LOD Laundromat: A Uniform Way of Publishing Other People's Dirty Data

Wouter Beek, Laurens Rietveld, Hamid R. Bazoobandi, Jan Wielemaker, and Stefan Schlobach

Dept. of Computer Science, VU University Amsterdam, NL
{w.g.j.beek,laurens.rietveld,h.bazoubandi,j.wielemaker,k.s.schlobach}@vu.nl

Abstract. It is widely accepted that *proper* data publishing is difficult. The majority of Linked Open Data (LOD) does not meet even a core set of data publishing guidelines. Moreover, datasets that are *clean* at creation, can get *stains* over time. As a result, the LOD cloud now contains a high level of *dirty* data that is difficult for humans to clean and for machines to process.
Existing solutions for cleaning data (standards, guidelines, tools) are targetted towards human data creators, who can (and do) choose not to use them. This paper presents the LOD Laundromat which removes stains from data without any human intervention. This fully automated approach is able to make very large amounts of LOD more easily available for further processing *right now*.
LOD Laundromat is not a new dataset, but rather a uniform point of entry to a collection of cleaned siblings of existing datasets. It provides researchers and application developers a wealth of data that is guaranteed to conform to a specified set of best practices, thereby greatly improving the chance of data actually being (re)used.

Keywords: Data Publishing, Data Cleaning, Data Reuse, Standards Conformance, Dataset Descriptions

1 Introduction

Uptake of Linked Open Data (LOD) has seen a tremendous growth over the last decade or so. 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 [3,4]. 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., via `owl:sameAs`). There is thus an incentive that goes beyond that of the original dataset creators to clean stains in LOD.

Existing solutions for cleaning LOD (standards, guidelines, tools) are targeted towards human data creators, who can (and do) choose not to use them. Therefore, despite these efforts, much of LOD is still difficult to use today, mostly because of mistakes for which solutions exist. We believe that this poses an unnecessary impediment to the (re)use of LOD for academic and commercial purposes.

This paper presents the LOD Laundromat, which takes *immediate* action by targeting the *data* directly, not its maintainers. By cleaning stains in LOD without any human intervention, LOD Laundromat is able to make very large amounts of LOD more easily available for further processing *right now*. The collection of cleaned datasets that LOD Laundromat produces are standards- and guidelines-compliant siblings of existing, idiosyncratic datasets.

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 tens of thousands of data files (and counting), and of billions of triples (and counting).
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 discovery to serialization in a standards-compliant canonical format that enables easy reuse.

Besides making LOD standards-compliant, 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 postprocessors to parse the data, e.g. through a syntax that is Regular Expression-friendly.

The LOD Laundromat is available at `http://lodlaundry.github.io`. The collection of datasets that is comprises is continuously being extended. Anyone can add seed points to the LOD Laundry Basket[1] by using a Web form. The fully automated LOD Washing Machine[2] takes seed points from the LOD Laundry Basket and cleans them. Cleaned datasets are disseminated in the LOD Wardrobe[3]. Various visualizations about the cleaned data are available as well.[4]

Section overview This paper is organized as follows: section 2 gives an overview of related work. Section 3 specifies the requirements we pose for clean and useful data, and briefly explores alternative approaches, thereby describing the context in which the LOD Laundromat operates. Section 4 details the major operationalization decisions that allow the data cleaning process to be fully automated. Section 5 elaborates on the information we publish in what way. Since the LOD

[1] `http://lodlaundry.github.io/laundryBasket`
[2] `https://github.com/LODLaundry/LOD-Washing-Machine`
[3] `http://lodlaundry.github.io/wardrobe`
[4] `http://lodlaundry.github.io/visualizations`

Laundromat republishes large amounts of existing data in a clean way, section 6 gives an overview of some of the stains that LOD Laundromat has been able to remove from dirty datasets. Section 7 concludes and mentions future work.

2 Related Work

Below we discuss several aspects of the state of the art. First, we discuss the different standards and best practices with respect to Linked Data publishing. Second, we discuss existing Linked Data collections and crawlers. Finally, we show the available Linked Data catalogs, their advantages and disadvantages.

2.1 Standards

The VoID standard is a vocabulary to formally describe datasets. It supports general metadata (e.g. the homepage of a dataset), access metadata (e.g. which protocols are available), possible links with other datasets, as well as structural metadata. This structural metadata includes items such as exemplary resources and statistics such as the number of triples, properties and classes.

An extension to the structural metadata of the VoID description are the dataset metrics presented by Bio2RDF [2]. These dataset metrics are more detailed than the VoID description (e.g. the number of unique objects linked from each predicate).

These standards are useful from both the data publisher and the data consