely manner. The original TPF API applies several 'sanity checks' on the data documents before hosting them. However, with 650,000 documents in the configuration file, this process requires minutes of processing time. Because the LOD Laundromat pipeline guarantees 'sane' HDT files, we avoid this issue by extending the TPF API with an optimized loading procedure which disables these sanity checks. As a result, re-loading the configuration whenever a new dataset is cleaned, requires seconds instead of minutes.

4.5 EVALUATION

We use the architecture described in the previous section to present a working implementation where we publish the LOD Cloud via Triple Pattern Fragments. In this section, we evaluate this deployment and validate the solutions from Section 4.3.

Solution 1 *Allow all Semantic Web documents to be queried through a uniform interface that is standards-compatible and self-descriptive.*

Solution 1 is evaluated analytically. Currently, 650,000 datasets are hosted as live query endpoints. Although this does not include all existing Semantic Web data, these numbers show that our approach can realistically be applied on Web scale (see Solution 3 for usage numbers).

Since the Triple Pattern Fragments APIs are generated for all data in the LOD Wardrobe, data queryable by LDaaS inherits the completeness and *data* standards-compliance properties of the LOD Laundromat for these compliance properties). *Query* standards-compliance — on the other hand — is attained only partially, since the server-centric paradigm of the SPARQL specification is purposefully deviated from in the current approach in order to fulfill Solution 2.[4] This primarily involves those parts of the SPARQL standard that require the Closed World Assumption (something the authors consider to be at odds with the basic tenets of Semantic Web philosophy) and common data manipulation functions that can be easily implemented by a client (e.g., sorting a list, calculating a maximum value).

The Linked Data Fragments API is self-descriptive, employing the Hydra vocabulary for hypermedia-driven Web APIs.[5] Hydra descriptions allow machine processors to detect the capabilities of the query

4 Even though there is a client-side rewriter that allows SPARQL queries to be performed against an LDF server backend, the standards-compliance of this rewriter is not assessed in this work.

5 See http://www.hydra-cg.com/spec/latest/core/

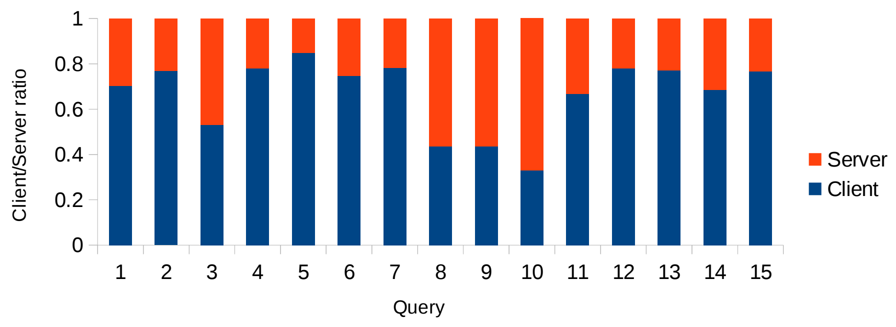


Figure 4.2: Processing time is shared between client and server.

endpoints in an automated way. In addition, the LDaaS query endpoints do not impose restrictions on the number of operations that may be performed or the number of results that can be retrieved. This allows full data graphs to be traversed by machine processors. Also, pagination is implemented in a reliable way, as opposed to SPARQL endpoints which cannot guarantee consistency with shifting `LIMIT` and `OFFSET` statements.

Finally, uniformity is guaranteed on two-levels: data and interface. The former leverages the LOD Laundromat infrastructure as an enabler for homogeneous deployment strategies. Thus, when an agent is able to process one data document, it is also able to query 600K+ data documents. The latter denotes that through Triple Pattern Fragments, processing queries only relies on HTTP, the uniform interface of the Web. Queries are processed in exactly the same way by all endpoints, in contrast to the traditional Semantic Web deployment where different endpoints implement different standards, versions or features.

Solution 2 *Strike a balance between server- and client-side processing, and automatically deploy all Semantic Web data as live query endpoints. If clients desire more flexibility, they can download the full data dumps as well.*

The SPARQL protocol relies on servers to do the heavy lifting: the complete computational processing is performed on the server, and the client is only responsible for sending the request and receiving the SPARQL results. The TPF API, used by LDaaS, takes a different approach. Executing SPARQL queries on the TPF API requires the client to perform joins between triple patterns, and e.g. apply filters or aggregations. As a result, the computational processing is shared between the client and the server, putting less strain on the server.

To quantify this balancing act between server and client-side processing of LDaaS, we evaluated a set of queries from the SP^{2}B benchmark, on a (synthetic) dataset of 10 million triples[6], added to the LOD

6 Experiments showed that these results do not differ greatly between SP^{2}B datasets of different sizes

Laundromat. We measure the client-side and server-side processing time, both running on the same hardware, and excluding network latency. The results, shown in figure 4.2, confirm that the computation is shared between client and server. More specifically, the client does most of the processing for the majority of these SP^2B SPARQL queries.

Solution 3 *Provide a service to take care of the tasks that have proven to be problematic for data publishers, having an effective cost model for servicing a high number of data consumers.*

Apart from facilitating common tasks (cleaning, ingesting, publishing), the LOD Laundromat operates under a different cost model than public SPARQL endpoints. Since its launch, the LOD Laundromat served more than 4,343 users who downloaded 7,909,738 documents and who issued more than 12,696,797 TPF API requests.

We consider the hardware costs of disk space and RAM usage below.

disk space Currently, 649,834 datasets (38,606,408,433 triples) are hosted as Triple Pattern Fragments. The required storage is 656GB in the compressed HDT format, or on average 1.03MB per document or 18.24 bytes per triple. The disk space used to store the equivalent gzip-compressed N-Triples (or N-Quads) files is 313GB (0.49MB per document or 8.70 bytes per triple). Such compressed archives do not allow for efficient triple-pattern queries, which the HDT files can handle at high speed.

memory usage The TPF server consists of 10 independent worker processes. Because JavaScript is single-threaded, it does not have a concurrency policy for memory access, so each worker needs its own space to allocate resources such as the metadata for each of the 649,834 datasets. However, no further RAM is required for querying or other tasks, since they are performed directly on the HDT files. We have allocated 4 GB per worker process, which was experimentally shown to be sufficient, bringing the total to 40 GB of RAM.

Solution 4 *Allow federated queries to be evaluated across multiple datasets. Allow metadata descriptions to be used in order to determine which datasets to query.*

Finally, we ran FedBench [93] to test the employability of the resulting TPF interfaces for answering federated SPARQL queries. A total of 9 datasets[7], excluding the isolated SP^2B dataset, were added to the LOD Laundromat, completing our publishing workflow. Also, we extended the existing TPF client to distribute each fragment request to a predefined list of interfaces and aggregate the results.

7 https://code.google.com/p/fbench/wiki/Datasets

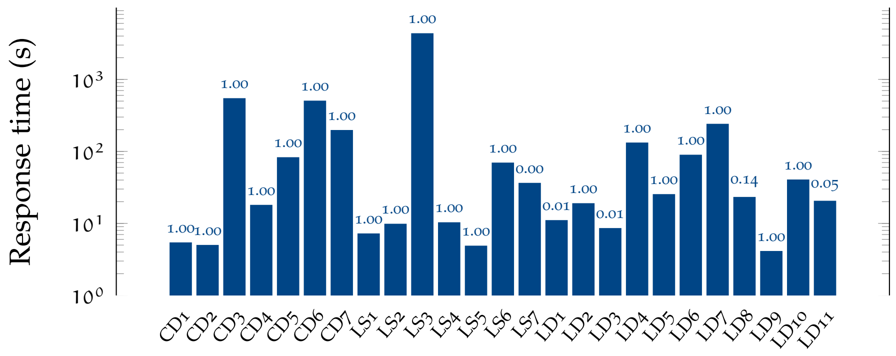


Figure 4.3: All FedBench queries complete slowly, but successfully, with high average recall (shown on top of each bar) when ran on the deployed LDaaS.

We executed the *Cross Domain (CD)*, *Linked Data (LD)*, and *Life Science (LS)* query sets in three runs, directly on `ldf.lodlaundromat.org` from a desktop computer on an external high-speed university network. Fig. 4.3 shows the average execution time for each query and the corresponding recall. All queries were successfully completed with an average result recall of 0.81, which confirms the ability to evaluate federated queries. The imperfect recall is a result of an occasional request timeout in queries (LS7, LD1, LD3, LD8, LD11), which, due to limitations of the current implementation, can drop potential results. Next, general execution time is magnitudes slower compared to state-of-the-art SPARQL Endpoint federation systems [90]. However, this is expected considering *a*) the LDF paradigm which sacrifices query performance for low server cost, and *b*) the greedy implementation where the set of sent HTTP requests is a naive Cartesian product between the set of fragments and the datasets. Nevertheless, several queries (LD9, LS5, CD2, CD1, LS1, LD3, LS2) complete within 10s, which is promising for future development in this area.

4.6 CONCLUSION

After the first 14 years of Semantic Web deployment the promise of a single distributed and heterogeneous data-space remains largely unfulfilled. Although RDF-based data exists in ever-increasing quantities, large-scale usage by intelligent software clients is not yet a reality. In other words, hosting data online is easy, but publishing Linked Data via a queryable API such as SPARQL appears to be too difficult.

To decrease the Linked Data providers' hardware costs (see problem 2 from chapter 1), we reduced the dataset size in chapter 3 using SampLD. In this chapter, we reduced the query complexity instead. We formulated sustainable solutions, which we proposed and implemented as a redeployment architecture for the Linked Open Data cloud. By combining the LOD Laundromat with a low-cost server-side interface (Triple Pattern Fragments), we were able to realize this with minimal engineering.

In doing so, we *a*) closed the API gap by providing low-cost structured query capabilities to otherwise static datasets; *b*) did so via a uniform, self-descriptive, and standards-compatible interface; *c*) enabled in turn federated queries across a multitude of datasets, and *d*) provide a service for Linked Data providers to use. More important than the deployment we provide is the wide applicability of the open source technology stack, whose architecture is detailed in this chapter.

However, this queryable LOD Laundromat API is not a full replacement for the SPARQL query language: the simplicity of TPF makes it technologically scalable, but reduces the query complexity to a large extend. TPF is sufficient for e.g. resource look-ups and dataset browsing. But efficiently querying for aggregate information, or using a multitude of joins, still requires the more comprehensive SPARQL language. Therefore, we see the TPF API and the SPARQL protocol as two complementary approaches.

LINKED DATA DEVELOPER

5 THE YASGUI FAMILY OF SPARQL CLIENTS

The previous chapter detailed how we deployed the LOD Laundromat documents via Triple Pattern Fragments. Where the scalability of Triple Pattern Fragments has its benefits, SPARQL offers more flexibility and is considered the de-factor standard. Because the size and complexity of Linked Data and its technology stack makes it difficult to query, access to Linked Data via SPARQL could be greatly facilitated if it were supported by a query editor with a strong focus on usability.
In this chapter we present the YASGUI family of SPARQL clients, that together compose a robust, feature-rich and developer friendly SPARQL editor. The modular approach of YASGUI goes beyond the intended user group of developers, and also enables Linked Data providers to re-use YASGUI components and improve access to their endpoints.
We show that over the past years the YASGUI family has had significant impact on the landscape of Linked Data management and consumption: YASGUI components are integrated in state-of-the-art triple-stores and Linked Data applications, and used as front-end by a large number of Linked Data publishers. Additionally, we show that the YASGUI web service – which provides access to any *SPARQL endpoint – has a large and growing user base among Linked Data consumers.*

This chapter is an updated version of:

Laurens Rietveld and Rinke Hoekstra. The YASGUI Family of SPARQL Clients. *Semantic Web Journal*, 2015

Contributions to this paper:

I am the main designer and creator of the YASGUI clients and main author of the paper

5.1 INTRODUCTION

Web developers can rely on advanced development tools such as in-browser debugging, integrated development environments, high-level libraries in all but the most austere programming languages, and increasingly simple and lightweight web-services. These are essential enablers for broad take up in industry, and vice versa the use of web technology in industry drives the development of ever more developer-friendly tools. Semantic Web and Linked Data technologies have some catching up to do, and steps in this direction are under way. An example is the recent start of the W3C Linked Data Platform working group[1] that aims to bring triple-store querying closer to the RESTful paradigm.

PROBLEM Several good Linked Data programming libraries exist, but uptake of these still relies on a thorough understanding of SPARQL and the other members of the Semantic Web technology stack. The situation for SPARQL is worse. Existing SPARQL clients convey a rather narrow interpretation of what a SPARQL client interface should do: POST (or GET) a SPARQL query string to an endpoint URL. As a result, these implementations do not offer functionality that goes far beyond a simple HTML form (see section 5.2).

The curious developer or potential enthusiast who wants to have a first taste of Linked Data is easily scared away by the current set of SPARQL clients. For Semantic Web developers designing and testing SPARQL queries is often a cumbersome and painful experience: "All who know the RDF namespace URI by heart raise their hands now!", and "Where is that Linked Data?". Many will know the DBpedia endpoint URL, but can perhaps recall only a handful of endpoints in total.

Existing clients offer only a small selection of the features that we, as a community, could offer to both ourselves as well as new users of Semantic Web technology (See problem 3 from the introduction). We propose to overcome this hurdle by means of simple, lightweight and user friendly clients for interacting with Linked Data that integrate with existing services in the field.

CONTRIBUTIONS This was our main motivation for designing and building Yet Another SPARQL GUI (YASGUI), [2] first introduced in a workshop paper [84]. YASGUI is a web-based SPARQL client that can be used to query both remote and local endpoints. It integrates linked data services and web APIs to offer features such as auto-completion and endpoint lookup. It supports query retention – query texts persist across sessions – and query 'permalinks', as well as syntax checking and highlighting. YASGUI is easy to deploy locally, and it is robust. Because of its dependency on third party services, we have paid extra attention to graceful degradation when these services are inaccessible or produce unintelligible results. The YASGUI family of SPARQL clients gives Linked Data developers a robust, feature-rich and user friendly SPARQL editor, and enables Linked Data providers to improve ease of access for their SPARQL endpoints.

Since its release in 2013, YASGUI has grown into a family of reusable components, and now includes two new, fully client side components (YASQE and YASR) that can be used independently. YASGUI has had considerable impact in the field. Our components

1 See `http://www.w3.org/2012/ldp`
2 See `http://yasgui.org`

have found their way into several third-party tools, and are now incorporated in three popular triple-stores. YASGUI is currently used by several important data providers, and has shown to provide a useful information source for further research (see chapter 8).

CONTENT This chapter is structured as follows. Section 5.2 provides an overview of the features present in the current state of the art in SPARQL user interfaces. Section 5.3 compares and explains the features and design considerations of YASGUI, and its YASQE and YASR components. Impact of the YASGUI family is discussed in section 5.4. We conclude in section 5.5.

5.2 STATE OF THE ART IN SPARQL USER INTERFACES

The features of SPARQL clients can be categorized under three main headers: *syntactic* features (auto-completion, syntax highlighting and validation), *applicability* features (endpoint or platform dependent/independent) and *usability* (query retention, results rendering and download, quick evaluation). Table 5.1 lists seventeen SPARQL clients – that range from very basic to elaborate – and depicts what features they implement. This section describes these features in more detail, and discusses whether and how the clients of Table 5.1 implement these features.

We excluded query interfaces that were not fully reproducible (SPARQLinG [57], ViziQuer [111], SPARQLViz [24] and NITELIGHT [96]) or only cover a subset of the SPARQL standard (iSPARQL [3])

5.2.1 *Syntactic Features*

Most modern applications that feature textual input support some form of *auto-completion*. Examples are the Google website which shows an auto-completion list for your search query, or your browser which (based on forms you previously filled in) shows auto-complete lists for text inputs. One advantage of auto-completion is that it saves you from writing the complete text. Another advantage is the increase in transparency, as the auto-completion suggestions may contain information the user was not aware of. The latter is particularly interesting for SPARQL, where users might not always know the exact namespace prefix they would like to use, or where the user might not know all available properties in a triple-store. Several SPARQL interfaces offer naive auto-completion functionalities, such as the Flint SPARQL Editor[4] which auto-completes SPARQL syntax and functions. Other interfaces offer more auto-completion functionali-

3 See http://dbpedia.org/isparql/
4 See http://openuplabs.tso.co.uk/demos/sparqleditor

Feature	4Store	OpenLink Virtuoso	Stardog	ClioPatria	SNORQL	SPARQLer	Apache Jena	Sesame Workbench	Sesame2 Windows Client	TopBraid Composer	Glint	Twinkle	SparqlGUI	SparQLed	Gosparqled	Squebi	Flint SPARQL Editor	YASGUI Family
Syntactic features																		
Auto-completion	-	-	-	+	-	-	+	-	-	-	-	-	-	+	+	+	+	+
Syntax Highlighting	-	-	+	+	-	-	+	+	-	+	+	-	-	+	+	+	+	+
Syntax Validation	-	-	-	+	-	-	+	+	-	-	-	-	-	+	+	-	+	+
Applicability features																		
Multiple Endpoints	-	-	-	-	-	-	-	-	+	-	+	+	± [a]	-	-	± [a]	± [a]	+
Platform independent	+	+	+	+	+	+	+	+	-	+	-	+	-	+	+	+	+	+
Available as library	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+
Usability features																		
Query retention	-	-	-	+	-	-	+	+	+	-	+	-	+	-	+	-	-	+
File upload	-	-	+	+	-	-	+	+	± [b]	+	-	+	+	-	-	-	-	- [c]
Results rendering	-	± [d]	+	+	+	± [d]	+	+	± [d]	+	± [d]	± [d]	± [d]	+	+	+	+	+
Chart Visualizations	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
Results download	+	+	+	+	+	+	+	+	+	+	+	+	+	-	+	+	-	+
Endpoint Search	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+

a Can deal with a limited number of endpoints, e.g. only CORS enabled ones.
b File upload requires a local triple store that implements the OpenRDF SAIL API, e.g. OpenRDF Sesame or OpenLink Virtuoso.
c File upload is a planned feature, using cloud triple-store services (e.g. dydra.com)
d The rendering does not use hyperlinks for URI resources.

Table 5.1: SPARQL client feature matrix

ties using external APIs, such as Squebi[5] for prefix auto-completion, and ClioPatria[6] and Apache Jena[7] for prefix and property/class auto-completions. Editors such as SparQLed [28] and Gosparqled[8] offer even more reliable auto-completions of resources, though this requires a dedicated back-end server.

Syntax highlighting is a functionality that most programming language editors have. It allows users to distinguish between different elements of the language: properties, variables, strings, etc. The same advantage can hold for query languages such as SPARQL, where you would like to distinguish between literals, IRIs, query variables, function calls, etc. The few SPARQL editors that support syntax highlighting are the Flint SPARQL Editor (or its derivatives) and Squebi, which both use the CodeMirror JavaScript library[9] to bring color to SPARQL queries.

Most Integrated Development Environments (IDEs) provide feedback when code contains syntax errors (i.e. runtime *syntax validation*). Feedback is immediate, which means the user can spot syntax errors in the code without having to execute them. Again, such functionality is useful for SPARQL editing as well. Immediate feedback on a SPARQL syntax means the user can spot invalid queries without having to execute it on a SPARQL endpoint. The quality of such feedback is often better compared to endpoint error messages: an IDE can pinpoint the error location in the user interface, where the returned errors from endpoints (depending on the triple-store) can differ greatly in both specificity and quality. The Flint, SparQLed, Gosparqled, Sesame Workbench, Apache Jena and ClioPatria SPARQL editors support immediate live syntax checking by means of JavaScript SPARQL parsers.

5.2.2 *Applicability Features*

There are only six clients that allow access to multiple endpoints. Most triple-stores provide a client interface, linking to that specific endpoint. They are *endpoint dependent*. Examples are 4Store [46], ClioPatria, StarDog[10], Apache Jena, OpenLink Virtuoso[11], OpenRDF Sesame Workbench [25] and SPARQLer[12]. More generic endpoint *independent* clients are the Sesame2 Windows Client [25], Glint[13],

5 See `https://github.com/tkurz/squebi`
6 See `http://cliopatria.swi-prolog.org/`
7 See `https://jena.apache.org/`
8 See `https://github.com/scampi/gosparqled`
9 See `http://codemirror.net/`
10 See `http://stardog.com/`
11 See `http://virtuoso.openlinksw.com/`
12 See `http://www.sparql.org/`
13 See `https://github.com/MikeJ1971/Glint`

Twinkle[14] and SparqlGUI[15]. Other applications only provide access to *some* SPARQL endpoints. The Flint SPARQL Editor and Squebi only connect to endpoints that are CORS enabled (i.e. support cross-domain JavaScript access). This is a problem because we observe that 38% of all available endpoints[16] are *in*-accessible via cross-domain JavaScript. Other editors support only XML or JSON as query results, such as SNORQL[17] (part of D2RQ [21]), which only supports query results in SPARQL/JSON format.

Platform In-dependence increases the accessibility of a SPARQL client. The user can access the client on any operating system. Web interfaces are a good example, as a site should work on any major browser (Internet Explorer/Firefox/Chrome), and at least one of these browsers is available for any type of common operating system. Examples are the SPARQL interfaces of Virtuoso, 4Store and the Gosparqled. Another example of multi-platform support is the use of a .jar file (e.g. Twinkle), as all major operating systems support java. Examples of single-platform applications are Sesame2 Windows Client and SparqlGUI: they require Windows.

Interfaces that are open-source and *available as standalone library*, are easy to integrate into other projects and libraries. Most of the presented interfaces are either closed source, or not published as an independent library. The SPARQL interfaces that do enable such re-use, are SparQLed, Gosparqled, Squebi and the Flint SPARQL Editor.

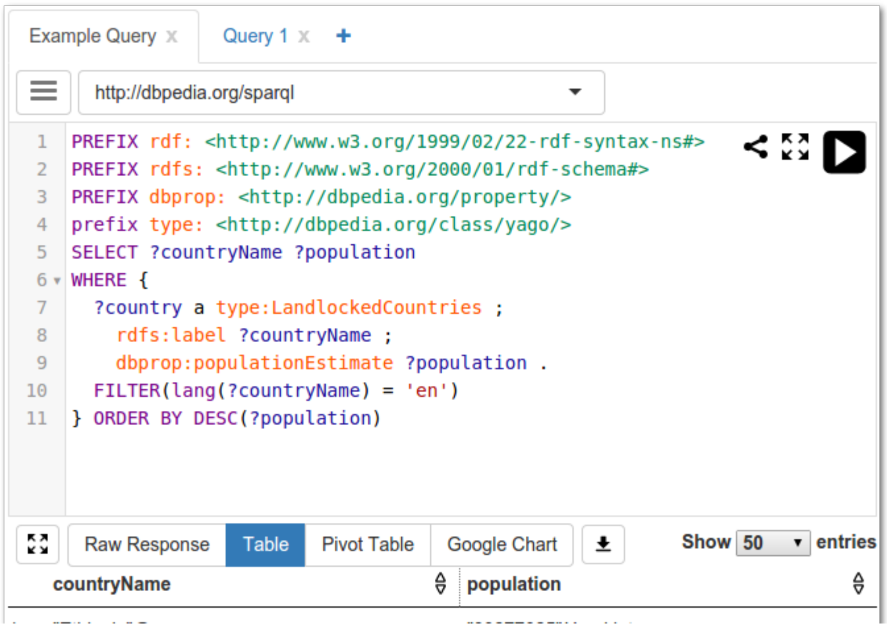
5.2.3 *Usability Features*

Query retention enables re-use of important or often used queries, and allows users to close the application, and resume working on the query later.

Quick evaluation or testing of a graph generated by the user should not require the hassle of installing a local triple-store. Ideally, this functionality would be embedded in the SPARQL client application itself. Most applications that require a local installation on the users computer support this feature, such as Twinkle. The Sesame Windows Client supports file uploads as well, though it requires a local triple-store that implements the OpenRDF SAIL API.

Query results (such as JSON or XML) for `SELECT` queries are often relatively difficult to read and interpret, especially for a novice. A *rendering* method which is easy to interpret and understand is a table. All applications except 4Store support the rendering of query results into a table. Because of the use of persistent URIs, we would expect navigable results for resources, e.g. in the form of drawing the URIs

14 See http://www.ldodds.com/projects/twinkle/
15 See http://www.dotnetrdf.org/content.asp?pageID=SparqlGUI
16 See sparqles.okfn.org
17 See https://github.com/kurtjx/SNORQL/



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbprop: <http://dbpedia.org/property/>
prefix type: <http://dbpedia.org/class/yago/>
SELECT ?countryName ?population
WHERE {
  ?country a type:LandlockedCountries ;
    rdfs:label ?countryName ;
    dbprop:populationEstimate ?population .
  FILTER(lang(?countryName) = 'en')
} ORDER BY DESC(?population)
```

Figure 5.1: The YASGUI interface

as hyperlinks. This feature is not supported by some applications, such as Virtuoso, Twinkle or SparqlGUI.

Rendering the results in a tabular fashion might not suit every use case. Instead, aggregating and visualizing the SPARQL results as *charts* may be preferable. Existing Chart solution exists, such as SgVizler [95] (which indirectly uses Google Charts and D3.js), but none of the existing SPARQL editors support such drawing of charts.

Downloading the results as a file allows for better re-use of these results. A user might want to avoid running the same heavy query more than once, and store the results locally instead. Additionally, the results of `CONSTRUCT` queries are often used in other applications or triple-stores. Saving the user from needing to copy & paste query results clearly improves user experience as well. The only applications that do not support the downloading of results, are the Flint SPARQL editor and SparQLed.

Most of the clients described above are restricted to one simple task: accessing information behind a SPARQL endpoint. However, equally important to this task is assisting the user in doing so. Looking at the table, the most elaborate editors are Squebi, Flint, Gosparqled and SparQLed. However, these all fall short in usability features, and most importantly: the ability to access *any* SPARQL endpoint. We conclude that currently no single endpoint independent, accessible, user-friendly SPARQL client exists.

5.3 THE YASGUI FAMILY

The preceding section shows how current SPARQL clients fall short in supporting Linked Data access. This is both a *publisher* and *consumer* problem. From a consumer perspective, Linked Data access is difficult because available SPARQL interfaces simply do not suffice. Publishers face the problem that no SPARQL interface libraries exist that would facilitate access and lower the threshold for the potential users of their data.

This section presents the open source[18] YASGUI family of SPARQL clients, consisting of components targeted at both publishers and consumers. The main component is a rewritten, modularized and extended version of YASGUI, first published in [84]. YASGUI is a user-friendly web-based interface for interacting with any SPARQL endpoint. It is targeted towards consumers of linked data, and is available online at `http://yasgui.org`.

For publishers, we provide three JavaScript packages: the complete YASGUI interface, the part of YASGUI responsible for writing the SPARQL query (YASQE, or 'Yet Another SPARQL Query Editor'), and the part of YASGUI responsible for visualizing the SPARQL results (YASR, or 'Yet Another SPARQL Result-set visualizer). To increase the ease of integration by publishers and developers, all the JavaScript libraries are available via the NodeJS Package Manager (NPM), the JavaScript dependency manager Bower, and via the JsDelivr[19] and CDNjs[20] Content Delivery Networks.

Below, we first discuss the YASQE and YASR components, the used technology and services, and how the features of both JavaScript libraries compare to the tools presented in the previous section. We then present how both libraries are combined to form the YASGUI library.

5.3.1 *YASQE*

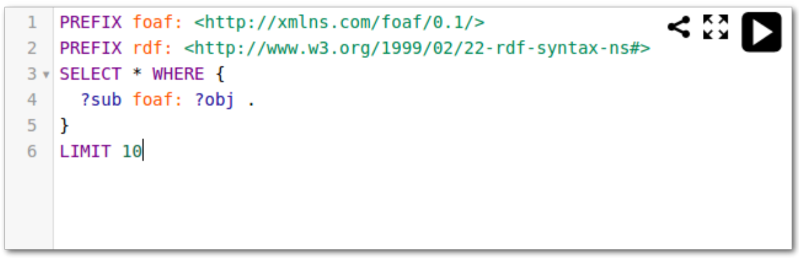

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT * WHERE {
  ?sub foaf: ?obj .
}
LIMIT 10
```

Figure 5.2: The YASQE interface

18 See `https://github.com/YASGUI`
19 See `http://www.jsdelivr.com/`
20 See `https://cdnjs.com/`

YASQE[21] (See Figure 5.2) is an extensive JavaScript library, targeted at Semantic Web publishers. YASQE takes a simple HTML text area, and – with one JavaScript command – transforms it into a full featured IDE-like SPARQL query editor.

YASQE is based on the CodeMirror JavaScript library[22], an extensive HTML text editor. Using CodeMirror and the JavaScript SPARQL grammar from the Flint SPARQL Editor, YASQE is able to tokenize, highlight, validate, and dissect SPARQL queries. If needed, users are presented with immediate validation errors of their queries, and information on the type of validation error. Additionally, YASQE provides several *auto-completion* services: Full namespace IRIs are completed as you type, using the Prefix.cc web service. Properties and classes are auto-completed as well, using the Linked Open Vocabularies [9] (LOV) API.

Using HTML 5 functionalities, YASQE stores that application state, making it *persistent* between user sessions: a returning user will see the screen as it was when she last closed the YASQE browser page.

Furthermore, YASQE provides query *permalink* functionality: For a given query, YASQE generates a link. Opening the link in a browser opens YASQE with the specified query filled in. We believe this is a welcome feature for people working together with a need to share queries.

Finally, YASQE has built-in support for submitting SPARQL queries to endpoints. By providing an abstract layer on top of the HTTP protocol, publishers and developers do not have to implement their own (error-prone) HTTP requests to SPARQL endpoints.

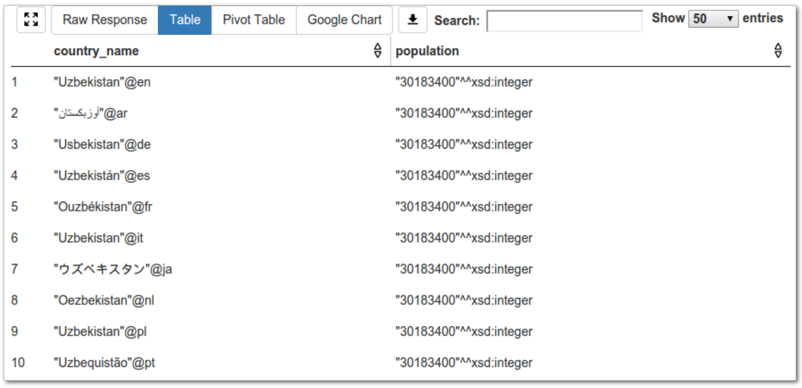
YASQE is developed to cater for many different publishing use cases, where not all features are needed all the time. To this end, YASQE is both configurable and extensible. Configurable, because publishers can toggle any of the above features on or off. And extensible, e.g. by modifying SPARQL auto-completion methods. The extendability is concretely illustrated by the Gosparqled editor,[23] which takes YASQE as its main component, and adds custom auto-completion functionality.

5.3.2 *YASR*

The YASR JavaScript library[24] (See Figure 5.3), aimed at publishers as well, parses and visualizes *any* SPARQL query response.

The W3C specifies several SPARQL result formats, including XML, JSON, CSV, Turtle and RDF/XML. To decrease the load on the publisher or developer, YASR consumes any of these data formats, by

21 See http://yasqe.yasgui.org
22 See http://codemirror.net
23 See https://github.com/scampi/gosparqled.
24 See http://yasr.yasgui.org



	country_name	population
1	"Uzbekistan"@en	"30183400"^^xsd:integer
2	"أوزبكستان"@ar	"30183400"^^xsd:integer
3	"Usbekistan"@de	"30183400"^^xsd:integer
4	"Uzbekistán"@es	"30183400"^^xsd:integer
5	"Ouzbékistan"@fr	"30183400"^^xsd:integer
6	"Uzbekistan"@it	"30183400"^^xsd:integer
7	"ウズベキスタン"@ja	"30183400"^^xsd:integer
8	"Oezbekistan"@nl	"30183400"^^xsd:integer
9	"Uzbekistan"@pl	"30183400"^^xsd:integer
10	"Uzbequistão"@pt	"30183400"^^xsd:integer

Figure 5.3: The YASR interface

parsing the results and wrapping them in an internal data representation. A first parse attempt is based on the Content-Type specified by the HTTP response. When such a Content-Type header is missing or appears to be invalid, YASR tries to parse the SPARQL results on a best-effort basis.

YASR has to deal with the wide variety of possible *errors* returned by endpoints. The SPARQL protocol specifies what the endpoint request and response should look like, but leaves error handling unspecified: what HTTP error code should be sent by an endpoint, and how should error messages be communicated? As a result, triple-stores come with various ways of conveying errors. Some endpoints return the error as part of an HTML page (with the regular 200 HTTP code), or as a SPARQL query result. Others only return an HTTP error code, where only some include a reason phrase together with the error code. The latter is a best practice for RESTful services. The absence of a standard, and the failure to adhere to best practices, makes a generic robust error handling solution messy and difficult to implement. Developing such a solution requires coding and testing by trial and error. YASR decreases the publishers and developers load by wrapping such SPARQL errors in an internal data representation.

The result of the procedures described above is a JavaScript library which is capable of handling any SPARQL response, moving the burden of writing SPARQL result-set parsers and error handlers away from the publisher.

As Table 5.1 shows, most SPARQL clients support both *rendering* and *downloading* of query results to some extent, which YASR supports as well. Users are provided with an extensive number of visualizations: A table renderer for `SELECT` query responses, and another renderer for visualizing the raw highlighted query response. Next to these two simple visualizations, YASR supports visualization via Google Charts, including line, bar and scatter plots, geographical maps, and several others. YASR supports a pivot-table functionality as well, allowing users to perform simple post-processing tasks on the SPARQL results. This functionality mimics functionality found in office suites such as Microsoft Excel and OpenOffice Calc, as users can cross-reference variables, aggregate on e.g. frequency counts or values, and plot these aggregated numbers on charts.

Most of the YASR visualizations are available for download, enabling offline re-use. The download options include CSV for tabular data, the as-is raw response, or the SVG renderings of charts.

Just as YASQE, YASR aims to be as extendable and configurable as possible. Publishers can easily toggle several visualizations on and off. Thanks to the modular architecture of YASR, adding a custom visualization is easy, as developers can ignore the different SPARQL response serializations and use the internal YASR response representation directly. This is illustrated by the Visu tool[25], which extends YASR by incorporating Google Chart visualizations. In turn, the Visu features have been integrated into YASR.

5.3.3 *YASGUI*

5.3.3.1 *JavaScript Library*

The YASGUI JavaScript library[26] (See Figure 5.1) includes YASQE and YASR, adds user functionality, and wraps the libraries in a tabbed graphical user interface. Next to the features described above, YASGUI includes several usability features described below.

To increase the findability of SPARQL endpoints, YASGUI uses the SPARQLES [26] service to provide endpoint search functionality. SPARQLES is a web service which monitors the up-time and characteristics of SPARQL endpoints, in effect providing a list of *available* SPARQL endpoints. However, YASGUI only uses this information in a static fashion, as SPARQLES does not publish this information dynamically via e.g. a SPARQL endpoint or regular API. Other services and endpoint catalogs exist such as DataHub.io, but these include endpoints which are often down and unavailable, and these catalogs do not publish their data via an API accessible by JavaScript.

YASGUI also supports *user*-configurable requests. For instance, some endpoints may only support the XML results format, or allow the use of additional request parameters such as the 'soft-limit' of 4Store or different reasoning levels of StarDog. Such endpoints can only be used to their full potential if users are able to specify these additional arguments manually. Therefore, YASGUI supports the specification of an arbitrary number of request parameters for every endpoint.

Where YASQE and YASR make the application states persistent between browser sessions, YASGUI goes a step further. YASGUI keeps

25 See `https://github.com/jiemakel/visu`
26 See `http://doc.yasgui.org`

track of queries and endpoints you have accessed in the past, and allows you to restore these queries from your local history.

The features described above are all bundled in the YASGUI JavaScript library. For those publishers that require more elaborate features going beyond the possibilities of client-side JavaScript, we provide a server-side back-end as well. This light-weight back-end is written in JavaScript and runnable as a NodeJS server.

As mentioned in section 5.2.2, client-side web applications such as the FLINT SPARQL Editor are *endpoint independent*, but only work for endpoints that enable Cross-Origin Resource Sharing (CORS)[27]. To overcome this limitation, YASGUI (optionally) includes this server-side proxy to access SPARQL endpoints which are otherwise not accessible via client-side JavaScript. For endpoints which *do* support cross domain JavaScript, YASGUI executes the queries from the clients side directly.

The YASGUI server also acts as a URL shortener. Web developers deploying YASGUI can choose to use this shortener, or configure YASGUI to use one of the available web URL shorteners. The rational behind a custom YASGUI shortener is that common web shorteners can suffer from link rot (they might disappear), they often require API key access, are not accessible from client-side JavaScript directly, and often have a limitation to the number of characters in a URL.

5.3.3.2 *Web Service*

Other than enabling Linked Data publishers to improve access to their SPARQL endpoints, we provide a running YASGUI instance as a web service as well[28]. This YASGUI instance, which includes a back-end server for CORS-disabled endpoints, presents users with a single usable editor for all SPARQL endpoints. This web service functions much like a local application: just as the regular YASGUI library, it can access SPARQL endpoints installed locally. Even more, in modern browsers, this application is still accessible when disconnected from the internet.

5.4 IMPACT

In our earlier work [84, 85] we voiced our expectation that YASGUI will fill a void in the tool chain of Linked Data consumers and publishers. As in the previous sections, we tried to substantiate this expectation by giving an in depth comparison with other, similar tools, and showing that YASGUI is substantially more feature rich than the competition. Nonetheless, it was just an expectation: because YASGUI hadn't been around for very long, we could not show that this

27 See `http://www.w3.org/TR/cors/`
28 See `http://yasgui.org`

expectation rang true. This section gives a brief overview of the impact the YASGUI family has had on the landscape of Linked Data management.

5.4.1 *Integration in Triple-stores*

Making YASQE and YASR available as highly configurable, lightweight, JavaScript-based front-ends for SPARQL interfaces has turned out to significantly lower the threshold for bundling YASGUI functionality with triple stores. YASQE and YASR have now made their way into three major triple stores:

APACHE JENA
: Includes both YASQE and YASR in the new Apache Jena-Fuseki 2 SPARQL interface

OPENRDF SESAME
: Includes YASQE as its main query editor.

CLIOPATRIA
: Includes both YASQE and YASR as query editor

5.4.2 *Integration in Other Applications*

The YASGUI family reduces the effort required from other developers to program against the idiosyncrasies of SPARQL endpoints and SPARQL responses. It thereby enables developers of SPARQL applications to kick-start their user interfaces by integrating or building on top of the YASGUI tools. Until now, we have been able to find the following usage of our work in five other applications:

GOSPARQLED
: An extension of YASQE, which provides (via a back-end server) smart, context-dependent auto-completions for properties and classes.

VISU
: The first library to extend YASR with Google Chart functionality. Now published together with the YASQE editor.

SNAPPER [29]
: An online Turtle and N-Triples editor, connecting to APIs which implement the SPARQL Graph Store Protocol. The tool uses several SPARQL queries to e.g. fetch items for auto-completions. Snapper allows users to configure such queries by means of YASQE.

29 See http://jiemakel.github.io/snapper

SEFARAD [30]
A data exploration tool-set which includes a SPARQL editor for templated SPARQL queries. This SPARQL editor is based on YASQE and YASR

BRWSR [31]
A lightweight Linked Data browser which incorporates YASQE and YASR to provide SPARQL access

5.4.3 *Linked Data Providers*

YASGUI components are used by a large number of Linked Data providers, in both open and closed, and non-profit and for-profit environments. Below we present a (non-exhaustive) list of Linked Data providers that use YASGUI components. We excluded those providers that already publish YASGUI components via their default endpoint interface, as discussed in Section 5.4.1.

HEALTHDATA.GOV [32]
A US federal government website managed by the department of Health & Human Services. Access to the healthcare data is provided via YASGUI.

SMITHSONIAN [33]
The Smithsonian American Art museum publishes art and artwork collections data as Linked Open Data [100]. YASGUI is used to provide access to the corresponding SPARQL endpoint.

ZBW [34]
The German National Library of Economics provides access to catalog information using YASQE and YASR.

LINKED OPEN VOCABULARIES [35]
Linked Open Vocabularies is a vocabulary catalog, which publishes their data via SPARQL endpoint and via regular APIs. As discussed in section 5.3, YASQE uses the LOV API for autocompletion functionality. In turn, LOV uses both YASQE and YASR to provide access to their SPARQL endpoint.

LOD LAUNDROMAT [36]
The LOD Laundromat meta-data SPARQL endpoint –presented

30 See `https://github.com/gsi-upm/Sefarad/`
31 See `https://github.com/Data2Semantics/brwsr`
32 See `http://www.healthdata.gov/sparql`
33 See `http://americanart.si.edu/collections/search/lod/about/sparql.cfm`
34 See `http://zbw.eu/labs/en/blog/publishing-sparql-queries-live` and `http://zbw.eu/beta/sparql-gui/`
35 See `http://lov.okfn.org/dataset/lov/sparql`
36 See `http://lodlaundromat.org/sparql/`

in chapter 6– uses the YASGUI client as part of the user interface.

METALEX [37]
: The MetaLex [53] service hosts almost all Dutch national regulations as Linked Data, and publishes these via a SPARQL endpoint. The SPARQL endpoint is accessible via the YASGUI interface.

CEDAR PROJECT [38]
: The CEDAR project publishes Dutch census data via a SPARQL endpoint, accessible via YASQE and YASR.

BUILDING BITS [39]
: A Semantic Web technology company which, for one of their customers, uses YASQE internally for accessing their triple-store.

KENNISNET [40]
: Kennisnet is a Dutch institute responsible for the basic education IT infrastructure. Some of the data that Kennisnet manages and publishes are exposed via regular APIs, but internally accessible via SPARQL, using YASQE and YASR

5.4.4 *Use by Consumers*

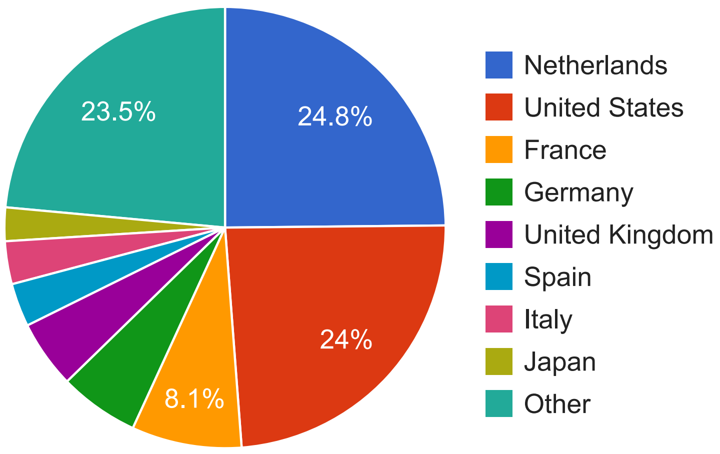


Figure 5.4: Locations of YASGUI users

The YASGUI web service is publicly available since October 2012, and we have gathered usage statistics from January 2013 onward. Over this period, we tracked (if permitted to do so) *at least* 6,670

37 See http://doc.metalex.eu/query
38 See http://lod.cedar-project.nl/cedar/data.html
39 See http://www.buildingbits.nl/
40 See http://www.kennisnet.nl/

unique visitors from over 96 countries (See figure 5.4), who executed 91,008 queries on around 1,645 endpoints.

We observe that the use of the YASGUI web service is increasing: we tracked 4.300 user sessions in 2013, which doubled to 8.400 user sessions in 2014. Note that these are conservative statistics, as only 56% of the users allowed us to track their information.

5.4.5 *Research Impact*

As the USEWOD challenge [15] shows, query logs enable research in the area of the *use* of Linked Data. This challenge distributes server SPARQL query logs from 6 endpoints (including DBpedia and Bio2RDF), and has seen an impact beyond the workshop as several research papers have been published using the USEWOD query log collection.

The YASGUI service query logs contribute to this research area for two reasons. First, the YASGUI logs are solely written by real persons, allowing us to distinguish man-made queries from (routine) machine use. This is something that cannot be done using server logs alone. Secondly, the USEWOD logs cover only 6 public endpoints, while the YASGUI logs cover both open and closed Linked Data; i.e. all endpoints listed by SPARQLES as well as local, private endpoints.

5.5 CONCLUSION

The size and complexity of the Semantic Web make it difficult to query, and requires tools with an extensive feature set (Problem 3). In this chapter we presented the state of the art in SPARQL user interfaces, and showed most of these are rather austere clients with little focus on usability, extendability, and feature completeness. Most striking is that their functionality is largely complementary: we have the SNORQL client for associative browsing, the Squebi editor for highlighted queries, several libraries which are accessible as SPARQL interface libraries, and other tools whose major selling point is access to *any* SPARQL endpoint. This large collection of tools, each with their own specific 'area of expertise', makes it hard for Linked Data developers to find and use the right tool for their task. Increasing user accessibility to the Semantic Web would require a tool-set which combines as much of these features as possible.

This is why we introduced the YASGUI family, which allows Linked Data developers to access *any* SPARQL endpoint – both remote and local –, and includes all the features present in the JavaScript libraries such as auto-completions, endpoint lookup, persistent user sessions, and syntax validation.

Since its launch more than two years ago, three triple-stores integrated YASGUI in their endpoint front-end, and several developers either adapted or included YASGUI components in new Linked Data applications. A large number of Linked Data providers use YASGUI components as their SPARQL endpoint interface, and close to a hundred thousand queries have been executed via the YASGUI web service on hundreds of SPARQL endpoints. The logs collected from this web service proved to be a useful data source for further research. This shows that the YASGUI family made a large impact on the landscape of Linked Data management.

6 META-DATA FOR A LOT OF LOD

In chapter 2 we described an infrastructure for collecting and publishing a centralized copy of the LOD Cloud, accessible via a single service: the LOD Laundromat. However, we do not know what kind of data we republish. Even simple statistics such as a dataset's size in bytes or triples, or more structural network measurements, are unknown. And finding a particular LOD Laundromat document for a given IRI is problematic as well. Linked Data developers who are interested in stress-testing applications against datasets with certain structural properties, or who are interested in datasets describing a particular IRI, are consequently still left in the dark.
Therefore, in this chapter we extend the LOD Laundromat with a Meta-Dataset that includes a IRI/namespace → dataset index, structural dataset characteristics, and provenance information about the performed processing steps.

This chapter is an updated version of:

> Laurens Rietveld, Wouter Beek, Stefan Schlobach, and Rinke Hoekstra. Meta-Data for a lot of LOD. *Semantic Web Journal*, To be published. http://semantic-web-journal.net/content/meta-data-lot-lod

Contributions to this paper:

> I am the main creator and designer of the Meta-Dataset and main author of the paper.

6.1 INTRODUCTION

Chapter 2 showed that using multiple Linked Open Datasets currently requires the hassle of finding a download location, hoping the downloaded data dumps are valid, and parsing the data in order to analyze or compare it based on some criterion. It is even more difficult to search for datasets based on characteristics that are relevant for machine-processing, such as syntactic conformance and structural properties such as the average outdegree of nodes. What is needed is a uniform representation of the *dataset* and a uniform representation of *dataset descriptions*.

The LOD Laundromat realizes the first: it (re)publishes the largest (collection of) dataset(s) on the Web of Data (over 38 billion triples and counting). The LOD Laundromat Meta-Dataset presented in this chapter realizes the second.

PROBLEM Currently, the creation of meta-data describing the datasets is left to the original data publisher. We see that many data publishers do not publish a dataset description that can be found by automated means, and that those data descriptions that *can* be found do not always contain all (de-facto) standardized meta-data. More importantly, the meta-data values are generally not comparable between datasets since different data publishers may interpret and calculate the same meta-data property differently. For instance, it is not generally the case that a dataset with a higher value for the `void:triples` property contains more triples: this value might be outdated with respect to the original dataset, or it might have been incorrectly calculated. Because of such incompatibilities between existing dataset descriptions, it is difficult to reliably analyze and compare datasets on a large scale.

CONTRIBUTIONS Therefore, next to the uniform *dataset* representations that are published by the LOD Laundromat, we need the same uniform representation for publishing *dataset meta-data*. In addition to uniformity, even straightforward meta-data should come with provenance annotations that describe how meta-data was generated. Such provenance annotations provide the context under which the meta-data was generated, and allow consumers meaningfully compare meta-data descriptions. The here presented LOD Laundromat Meta-Dataset brings exactly this: a uniform collection of dataset metadata that describes the structural properties of very many (over 650,000) Linked Data Documents containing over 38 billion triples. This Meta-Dataset is unique in its scale (both in terms of the 650,000 datasets it describes, and the number of meta-data properties), the consistent way in which meta-data properties are calculated, the explicit description of the computational processes used to calculate these properties, and the use cases it supports. As a result, it uniquely facilitates the analysis and comparison of very many datasets.

CONTENT In section 6.2 we give an overview of comparable datasets. In section 6.3 we identify shortcomings in existing meta-data standards and collections, and formulate a set of requirements for a dataset that would allow large collections of datasets to be analyzed, compared, and used. Section 6.4 presents the meta-data we publish, the model that is used to publish it, the external vocabularies used, a discussion in the context of the five stars of Linked Data Vocabulary use, and a clarification of how the LOD Laundromat Meta-Dataset is generated and maintained. Section 6.5 shows the applications and use cases that the LOD Laundromat Meta-Dataset supports. We conclude with section 6.6.

6.2 COMPARABLE DATASETS

SPARQL Endpoint Status[1] [26] presents an overview of dataset descriptions that can be found by automated means. These results show that even the uptake of the core meta-data properties (such as the ones from the VoID specification) is still quite low: only 12.9% of the analyzed SPARQL endpoints are described using VoID. Because of this apparent lack of LOD meta-data, several initiatives tried to fill this gap by creating uniform metadata descriptions for multiple datasets.

Firstly, LODStats provides statistical information for all Linked Open Datasets that are published in the CKAN-powered[2] Datahub catalog. It offers a wide range of statistics, e.g., including the number of blank nodes in a dataset and the average outdegree of subject terms. Unfortunately, only a small subset of those statistics are themselves being published as Linked Data. Secondly, Sindice provides statistical information similar to LODStats, but mostly analyzes smaller datasets that are crawled from Web pages. The meta-data provided by Sindice are similar to those in the VoID specification but they are not published in a machine-readable format such as RDF.

Although Sindice and LODStats provide a step in the right direction by uniformly creating metadata descriptions for many Linked Datasets, they only support a subset of existing metadata properties, they do not publish exhaustive metadata descriptions as Linked Data, and they do not publish structural information on the meta-data generation procedure. Also, they are constrained to Linked Datasets that are published in only certain locations.

6.3 META-DATA REQUIREMENTS

In this section we present a requirements analysis for a dataset that satisfies our goal of supporting the meaningful analysis, comparison, and use, of very many datasets.

We explain problems with respect to meta-data specifications (section 6.3.1), dataset descriptions (section 6.3.2) and collections of dataset descriptions (section 6.3.3). Based on these considerations, the requirements are presented in section 6.3.4.

6.3.1 *Meta-data specifications*

Existing dataset vocabularies include VoID [1], VoID-ext [68], DCAT[3], and Bio2RDF [14]. VoID is a vocabulary for expressing metadata about Linked Datasets. It supports generic meta-data (e.g., the home-

1 See http://sparqles.ai.wu.ac.at/
2 http://ckan.org/
3 See http://www.w3.org/TR/vocab-dcat/

page of a dataset), access meta-data (e.g., which protocols are available), links to other datasets, exemplary resources, as well as dataset statistics (e.g., the number of triples). Only some of the VoID meta-data properties can be automatically generated. Others can only be given by human authors, –such as exemplary resources– since they depend on interpretation. Bio2RDF presents a collection of dataset meta-data properties that extends the set of VoID properties and provides more detail. For example, Bio2RDF includes properties that describe how often particular types are used in the subject position and in the object position for a given property; e.g. property `ex:livesIn` links 10 subjects of type `ex:Person` to 6 objects of type `ex:City`. The use of such descriptive properties can increase the size of a Meta-Dataset significantly when the described dataset has a large number of classes and properties. VoID-ext extends the set of meta-data properties that are found in VoID as well. It includes the in- and outdegree of entities, the number of blank nodes, the average string length of literals, and a partitioning of the literals and IRIs based on string length. The Data Catalog Vocabulary (DCAT) is a vocabulary for describing datasets on a higher level. I.e., it includes properties such as the dataset title, description and publishing/modification date. Such information is difficult to reliably extract from the dataset in an automated fashion.

We observe the following problems with these existing meta-data specifications:

First, some existing meta-data properties are subjective. For example, `void:entities` is intended to denote a subset of the IRIs of a dataset based on "arbitrary additional requirements" imposed by the authors of the dataset description. Since different authors may impose different requirements, the number of entities of a dataset may vary between zero and the number of resources.

Secondly, some existing meta-data properties are defined in terms of undefined concepts. For example, LODStats specifies the set of vocabularies that are reused by a given dataset. The notion of a 'reused vocabulary' is itself not formally defined but depends on heuristics about whether or not an IRI belongs to another dataset. LODStats calculates this set by using relatively simple string operations according to which IRIs of the form `http://<authority>/<string>/<value>` are assumed to belong to the vocabulary denoted by `http://<authority>/<string>`. Although this is a fair attempt at identifying reused vocabularies, there is not always a bijective map between datasets and IRI substrings that occur in datasets. The number of links to other datasets suffers from the same lack of a formal definition.

6.3.2 *Dataset descriptions*

We observe the following problems with existing dataset descriptions: First, uptake of dataset descriptions that can be found by automated means is still quite low (section 6.2). Secondly, for reasons discussed above, the values of meta-data properties that do not have a well-founded definition cannot be meaningfully compared across datasets. E.g., if two dataset descriptions contain different values for the `void:entities` property it is not clear whether this denotes an interesting difference between the two datasets or whether this is due to the authors having different criteria for identifying the set of entities. Thirdly, even the values of well-defined meta-data may have been calculated in different ways by different computational procedures. We observe that there are significant discrepancies between meta-data which occurs *in* the original dataset description and those from the LOD Laundromat. For example, a dataset about a Greek fire brigade contains 3,302,302 triples according to its original VoID description[4], but 4,134,725 triples according to the LOD Laundromat Meta-Dataset[5].

Similar discrepancies exist between meta-data values that occur in different dataset description *collections*, e.g. between LODStats and the LOD Laundromat Meta-Dataset.[6]

Since it is difficult to assess whether a computational procedure that generates meta-data is correct, we believe it is necessary that all generated meta-data is annotated with provenance information that describes the used computational procedure. Although relatively verbose, this approach circumvents the arduous discussion of which version of what tool is correct/incorrect for calculating a given meta-data value. We assume that there will always be multiple values for the same meta-data property. The fact that there are different values, and that these have been derived by different means, is something that has to be made transparent to the consumer of this meta-data. The onus is on the data consumer to trust one computational procedure for calculating a specific meta-data value more than another. This requires provenance that details the mechanism behind the calculated meta-data.

6.3.3 *Dataset description collections*

We observe two problems with existing collections of dataset descriptions: Firstly, even though the meta-data may be calculated consistently within a collection, the computational procedure that is used is

4 See `http://greek-lod.auth.gr/Fire/void.ttl`
5 See `http://lodlaundromat.org/resource/0ca7054f382b29319c82796a7f9c3899`
6 E.g., according to LODStats the dataset located at `http://www.open-biomed.org.uk/open-biomed-data/bdgp-images-all-20110211.tar.gz` contains 1,080,060 triples while the LOD Laundromat Meta-Dataset states 1,070,072.

not described in a machine-processable format (if at all). This means that values can only be compared within the collection, but not with dataset descriptions external to the collection (e.g. occurring in other collections). Secondly, meta-data that is calculated within existing collections is not always published in a machine-interpretable format (e.g. LODStats).

6.3.4 *Requirements*

Based on the above considerations, we formulate the following requirements which allow multiple datasets to be meaningfully compared based on their meta-data:

1. The LOD Laundromat Meta-Dataset must cover very many datasets in order to improve data comparability.
2. The Meta-Dataset should reuse official and de-facto meta-data standards as much as possible, in order to be compatible with other dataset descriptions and to promote reuse.
3. The Meta-Dataset must be generated algorithmically in order to assure that values are calculated in the same way for every described dataset.
4. Only those meta-data properties must be used that can be calculated efficiently, because datasets can have peculiar properties that may not have been anticipated when the meta-data properties were first defined.
5. The LOD Laundromat Meta-Dataset must contain provenance annotations that explain how and when the meta-data was calculated.
6. The meta-data must be disseminated as LOD and must be accessible via a SPARQL endpoint.
7. The LOD Laundromat Meta-Dataset must be able to support a wide range of real-world use cases that involve analyzing and/or comparing datasets such as Big Data algorithms that process LOD.

6.4 THE LOD LAUNDROMAT META-DATASET

In this section we present the meta-data we publish, the model we use, and how we generate this dataset.

6.4.1 *Published Meta-Data*

The LOD Laundromat Meta-Dataset is generated in adherence to the requirements formulated in section 6.3. Since there are multiple ways in which these requirements can be prioritized and made concrete, we will now discuss the considerations that have guided the generation of the meta-data.

Firstly, there is a trade-off between requirements 2 and 3: since the Meta-Dataset has to be constructed algorithmically, only well-defined meta-data properties can be included.

Secondly, there is a conflict between requirements 1 and 4 on the one hand, and requirement 2 on the other: since the LOD Laundromat Meta-Dataset must describe many datasets, some of which are relatively large, and we want calculations to be efficient, we chose to narrow down the set of meta-data properties to those that can be calculated by *streaming* the described datasets. This excludes properties that require loading (large parts of) a dataset into memory, e.g. in order to perform joins on triples.

Thirdly, because of the scale at which the LOD Laundromat Meta-Dataset describes datasets, it is inevitable that some datasets will have atypical properties. This includes datasets with extremely long literals, datasets where the number of unique predicate terms is close to the total number of predicate terms, or datasets where the number of unique literal datatype equals the total number of literals. It is only when meta-data is systematically generated on a large scale, that one finds such corner cases. These corner cases can make dataset descriptions impractically large. This is especially true for meta-data properties that consist of enumerations. E.g., for some datasets the partition of all properties, as defined by VoID-ext and Bio2RDF, is only (roughly) a factor 3 smaller than the described dataset itself (and this is only one meta-data property). Or, take as example the `void-ext:subjectPartition`, that refers to a partition that contains triples for a certain subject. Using such partitions for all the subjects in a dataset would generate a Meta-Dataset that equals the size of the original dataset. Therefore, in order to keep data descriptions relatively small w.r.t. the dataset described, the Meta-Dataset does not include properties whose values are dataset partitions.

Under these restrictions, the Meta-Dataset is able to include a large number of datasets while still being relatively efficient to construct. Implementation-wise, the generation of the Meta-Dataset takes into account the many advantages that come from the way in which LOD Laundromat (re)publishes datasets. LOD Laundromat allows datasets to be opened as gzip-compressed streams of lexicographically sorted N-Triples and N-Quads. Since these streams are guaranteed to contain no syntax error nor any duplicate occurrences of triples, they can be processed on a line-by-line / triple-by-triple basis, making it convenient to generate meta-data for inclusion in the LOD Laundromat Meta-Dataset. Because of these advantages, the meta-data server (with 5TB SSD Disk space, 8-core CPU and 256GB memory) manages to stream and analyze 400.000 triples per second.

Table 6.1 gives an overview of the meta-data properties included in the LOD Laundromat Meta-Dataset, together with those that are included in existing dataset description standards. As can be seen from the table, the only meta-data properties that are excluded from our dataset (because of computational issues) are the distinct number of classes that occur in either the subject, predicate, or object position, as specified in VoID-ext. These three meta-data properties cannot be calculated by streaming the data a single time. In addition, all meta-data properties whose values must be represented as partitions are excluded in order to preserve brevity for all dataset descriptions, and to maintain scalability. Considering these limitations, the meta-data properties presented in Bio2RDF are similar to those in VoID and VoID-ext. Therefore, Bio2RDF is not referenced in our vocabulary. The generation of several statistics (e.g. the distinct number of IRIs) requires in-memory lists. To reduce this memory consumption, we use an efficient in-memory dictionary (RDF Vault [11]).

Since we want the LOD Laundromat Meta-Dataset to be maximally useful for a wide range of use cases (requirement 7), we have added several meta-data properties that do not occur in existing specifications:

1. Next to the number of distinct IRIs, blank nodes and literals (i.e., *types*), we also include the number of (possibly non-distinct) occurrences (i.e., *tokens*).
2. Existing vocabularies specify the number of properties and classes (although they do so incorrectly, see section 6.3). The Meta-Dataset also includes the number of classes and properties that are *defined* in a dataset, such as `<prop> rdf:type rdf:Property`
3. Existing dataset description vocabularies such as VoID-ext use arithmetic means to describe number series such as the literal lengths in given document. The LOD Laundromat Meta-Dataset uses more detailed descriptive statistics, that include the median, minimum, maximum and standard deviation values as well.
4. Similar statistics are provided for network characteristics such as Degree, In Degree and Out Degree.

Considering that only 0.5% of the datasets publish a corresponding dataset license via RDF, we exclude this information for now. We expect these dataset licenses to increase in use and popularity though, and will include this meta-data in a future crawl.

Figure 6.1 illustrates one of the published meta-data properties: the average out degree of datasets. The figure illustrates our previous remark that analyzing many datasets will inevitably include datasets with atypical properties or 'corner cases'. E.g., the dataset with the highest average out degree, contains 10.004 triples, and only one subject, thereby strongly skewing the dataset distribution. Such a-typical

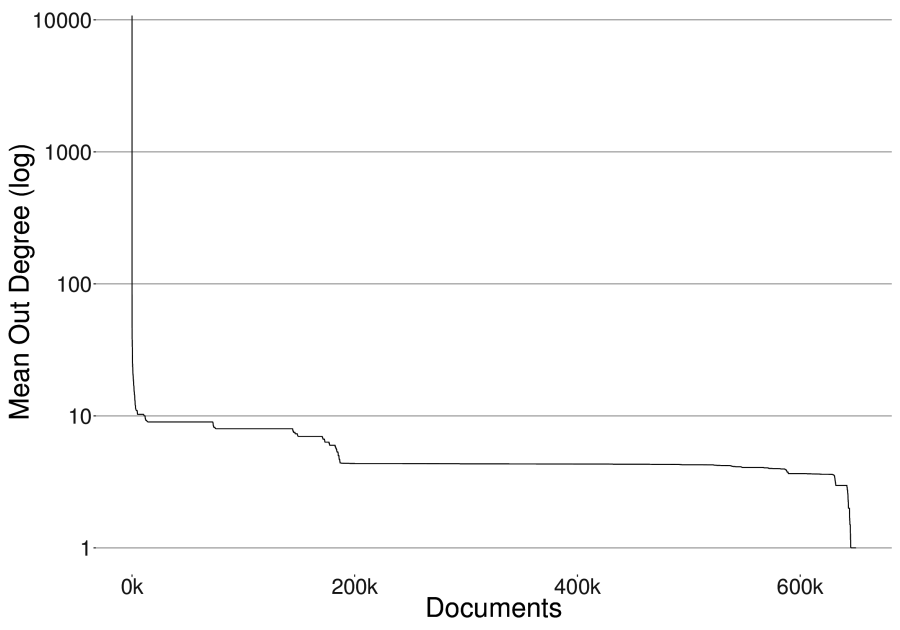


Figure 6.1: Average out degree distribution of LOD Laundromat documents

properties of datasets are potentially important as e.g. a means of explaining deviating evaluation results between datasets –as shown in chapter 7. Note, that generating the data behind this figure requires the following SPARQL query, illustrating the ease of use:

```
SELECT * {[] llm:outDegree/llm:mean ?mean}
```

Besides publishing the meta-data, and in line with requirement 5, the Meta-Dataset contains a provenance trail of how the meta-data was generated. The provenance trail includes a reference to the code that was used to generate the meta-data. For this we use a Git commit identifier in order to uniquely identify the exact version that was used. The provenance trail also includes all the steps that preceded the calculation of the meta-data:

1. Where the file was downloaded (either the original URL or the archive that contained the file).
2. When the file was downloaded (date and time).
3. Metadata on the download process, such as the status code and headers from the original HTTP reply. For archived data the applied compression techniques (possibly multiple ones) are enumerated as well.
4. Detailed metadata on the data preparation tasks performed by the LOD Laundromat in order to clean the data. This includes the number of bytes that were read (not necessarily the same as the value for `Content-Length` HTTP header) and syntax errors that were encountered (e.g., malformed syntax, unrecognized encoding, undefined prefixes).

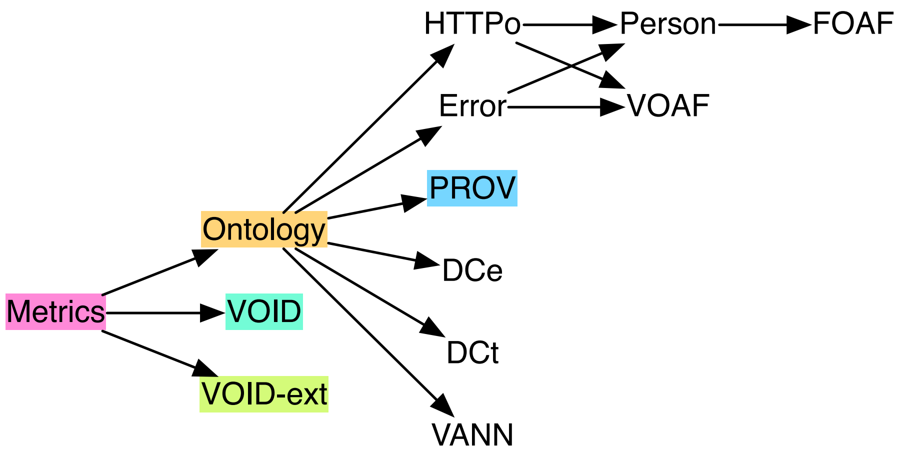


Figure 6.2: Dependencies of LOD Laundromat Meta-Dataset vocabulary

5. The number of duplicate triples in the original dataset.
6. A reference to the online location where the cleaned file is stored, and from which the meta-data is derived.

Other relevant meta-data includes links between IRIs (13 billion) or namespaces (184 million) and the documents these occur in, as this greatly increases the findability of Linked Datasets. Publishing this meta-data via RDF and SPARQL would require significant hardware resources. Instead, we store these indexes in the scalable disk-based column-store RocksDB[7], hosted on SSD hard disks for fast access.

6.4.2 *Model*

The meta-data is specified in the LOD Laundromat Meta-Dataset vocabulary[8]. Of the 26 meta-data properties that are included, 22 are linked to one or more other dataset description vocabularies. Figure 6.2 shows the dependencies between our Meta-Dataset vocabulary and other vocabularies. The referenced dataset description vocabularies are VoID and VoID-ext. Figure 6.3 shows an example dataset description that illustrates the structure of this Meta-Dataset[9]. The Meta-Dataset also includes information about the vocabulary *itself*, such as its license (Creative Commons[10]), last modification date, creators, and homepage. As such, it implements the first 4 of the 5 stars for vocabulary re-use [59]. The fifth star (re-use *by* other vocabularies) is not reached yet because the vocabulary is quite recent. However, the LOD Laundromat Meta-Dataset has been submitted to the Linked

7 See http://rocksdb.org/
8 See http://lodlaundromat.org/metrics/ontology/
9 For brevity, only a subset of the available meta-data properties are included in this figure
10 See http://creativecommons.org/licenses/by/3.0/

Table 6.1: An overview of dataset meta-data properties, grouped by the vocabularies that define them and dataset description collections that include them. For brevity's sake, properties whose values are dataset partitions and properties that require manual intervention are excluded

Meta-data Property	**VoID**	**Bio2RDF**	**VoID-ext**	**LOD Laundromat**
Triples	v	v	v	v
Entities	v	v	v	v
Distinct Classes	v	v	v	v
Distinct Properties	v	v	v	v
Distinct Subject	v	v	v	v
Distinct Objects	v	v	v	v
Distinct RDF Nodes			v	v
Distinct IRIs			v	v
IRIs				v
Distinct Blank Nodes			v	v
Blank Nodes				v
Distinct Literals	v		v	v
Literals				v
Distinct URIs in subject position			v	v
Distinct Blank Nodes in subject pos.			v	v
Distinct URIs in object position			v	v
Distinct Blank Nodes in object pos.			v	v
Distinct Literal Data-Types			v	v
Distinct Literal Languages			v	v
Length statistics of IRIs			v	v
Length statistics of IRIs in subject pos.			v	v
Length statistics of IRIs in predicate pos.			v	v
Length statistics of IRIs in object pos.			v	v
Length statistics of Literals			v	v
Degree Statistics				v
Indegree Statistics				v
Outdegree Statistics				v
Defined Classes				v
Defined Properties				v
Distinct Classes in the subject pos.			v	
Distinct Classes in the predicate pos.			v	
Distinct Classes in the object pos.			v	

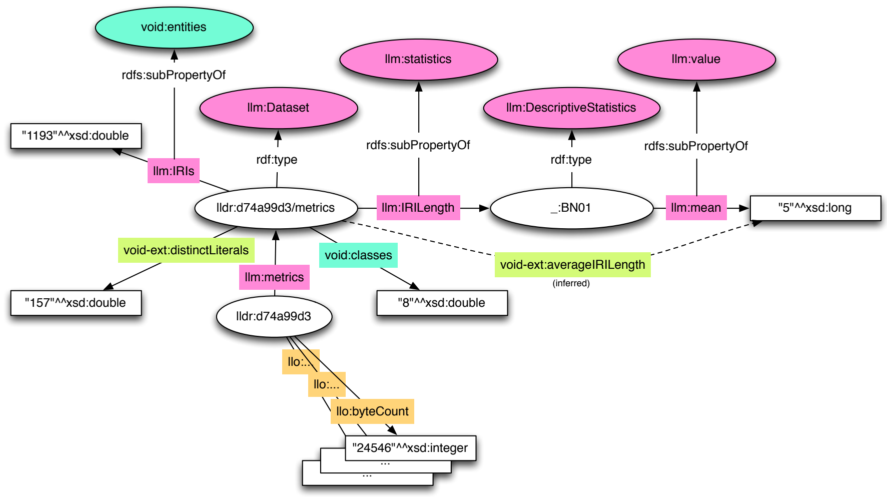


Figure 6.3: Example (partial) dataset meta-data description, color-coded using vocabularies from Figure 6.2

Open Vocabulary catalog [11], thereby hopefully supporting its re-use and findability.

The provenance information of datasets is described using the PROV-O vocabulary [64], a W3C recommendation. Figure 6.4 presents an overview on how PROV-O is used by the LOD Laundromat Meta-Dataset. Similar vocabularies exist, such as the VoiDp [78] vocabulary which matches the provenance of Linked Datasets with the VoID vocabulary. However, because VoiDp uses a predecessor of the PROV-O standard, we model our provenance in PROV-O directly. The Provenance Vocabulary [49] aims to describe the provenance of Linked Datasets as well, but is too specific for our use considering the wide range of provenance (see below) we describe.

As the LOD Laundromat cleaning process is part of the provenance trail, we model this part of the dataset using separate vocabularies: Firstly, the LOD Laundromat vocabulary[12] describes the crawling and cleaning process of LOD Laundromat. This description includes the download time and date of the original document, and therefore specifies which version of the original document is described by the Meta-Dataset. Secondly, the HTTP vocabulary[13] describes HTTP status codes. Thirdly, the error ontology[14] models all exceptions and warnings, and is used by the LOD Laundromat vocabulary to represent errors that occur during the crawling and cleaning process. Each of these vocabularies are linked to other vocabularies. E.g., the HTTP vocabulary is an extension of the W3C HTTP in RDF vocabulary[15].

11 http://lov.okfn.org/
12 http://lodlaundromat.org/ontology/
13 http://lodlaundromat.org/http/ontology/
14 http://lodlaundromat.org/errors/ontology/
15 http://www.w3.org/2011/http

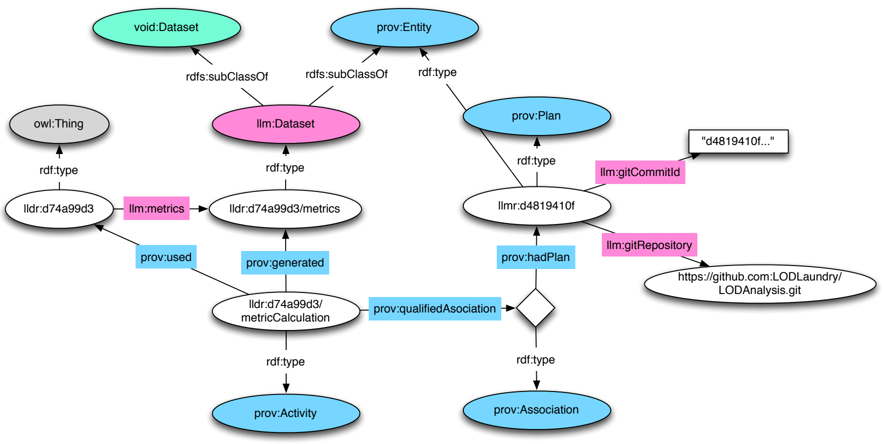


Figure 6.4: Provenance model illustration

6.4.3 *Naming Scheme*

The LOD Laundromat Meta-Dataset uses the following naming scheme. As a running example, we take a Semantic Web Dog Food file that is crawled by LOD Laundromat[16].

- The LOD Laundromat document identifier for this dataset is generated by appending an MD5 hash of the data source IRI to `http://lodlaundromat.org/resource/`[17].
- The calculated structural properties of this dataset are accessible by appending `/metrics` to the LOD Laundromat document identifier[18].
- Provenance that describes the procedure behind the metrics calculation is accessible by appending `metricCalculation` to the LOD Laundromat identifier[19].

6.4.4 *Dissemination*

The LOD Laundromat continuously crawls and analyses Linked Data dumps. In order to get a maximum coverage of the LOD Cloud, it searches both linked data catalogs and the LOD Laundromat datasets themselves for references to datadumps. Because it does not claim to have a complete seed list that links to all LOD in the world, users have the option to manually or algorithmically add seed-points to the LOD Basket[20].

16 See `http://data.semanticweb.org/dumps/conferences/iswc-2013-complete.rdf`
17 See `http://lodlaundromat.org/resource/05c4972cf9b5ccc346017126641c2913`
18 See `http://lodlaundromat.org/resource/05c4972cf9b5ccc346017126641c2913/metrics`
19 See `http://lodlaundromat.org/resource/05c4972cf9b5ccc346017126641c2913/metricCalculation`
20 `http://lodlaundromat.org/basket/`

The code[21] used to generate the LOD Laundromat Meta-Dataset runs immediately after a document is crawled and cleaned by the LOD Laundromat, and is directly published via a public SPARQL endpoint[22]. SPARQL is preferred over HDT as publishing method, because HDT files are static and do not support updates. In line with requirement 6, each daily version of the Meta-Dataset is extracted from the SPARQL endpoint and published as data dump[23], in the same standardized N-Quad serialization format of the LOD Laundromat.

Considering some meta-data is too verbose and expensive to host as RDF, we publish non-RDF data as well. Specifically, we publish the mapping between all the IRIs and namespaces to the corresponding LOD Laundromat documents these occur in. We provide access to this meta-data via a custom RESTful API[24].

6.4.5 *Dataset Statistics*

Since the release of LOD Laundromat in September 2014 and the release of the Meta-Dataset in January 2015, we registered 2,119,218 document downloads, and 20,606,194 SPARQL queries on the Meta-Dataset. As mentioned before, the LOD Laundromat crawled and re-publishes over 650,000 documents containing over 38 billion triples. The meta-data of these crawled documents are published in the LOD Laundromat Meta-Dataset, and now contains over 110 million triples, accessible via a data dump and SPARQL endpoint.

6.5 USE CASES

The LOD Laundromat Meta-Dataset is intended to support a wide array of non-trivial use cases. And where we focused on the Linked Data developer specifically, other Linked Data consumers can benefit from this Meta-Dataset as well:

The first use case we present is the evaluation of Semantic Web (SW) algorithms. In contemporary SW research novel algorithms are usually evaluated against only a handful of – often the same – datasets (i.e., mainly DBpedia, Freebase, and Billion Triple Challenge). The risk of this practice is that – over time – SW algorithms will be optimized for datasets with specific distributions, but not for others. In chapter 7 we re-evaluate parts of three SW research papers using *Frank*, a bash interface that connects with the LOD Laundromat. We show how the LOD Laundromat Meta-Dataset can be used to relate datasets to their overall structural properties, and how SW evalua-

21 Publicly available at `https://github.com/LODLaundry/LODAnalysis`
22 `http://lodlaundromat.org/sparql`
23 `http://download.lodlaundromat.org/dump.nt.gz`
24 See `http://index.lodlaundromat.org/`

tions can be performed on a much wider scale, leading to results that are more indicative of the *entire* LOD Cloud. This use case combines the strength of both the collection of the LOD Laundromat and the LOD Laundromat Meta-Dataset.

Similarly to *evaluating* SW algorithms, the LOD Laundromat Meta-Dataset can also be used to *tune* Linked Data applications or prune datasets with the desired property at an early stage, i.e., without having to load and interpret them. An example of this is PrefLabel[25], an online service that returns a human-readable label for a given resource-denoting IRI. The index behind the PrefLabel Web service is populated by streaming and analyzing LOD Laundromat datasets for RDFS label statements in datasets. PrefLabel uses the LOD Laundromat Meta-Dataset by pruning for datasets that do not contain RDF literals at all. This crude way of using the Meta-Dataset already excludes 20% of all the triples that are in the LOD Laundromat today, thereby significantly optimizing the algorithm. The following SPARQL query is used by PrefLabel to prune the list of documents:

```
SELECT ?doc WHERE {
  ?doc llm:metrics/llm:literals ?lit;
  FILTER(?lit = 0)
}
```

Another use case involves using the LOD Laundromat Meta-Dataset to analyze and compare datasets, e.g., in order to create an overview of the state of the LOD Cloud at a given moment in time. A common approach (see e.g. [52, 61, 91]) is to crawl Linked Data via dereferenceable URIs using tools such as LDspider [58], and/or to use catalogs such as datahub to discover the Linked Datasets. Both dereferenceable URIs and dataset catalogs come with limitations: most Linked Data URIs are not dereferenceable, and the dataset catalogs only cover a subset of the LOD Cloud. The LOD Laundromat on the other hand provides access to more than dereferenceable URIs only, and aims to provide a complete as possible dataset collection. The corresponding Meta-Dataset provides a starting point for e.g. selecting datasets by Top Level Domain, serialization format, or structural properties such as number of triples. In chapter 7 we re-evaluate (part of) exactly such a Linked Data Observatory paper [91], where we use the Meta-Dataset and LOD Laundromat to find the documents and extract namespace statistics.

Next to the *structural* meta-data properties, the *provenance* meta-data provides an interesting data source as well. It enables e.g. an overview of common HTTP exceptions for Linked Data files, as shown in figure 6.5. Related visualizations are available online[26], including their corresponding SPARQL queries.

25 http://preflabel.org
26 See http://lodlaundromat.org/visualizations

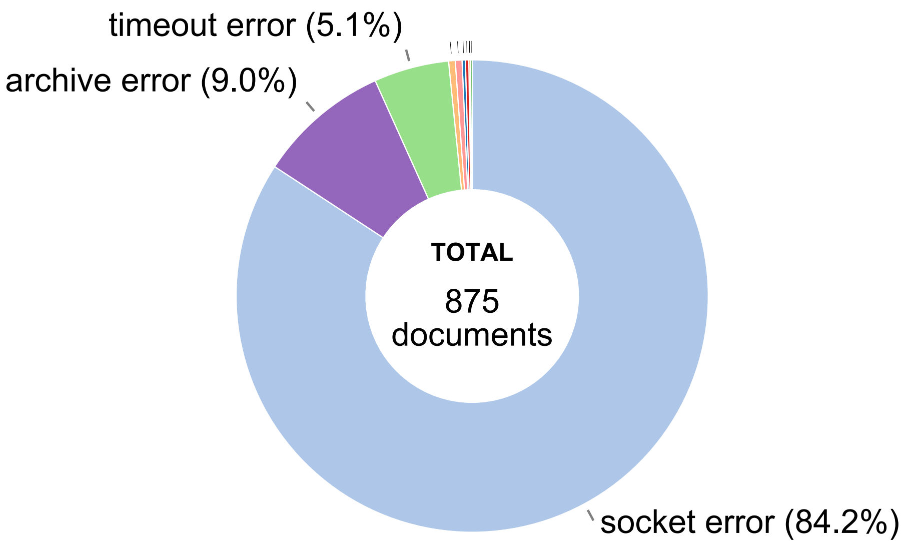


Figure 6.5: Frequent HTTP exceptions during the LOD Laundromat crawling process

6.6 CONCLUSION

The current state-of-the-art for dataset descriptions leaves a lot to be desired: data providers often do not publish dataset meta-data, catalogs are incomplete, and meta-data descriptions often lack structural information about the meta-data generation procedure. As a result, datasets are difficult to locate and dataset descriptions are difficult to compare (Problem 4).

The LOD Laundromat Meta-Dataset improves this state-of-the-art by providing access to a large set of uniformly represented datasets descriptions, acting as an enabler for Linked Data developers: finding or comparing linked datasets with certain structural properties is now possible by executing a SPARQL query, and finding documents for any given IRI or namespace is as easy as issuing a simple HTTP `GET` request. And even better: because the dataset descriptions are linked to their uniform dataset representations, the underlying data is easily accessible as well.

LINKED DATA SCIENTIST

7 LOD LAB: EXPERIMENTS AT LOD SCALE

*Typically, Linked Data research evaluates and optimizes algorithms for only a handful of datasets such as DBpedia, BSBM, DBLP and only a few more. Considering the potential impact of structural properties on experiment results (as we showed in SampLD, chapter 3), current practice does not generally take the true **variety** of Linked Data into account. With hundreds of thousands of datasets out in the world today the results of Semantic Web evaluations are less generalizable than they should and — this chapter argues — can be. In this chapter we present LOD Lab: an evaluation paradigm that uses the LOD Laundromat, together with the Meta-Dataset presented in the previous chapter, to make algorithmic evaluation against hundreds of thousands of datasets the new norm.*

This chapter is an combined version of:

Laurens Rietveld, Wouter Beek, and Stefan Schlobach. LOD Lab: Experiments at LOD Scale. In *Proceedings of the International Semantic Web Conference (ISWC)*. Springer, 2015

and

Wouter Beek and Laurens Rietveld. Frank: The LOD Cloud at your Fingertips. In *Developers Workshop , Extended Semantic Web Conference (ESWC)*, 2015

Contributions to these papers:

I created the *Frank* interface, evaluated the three discussed publications, and I am the main author of the paper.

7.1 INTRODUCTION

While the exact size of the Linked Data Cloud is unknown, there is broad agreement that the volume of data published according to Linked Open Data standards has to be counted in tens, if not hundreds, of billions of triples by now, originating from hundreds of thousands of datasets from various domains and provenance. This amount and broadness of information makes the Linked Open Data Cloud ideal for testing various types of algorithms and an exciting object of study. As this is widely recognized it is no surprise that many research papers have been published in the recent past that use parts of this enormous and rich collection.

PROBLEM Unfortunately, true large-scale evaluation, both in terms of volume and variety have proven to be much harder to come by than one would expect. One of the main reasons for this is the heterogeneity and user-unfriendliness of the most wide-spread dissemination strategy for Linked Data today: datadumps. Most researchers will recognize the problem of dealing with various serialization formats and juggling with syntax errors as well as other data document-specific idiosyncrasies. With the core research being on algorithms and evaluations, data collection, cleaning and harmonization can easily become a barrier too high to overcome.

To avoid these tedious and painful efforts of integrating hundreds of thousands of heterogeneous datasets most current studies with evaluations focus on data published through APIs, e.g., using SPARQL. Although this often provides high-volume datasets for testing, this leads to a strange imbalance in current practice: of the hundreds of thousands of available datasets [13], only around 260 are available through live query endpoints [26], and of the latter less than 10% dominate the evaluation landscape (see Section 7.2). As such, question-marks have to be put on the generalizability and maybe even validity of many of the results.

CONTRIBUTIONS The technological developments presented in the previous chapters have changed the situation significantly: LOD Laundromat (Chapter 2) presents a wealth of clean standards-compliant documents and accessible as queryable interfaces (Chapter 4), and disseminates the related meta-data via SPARQL and other RESTful APIs (Chapter 6). While these Web Services provide a good interface for some use cases, e.g. downloading a specific data document, the large-scale evaluation of a Semantic Web algorithm against thousands of data documents is still relatively time consuming.

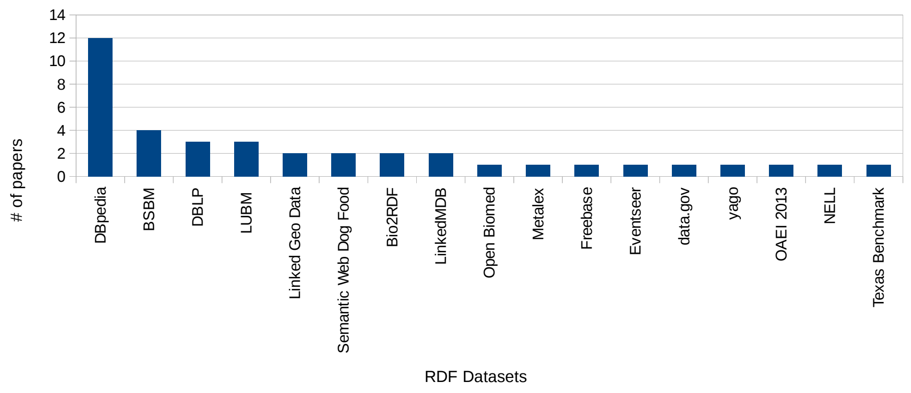
This is why we present LOD Lab: an integrated approach towards running Linked Data evaluations in the large. The LOD Lab approach is implemented by pairing the LOD Laundromat backend with *Frank*, an open-source[1] and simple yet flexible programming interface for conducting large-scale experiments over heterogeneous data.

Since the LOD Lab approach defaults to running Semantic Web evaluations against hundreds of thousands of data documents, it introduces a problem that would have been considered a luxury problem even two years ago: now that 650,000 datasets are available, choosing suitable ones for specific experiments becomes a non-trivial task. Fortunately, *Frank* facilitates informed selection by filtering on domain vocabularies and by using metadata about the scraping and cleaning process as well as metadata about the structural properties of the data.

In this chapter we present a new way of conducting Linked Data experiments that incorporates both volume and variety while at the same time allowing the set of considered data documents to be lim-

1 See `https://github.com/LODLaundry/Frank`

Figure 7.1: Overview of datasets used in evaluations of papers accepted in the ISWC 2014 research track. For each dataset the number of articles that use it is shown.



ited according to structural constraints. This chapter demonstrates the viability of the LOD Lab evaluation approach by rerunning three experiments reported in recent Semantic Web conference publications, but now by using hundreds of thousands of data documents.

CONTENT This chapter is structured as follows. In section 7.2 we present the motivation behind our approach, and in section 7.3 we present related work on Linked Data evaluation methods at scale. Section 7.4 presents the usage, functionality and implementation of *Frank*, which is used for running large-scale Linked Data evaluations from the command-line. We demonstrate the viability of the LOD Lab evaluation approach in section 7.5, by rerunning three experiments reported in recent Semantic Web conference publications, but now by using hundreds of thousands of data documents. We conclude in section 7.6.

7.2 MOTIVATION

Figure 7.1 gives an overview of the datasets that are used in 20 papers that were accepted in the ISWC 2014 research track. It only includes papers that evaluate Linked Datasets, excluding ones that evaluate algorithms on relatively small ontologies, non-RDF datasets or streamed data. The figure shows that 17 datasets are used in total. The number of datasets per article varies between 1 and 6 and is 2 on average.

The figure shows that most evaluations are conducted on only a handful of datasets. Even the total collection of datasets that are used in these 20 papers is not very large. This implies that many papers evaluate against the same datasets, most often DBpedia. This means that it is generally unclear to what extent published results will transfer to other datasets, specifically those that are only very rarely evaluated against. This is the problem of the *generalizability* of Semantic Web research results (Problem 1).

Problem 1 *By using very few datasets in scientific evaluations, the generalizability of Semantic Web research results is often unknown.*

The reason for Problem 1 is that current evaluation practice does not scale over heterogeneous data, i.e. we face a problem of *variety*. The problem is no longer with the *volume* of the data since most of the datasets that are never evaluated against are smaller than some of the datasets that are currently used in evaluations. While it is sufficiently easy to obtain, load and evaluate one dataset, contemporary practice shows that it is still difficult to do the same thing for very many datasets.

One critique that may be leveled against our identification of Problem 1 is that the most often used datasets are evaluated most often and that evaluation practice is simply in line with data usefulness or relevance. However, most of the algorithms and approaches that are evaluated in Semantic Web research target generic applicability. Specifically, none of the above 20 papers claims to develop a *dataset-specific* approach. Moreover, that a dataset is popular does not imply that results obtained over it are indicative of Linked Data in general and can be transferred to other datasets. This is especially true for Linked Data where the expressiveness of the language allows datasets to differ considerably.

Empirical surveys have documented the restricted state of today's Semantic Web deployment. [26, 56] Many datasets are only available as data dumps, lack dereferenceable IRIs, cannot be downloaded due to HTTP errors, cannot be unpacked due to archive errors, or cannot be loaded into Semantic Web tools due to syntax errors. These idiosyncrasies imply in practice that the human costs to run experiments usually increases linearly with the number of datasets. This implies that eager researchers can use one, two, or even six datasets in their evaluations. There is no way, though, to expect hundreds, thousands or even hundreds of thousands of datasets in their evaluations. This lack of variety is due to the fact that the use of every single dataset requires some manual operations (and often repeatedly very similar operations) in order to overcome the aforementioned idiosyncrasies (Hypothesis 1).

Hypothesis 1 *The main reason why experiments are run on very few datasets is that for every dataset a certain amount of manual labor is needed.*

If Hypothesis 1 is correct, then the solution to Problem 1 is to make the human cost of using datasets independent from the number of datasets that is used (Solution 1). The human cost involved in evaluating against datasets should not only be independent of the number of datasets, but should also be low. Both these features can be achieved by fully automating the tasks of obtaining, loading, and using datasets. The LOD Laundromat solves this problem by providing a fully automated infrastructure for disseminating heterogeneous datasets in a unifom and standardized format. It (re)publishes data as cleaned datadumps and, more recently, through Web Services. Neither method is suitable for large-scale evaluation, which requires tools support for fetching, selecting and application of custom algorithms over the appropriate subset of datasets from the LOD Laundromat.

Solution 1 *Make the human effort needed to obtain, load, and use a collection of datasets independent from the size of the collection.*

While running more evaluations against hundreds of thousands of datasets will increase the generalizability of Semantic Web approaches, it also creates a new problem: selectivity (Problem 2). Not every evaluation needs to be, should be nor can be performed on all the available datasets published through the LOD Laundromat. So the question arises which datasets to choose.

Problem 2 *There are currently no means to select those datasets that are pertinent to a given algorithm or approach based on properties of the data.*

The ability to select datasets based on properties of the data also relates to another problem. It is well known, and supported by our results in Section 7.5 that evaluation outcomes sometimes differ radically for different datasets. Even though this is an interesting observation in itself, it is more pertinent to inquire as to *why* and *how* performance differs over datasets. This is a topic that has traditionally not been touched upon very often in the context of Semantic Web evaluations. LOD Lab will radically simplify future studies in the Semantic Web community to gain insight in how the performance of Semantic Web approaches relates to properties of the data (Problem 3).

Problem 3 *Current evaluations do not relate evaluation outcomes such as the performance of the evaluated algorithm or approach to properties of the data.*

The solution to Problems 2 and 3 is to allow datasets to be selected based on various criteria (Solution 2). These criteria should include a dataset's metadata (e.g., when it was crawled) and structural properties of the data (e.g., the number of unique triples it contains).

Solution 2 *Allow datasets to be selected based on their properties, including the dataset metadata, and structural properties of the data.*

7.3 RELATED WORK

7.3.1 *Evaluation Frameworks and Benchmarks*

Evaluation frameworks and benchmarks have played an important role in Semantic Web research. Many of the previous efforts focused on evaluation of storage and query answering, e.g., in the area of RDF processing and SPARQL query answering, such as the Berlin Benchmark [20], SP^2Bench [92], LUBM [44] and Fedbench [93] or LDBC [23]. Those benchmarks usually provide datasets and corresponding query sets, in order to level the playing field and allow for a fair comparisons between tools. Such approaches are a useful source for particular Linked Data research areas. However, most of these approaches present a static or even synthetic dataset. LOD Lab differs from the above by allowing experiments over an extremely high percentage of the real datasets that were published.

Relevant is the Ontology Alignment Evaluation Initiative [34] (OAEI) which presents datasets, and gold standards to relate results to, and a framework for doing so. Most importantly, the OAEI has been using the SEALs[2] evaluation platform for years now. SEALs supports experiments on ontology alignment with similar functionality as the LOD Lab supports scalability analytic experiments over multiple various heterogeneous data sources.

7.3.2 *Dataset Collections*

The most common large dataset collection to date is a Linked Data crawl published as the Billion Triple Challenge [48] (BTC). The key goal of the Billion Triple Challenge is *'to demonstrate the scalability of applications, as well as the capability to deal with the specifics of data that has been crawled from the public web'*. BTC has indeed proven to facilitate such research, and it has been used in a wide range of papers. The latest BTC dataset collection was published in 2012, and contains 1.4 billion triples. But lets be frank: where this volume used to be 'large', it has now suffered from inflation and is superseded by several larger datasets. Additionally, BTC suffers from the same idiosyncrasies found in other parts of the LOD Cloud: several BTC files contain a sizable number of duplicates and serialization errors[3]. Although the BTC has proven successful for testing algorithms for 'large' data, it lacks the meta-data for dealing with variety: neither dataset characteristics or detailed crawling provenance are available.

Another collection of datasets is LODCache, a Linked Data crawl published via a SPARQL endpoint, exposing (at the time of writing) 34.5 billion triples. Though an interesting source of data, the limita-

2 `http://www.seals-project.eu/`

3 See `http://lodlaundromat.org/resource/c926d22eb49788382ffc87a5942f7fb3`

tions that the endpoint imposes makes extracting and downloading these datasets difficult. Additionally, no information is published on the crawl mechanism behind it, and the web service lacks meta-data of both the crawl and datasets as well. I.e., this service provides data in a large volume, but lacks the meta-data to *select* datasets.

7.3.3 *Collecting data on scale*

Some resort to crawling Linked Data themselves considering the lack of available dataset collections. A common tool for this approach is LDspider [58], a Linked Data crawler which supports a wide range of RDF serialization formats, and traverses the Linked Data cloud automatically. This approach requires a large seed list of dataset locations, considering an automatic crawl would need many dereferenceable IRIs to automatically discover new datasets. Therefore, LDspider is suitable for some, but crawling larger parts of the LOD Cloud both requires manual effort for curating the seed list, as well as a significant hardware investment.

7.4 IMPLEMENTATION

LOD Laundromat provides a wealth of data, including the corresponding metadata such as crawling provenance and structural properties of data documents –as described in chapter 6. LOD Laundromat data can be accessed by writing a custom script that queries the metadata endpoint to fetch pointers to the relevant data documents. Those pointers either give access to the complete data document or to the Linked Data Fragment API for that particular document. The problem with this approach is that a user needs to be acquainted with the scraping and structural metadata schema used by LOD Laundromat. Since the latter is quite elaborate, designed with versatility rather than usability in mind, the Web Services do not implement Solution 1.

We therefore introduce *Frank*, a Bash interface that makes it easy to run evaluations against very large numbers of datasets. By implementing *Frank* in Bash it can be used by all except Windows users who do not want to install Cygwin[4]. Since *Frank* is a plain text file it requires no installation and no inclusion in a software repository or app store, nor does it depend on a specific programming paradigm. As with any Bash script, in- and output can be straightforwardly piped from and to other programs and scripts.

Frank implements Solution 1 since it allows evaluations over hundreds of thousands of data documents to be run by typing a single command (see Section 7.5 for the commands that were use to scale-up existing experiments). *Frank* implements Solution 2 by offering mech-

4 See https://www.cygwin.com/

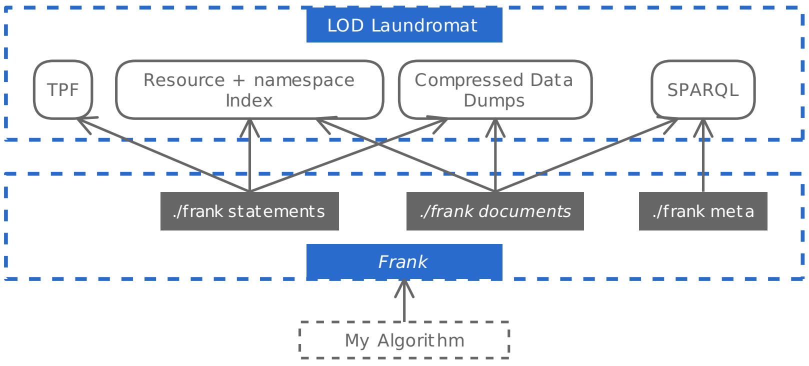


Figure 7.2: The implementation architecture for *Frank* and its dependencies on the LOD Laundromat Web Services.

anisms to select datasets according to their metadata, and structural properties (see below for the concrete properties that are supported).

Below, we discuss the three main features of *Frank*: streamed triple retrieval, streamed document retrieval, and metadata retrieval. For brevity, we sometimes abbreviate document identifiers (MD5 hashes) and use common RDF prefix shortening in results.

7.4.1 *Streamed triple retrieval*

`frank statements` allows individual atomic statements or triples to be retrieved. When called without arguments this outputs all 38 billion triples by fetching and unpacking the Gzipped LOD Laundromat data dumps in a streaming fashion. Access to a single statement is possible by using the power of Bash streams and pipes:

```
$ ./frank statements | head -n 1
<http://csarven.ca/#i> foaf:givenName "Sarven" .
```

In the above example, *Frank* is asked for any instantiation for the subject, predicate and object. The results are returned in a stream of arbitrary length, containing an arbitrary number of solutions. Since *Frank* uses the standard conventions for output handling, other processes can utilize the resultant triples by simply reading from standard input. Because *Frank* returns answers with anytime behavior, i.e., one-by-one, processes that utilize its output are able to run flexibly. Specifically, no cumbersome writing to file and/or waiting for complete result sets is needed.

If called with the command-line flags `--subject`, `--predicate`, and/or `--object`, only triples that contain the specified subject-, predicate- and object-term are returned. These three flags mimic the expressivity of the Linked Data Fragment Web API, discussed in chapter 4. They are expressively equivalent to SPARQL queries with a single-line Basic Graph Pattern (BGP) [47]. LDF supports streamed processing though a self-descriptive API that uses pagination in order to serve large results in smaller chunks. If called with a subject, predicate and/or object flag, `frank statements` interfaces with the LOD Laundromat index, discussed in chapter 6, which contains a mapping between all LOD Laundromat resources and documents. For these documents, *Frank* connects with the Linked Data Fragments API for, handling the LDF pagination settings in order to ensure a constant stream of triples. For example, in order to retrieve only persons:

```
$ ./frank statements \
    --predicate rdf:type \
    --object foaf:Person \
    --showGraph \
  | head -n 2
<http://csarven.ca/#i> rdf:type foaf:Person ll:85d...33c.
dbp:Computerchemist rdf:type foaf:Person ll:0fb...813.
```

Notice that we have instantiated the predicate and object terms and have requested the graph from which a triple originates. Or, in this case, the graph from which a person was retrieved. These graphs are the LOD Laundromat identifiers that stand for the cleaned documents containing the respective FOAF persons. To query a specific graph (or a specific collection of graphs), these LOD Laundromat document identifiers can be added as arguments to `frank statements`:

```
$ ./frank statements
    --predicate rdf:type \
    --object foaf:Person \
    http://lodlaundromat.org/resource/85d...33c
<http://csarven.ca/#i> rdf:type foaf:Person .
...
```

7.4.2 *Streamed document retrieval*

Besides querying for individual triples, *Frank* can also load entire data documents. The advantage of loading documents is that a document is a collection of triples that is published with a certain intent. Even though data documents can — in theory — be assembled randomly, in pratice there is some cohesion present in a document that cannot be found in a random collection of triples (this may be called a social aspect of RDF data).

`frank documents` interfaces with the LOD Laundromat SPARQL endpoint and index in order to find data documents. The following selection mechanisms are supported:

- Flags `--minTriples` and `--maxTriples` filter data documents based on the number of unique triples they contain.
- Filtering on the average minimum and maximum degree (as well as *in* and *out* degree), e.g. `--minAvgDegree`

- Flag `--namespace` connects to the LOD Laundromat namespace index, and only returns documents using that particular namespace. This allows for coarse selectivity of domains. For instance datasets that are possibly relevant to the bioinformatics domain can be filtered based on the `drugbank` and `chebi` namespaces. The namespace flag accepts both full URIs and de-facto RDF prefixes[5] that denote namespaces.
- Flag `--sparql` allows an arbitrarily complex SPARQL query to be evaluated against the LOD Laundromat backend. While not very user-friendly, this flag allows less often used selection criteria to be applied. Since we log SPARQL queries at the backend, we are able to add flags to *Frank* based on often requested queries.

Data documents are identified in the following two ways:

1. The URI from which the data document, cleaned by the LOD Laundromat, can be downloaded (`--downloadUri`). These clean data documents are disseminated by the LOD Laundromat as Gzipped N-Triples or N-Quads. The statements are unique within a document so no bookkeeping with respect to duplicate occurrences needs to be applied. Statements are returned according to their lexicographic order. These statements can be processed on a one-by-one basis which allows for streamed processing by *Frank*.
2. The Semantic Web resource identifier assigned by LOD Laundromat for this particular document (`--resourceUri`).

When neither `--downloadUri` nor `--resourceUri` are passed as arguments *Frank* returns both separated by a white-space.

The streaming nature of *Frank* enables combinations of streamed triple and document retrieval. The following command returns a stream of documents with an average out-degree of 15 that contain at least 100 unique RDF properties. The stream consists of N-Quads where every triple ends in a newline and within-triple newlines are escape according to the N-Quads standard. The graph name of each quadruple is the LOD Laundromat document identifier.

```
$ ./frank documents \
    --resourceUri \
    --minAvgOutDegree 15 \
    --sparql "?doc llm:metrics/llm:distinctProperties ?prop.
             (FILTER ?prop > 100)"
  | ./frank statements --showGraph
```

5 Prefixes are taken from `http://prefix.cc`.

7.4.3 *Metadata*

`frank meta` retrieves the metadata description of a given data document. It interfaces with the SPARQL endpoint of LOD Laundromat and returns N-Triples that contain provenance and structural properties for that particular document.

For example, the following returns metadata for one particular document.

```
$ ./frank documents --resourceUri | head -n 1 | frank meta;
ll:85d...33c ll:triples "54"^^xsd:int .
ll:85d...33c llo:added "2014-10-10T00:23:56"^^xsd:dateTime .
...
```

These structural properties include:

- VoID description properties such as the number of triples, entities, and the number of used properties and classes
- Additional properties not included in VoID directly, such as the number of *defined* properties and classes, and the number of literals, IRIs, and blank nodes.
- Network properties such as degree, in degree and out degree. For each of these properties we present descriptive statistics including the minimum, maximum, median, mean and standard deviation.
- Details on the IRI and literal lengths, with similar descriptive statistics.
- LOD Laundromat provenance information such as the original download location, warnings and errors

7.5 EVALUATION

To illustrate the use of the LOD Lab for evaluation purposes, we reevaluate parts of three previously published papers. A paper presenting an efficient in-memory RDF dictionary (Section 7.5.1), a paper compressing RDF in a binary representations (Section 7.5.2), and a paper exploring Linked Data best practices (Section 7.5.3). We do not aim to completely reproduce these papers, as we merely intend to illustrate LOD Lab and how *Frank* can be used by others.

Below we discuss these papers in detail and highlight the parts of their experiment we reproduce. For these experiments we illustrate how we used *Frank*, and we present the reevaluated results. The source-code of these evaluations are publicly available[6].

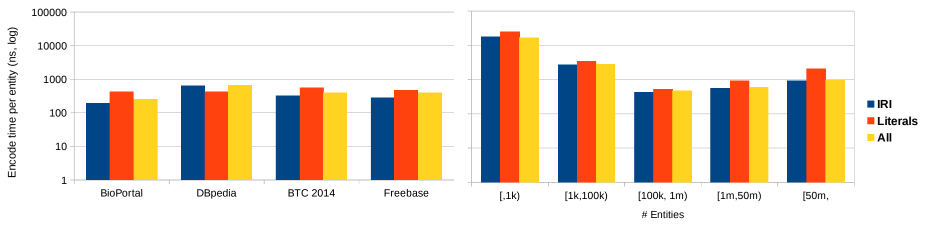
6 See https://github.com/LaurensRietveld/FrankEvaluations

7.5.1 *Paper 1: RDF Vault*

'A Compact In-Memory Dictionary for RDF data' [11] is a recent paper from the 2015 Extended Semantic Web Conference, which presents RDF Vault. RDF Vault is an in-memory dictionary, which takes advantage of string similarities of IRIs, as many IRIs share the same prefix. The authors take inspiration from conventional Tries (tree structures for storing data), and optimize this method for RDF data.

The authors measure the average encoding time per entity (time it takes to *store* a string in RDF Vault), average decoding time per entity (time it takes to *get* this string), and the memory use. Additionally, the authors make a distinction between these measurements for literals and IRIs, considering literals often lack a common prefix. In the original paper, RDF vault is compared against several baselines (e.g. a classical in-memory dictionary), and evaluated against the following 4 datasets: Freebase, the Billion Triple Challenge datasets, DBpedia and BioPortal.

We use *Frank* to re-evaluate the encoding time of RDF Vault (using the original implementation) against a larger number of datasets: for each document, we measure the average encoding time of literals, IRIs, and both combined. In order to compare these results meaningfully with the results from the original paper, we group the documents by number of entities, and present the encoding/decoding time for each group.



(a) Results from [11] (b) LOD Lab results

Figure 7.3: Average encoding time per entity (ns)

In figure 7.3 we present the original RDF vault results on the left side, and the results obtained via *Frank* on the right side. We collected the results from frank by piping all documents to the evaluation script as follows, where
`./rdfVaultEncodeDocument.sh` is a Bash script that reads the *Frank* documents from the standard input, and applies RDF Vault for each of these documents.

```
$ ./frank documents --downloadUri | ./rdfVaultEncodeDocument.sh
```

Both figures show the average encoding time of IRIs, Literals, and both combined. Our results are based on 100,000 LOD Laundromat documents[7], where we grouped documents in buckets by the number of encoded entities. The original results differ between datasets: the average encoding time of IRIs in BioPortal are 1/3 of the DBpedia encoding times. Our results show the influence of the dataset size on the encoding times (particularly considering the y log scale). Smaller datasets of less than 1,000 entities may take up to 30.000 nano seconds per entity. Similarly, datasets with between 1,000 and 100,000 entities show longer encoding times than the original paper as well. For dataset sizes which correspond to the original paper, the results are similar. The re-evaluation of these results clearly show the effect of the dataset size on encoding times. That effect was not investigated in the original paper, because the experiments were only done on a handful of datasets. As we have shown, *Frank* trivially allows to run the original experiments on hundreds of thousands datasets, immediately giving an insight in the unexpected non-monotonic relation between dataset size and encoding time per entity.

Other structural dimensions might be relevant for this paper as well, such as the number of literals in a dataset or the standard deviation of IRI or literal lengths. All these dimension are accessible using the LOD Laundromat meta-data and the *Frank* interface. E.g., to run the vault experiments for dataset with a high standard deviation in IRI lengths, run:

```
$ ./frank documents \
  --downloadUri \
  --query "{?doc llm:metrics/llm:IRILength/llm:std ?std .
           FILTER(?std >  50)}"
| ./rdfVaultEncodeDocument.sh
```

7.5.2 *Paper 2: RDF HDT*

'Binary RDF Representation for Publication and Exchange (HDT)' [35] is an often cited paper (56 at the time of writing) from the journal of Web Semantics. HDT is a compact binary RDF representation which partitions RDF datasets in three components: Header information, a dictionary, and the actual triples structure. The important gain of HDT is that the HDT files are queryable *in their compressed form* using simple SPARQL triple patterns.

In the original paper, the performance of HDT is evaluated by measuring the compression ratio of HDT compared to other compression algorithms (e.g. Gzip and Bzip2), the compression time, and by measuring the number of entries in the dictionary compared to the total number of triples. The datasets used in this evaluation are Geonames, Wikipedia, DBTune, Uniprot and DBpedia-en. A part of the evaluation is evaluated against the 400 largest datasets in the Billion Triple

7 Due to the runtime of RDF Vault and time constraints we were unable to re-evaluate this on the complete LOD Laundromat set

Challenge (BTC). This is a fairly complete evaluation, considering the number of datasets, and the use of BTC datasets.

The results we re-evaluate[8] are the compression ratios presented in [35] which were evaluated on Uniprot datasets from different sizes (1, 5, 10, 20, 30 and 40 million triples). We re-evaluate this particular research result using *Frank* by finding dataset of similar sizes (± 10%) and by measuring the compression ratio.

The LOD Laundromat documents are fetched using *Frank* and filtered to match the Uniprot dataset sizes. E.g., to select LOD Laundromat documents matching the 1 million Uniprot dataset, *Frank* searches for documents of 1 million with a deviation of 10%, and streams these document to a shell script which downloads and compresses these documents using HDT.

```
$ ./frank documents --minTriples 950000 --maxTriples 1050000
     | ./hdtCompressDocument.sh
```

Triples (millions)	**Original: Uniprot**			**LOD Lab**		
	# docs	Size (MB)	Compression Ratio	# docs	Average Size (MB)	Average Compression Ratio
1	1	89	3.73%	179	183	11.23%
5	1	445	3.48%	74	800	4.99%
10	1	893	3.27%	50	1,643	5.43%
20	1	1,790	3.31%	17	3,329	4.15%
30	1	2,681	3.27%	19	4,880	5.09%
40	1	3,575	3.26%	8	6,587	7.25%

Table 7.1: HDT Compression rates: Results from [35] on Uniprot (left side) vs. results from *Frank* (right side)

Table 7.1 shows the compression ratio for Uniprot datasets on the left side, and the average compression ratio for LOD Laundromat documents on the right side. There is a large difference between Uniprot and the LOD Laundromat datasets in both compression ratio and average document size. Another interesting observation is the high average compression ratio of LOD Laundromat documents around 1 million, compared to other LOD Laundromat documents.

To better understand such differences, we use *Frank* to evaluate RDF HDT along another dimension: the average degree of documents. We did so by searching for three buckets of datasets. Those with a low

8 We re-evaluated the latest HDT version accessible at https://github.com/rdfhdt/

(1-5), medium (5-10) and high (10+) average degree, all with at least 1 million triples:

```
$ ./frank documents --minAvgDegree 5 --maxAvgDegree 10 \
       --minTriples 1000000
     | ./hdtCompressDocument.sh
```

Avg. Degree	# docs	Compression Ratio
1-5	92	21.68%
5-10	80	6.67%
10-∞	99	4.85%

Table 7.2: HDT Compression rates grouped by avg degree

The results (See Table 7.2) show that an increase in degree of a document comes with a decrease in compression ratio.

These experimentation on a large numbers of datasets across a large number of dimensions is made easy by *Frank*, and allows researchers to both tune their algorithms to different document characteristics, as well as better understand their algorithms behavior under different conditions.

7.5.3 *Paper 3: Linked Data Best Practices*

Other than using the LOD Lab for empirical evaluations, we show how it can be used for explorative and observational papers as well. The most cited paper of the International Semantic Web Conference 2014 is 'Adoption of the Linked Data Best Practices in Different Topical Domains' [91], where the authors analyze Linked Data best practices by crawling the LOD Cloud (using LDspider [58]). Seed items for this crawl come from public catalogs, the Billion Triple Challenge, and datasets advertised on public LOD mailing lists. The crawl included 900,129 documents (IRIs that were dereferenced) and 8,038,396 resources. Documents are grouped to 1014 datasets using information from catalogs, or Pay-Level-Domain (PLD) otherwise. The paper presents a large and diverse set of statistics, including:

1. The number of resources per document
2. Datasets grouped by topical domain. These domains are fetched from online catalogs if any, and manually annotated otherwise
3. Indegree and outdegree of datasets
4. The links occurring between datasets, and the type of predicates used for linking

5. The use of vocabularies in datasets

The crawling mechanism behind these statistics strongly relies on dereferenceable IRIs. As a consequence, there is a strong link between a crawled document and the IRI it is crawled from: we know which IRI is the 'authority' for a document. This offers opportunities for e.g. grouping the datasets by PLD and finding links between datasets. This crawling mechanism differs from the LOD Laundromat, which mostly consists of (often compressed) data dumps. As a result, in LOD Laundromat, the URL `http://data.dws.informatik.uni-mannheim.de/dbpedia/2014/en/` (the official DBpedia download location) does not directly match with `http://dbpedia.org/resource/Amsterdam`, making it difficult to know the authoritativeness of the download dump IRI. I.e., the LOD Laundromat crawls many more documents and triples (including those not accessible as dereferenceable IRI), but lacks information on the authoritativeness of IRIs. Vice versa, the used crawl in [91] crawls only a fraction of the LOD Laundromat size, but retains the notion of authority. As a result, the original paper has statistics on DBpedia as a whole, where the LOD Lab results are separate for each independent DBpedia data dump.

These differences in features between both crawling mechanisms restricts the ability of *Frank* to reproduce all of the statistics from [91]. However, we chose to focus on re-evaluating the used vocabularies on the LOD Cloud, which does not suffer from these difference in crawling mechanisms. Instead, *Frank* offers a more complete perspective on the use of vocabularies, considering the number of crawled triples.

We reproduced this experiment by simply streaming all the LOD Laundromat download URIs to a script counting the namespaces[9]:

```
$ ./frank documents --downloadUri \
    | ./countNamespacesForDocument.sh
```

Table 7.3 shows the 10 most frequent occurring namespaces in documents. In the original paper these counts are grouped by dataset (i.e. groups of documents), where we present these statistics on a document level alone.

This table shows striking differences: where the *time* namespace is used in 68.20% of the LOD Laundromat documents, it does not occur in the top 10 list of [91]. Similarly, the *cube* namespace occurs in 23.92% of LOD Laundromat documents, and is missing from the original top 10 list as well.

The crawling method behind both approaches, and the method used by [91] to group documents as datasets can explain these discrepancies. Therefore, we do not claim to have the right answer for

9 Using the namespace list of `http://prefix.cc`, similar to the original paper

Original [91]			LOD Lab		
Prefix	#datasets	% datasets	Prefix	#docs	% docs
rdf	996	98.22%	rdf	639,575	98.40%
rdfs	736	72.58%	time	443,222	68.19%
foaf	701	69.13%	cube	155,460	23.92%
dcterm	568	56.01%	sdmxdim	154,940	23.84%
owl	370	36.49%	worldbank	147,362	22.67%
wgs84	254	25.05%	interval	69,270	10.66%
sioc	179	17.65%	rdfs	30,422	4.68%
admin	157	15.48%	dcterms	26,368	4.06%
skos	143	14.11%	foaf	20,468	3.15%
void	137	13.51%	dc	14,423	2.22%

Table 7.3: Top 10 namespaces used in documents

these kind of statistics. Instead, we show that the LOD Lab approach allows for large scale comparisons for these kinds of Linked Data observational studies.

7.6 conclusion

The distributed nature of the Semantic Web, the wide range of serialization formats, and the idiosyncrasies found in datasets, make it difficult for Linked Data scientists to use the Semantic Web as a true large-scale evaluation platform. As a consequence, most research papers are only evaluated against a handful of datasets, and do not take the true variety of Linked Data into account (Problem 5).

In this chapter we presented LOD Lab, a new way of conducting Linked Data experiments that incorporates both volume and variety while at the same time allowing the set of considered data documents to be limited according to domain-specific and/or structural constraints. This is achieved by using the LOD Laundromat backend together with the simple yet versatile programming interface *Frank* that allows large-scale Linked Data evaluations to be run from the command-line.

The viability of the LOD Lab approach was demonstrated by scaling up three experiments reported in recent Semantic Web conference publications. These re-evaluations show that evaluations over Linked Data can now be performed without the human effort having to increase linearly in terms of the number of datasets involved. In addition, the re-evaluations show that the combination of volume, variety and selectivity facilitates a more detailed analysis of Semantic Web algorithms and approaches by relating evaluation outcomes to properties of the data.

8 YASGUI AS A MEASURING DEVICE FOR LINKED DATA USE

Where the previous chapter analyzed Linked Data from a ***data-centric*** *perspective, this chapter takes a* ***usage*** *perspective. Information and research about the use of Linked Data by consumers is limited. An important reason behind this lack of information is the unavailability of consumer usage data. This problem became evident in our SampLD (chapter 3) experiment setup, where the breadth of the evaluation was strongly restricted by the low number of available query logs. The low number of available query logs restrict the possibilities of Linked Data scientists to study the use of Linked Data at large.*
In this chapter we track Linked Data usage at the client *side, using the YASGUI query editor from chapter 5 as a measuring device for interactions with the Linked Data Cloud. It enables us to determine what part of the Linked Data Cloud is actually used, what part is open or closed, the efficiency and complexity of queries, and how these results relate to commonly used dataset statistics.*

This chapter is an updated version of:

Laurens Rietveld and Rinke Hoekstra. YASGUI: Feeling the Pulse of Linked Data. In *Knowledge Engineering and Knowledge Management*, pages 441–452. Springer, 2014

Contributions to this paper:

I am responsible for the main contributions of this paper and the analysis of query logs.

8.1 INTRODUCTION

As the Linked Data cloud grows both in size and complexity, it becomes increasingly interesting to study how, and what parts are being used for which purpose. There are currently two approaches: the study of query logs, such as provided by the USEWOD series, and of gathering dataset statistics [18, 31].

PROBLEM Both approaches only partially fulfill their intended purpose because 1) they are restricted to a small number of datasets and 2) the information is collected at the publisher rather than the user-end of the development pipeline. What is missing for analytics over the Linked Data cloud is a dataset independent data collection point, which can act as a kind of observational lens.

Take as analogy the query logs collected by search engines, such as Google or Yahoo. These have become the primary proxies for studying information need on the World Wide Web. This has to do with the unique position those engines have as *the* central filters through which users access the otherwise distributed information. Indeed, the business model of web search giants is founded on their ability to adequately target advertisements to users, based on their search behavior. For the Web of Data, not a single such entry point currently exists.

CONTRIBUTIONS This chapter uses statistics generated by the YASGUI service from the previous section, which has the potential for becoming such an observational lens for the Linked Data cloud. When given permission to do so, it acts as a measuring device for Linked Data, by tracking the actions of users. This provides insight in how we interact with Linked Data. As YASGUI works for every SPARQL endpoint, it can collect information on more than the Linked Data cloud we were previously aware of, including endpoints inaccessible from the internet. We show how the information collected through this SPARQL interface increases our knowledge of Linked Data, such as which part of the Linked Data cloud is actually used, what part is open and accessible, the complexity of man-made queries, and the most commonly used namespaces.

The matter of uptake is the critical factor as to whether or not YASGUI will eventually collect sufficient, valid, and unbiased data, and can become a proper observational lens. In Section 8.3 we argue that there are sufficient incentives for users to use it as their point of entry for the Linked Data cloud as it is the most user friendly, intuitive and interactive interface to date.

CONTENT This chapter is structured as follows. Section 8.2 discusses related approaches to the study of the Linked Data cloud. Section 8.3 outlines our methodology. Section 8.4 discusses how the use of YASGUI allows us to analyze the Linked Data cloud, and what we can observe from the data we gathered since its launch. We conclude in section 8.5.

8.2 RELATED WORK

WHERE IS THE LINKED DATA The most well known depiction of Linked Data is a "cloud" of 311 connected ("linked") datasets [18]. The size of circles depends on the size of the datasets, and links represent the reuse of identifiers between datasets. Not only is the latest version outdated (November 2011), it is also rather limited in that it is based on metadata that were manually registered in the Datahub CKAN catalog and which have an open license. This makes the analysis quite unreliable and static: there is no check as to whether the size and number of links registered correspond to reality, and there is no indication of whether the data is actually being used.

LODStats assesses the availability of the information in the Datahub. It attempts to access or download registered datasets, and extracts structure and schema characteristics. Results show that for various reasons, only a fraction of the registered data is accessible in practice. Similar to Ding et al. [32], LODStats provides statistics of the popularity of *namespaces* (and thus vocabularies) across a large body of RDF. However, counting namespace occurrence does not give insight in the *spread* of a namespace: is it popular within an isolated *cluster* of interlinked datasets, or is its use evenly spread out?

Hogan et al. [56] performed an in depth analysis of the quality of Linked Data that was crawled from the Web as part of the Billion Triple Challenge in 2011[1], focusing in particular on the adherence of the datasets to Linked Data principles such as dereferenceability of IRIs. These efforts show that accessibility is hampered by the reliability of services hosting the data. Also, the quality and standards-compliance of Linked Data published is relatively low, given the number of tools that support Linked Data manipulation [13]. SPARQL Endpoint Status continuously tracks the up-time of SPARQL endpoints, which features they support, and which endpoints publish dataset statistics. This is useful for observing the current state of accessible SPARQL endpoints, though again, the set of endpoints is limited to those published on CKAN. Sindice collects data from the Web of Data by crawling web pages for RDFa and Microformat markup. It also collects data from endpoints through a manual procedure (only 8 out of 311 CKAN datasets are indexed). Sindice provides an extensive amount of information about the Web of Data, taking a broader perspective than focusing on SPARQL endpoints alone. In short, we have an incomplete knowledge of what Linked Data is, and how much resides where.

USAGE OF LINKED DATA To better understand the usage of Linked Data, the USEWOD [16] workshop series initiated a challenge to analyze server logs from six well known SPARQL query endpoints (datasets): DBpedia, Semantic Web Dog Food, BioPortal, Bio2RDF, Open-BioMed, and Linked Geo Data.[2] Clearly this only covers a small portion of the number of datasets registered in the Datahub, making it difficult to extrapolate to the full size of the Web of Data. Also, the query logs make no distinction between 'machine queries' – queries executed by applications – and manual interaction with Linked Data [72]. In previous work [85], we quantified exactly this difference, by comparing the YASGUI set of man-made queries with

1 See http://km.aifb.kit.edu/projects/btc-2011/.

2 See http://dbpedial.org, http://data.semanticweb.org, http://bioportal.bioontology.org/, http://www.open-biomed.org.uk/, and http://linkedgeodata.org, respectively.

queries taken from server logs (containing mostly machine queries). We showed that queries from each sets differ greatly in size, the range of SPARQL features they use, and complexity.

8.3 METHODOLOGY

The discussion of related work shows that we can only sketch a reliable picture of the Linked Data cloud that includes both the presence and use of datasets if we tap into where interaction with the Linked Data cloud occurs: on the client side. For this purpose, YASGUI is a knife that cuts both ways: it is a tool that makes it easier to interact with Linked Data, and it allows us to gather an unprecedented wealth of usage data if users opt-in. Other SPARQL clients are in a position to track *user* queries specifically, but –as we showed in chapter 5– they lack the feature richness needed to study SPARQL usage across datasets and to attract sufficient numbers of users. To the best of our knowledge, none of these clients track and analyze user data at scale. And despite the increasing number of publishers who improving their user interface using YASQE and YASR, we have only seen an increase in the number of user sessions on the yasgui.org web service.

Our method follows two steps, we 1) developed the YASGUI SPARQL client that can attract users and allows access to all SPARQL endpoints, we then 2) ask permission to log user queries, and analyze these queries along various dimensions such as type, namespaces, endpoints, complexity, etc. We discuss these dimensions in more detail below.

We use Google Analytics[3] to log the actions of users that explicitly allow us to do so: every user is presented with an opt-out form in which users may choose to disable logging completely, or to disable logging of endpoints and queries only. User actions include the queries a user executes, the endpoint they use, the time it takes to get the query response[4], the use of the URL shortener service, and more general information such as (an estimate of) the user's location and the local time.

Given these logs we can study the following:

1. How do the SPARQL endpoints registered in CKAN relate to the endpoints used in YASGUI? How big is the overlap?
2. Looking at the datasets hosted by these endpoints, what part of the dataset is actually needed to answer the queries posed against it?
3. What namespaces are most commonly used in the the queries?

3 See http://www.google.com/analytics/

4 Logging the execution time of queries is added recently. Therefore, these results are not included in this chapter

Queries		Complexity	
Total Queries	91,008	⩾ 1 joins	49.08%
Valid Queries	64,643	⩾ 1 `VCC` pattern	52.66%
Unique Queries	40,347	⩾ 1 `VCV` pattern	49.47%
		⩾ 1 `VVV` pattern	13.78%
`SELECT`	95.48%	⩾ 1 `CVV` pattern	11.69%
`DESCRIBE`	0.61%	⩾ 1 `VVC` pattern	10.64%
`ASK`	1.69%	⩾ 1 `CCV` pattern	9.38%
`CONSTRUCT`	2.22%	⩾ 1 `CCC` pattern	0.66%
`INSERT`	0.00%	⩾ 1 `CVC` pattern	0.3%

Accessible endpoints	#	%
CKAN endpoints	129	72.36%
Not in CKAN	222	8.30%

Inaccessible endpoints	#	%
Probably incorrect	799	1.12%
Private (local) endpoint	248	16.70%
Only contains public data	247	2.65%

Figure 8.1: Statistics on the use of Linked Data as measured from the YASGUI logs

4. How complex are the queries, how many are there, what tasks are they used for?

At a more *fine-grained* level, we analyze the complexity of the query sets, using the methods described in [37, 81]. We look at two aspects: the triple pattern structure and the number of joins. The number of triple patterns used in queries, as well as the structure of these triple patterns is a good indication of the complexity of queries. We use the method described in [37] to determine types of joins, and the number of joins per query. Each element in a triple can be a variable (V), or a constant (C). For instance, `[] rdf:type ?object` can be classified as `V C V`. When two triple patterns have one variable in common, the query engine would need to join both. Given the features of YASGUI compared to other clients, we expected that our queries have a higher complexity than SPARQL queries obtained from server-side logs.

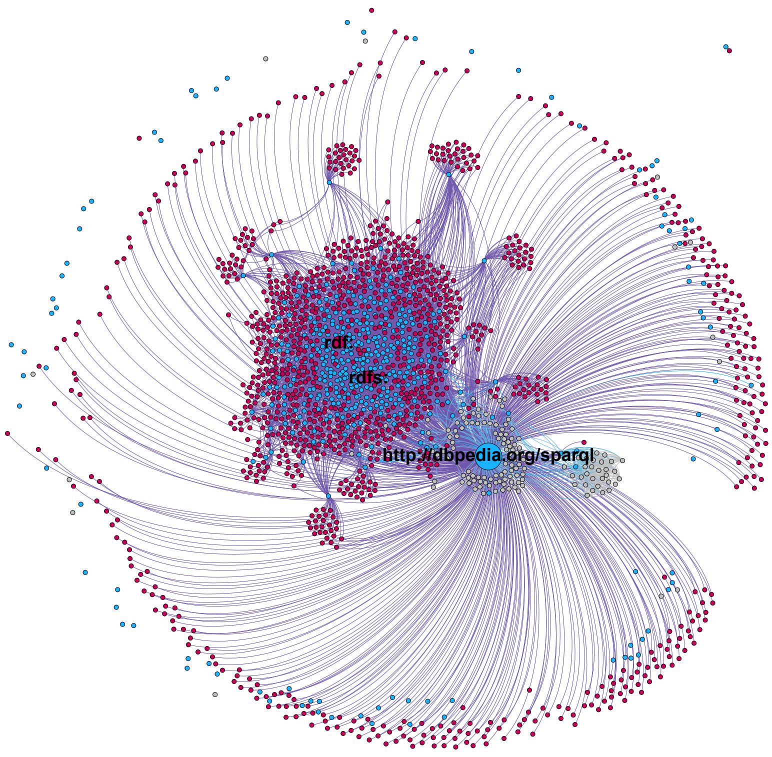


Figure 8.2: Usage of datasets in the 'Linking Open Data' cloud by YASGUI users, and non-listed endpoints, related to namespaces. Red circles are namespaces, grey circles are datasets registered at the Datahub, blue circles are datasets that appear in YASGUI queries (not all of which are in the Datahub). The size depends on the number of links stated in the Datahub, combined with the number of links with namespaces.

8.4 RESULTS

Since the public launch of YASGUI 2 years ago, it has attracted 6,670 unique visitors from over 96 countries[5]. Until now, 1,700 (56%) of our users allow full logging, 37% disabled logging of endpoints and queries only, where the remaining 7% disabled logging altogether. Of the 56% who allowed logging, we tracked 91,008 queries, executed against 1,645 SPARQL endpoints. This means that in total, an estimated additional 70,000 queries were executed through YASGUI without our knowledge. To give some context to the number of visitors: the Semantic Web Dog Food project lists 11,184 unique persons.[6]

5 Statistics are from July 2015
6 Number taken from http://data.semanticweb.org/ (July 2015)

8.4.1 *Endpoint Usage*

In the well known Linked Open Data cloud [18], edges between datasets indicate links between resources of both datasets, and the node size indicates the number of triples of the dataset. As mentioned in section 8.2, this dataset catalog has strong curation limitations. With our logs, we can observe the actual usage of endpoints in the LOD-cloud. The datasets in Figure 8.2 are retrieved from CKAN using the same code of the original LOD diagram[7]. The gray endpoints have never been accessed via YASGUI, where the blue nodes are, indicating that only a small part (20%) of the endpoints in this particular LOD-cloud representation is used. Note, however, that not all datasets in CKAN are hosted online, which means that this overlap may be larger in reality.

We divide the endpoints into five categories (See Table 8.1). To filter typographic errors, we reduce the list of 1,645 to a list of 846 endpoints that are accessed more than two times. For this list, we check whether each endpoint is accessible and whether it occurs in the Datahub catalog. Inaccessible datasets do not only contain private or closed data: users might store a copy of a CKAN dataset locally for analysis. Therefore, we analyze the namespaces in the corresponding queries of these endpoints: whenever a namespace does not occur in the prefix.cc[8] collection, we assume this endpoint contains private data. This gives us 248 endpoints, from which we can derive that 16.70% of all queries are executed on an endpoint containing private data. In other words, from the YASGUI usage perspective, 80.66% of the Linked Data Cloud is open, where the other 19.34% is closed.

8.4.2 *Dataset Usage*

We take inspiration from our experiment setup in SampLD (Section 3.5), where rewrite SPARQL `SELECT` queries to `CONSTRUCT` queries to measure what part a data set is touched by queries. We apply this same methodology on a larger scale, using the public SPARQL endpoints logged in YASGUI. This gives us, for each pair of query and endpoint, the triples needed to answer the original `SELECT` query (see Figure 8.3). We performed this analysis for the 27 most often used public datasets, of which **17** endpoints were or became inaccessible during our experiments. This shows that for most endpoints, less than 0.2% of the dataset is actually needed to answer our queries. DBpedia (the most popular endpoint) requires only 0.18% of its size, to answer 54,614 queries.

7 See `https://github.com/lod-cloud/datahub2void` (6 May 2014)
8 See `http://prefix.cc`

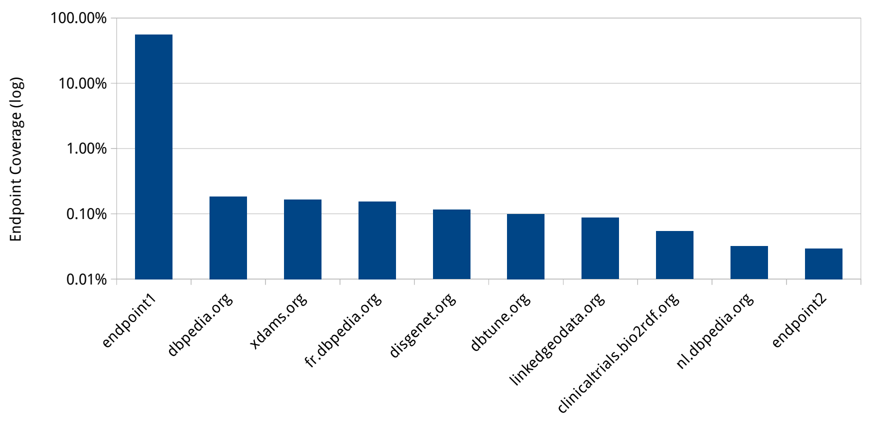


Figure 8.3: Query coverage of 10 most frequently used and accessible datasets in YASGUI (log scale). Endpoints not available through the Datahub are anonymized

8.4.3 *Namespace Usage*

The query logs allow us to see what type of information from the Web of Data is used. Namespaces are good candidates to look at, as they reflect the use of often domain specific vocabularies. Table 8.1 shows the 10 most common namespaces used in all the queries. The RDF type and RDF schema namespaces are the most popular ones. Table 8.2 compares the pre-LOD statistics of [32] with that of users on Prefix.cc and YASGUI (Prefix.cc provides no numbers, only a ranking). Six out of eight original namespaces are still high ranked in the Linked Data age. The highly ranked RSS namespace in Prefix.cc can be explained by (non-semantic) web developers.

Table 8.3 compares the namespace use between YASGUI queries and LOD Laundromat documents. The higher ranked RDF Schema and OWL namespace (8th) in the YASGUI ranking indicates that users do rely on schema information. Using the pairing of namespaces and datasets, we can create a map of the commonalities between datasets. Where table 8.2 showed that DBpedia-based namespaces are frequently used in queries, table 8.3 shows that they are less frequently used across datasets.

8.4.4 *Query Analysis*

Table 8.1 shows a number of statistics based on a total of 91,008 queries collected via YASGUI. After filtering invalid queries using the Jena[9] query parser, this number drops to 64,643 queries. This large number of invalid queries is partly due to the strict parsing of Jena. Some queries may not conform to the SPARQL standard, but return valid SPARQL results for certain endpoints regardless. For example,

9 See `http://jena.apache.org/`

Namespace	#	%
http://www.w3.org/2000/01/rdf-schema#	29,219	45.20%
http://www.w3.org/1999/02/22-rdf-syntax-ns#	28,451	44.01%
http://dbpedia.org/ontology/	18,223	28.19%
http://dbpedia.org/property/	16,395	25.36%
http://dbpedia.org/resource/	16,146	24.98%
http://xmlns.com/foaf/0.1/	7,851	12.15%
http://www.w3.org/2001/XMLSchema#	5,602	8.67%
http://www.w3.org/2002/07/owl#	4,904	7.59%
http://purl.org/dc/terms/	4,624	7.15%
http://dbpedia.org/	3,739	5.78%

Table 8.1: YASGUI: Top 10 namespaces occurring in queries

Namespace	Ding et al.	prefix.cc	YASGUI
http://www.w3.org/1999/02/22-rdf-syntax-ns	1	2	2
http://www.foaf-project.org (or http://xmlns.com/foaf/0.1/)	2	3	6
http://purl.org/dc/elements/1.1	3	5	13
http://www.w3.org/2000/01/rdf-schema	4	6	1
http://webns.net/mvcb	5	39	*none*
http://purl.org/rss/1.0	6	9	*none*
http://www.w3.org/2001/vcard-rdf/3.0 (or http://www.w3.org/2006/vcard/ns)	7	32	19
http://purl.org/vocab/bio/0.1	8	52	1,015

Table 8.2: Top 8 namespace rankings, comparing Ding et al. [32], Prefix.cc and YASGUI.

a query containing a 'bif:' IRI, supported by Virtuoso endpoints, is marked as invalid. When we remove duplicate queries from the query set, 40,347 queries remain. The numbers below involves the complete list (i.e. including duplicates) of valid queries.

We observe that the majority of queries executed via YASGUI are `SELECT` queries. Both `ASK` and `DESCRIBE` queries, amount to a fraction of the YASGUI query logs (1.69% and 0.61% respectively). We believe this shows that users prefer the more common `SELECT` keyword instead. Rather than the boolean value returned by an `ASK` query, the user may evaluate the query results from the `SELECT` query as-is. We expect this is due to the familiarity users have with `SELECT` queries; only a few of them will opt for an `ASK` or `DESCRIBE` query. Interestingly, the number of executed `CONSTRUCT` queries amounts to only 2.22%, which might indicate that data re-use via SPARQL queries is uncommon. The YASGUI logs show that roughly 5% out of the `SELECT` queries is accounted for by SNORQL-style queries.

Namespace	LOD Laundromat		YASGUI	
http://www.w3.org/1999/02/22-rdf-syntax-ns#	1	98.40%	2	70.84%
http://www.w3.org/2000/01/rdf-schema#	2	68.19%	1	72.99%
http://xmlns.com/foaf/0.1/	3	23.92%	6	17.44%
http://purl.org/dc/terms/	4	23.84%	9	10.42%
http://www.w3.org/2002/07/owl#	5	22.67%	8	10.99%
http://www.w3.org/2003/01/geo/wgs84_pos#	6	10.66%	12	6.72%
http://rdfs.org/sioc/ns#	7	4.68%	893	0.00%
http://webns.net/mvcb/	8	4.06%	*none*	0.00%
http://www.w3.org/2004/02/skos/core#	9	3.15%	10	8.15%
http://rdfs.org/ns/void#	10	2.22%	44	0.63%

Table 8.3: LOD Laundromat: Top 10 namespaces based on occurrences in documents

Another observation concerns the complexity of SPARQL queries. Figure 8.1 shows that 49.08% of the queries contain one or more joins, and the most common triple patterns consists of VCC and VCV triple patterns. Such statistics can be used for optimizing man-made queries, and tell us more about how people query Linked Data. When we take a closer look at the individual queries contained in the logs, we see that we can glean information about more than the queries only. First, following [67], which recommends to minimize unnecessary `OPTIONAL` clauses, we observe that 72.66% of executed queries that contains the optional `OPTIONAL` keyword are inefficient: we compared query results *with* and *without* the `OPTIONAL` to detect these. This high percentage may be partly explained if we consider that SPARQL clients are often used for exploratory tasks. Finding task-trails in query logs [109] will allow us to better detect this behavior.

8.5 CONCLUSION

Where chapter 7 takes a data-centric approach for increasing the variety in Linked Data research, this chapter takes a different angle and uses YASGUI as a measuring device to increase the variety from a *usage* perspective.

Only 2 years after the release of YASGUI, we have collected a query log that spans a larger part of the web of Linked Open Data compared to other query catalogs. This gives unprecedented insight into how we actually use the Linked Data cloud, and what part of the Linked Data cloud we use. Using the collected data, we were able to analyze the efficiency of queries, what part of the used Linked Data cloud is open or closed, what part of these datasets we use, the complexity of queries, and the shared use of namespaces over all the endpoints. We are aware the results presented in this chapter are not (yet) fully representative and unbiased. However, alternative dataset statistics suffer from the same problem: these are either based on (outdated) dataset catalogs, or on an opt-in basis, making these statistics incomplete.

The number of yasgui.org user sessions increases each year, despite data providers hosting their own version of YASGUI (or the related YASQE and YASR libraries). Therefore we believe that with an increase in uptake of YASGUI, we will be able to make our claims even stronger and understand the use of Linked Data even better. More data allows us to recognize more fine-grained patterns, e.g. to identify a relation between the structure of a dataset and its queries, which categories of queries exist, and how these query categories relate to typical tasks. This chapters shows first steps in this direction.

To conclude, this chapter introduces a tool, dataset and methodology that increase our knowledge of the use of Linked Data. The query catalog is an improvement in variety over other public query catalogs, and allows for analyzing the Linked Data cloud in the broadest sense: what datasets exist, how are they used, and for what purpose? The amount of data we gathered in this short period of time, and the increasing uptake of YASGUI, promises an even clearer picture of Linked Data in the future.

9 CONCLUSIONS

This chapter summarizes and discusses our results, and presents directions for future work.

9.1 RESULTS

The key contribution of this thesis is a number of advancements that enables Linked Data publishing, consumption and research. We decreased the hardware costs and human effort of Linked Data publishing, and increased the utility and accessibility for Linked Data consumption. In the introduction (Chapter 1) we introduced five problems to structure these contributes. These problems were studies from the perspective of three different stakeholders:the Linked Data provider, Linked Data developer, and Linked Data scientist. In this section, we discuss how we addressed these five problems, the limitations of our solutions, and the future work on each of these problems.

9.1.1 *Impractical Linked Data re-use (Problem 1)*

The number of Linked Datasets that do not follow standards and best practices makes Linked Data re-use impractical.

Publishing Linked Data as static files seems straightforward, but as we showed in chapter 2, even this method can be difficult in practice: many Linked Datasets still do not adhere to standards and best practices. And although the state-of-the-art in standards, guidelines and tools may incrementally improve the state of Linked Data, they do not offer an immediate solution for publishing clean and standards-compliant Linked Data.

We developed a centralized service called the LOD Laundromat, that re-publishes a clean version of as many Linked Open Datasets as possible. The republished files are served in a canonical compressed N-Triples or N-Quads format, and either crawled via Linked Open Datasets or manually added by users. This approach presents Linked Data providers with a cleaned and hosted version of their datasets. We focused on this problem from the Linked Data providers viewpoint, though chapters 4, 6 and 7 showed that the LOD Laundromat finds a broader usage: developers are presented with a wealth of uniform clean data, and scientists can now study and use Linked Data at large with minimum effort.

There are no inherent limitations with respect to the variety and quantity of Linked Data that LOD Laundromat re-publishes: the low hardware footprint of the published compressed data files (313GB in disk space) scale linearly with the number of crawled triples, and can be hosted on most consumer-grade hardware.

LOD Laundromat takes a pragmatic approach towards dataset dynamics: dataset snapshots are re-published, but future versions of that dataset that are published under the same URL may not necessarily be included. Several research problems have to be solved in order to store multiple incremental versions of datasets, including detecting whether a dataset changes (HTTP `last-modified` headers are often incorrect), and storing delta's between several versions of the same dataset. Particularly blank nodes are cumbersome in this respect, as multiple serializations of *the same* dataset can result in syntactically *different* files. These problems are difficult to solve, without losing a key feature that keeps LOD Laundromat scalable: streamed conversion of Linked Datasets to their clean siblings.

The LOD Laundromat is not *the* solution for publishing Linked Data, nor does it intend to be. The distributed nature of Linked Data results in many different dataset owners, thus many different places to introduce errors. This is similar to the World Wide Web, which suffers from an enormous amount of publishers, each hosting possibly non-standards-compliant websites. Web browser companies have learned to deal with processing such websites on a best-effort basis. Most RDF parsers though are less flexible and may return a fraction of the triples from the original (syntactically invalid) document. The Linked Data community therefore needs a better overview and benchmark on how RDF parsers resolve syntax errors, and how to resolve syntactic problems on a best-effort basis with minimum information loss.

9.1.2 *Queryable Linked Data is expensive to host (Problem 2)*

The expressivity of SPARQL demands powerful triple-stores, and makes large Linked Datasets expensive to host.

The de-facto standard for hosting queryable Linked Data is the SPARQL endpoint. Its flexibility and rich querying language offers advantages for Linked Data providers, but comes at a cost: these SPARQL triple-stores are expensive to host. We presented two solutions to this problem.

First, we presented a sampling method called SampLD (Chapter 3), that reduces the dataset size, and thus reduces the hardware costs for hosting a SPARQL endpoint. SampLD aims at finding the smallest part of the data that entails as many of the answers to typical SPARQL queries as possible. SampLD includes a number of sampling methods, each using different properties of the graph structure. Our evaluation showed that we can determine to a surprisingly high degree which triples in the dataset can safely be ignored and which cannot. This approach is suitable for Linked Data providers that would like to host Linked Data via the flexible SPARQL language, and who are willing to sacrifice completeness.

A second, orthogonal, solution is to reduce query language complexity instead of reducing the completeness of query results (Chapter 4). We did so by combining the LOD Laundromat with a Triple Pattern Fragment API that only supports simple triple pattern queries. This resulted in a low cost Linked Data API that consumes a fraction in memory and processing power compared to SPARQL triple-stores. This publishing process provides opportunities beyond that of the Linked Data provider: developers can build applications on top of a wide range of LOD Laundromat datasets and APIs, where scientists can browse, query and analyze Linked Datasets that are otherwise only hosted as static files.

Both approaches come with their limitations. Where SampLD creates datasets that are cheap to *host*, it requires significant hardware resources to *generate* the sample. And where Triple Pattern Fragments allows for scalable Linked Data querying, it requires effort from clients to answer questions more complex than triple patterns. Additionally, both approaches do not support incremental updates to the dataset.

Research towards scalable solutions for hosting queryable Linked Data mostly focus on the efficiency of SPARQL triple-stores. The two approaches we presented take a completely different perspective to this problem, and provides several opportunities to investigate further. This includes questions on how to propagate incremental dataset changes to the corresponding sample, and how to update the HDT file used by Triple Pattern Fragments in an incremental fashion.

The Triple Pattern Fragments approach offers interesting opportunities to distribute the computation between clients and servers. Where Linked Data is distributed by nature, the computation always takes place on the server side. Recent attempts [74] show that Triple Pattern Fragments can be used to answer SPARQL queries, where complex operations such as joins are performed on the *client* instead of the server. Such developments can help Linked Data providers by offering cheap scalable backends with SPARQL expressivity, where clients do the heavy lifting.

9.1.3 *Formulating SPARQL queries is difficult (Problem 3)*

The expressivity and complexity of SPARQL makes formulating queries difficult.

The complexity and expressiveness of SPARQL makes it an unforgiving and difficult query language. But as we showed in chapter 5, the state-of-the-art in query editors does not provide the tooling and features that web developers are accustomed to. We improved the state-of-the-art by developing YASGUI, a SPARQL query editor accessible from the browser, that has many features known to web developers such as syntax highlighting, syntax checking, and auto-completion functionality.

The service we provide cuts both ways: it provides Linked Data developers, scientists and providers with an interface that improves accessibility to SPARQL triple-stores, but it also presents a unique opportunity to monitor Linked Data usage (See chapter 8) via the centralized YASGUI service.

The flexible schema of Linked Datasets and SPARQL make it difficult to know the data behind a SPARQL endpoint a-priori. YASGUI attempts to improve accessibility of SPARQL triple-stores by using auto-completion services such as the Linked Open Vocabulary API. However, this does not completely solve the black-box problem of SPARQL endpoints: the suggested auto-completions are not based on the dataset in question, and are not related to the query that is being executed. Ideally, YASGUI provides these suggestions using the SPARQL endpoint itself (something that SPARQL in principle supports), but experience shows that several SPARQL triple-stores are not efficient enough to answer such queries in a timely (< 1 second) manner. VoID descriptions for SPARQL endpoints partly solve this issue, but uptake of VoID is limited. Another solution to this black-box problem involves a dedicated service that is tightly coupled to the corresponding dataset [27].

We expect a generic solution to the black-box problem of SPARQL endpoints to entail both the use of dataset descriptions descriptions, and more efficient SPARQL triple-stores that respond to auto-completion SPARQL queries in a timely manner.

Most older multi-purpose query editors require local installation, while YASGUI is built for the web infrastructure. We see this trend towards tooling in web infrastructure in other Linked Data areas as well. The ontology editor Protégé is now available in a web version, and the turtle editor Snapper[1] is completely web-based as well. This move towards web applications is enabled by recent releases of JavaScript libraries, such as the RDF parser N3.js[2] and the SPARQL parser SPARQL.js[3] These developments are promising for Linked Data uptake and development, as these applications do not require local installation of applications, are operating system independent, allow for isomorphic applications (where clients and servers share

1 See http://jiemakel.github.io/snapper
2 See https://github.com/RubenVerborgh/N3.js
3 See https://github.com/RubenVerborgh/SPARQL.js

the same code), and are part of the same infrastructure of Linked Data: that of the World Wide Web.

9.1.4 *Problematic Access to Linked (Meta-)Data (Problem 4)*

The distributed nature of Linked Data and its corresponding meta-data makes locating and accessing the right Linked Data difficult.

The distributed nature of Linked Data and the absence of structural descriptions of datasets makes locating and accessing Linked Datasets difficult. A centralized solution such as the LOD Laundromat does not directly solve this issue, as finding datasets according to structural criteria still requires manual processing and exploration.

Therefore we extended the LOD Laundromat with a structural Meta-Dataset that includes an IRI and namespace to dataset index, dataset characteristics, and provenance information about the performed processing steps. We published this Meta-Dataset via SPARQL and public JSON APIs.

The limitations of this approach is the same as the LOD Laundromat: despite the wide scope of LOD Laundromat, it does not cover the *complete* array of available Linked Datasets, nor does it support dataset dynamics.

9.1.5 *Variety of Linked Data Research (Problem 5)*

Linked Data research does not take the true variety of Linked Data into account

As shown in chapter 7, Linked Data research is suffering from the unavailability of resources and tools to study Linked Data and its use at large. As a result, –from both the usage and data perspective–, Linked Data research does not take the true variety into account.

The first approach we presented in chapter 8 focused on Linked Data *use* –a research area that is strongly restricted by the limited number of available query logs. Our work on SampLD (chapter 3)corroborates the limited availability of usage data, as this evaluation was strongly limited by the number of available query logs. Where these query logs are all collected from SPARQL servers, we proposed to track Linked Data usage from the *client* side using the YASGUI query editor as a measuring device. We show how the queries collected by YASGUI enable us to investigate usage patterns that are difficult to measure otherwise.

The second approach taken in chapter 7 focuses on increasing the variety from a *data-centric* perspective. We showed that existing research only evaluates on a handful of datasets, and we presented an alternative approach for running experiments on a much broader scale, using *Frank*. The *Frank* interface connects to the LOD Laundromat, the corresponding Meta-Dataset, and the exposed LOD Laundromat Triple Pattern Fragments API. It enables users to select documents or triples from the command-line, based on properties such as dataset characteristics or simple triple pattern fragments. By re-evaluating three recent experiments using *Frank*, we showed that simply rerunning existing experiments within this new evaluation paradigm brings up interesting research questions as to how algorithmic performance relates to (structural) properties of the data.

Both approaches improve the state-of-the-art by allowing a broader perspective on Linked Data, but these approaches suffer from limitations as well. Linked Data usage is not restricted to SPARQL queries alone, and includes e.g. statistics on visited dereferenceable IRIs. Such statistics are tracked per dataset on the server side, and difficult to monitor on a broader scale. Additionally, our LOD Lab approach is restricted by the same limitations as the LOD Laundromat, in that it does not cover the *complete* array of available Linked Datasets.

9.2 FINAL OBSERVATIONS

In the remainder of this conclusion we discuss some general observations related to the topics discussed in this thesis.

9.2.1 *Structure Matters*

Besides the semantics expressed via RDF Data Models, there is implicit information captured by the network structure that may not be explicitly represented in the model itself. As shown in this thesis, this structural information can be used in different scenarios. SampLD (chapter 3) shows how a network analysis of these structural properties can be used to estimate the relevance of triples. And LOD Lab (chapter 7) shows how structural properties of datasets influence experiment results of recent Linked Data publications. Both approaches solely use structural information, and ignore the semantics of the graph. These approaches also show that structural properties of Linked Data are closely related to the field of network analysis. considering that RDF graphs can be approximated as directed graphs with labeled edges. Our work on SampLD shows that RDF graphs are not directly suited to all network analysis algorithms though: RDF predicates can be considered edge labels of a directed graph, but it would ignore the context of that predicate. To use a wider range of network analysis algorithms, a conversion between RDF to a more suitable network representation is required. The list of RDF rewrite methods introduced in SampLD is not complete though. Other rewrite methods are possible, such as converting an RDF graph to a bipartite network [51]. To bridge the gap between the field of network analysis and RDF, a more exhaustive study is needed to better understand the information loss of these rewrite algorithms, and the effect on the resulting network structure.

The implicit information captured by the structure of RDF graph is interesting, considering that structurally different RDF graphs can be semantically equal. In other words, this implicit structural information is introduced by design decisions of the dataset creator. The relation between modeling decisions and structural graph properties is only investigated in the context of RDF schemas [102]. To better understand these relations, a similar study is required in the context of complete datasets.

9.2.2 *Centralization as Enabler*

The decentralized approach of Linked Data allows for a web of independent interlinked datasets. As we showed in this thesis, this characteristic comes at a cost: data owners are responsible for publishing standards-compliant data (but often do not), the distributed datasets are hard to locate, and the use of these distributed datasets can only be monitored at a local level. This is largely similar to the regular World Wide Web, where publishers are responsible for publishing standards-compliant HTML, websites are hard to locate without a central search engine, and website usage is only monitored at per website. The World Wide Web shows how these problems can be solved without losing the key features of a distributed web: companies and initiatives such as Google, DuckDuckGo and Yahoo increase the findability of websites by performing web-scale indexing, and act as a gateway to the –still distributed– internet. Despite the overlap between the Linked Data architecture and that of the World Wide Web, these problems were left unresolved for Linked Data.

Both YASGUI and LOD Laundromat show that centralization can act as enabler for Linked Data research and use, without going against the decentralized philosophy of Linked Data.

In YASGUI, we did so by providing a centralized SPARQL query editor as service, that in turn accesses the large range of distributed SPARQL endpoints. This acts as a funnel for Linked Data users, where logging information that is otherwise tracked per-datasets, is now tracked in a single location. This approach enabled research on how users use Linked Data at a web scale; something that was previously impossible.

LOD Laundromat and its Meta-Dataset illustrated the added value of centralization as enabler as well. The indexes provided by the Meta-Dataset increase findability of Linked Data, and the data we re-publish solved idiosyncrasies in Linked Datasets. This enabled the *Frank* interface to Linked Data, and it enabled evaluations on a large number of datasets using LOD Lab–both of which are difficult to realize without a centralized means of access. Because the LOD Laundromat does not replace distributed Linked Datasets, but merely provides a centralized clean copy, it keeps the advantages of Linked Data intact while enabling research that was otherwise problematic to achieve.

9.2.3 *Academic Prototyping*

In our experience, Linked Data software development often takes the prototyping approach, where the tool or library is not robust, documented, and maintainable. Our exploration for SPARQL query editors related to YASGUI showed several characteristics that some of these tools and libraries suffer from:

CLOSED-SOURCE The software is not open source, making it impossible to maintain, improve or re-use by third-parties.

UNCLEAR LICENSING Those tools and libraries that *are* open-source, do not always come with an open-source license. This is particularly problematic for companies that want to re-use such software.

INCOMPATIBILITY Software might be built for a particular version of Java, or only a particular operating system. Re-use of these tools in other environments becomes difficult.

UNDOCUMENTED CODE This discourages other developers to re-use or maintain the project. This is particularly the case for libraries aimed at programmatic that do not publish API documentation

UNDOCUMENTED USE No documentation on how to use and start the software, e.g. with some example code snippets.

NO DEPENDENCY MANAGEMENT Several packages did not use dependency management, that automatically pulls in libraries it depends on. Compiling such a tool often requires finding and trying older versions of dependencies on a trial-and-error basis

These characteristics are problematic for the following reasons:

1. Publications can be difficult to reproduce when the corresponding software is difficult to understand and execute.
2. Linked Data evangelists are keen on seeing an uptake of Linked Data in industry. Where industry is used to a certain level of tooling, the academic research projects often fail to bridge this gap.
3. Where current research methods and standards are aimed towards incremental research, this is not the case for academic

software: re-use and modification of tools is cumbersome without documentation, incompatibilities, or troubles with dependency management.

4. Most importantly, we (as researchers) make our *own* lives difficult by not providing us with the tools and libraries we need.

The reason behind the lack of proper software in academia is unclear, but part of the problem probably lies in the academic incentive model where software is not rewarded in a similar fashion as scientific publications. This problem seems to have been recognized by others, as the recent developer workshops at Semantic Web conferences, and 'Tools & Systems' tracks at journals and conferences try to change exactly this incentive model by providing a platform for Linked Data developers.

9.3 ACCESSIBLE & SCALABLE LINKED DATA

This thesis presents a number of advancements for building Linked Data based services. The presented solutions are targeted at both consumers and publishers of Linked Data, and are a step towards a web of Linked Data that is more accessible and technically scalable. We believe this future Web of Linked Data can retain its positive aspects and co-exist with centralized services. The functionality of these centralized services can go beyond what we showed in this thesis. They can provide web-scale Linked Data indexing, querying, searching and hosting, and provide a means to move the current (mostly read-only) Web of Data towards that of a read-write Web of Data.

BIBLIOGRAPHY

[1] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing Linked Datasets: On the Design and Usage of VoID, the "Vocabulary Of Interlinked Datasets" . In *Linked Data on the Web Workshop*, 2009.

[2] R. Angles Rojas, P. Minh Duc, and P. A. Boncz. Benchmarking Linked Open Data Management Systems. *ERCIM News*, 96:24 – 25, January 2014.

[3] Kemafor Anyanwu, Angela Maduko, and Amit Sheth. Sem-Rank: ranking complex relationship search results on the semantic web. In *Proceedings of the 14th international conference on WWW*, pages 117–127. ACM, 2005.

[4] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The Semantic Web*, pages 722–735. Springer, 2007.

[5] Sören Auer, Jan Demter, Michael Martin, and Jens Lehmann. LODStats—an extensible framework for high-performance dataset analytics. In *Proceedings of EKAW*, volume 7603 of *Lecture Notes in Computer Science*. Springer, 2012.

[6] Sören Auer, Jan Demter, Michael Martin, and Jens Lehmann. LODStats–an extensible framework for high-performance dataset analytics. In *Knowledge Engineering and Knowledge Management*, pages 353–362. Springer, 2012.

[7] Sören Auer, Jens Lehmann, and Sebastian Hellmann. Linkedgeodata: Adding a spatial dimension to the web of data. In *The Semantic Web-ISWC*, pages 731–746. Springer, 2009.

[8] Ching Avery. Giraph: Large-scale graph processing infrastructure on Hadoop. *Proceedings of the Hadoop Summit. Santa Clara*, 2011.

[9] Thomas Baker, Pierre-Yves Vandenbussche, and Bernard Vatant. Requirements for vocabulary preservation and governance. *Library Hi Tech*, 31(4):657–668, 2013.

[10] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[11] Hamid R Bazoobandi, Steven de Rooij, Jacopo Urbani, et al. A Compact In-Memory Dictionary for RDF data. In *The Extended Semantic Web Conference – ESWC*. Springer, 2015.

[12] Wouter Beek and Laurens Rietveld. Frank: The LOD Cloud at your Fingertips. In *Developers Workshop , Extended Semantic Web Conference (ESWC)*, 2015.

[13] Wouter Beek, Laurens Rietveld, Hamid R Bazoobandi, Jan Wielemaker, and Stefan Schlobach. LOD Laundromat: A Uniform Way of Publishing Other People's Dirty Data. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2014.

[14] François Belleau, Marc-Alexandre Nolin, Nicole Tourigny, Philippe Rigault, and Jean Morissette. Bio2RDF: towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, 41(5):706–716, 2008.

[15] Bettina Berendt et al. Usage analysis and the web of data. In *ACM SIGIR Forum*, volume 45, pages 63–69. ACM, 2011.

[16] Bettina Berendt, Laura Hollink, Markus Luczak-Rösch, Knud Möller, and David Vallet. Proceedings of USEWOD2013 - 3rd International Workshop on Usage Analysis and the Web of Data. In *10th ESWC - Semantics and Big Data, Montpellier, France*, 2013.

[17] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The Semantic Web. *Scientific american*, 284(5):28–37, 2001.

[18] Chris Bizer, Anja Jentzsch, and Richard Cyganiak. State of the LOD Cloud. *Version 0.3 (September 2011)*, 1803, 2011.

[19] Christian Bizer and Richard Cyganiak. D2r server-publishing relational databases on the semantic web. In *Poster at the 5th International Semantic Web Conference*, pages 294–309, 2006.

[20] Christian Bizer and Andreas Schultz. The berlin sparql benchmark, 2009.

[21] Christian Bizer and Andy Seaborne. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. *World Wide Web Internet And Web Information Systems*, page 26, 2004.

[22] 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 International Conference on Management of Data* , pages 1247–1250. ACM, 2008.

[23] Peter Boncz, Irini Fundulaki, Andrey Gubichev, Josep Larriba-Pey, and Thomas Neumann. The Linked Data Benchmark Council Project. *Datenbank-Spektrum*, 13(2):121–129, 2013.

[24] Jethro Borsje and Hanno Embregts. Graphical query composition and natural language processing in an RDF visualization interface. *Erasmus School of Economics and Business Economics, Vol. Bachelor. Erasmus University, Rotterdam*, 2006.

[25] Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. Sesame: An architecture for storing and querying RDF data and schema information, 2001.

[26] Carlos Buil-Aranda et al. SPARQL Web-Querying Infrastructure: Ready for Action? In *The Semantic Web–ISWC*, pages 277–293. Springer, 2013.

[27] Stéphane Campinas. Live sparql auto-completion. In *ISWC Demo's*, volume 14, pages 477–480, 2014.

[28] Stephane Campinas et al. Introducing RDF graph summary with application to assisted sparql formulation. In *Database and Expert Systems Applications (DEXA)*, pages 261–266. IEEE, 2012.

[29] Stéphane Campinas, Thomas E. Perry, Dieggo Ceccarelli, Renaud Delbru, and Giovanni Tummarello. Introducing RDF Graph Summary with application to Assisted SPARQL Formulation. In *23rd International Workshop on Database and Expert Systems Applications*, 2012.

[30] Gong Cheng, Saisai Gong, and Yuzhong Qu. An empirical study of vocabulary relatedness and its application to recommender systems. In *The Semantic Web–ISWC 2011*, pages 98–113. Springer, 2011.

[31] Jan Demter, Sören Auer, Michael Martin, and Jens Lehmann. LODStats – An Extensible Framework for High-performance Dataset Analytics. In *Proceedings of the EKAW 2012*, Lecture Notes in Computer Science (LNCS) 7603. Springer, 2012.

[32] Li Ding, Lina Zhou, Tim Finin, and Anupam Joshi. How the Semantic Web is Being Used: An Analysis of FOAF Documents. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, HICSS '05, pages 113.3–, Washington, DC, USA, 2005. IEEE Computer Society.

[33] Ivan Ermilov, Michael Martin, Jens Lehmann, and Sören Auer. Linked Open Data Statistics: Collection and Exploitation. In *Knowledge Engineering and the Semantic Web*, pages 242–249. Springer, 2013.

[34] Jérôme Euzenat, Christian Meilicke, Heiner Stuckenschmidt, Pavel Shvaiko, and Cássia Trojahn. Ontology alignment evaluation initiative: Six years of experience. In *Journal on data semantics XV*, pages 158–192. Springer, 2011.

[35] 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.

[36] Thomas Franz, Antje Schultz, Sergej Sizov, and Steffen Staab. Triplerank: Ranking semantic web data by tensor decomposition. *The Semantic Web-ISWC*, pages 213–228, 2009.

[37] Mario Arias Gallego, Javier D Fernández, Miguel A Martínez-Prieto, and Pablo de la Fuente. An empirical study of real-world SPARQL queries. In *1st International Workshop on Usage Analysis and the Web of Data (USEWOD2011)*, 2011.

[38] Alan F Gates et al. Building a high-level dataflow system on top of Map-Reduce: the Pig experience. *Proceedings of the VLDB Endowment*, 2(2):1414–1425, 2009.

[39] Weiyi Ge, Jianfeng Chen, Wei Hu, and Yuzhong Qu. Object link structure in the semantic web. In *The Semantic Web: Research and Applications*, pages 257–271. Springer, 2010.

[40] John H Gennari, Mark A Musen, Ray W Fergerson, William E Grosso, Monica Crubézy, Henrik Eriksson, Natalya F Noy, and Samson W Tu. The evolution of protégé: an environment for knowledge-based systems development. *International Journal of Human-computer studies*, 58(1):89–123, 2003.

[41] Olaf Görlitz, Matthias Thimm, and Steffen Staab. SPLODGE: systematic generation of SPARQL benchmark queries for linked open data. In *The Semantic Web–ISWC*, pages 116–132. Springer, 2012.

[42] Thomas Gottron and Rene Pickhardt. A detailed analysis of the quality of stream-based schema construction on linked open data. In *Semantic Web and Web Science*, pages 89–102. Springer, 2013.

[43] Christophe Guéret, Shenghui Wang, Paul Groth, and Stefan Schlobach. Multi-scale Analysis of the Web Of Data: A Challenge to the Complex System's Community. *Advances in Complex Systems*, 14(04):587, 2011.

[44] 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):158–182, 2005.

[45] Chris Halaschek, Boanerges Aleman-meza, I. Budak Arpinar, and Amit P. Sheth. Discovering and Ranking Semantic Associations over a Large RDF Metabase. In *VLDB*, 2004.

[46] Steve Harris, Nick Lamb, and Nigel Shadbolt. 4store: The design and implementation of a clustered RDF store. In *5th International Workshop on Scalable Semantic Web Knowledge Base Systems*, pages 94–109, 2009.

[47] Steve Harris and Andy Seaborne. SPARQL 1.1 query language, March 2013.

[48] Andreas Harth. Billion Triples Challenge data set. Downloaded from http://km.aifb.kit.edu/projects/btc-2012/, 2012.

[49] 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.

[50] Bernhard Haslhofer and Antoine Isaac. data.europeana.eu: The Europeana linked open data pilot. In *International Conference on Dublin Core and Metadata Applications*, pages 94–104, 2011.

[51] Jonathan Hayes and Claudio Gutierrez. Bipartite Graphs as Intermediate Model for RDF. In *International Semantic Web Conference*, pages 47–61, 2004.

[52] Tom Heath and Christian Bizer. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, 1(1):1–136, 2011.

[53] Rinke Hoekstra. The MetaLex Document Server - Legal Documents as Versioned Linked Data. In Harith Alani and Jamie Tailor, editors, *Proceedings of the 10th International Semantic Web Conference (ISWC)*, page 16. Springer, 2011.

[54] Aidan Hogan, Andreas Harth, and Stefan Decker. ReConRank: A Scalable Ranking Method for Semantic Web Data with Context. In *2nd Workshop on Scalable Semantic Web Knowledge Base Systems*, 2006.

[55] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker, and Axel Polleres. Weaving the Pedantic Web. In *Linked Data on the Web Workshop*, 2010.

[56] Aidan Hogan, Jürgen Umbrich, Andreas Harth, Richard Cyganiak, Axel Polleres, and Stefan Decker. An empirical survey of Linked Data conformance. *J. Web Sem.*, 14:14–44, 2012.

[57] Frederik Hogenboom et al. RDF-GL: a SPARQL-Based graphical query language for RDF. In *Emergent Web Intelligence: Advanced Information Retrieval*, pages 87–116. Springer, 2010.

[58] Robert Isele, Jürgen Umbrich, Christian Bizer, and Andreas Harth. LDspider: An open-source crawling framework for the

Web of Linked Data. In *9th International Semantic Web Conference*. Citeseer, 2010.

[59] Krzysztof Janowicz, Pascal Hitzler, Benjamin Adams, Dave Kolas, Charles Vardeman, et al. Five stars of Linked Data vocabulary use. *Semantic Web*, 5(3):173–176, 2014.

[60] Tobias Käfer, Ahmed Abdelrahman, Jurgen Umbrich, Patrick O'Byrne, and Aidan Hogan. Observing Linked Data Dynamics. In *The Semantic Web: Semantics and Big Data*. Springer, 2013.

[61] Tobias Käfer, Jürgen Umbrich, Aidan Hogan, and Axel Polleres. Towards a dynamic Linked Data observatory. *LDOW at WWW*, 2012.

[62] Minoru Kanehisa et al. From genomics to chemical genomics: new developments in KEGG. *Nucleic acids research*, 34(suppl 1):D354–D357, 2006.

[63] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[64] Timothy Lebo, Satya Sahoo, Deborah McGuinness, Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. Prov-o: The prov ontology. *W3C Recommendation, 30th April*, 2013.

[65] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *The 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636, 2006.

[66] Michael Ley. Dblp: some lessons learned. *Proceedings of the VLDB Endowment*, 2(2):1493–1500, 2009.

[67] Antonis Loizou, Renzo Angles, and Paul Groth. On the formulation of performant sparql queries. *Web Semantics: Science, Services and Agents on the World Wide Web*, 31:1–26, 2015.

[68] Eetu Mäkelä. Aether – Generating and Viewing Extended VoID Statistical Descriptions of RDF Datasets. In *Proceedings of the ESWC 2014 demo track, Springer-Verlag*, 2014.

[69] Luca Matteis. Restpark: Minimal restful api for retrieving rdf triples. *arXiv preprint arXiv:1408.4793*, 2014.

[70] Albert Meroño-Peñuela, Christophe Guéret, Ashkan Ashkpour, and Stefan Schlobach. Cedar: The dutch historical censuses as linked open data. *Semantic Web Journal*, 2014.

[71] Ian Millard, Hugh Glaser, Manuel Salvadores, and Nigel Shadbolt. Consuming multiple linked data sources: Challenges and Experiences. In *First International Workshop on Consuming Linked Data* , November 2010. Event Dates: 2010-11-07.

[72] Knud Möller, Michael Hausenblas, Richard Cyganiak, and Siegfried Handschuh. Learning from linked open data usage: Patterns & metrics. In *WebSci10: Extending the Frontiers of Society On-Line*, pages 1–9, 2010.

[73] Knud Möller, Tom Heath, Siegfried Handschuh, and John Domingue. Recipes for semantic web dog food. The ESWC and ISWC metadata projects. In *The Semantic Web*, pages 802–815. Springer, 2007.

[74] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Federated SPARQL Queries Processing with Replicated Fragments. In *The Semantic Web-ISWC 2015-14th International Semantic Web Conference*, 2015.

[75] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. Dbpedia sparql benchmark–performance assessment with real queries on real data. *The Semantic Web–ISWC 2011*, pages 454–469, 2011.

[76] Chimezie Ogbuji. SPARQL 1.1 Graph Store HTTP Protocol. Recommendation, W3C, March 2013.

[77] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig Latin: A not-so-foreign Language for Data Processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* , pages 1099–1110. ACM, 2008.

[78] Tope Omitola, Landong Zuo, Christopher Gutteridge, Ian C Millard, Hugh Glaser, Nicholas Gibbins, and Nigel Shadbolt. Tracing the provenance of linked data using voiD. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, page 17. ACM, 2011.

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

[80] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics of SPARQL. In *Technical Report TR/DCC-2006-17, Department of Computer Science, Universidad de Chile*, volume 4, 2006.

[81] Francois Picalausa and Stijn Vansummeren. What are real SPARQL queries like? In *International Workshop on Semantic Web Information Management*, page 7. ACM, 2011.

[82] Laurens Rietveld, Wouter Beek, and Stefan Schlobach. LOD Lab: Experiments at LOD Scale. In *Proceedings of the International Semantic Web Conference (ISWC)*. Springer, 2015.

[83] Laurens Rietveld, Wouter Beek, Stefan Schlobach, and Rinke Hoekstra. Meta-Data for a lot of LOD. *Semantic Web Journal*, To be published. `http://semantic-web-journal.net/content/meta-data-lot-lod`.

[84] Laurens Rietveld and Rinke Hoekstra. YASGUI: Not just another SPARQL client. In *The Semantic Web: ESWC 2013 Satellite Events*, pages 78–86. Springer, 2013.

[85] Laurens Rietveld and Rinke Hoekstra. Man vs. Machine: Differences in SPARQL Queries. In *4th USEWOD Workshop on Usage Analysis and the Web of Data, ESWC*, 2014.

[86] Laurens Rietveld and Rinke Hoekstra. YASGUI: Feeling the Pulse of Linked Data. In *Knowledge Engineering and Knowledge Management*, pages 441–452. Springer, 2014.

[87] Laurens Rietveld and Rinke Hoekstra. The YASGUI Family of SPARQL Clients. *Semantic Web Journal*, 2015.

[88] Laurens Rietveld, Rinke Hoekstra, Stefan Schlobach, and Christophe Guéret. Structural Properties as Proxy for Semantic Relevance in RDF Graph Sampling. In *The Semantic Web–ISWC 2014*, pages 81–96. Springer, 2014.

[89] Laurens Rietveld, Ruben Verborgh, Wouter Beek, Miel Vander Sande, and Stefan Schlobach. Linked Data-as-a-Service: The Semantic Web Redeployed. In *Proceedings of the Extended Semantic Web Conference (ESWC)*. Springer, 2015.

[90] Saleem, Muhammad and Khan, Yasar and Hasnain, Ali and Ermilov, Ivan and Ngomo, Axel-Cyrille Ngonga. A fine-grained evaluation of sparql endpoint federation systems. *Semantic Web Journal*, 2014.

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

[92] M Schmidt, T Hornung, G Lausen, and C Pinkel. SP2Bench: A SPARQL performance benchmark, ICDE. *Shanghai, China*, 2009.

[93] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *The Semantic Web–ISWC*, pages 585–600. Springer, 2011.

[94] Michael Schmidt, Thomas Hornung, Michael Meier, Christoph Pinkel, and Georg Lausen. SP2Bench: A SPARQL performance benchmark. In *Semantic Web Information Management*, pages 371–393. Springer, 2010.

[95] Martin G Skjæveland. Sgvizler: A javascript wrapper for easy visualization of sparql result sets. In *Extended Semantic Web Conference*, 2012.

[96] Paul R Smart et al. A visual approach to semantic query design using a web-based graphical query designer. In *Knowledge Engineering: Practice and Patterns*, pages 275–291. Springer, 2008.

[97] Steve Speicher, John Arwe, and Ashok Malhotra. Linked Data Platform 1.0. Candidate recommendation, W3C, June 2014.

[98] Steffen Stadtmüller, Andreas Harth, and Marko Grobelnik. Accessing Information about Linked Data Vocabularies with vocab.cc . In *Semantic Web and Web Science*, pages 391–396. Springer, 2013.

[99] Seema Sundara et al. Visualizing large-scale RDF data using Subsets, Summaries, and Sampling in Oracle. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 1048–1059. IEEE, 2010.

[100] Pedro Szekely et al. Connecting the Smithsonian American Art Museum to the Linked Data Cloud. In *The Semantic Web: Semantics and Big Data*, pages 593–607. Springer, 2013.

[101] Guangming Tan, Dengbiao Tu, and Ninghui Sun. A parallel algorithm for computing betweenness centrality. In *Proc. of ICPP*, pages 340–347, 2009.

[102] Yannis Theoharis, Yannis Tzitzikas, Dimitris Kotzinos, and Vassilis Christophides. On Graph Features of Semantic Web Schemas. *Knowledge and Data Engineering, IEEE Transactions on*, 20(5):692–702, 2008.

[103] Alberto Tonon and Michele Catasta. TRank: Ranking Entity Types Using the Web of Data. In *The Semantic Web - ISWC*, pages 640 – 656, 2013.

[104] Giovanni Tummarello, Richard Cyganiak, Michele Catasta, Szymon Danielczyk, Renaud Delbru, and Stefan Decker. Sig.ma: Live Views on the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web* , 8(4):355–364, 2010.

[105] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle.

Low-Cost Queryable Linked Data through Triple Pattern Fragments. In *Proceedings of the 13th International Semantic Web Conference: Posters and Demos*, 2014.

[106] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle. Querying datasets on the web with high availability. In *The Semantic Web–ISWC 2014*, pages 180–196. Springer, 2014.

[107] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-Scale Querying through Linked Data Fragments. In *Proceedings of the 7th Workshop on Linked Data on the Web*, April 2014.

[108] Shenghui Wang and Paul Groth. Measuring the dynamic bidirectional influence between content and social networks. In *The 9th International Semantic Web Conference ISWC*, pages 814–829. Springer, 2010.

[109] Ingmar Weber and Alejandro Jaimes. Who Uses Web Search for What? And How? In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 15–24, New York, NY, USA, 2011. ACM.

[110] Jan Wielemaker. SWI-Prolog Semantic Web Library 3.0.

[111] Martins Zviedris and Guntis Barzdins. ViziQuer: a tool to explore and query SPARQL endpoints. In *The Semanic Web: Research and Applications*, pages 441–445. Springer, 2011.

SUMMARY

The World Wide Web is a huge collection of interlinked information. For us, humans, this information is readily accessible in formats we like, such as news articles rendered as text, audio broadcasts, and videos via services such as YouTube. Where human beings can understand these formats, software agents often cannot.

A complementary approach to the World Wide Web is that of Linked Data, where information is represented in a machine readable format. Linked Data uses the same underpinnings as the World Wide Web: both use the Hypertext Transfer Protocol (HTTP) to access and retrieve information. In Linked Data, URLs and string denote 'things', called resources. These resources can be anything, such as geographical locations, documents, abstract concepts like 'Democracy', or numbers and strings.

The Linked Data architecture enables consumption without a-priori knowledge of the schema and content, and enables publishing without knowing how a dataset is going to be used. This unknown (re)use is an intrinsic positive quality of Linked Data, but it presents problems as well for both consumers and publishers of Linked Data. Below, we discuss the five problems we identified together with the corresponding contributions of this thesis.

IMPRACTICAL LINKED DATA RE-USE Publishing Linked Data as static files seems straightforward, but even this method can be difficult in practice: many Linked Datasets still do not adhere to standards and best practices.

We developed a centralized service called the LOD Laundromat, that re-publishes a clean version of as many Linked Open Datasets as possible, providing a wealth of uniform clean data that can be used with little effort.

QUERYABLE LINKED DATA IS EXPENSIVE TO HOST The de-facto standard for hosting queryable Linked Data is the SPARQL endpoint. Its flexibility and rich querying language offers advantages for Linked Data providers, but comes at a cost: these SPARQL triplestores are expensive to host. We presented two solutions to this problem.

First, we presented a sampling method called SampLD, that reduces the dataset size, and thus reduces the hardware costs for hosting a SPARQL endpoint.

A second, orthogonal, solution is to reduce query language complexity instead of reducing the completeness of query results. We did so by combining the LOD Laundromat with a Triple Pattern Fragment API that only supports simple triple pattern queries. This re-

sulted in a low cost Linked Data API that consumes a fraction in memory and processing power compared to SPARQL triple-stores.

FORMULATING SPARQL QUERIES IS DIFFICULT The complexity and expressiveness of SPARQL makes it an unforgiving and difficult query language. But the state-of-the-art in query editors does not provide the tooling and features that web developers are accustomed to. We improved this state-of-the-art by developing YASGUI, a SPARQL query editor accessible from the browser, that has many features known to web developers such as syntax highlighting, syntax checking, and auto-completion functionality.

PROBLEMATIC ACCESS TO LINKED (META-)DATA The distributed nature of Linked Data and the absence structural descriptions of datasets makes locating and accessing Linked Datasets difficult. A centralized solution such as the LOD Laundromat does not directly solve this issue, as finding datasets according to structural criteria still requires manual processing and exploration.

Therefore we extended the LOD Laundromat with a structural Meta-Dataset that includes an IRI and namespace to dataset index, dataset characteristics, and provenance information about the performed processing steps.

VARIETY OF LINKED DATA RESEARCH Linked Data research is suffering from the unavailability of resources and tools to study Linked Data and its use at large.

The first approach we presented focused on Linked Data *use* –a research area that is strongly restricted by the limited number of available query logs. We enable tracking Linked Data usage from the *client* side using the YASGUI query editor as a measuring device, and show how the queries collected by YASGUI enable us to investigate usage patterns that are difficult to measure otherwise.

The second approach focused on increasing the variety from a *data-centric* perspective. We showed that existing research only evaluates on a handful of datasets, and we presented an alternative approach for running experiments on a much broader scale using the LOD Laundromat, the corresponding Meta-Dataset, and the exposed LOD Laundromat Triple Pattern Fragments API. By re-evaluating three recent publications, we this new evaluation paradigm brings up interesting research questions as to how algorithmic performance relates to (structural) properties of the data.

This thesis presents a number of advancements for building Linked Data based services. The presented solutions are targeted at both consumers and publishers of Linked Data, and are a step towards a web of Linked Data that is more accessible and technically scalable.

SAMENVATTING

Het Wereldwijde Web (WWW) is een grote collectie van informatie, waarbij onderling verbonden is. Voor eindgebruikers is deze informatie toegankelijk in verschillende bestandstypen. Zo zijn bijvoorbeeld nieuwsartikelen beschikbaar als text, en video's beschikbaar via diensten als YouTube. Hoewel deze verschillende bestandsformaten goed bruikbaar zijn voor mensen, is dit voor software agents daarentegen moeilijker te verwerken.

Naast het WWW bestaat Linked Data. Met Linked Data is het mogelijk om informatie te representeren zodat software agents dit kunnen 'begrijpen'. WWW en Linked Data gebruiken dezelfde fundering om toegang te krijgen tot de informatie, namelijk HTTP. Het verschil is dat Linked Data URLs en kleine stukken tekst gebruikt om 'dingen' te duiden, de zogenaamde 'resources'. Dit kan van alles zijn, variërend van geografische locaties en documenten tot abstracte concepten zoals democratie. De Linked Data architectuur maakt datagebruik mogelijk zonder a priori kennis van de inhoud te hebben. Bovendien is het mogelijk om data te publiceren zonder dat vooraf het publicatiedoel bekend hoeft te zijn. De onbekendheid over (her)gebruik van Linked Data heeft positieve en negatieve kanten: het is goed voor de kwaliteit van de data, maar stelt zowel gebruikers als publicisten voor problemen. Onderstaand presenteren we vijf problemen in deze context, en hoe dit proefschrift bijdraagt aan oplossingen voor deze problemen.

ONPRAKTISCH HERGEBRUIK VAN LINKED DATA

Het publiceren van Linked Data als statische bestanden lijkt eenvoudig, maar zelfs deze methode blijkt in de praktijk lastig: veel Linked Datasets voldoen niet aan de geldende standaarden. Wij hebben een gecentraliseerde dienst ontwikkeld die de LOD Laundromat heet, welke 'schone' versies van zoveel mogelijk Linked Datasets herpubliceert. Dit voorziet in een overvloed van uniforme schone datasets die met weinig moeite gebruikt kunnen worden door software agents.

BEVRAAGBARE LINKED DATA IS DUUR OM TE HOSTEN

De de facto standaard voor het hosten van bevraagbare Linked Data is de 'SPARQL endpoint'. SPARQL is een flexibele en rijke query taal die veel voordelen biedt, maar er zijn ook kosten aan verbonden: deze endpoints zijn duur om te hosten. Wij presenteren twee oplossingen voor dit probleem.

Als eerste presenteren we een 'sampling' methode genaamd SampLD welke de grootte vermindert –en daarmee ook de hardware kosten– voor het hosten van een SPARQL endpoint.

Een tweede orthogonale aanpak is om de complexiteit van de query-taal te verminderen, in plaats van het verminderen van de volledigheid van de query antwoorden. Deze aanpak bestaat uit het combineren van de LOD Laundromat met een 'Triple Pattern Fragments API' die alleen simpele triple patronen ondersteunt. Dit leidt tot een goedkope Linked Data API die een fractie van het geheugen en processor-kracht gebruikt vergeleken SPARQL endpoints.

FORMULEREN VAN SPARQL QUERIES IS MOEILIJK

De complexiteit en expressiviteit van SPARQL, maakt het een moeilijke query-taal. Veel van de meest recente query-bewerkers bieden niet de mogelijkheden waaraan web ontwikkelaars gewend zijn. Daarom hebben wij YASGUI ontwikkeld, een SPARQL query bewerker die toegankelijk is vanuit de browser en veel functies bevat die web ontwikkelaars bekend voorkomen, zoals het markeren en controleren van de syntax, en het aanbieden van suggesties tijden het schrijven van de query.

PROBLEMATISCHE TOEGANG NAAR LINKED (META-)DATA

Het gedistribueerde karakter van Linked Data en de afwezigheid van structurele dataset-omschrijvingen maakt het moeilijk om Linked Data te vinden en te benaderen. Een gecentraliseerde oplossing zoals de LOD Laundromat lost dit probleem niet direct op, omdat het vinden van datasets aan de hand van structurele eigenschappen nog steeds handmatige stappen vereist. Onze aanpak voor dit probleem is om de LOD Laundromat uit te breiden met een structurele Meta-Dataset, welke structurele eigenschappen van de verzamelde datasets bevat, een index om deze datasets te vinden, en herkomst informatie over hoe deze datasets verzameld en geanalyseerd zijn.

VARIËTEIT VAN LINKED DATA ONDERZOEK

Linked Data onderzoek heeft te lijden onder onbeschikbaarheid van documenten en methodes om Linked Data als groot geheel te analyseren. Voor dit probleem biedt het werk in dit proefschrift twee oplossingen.

De eerste aanpak die we presenteren richt zich op Linked Data *gebruik*: dit onderzoeksgebied is sterk beperkt door weinig beschikbare query logs. Wij maken het mogelijk om het gebruik van Linked Data te volgen vanaf de *gebruikers*-kant, door middel van de YASGUI query-bewerker. Wij laten zien hoe de verzamelde queries van YAS-

GUI het mogelijk maken om op grote schaal gebruikers patronen te analyseren.

De tweede aanpak richt zich op het vergroten van de variëteit vanuit een *data*-perspectief. We laten zien dat bestaand Linked Data onderzoek vaak wordt uitgevoerd op enkele datasets, en we presenteren een alternatieve aanpak voor het draaien van experimenten op veel grotere schaal, middels de LOD Laundromat, de bijbehorende Meta-Dataset, en de gepubliceerde LOD Laundromat Triple Pattern Fragments API. Door gedeeltes van drie recente publicaties te herevalueren, laten we zien dat dit nieuwe evaluatie paradigma interessante onderzoeksvragen oproept zoals hoe de prestaties van algoritmes in verband staan met (structurele) eigenschappen van de data.

Dit proefschrift biedt een aantal bijdragen aan voor het bouwen van Linked Data gebaseerde diensten. De oplossingen die we aandragen zijn zowel op de gebruiker als publicist gericht, en zijn een stap in de richting van een meer toegankelijk en technisch schaalbaarder Linked Data web.

SIKS DISSERTATIEREEKS

====
2010
====

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children

2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to
Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions

2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL)
Automatisch contracteren

2010-28 Arne Koopman (UU)
Characteristic Relational Patterns

2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE)
Correctness of services and their composition

2010-38 Dirk Fahland (TUE)
From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples

2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions

2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access

====
2011
====

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU)
Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT)
Cloud Content Contention

2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA)
Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU)
The Mind's Eye on Personal Profiles

2011-20 Qing Gu (VU)
Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD)
Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UVA)
System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UVA)
Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination with Virtual Humans

On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU))
Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU)
Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein(UVA)
Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification

2011-30 Egon van den Broek (UT)
Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach

2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU)
Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz(VU)
Exploring Computational Models for Intelligent Support of Persons with Depression

2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

====
2012
====

2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair(VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU)
Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications

2012-05 Marijn Plomp (UU)
Maturing Interorganisational Information Systems

2012-06 Wolfgang Reinhardt (OU)
Awareness Support for Knowledge Workers in Research Networks

2012-07 Rianne van Lambalgen (VU)
When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

2012-08 Gerben de Vries (UVA)
Kernel Methods for Vessel Trajectories

2012-09 Ricardo Neisse (UT)
Trust and Privacy Management Support for Context-Aware Service Platforms

2012-10 David Smits (TUE)
Towards a Generic Distributed Adaptive Hypermedia Environment

2012-11 J.C.B. Rantham Prabhakara (TUE)
Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

2012-12 Kees van der Sluijs (TUE)
Model Driven Design and Data Integration in Semantic Web Information Systems

2012-13 Suleman Shahid (UvT)
Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
2012-14 Evgeny Knutov(TUE)
Generic Adaptation Framework for Unifying Adaptive Web-based Systems
2012-15 Natalie van der Wal (VU)
Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

2012-16 Fiemke Both (VU)
Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT)
Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU)
Improving Solution Architecting Practices
2012-19 Helen Schonenberg (TUE)
What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN)
Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
2012-21 Roberto Cornacchia (TUD)
Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT)
Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT)
Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24 Laurens van der Werff (UT)
Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT)
Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UVA)
Making Sense of Legal Text

2012-27 Hayrettin Gurkok (UT)
Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT)
Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT)
Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD)
Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN) A Learning by Construction Approach for Higher Order Cognitive Skills
Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD)
Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OUN)
Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN)
Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU)
Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

2012-36 Denis Ssebugwawo (RUN)
Analysis and Evaluation of Collaborative Modeling Processes

2012-37 Agnes Nakakawa (RUN)
A Collaboration Process for Enterprise Architecture Creation

2012-38 Selmar Smit (VU)
Parameter Tuning and Scientific Testing in Evolutionary Algorithms

2012-39 Hassan Fatemi (UT)
Risk-aware design of value and coordination networks

2012-40 Agus Gunawan (UvT)
Information Access for SMEs in Indonesia

2012-41 Sebastian Kelle (OU)
Game Design Patterns for Learning

2012-42 Dominique Verpoorten (OU)
Reflection Amplifiers in self-regulated Learning

2012-44 Anna Tordai (VU)
On Combining Alignment Techniques

2012-45 Benedikt Kratz (UvT)
A Model and Language for Business-aware Transactions

2012-46 Simon Carter (UVA)
Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

2012-47 Manos Tsagkias (UVA)
Mining Social Media: Tracking Content and Predicting Behavior

2012-48 Jorn Bakker (TUE)
Handling Abrupt Changes in Evolving Time-series Data
2012-49 Michael Kaisers (UM)
Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

2012-50 Steven van Kervel (TUD)
Ontologogy driven Enterprise Information Systems Engineering

2012-51 Jeroen de Jong (TUD)
Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

====
2013
====

2013-01 Viorel Milea (EUR)
News Analytics for Financial Decision Support

2013-02 Erietta Liarou (CWI)
MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

2013-03 Szymon Klarman (VU)
Reasoning with Contexts in Description Logics

2013-04 Chetan Yadati(TUD)
Coordinating autonomous planning and scheduling

2013-05 Dulce Pumareja (UT)
Groupware Requirements Evolutions Patterns

2013-06 Romulo Goncalves(CWI)
The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

2013-07 Giel van Lankveld (UvT)
Quantifying Individual Player Differences

2013-08 Robbert-Jan Merk(VU)
Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

2013-09 Fabio Gori (RUN)
Metagenomic Data Analysis: Computational Methods and Applications

2013-10 Jeewanie Jayasinghe Arachchige(UvT)
A Unified Modeling Framework for Service Design.

2013-11 Evangelos Pournaras(TUD)
Multi-level Reconfigurable Self-organization in Overlay Services

2013-12 Marian Razavian(VU)
Knowledge-driven Migration to Services

2013-13 Mohammad Safiri(UT)
Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly

2013-14 Jafar Tanha (UVA)
Ensemble Approaches to Semi-Supervised Learning Learning

2013-15 Daniel Hennes (UM)
Multiagent Learning - Dynamic Games and Applications

2013-16 Eric Kok (UU)
Exploring the practical benefits of argumentation in multi-agent deliberation

2013-17 Koen Kok (VU)
The PowerMatcher: Smart Coordination for the Smart Electricity Grid

2013-18 Jeroen Janssens (UvT)
Outlier Selection and One-Class Classification

2013-19 Renze Steenhuizen (TUD)
Coordinated Multi-Agent Planning and Scheduling

2013-20 Katja Hofmann (UvA)
Fast and Reliable Online Learning to Rank for Information Retrieval

2013-21 Sander Wubben (UvT)
Text-to-text generation by monolingual machine translation

2013-22 Tom Claassen (RUN)
Causal Discovery and Logic

2013-23 Patricio de Alencar Silva(UvT)
Value Activity Monitoring

2013-24 Haitham Bou Ammar (UM)
Automated Transfer in Reinforcement Learning

2013-25 Agnieszka Anna Latoszek-Berendsen (UM)
Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

2013-26 Alireza Zarghami (UT)
Architectural Support for Dynamic Homecare Service Provisioning

2013-27 Mohammad Huq (UT)
Inference-based Framework Managing Data Provenance

2013-28 Frans van der Sluis (UT)
When Complexity becomes Interesting: An Inquiry into the Information eXperience

2013-29 Iwan de Kok (UT)
Listening Heads

2013-30 Joyce Nakatumba (TUE)
Resource-Aware Business Process Management: Analysis and Support

2013-31 Dinh Khoa Nguyen (UvT)
Blueprint Model and Language for Engineering Cloud Applications

2013-32 Kamakshi Rajagopal (OUN) Networking For Learning; The role of Networking in a Lifelong Learner's
Professional Development

2013-33 Qi Gao (TUD)
User Modeling and Personalization in the Microblogging Sphere

2013-34 Kien Tjin-Kam-Jet (UT)
Distributed Deep Web Search

2013-35 Abdallah El Ali (UvA)
Minimal Mobile Human Computer Interaction

2013-36 Than Lam Hoang (TUe)
Pattern Mining in Data Streams

2013-37 Dirk Borner (OUN)
Ambient Learning Displays

2013-38 Eelco den Heijer (VU)
Autonomous Evolutionary Art

2013-39 Joop de Jong (TUD)
A Method for Enterprise Ontology based Design of Enterprise Information Systems

2013-40 Pim Nijssen (UM)
Monte-Carlo Tree Search for Multi-Player Games

2013-41 Jochem Liem (UVA)
Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning

2013-42 Leon Planken (TUD)
Algorithms for Simple Temporal Reasoning

2013-43 Marc Bron (UVA)
Exploration and Contextualization through Interaction and Concepts

====
2014
====

2014-01 Nicola Barile (UU)
Studies in Learning Monotone Models from Data

2014-02 Fiona Tuliyano (RUN)
Combining System Dynamics with a Domain Modeling Method

2014-03 Sergio Raul Duarte Torres (UT)
Information Retrieval for Children: Search Behavior and Solutions

2014-04 Hanna Jochmann-Mannak (UT)
Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation

2014-05 Jurriaan van Reijsen (UU)
Knowledge Perspectives on Advancing Dynamic Capability

2014-06 Damian Tamburri (VU)
Supporting Networked Software Development

2014-07 Arya Adriansyah (TUE)
Aligning Observed and Modeled Behavior

2014-08 Samur Araujo (TUD)
Data Integration over Distributed and Heterogeneous Data Endpoints

2014-09 Philip Jackson (UvT)
Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language

2014-10 Ivan Salvador Razo Zapata (VU)
Service Value Networks

2014-11 Janneke van der Zwaan (TUD)
An Empathic Virtual Buddy for Social Support

2014-12 Willem van Willigen (VU)
Look Ma, No Hands: Aspects of Autonomous Vehicle Control

2014-13 Arlette van Wissen (VU)
Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains

2014-14 Yangyang Shi (TUD)
Language Models With Meta-information

2014-15 Natalya Mogles (VU)
Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare

2014-16 Krystyna Milian (VU)
Supporting trial recruitment and design by automatically interpreting eligibility criteria

2014-17 Kathrin Dentler (VU)
Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability

2014-18 Mattijs Ghijsen (VU)
Methods and Models for the Design and Study of Dynamic Agent Organizations

2014-19 Vinicius Ramos (TUE)
Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support

2014-20 Mena Habib (UT)
Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

2014-21 Kassidy Clark (TUD)
Negotiation and Monitoring in Open Environments

2014-22 Marieke Peeters (UU)
Personalized Educational Games - Developing agent-supported scenario-based training

2014-23 Eleftherios Sidirourgos (UvA/CWI)
Space Efficient Indexes for the Big Data Era

2014-24 Davide Ceolin (VU)
Trusting Semi-structured Web Data

2014-25 Martijn Lappenschaar (RUN)
New network models for the analysis of disease interaction

2014-26 Tim Baarslag (TUD)
What to Bid and When to Stop

2014-27 Rui Jorge Almeida (EUR)
Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty

2014-28 Anna Chmielowiec (VU)
Decentralized k-Clique Matching

2014-29 Jaap Kabbedijk (UU)
Variability in Multi-Tenant Enterprise Software

2014-30 Peter de Cock (UvT)
Anticipating Criminal Behaviour

2014-31 Leo van Moergestel (UU)
Agent Technology in Agile Multiparallel Manufacturing and Product Support

2014-32 Naser Ayat (UvA)
On Entity Resolution in Probabilistic Data

2014-33 Tesfa Tegegne (RUN)
Service Discovery in eHealth

2014-34 Christina Manteli(VU) The Effect of Governance in Global Software Development: Analyzing
Transactive Memory Systems.

2014-35 Joost van Ooijen (UU)
Cognitive Agents in Virtual Worlds: A Middleware Design Approach

2014-36 Joos Buijs (TUE)
Flexible Evolutionary Algorithms for Mining Structured Process Models

2014-37 Maral Dadvar (UT)
Experts and Machines United Against Cyberbullying

2014-38 Danny Plass-Oude Bos (UT)
Making brain-computer interfaces better: improving usability through post-processing.

2014-39 Jasmina Maric (UvT)
Web Communities, Immigration, and Social Capital

2014-40 Walter Omona (RUN)
A Framework for Knowledge Management Using ICT in Higher Education

2014-41 Frederic Hogenboom (EUR)
Automated Detection of Financial Events in News Text

2014-42 Carsten Eijckhof (CWI/TUD)
Contextual Multidimensional Relevance Models

2014-43 Kevin Vlaanderen (UU)
Supporting Process Improvement using Method Increments

2014-44 Paulien Meesters (UvT)
Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.

2014-45 Birgit Schmitz (OUN)
Mobile Games for Learning: A Pattern-Based Approach

2014-46 Ke Tao (TUD)
Social Web Data Analytics: Relevance, Redundancy, Diversity

2014-47 Shangsong Liang (UVA)
Fusion and Diversification in Information Retrieval

====
2015
====

2015-01 Niels Netten (UvA)
Machine Learning for Relevance of Information in Crisis Response

2015-02 Faiza Bukhsh (UvT)
Smart auditing: Innovative Compliance Checking in Customs Controls

2015-03 Twan van Laarhoven (RUN)
Machine learning for network data

2015-04 Howard Spoelstra (OUN)
Collaborations in Open Learning Environments

2015-05 Christoph Bosch(UT)
Cryptographically Enforced Search Pattern Hiding

2015-06 Farideh Heidari (TUD)
Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes

2015-07 Maria-Hendrike Peetz(UvA)
Time-Aware Online Reputation Analysis

2015-08 Jie Jiang (TUD)
Organizational Compliance: An agent-based model for designing and evaluating organizational interactions

2015-09 Randy Klaassen(UT)
HCI Perspectives on Behavior Change Support Systems

2015-10 Henry Hermans (OUN)
OpenU: design of an integrated system to support lifelong learning

2015-11 Yongming Luo(TUE)
Designing algorithms for big graph datasets: A study of computing bisimulation and joins

2015-12 Julie M. Birkholz (VU)
Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks

2015-13 Giuseppe Procaccianti(VU)
Energy-Efficient Software

2015-14 Bart van Straalen (UT)
A cognitive approach to modeling bad news conversations

2015-15 Klaas Andries de Graaf (VU)
Ontology-based Software Architecture Documentation

2015-16 Changyun Wei (UT)
Cognitive Coordination for Cooperative Multi-Robot Teamwork

2015-17 André van Cleeff (UT)
Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs

2015-18 Holger Pirk (CWI)
Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories

2015-19 Bernardo Tabuenca (OUN)
Ubiquitous Technology for Lifelong Learners

2015-20 Loís Vanhée(UU)
Using Culture and Values to Support Flexible Coordination

2015-21 Sibren Fetter (OUN)
Using Peer-Support to Expand and Stabilize Online Learning

015-22 Zhemin Zhu(UT)
Co-occurrence Rate Networks

2015-23 Luit Gazendam (VU)
Cataloguer Support in Cultural Heritage

2015-24 Richard Berendsen (UVA)
Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation

2015-25 Steven Woudenberg (UU)
Bayesian Tools for Early Disease Detection

2015-26 Alexander Hogenboom (EUR)
Sentiment Analysis of Text Guided by Semantics and Structure

2015-27 Sándor Héman (CWI)
Updating compressed column-stores

2015-28 Janet Bagorogoza (TiU)
Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO

2015-29 Hendrik Baier (UM)
Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains

2015-30 Kiavash Bahreini (OUN)
Real-time Multimodal Emotion Recognition in E-Learning

colophon

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of February 12, 2016 (`classicthesis`).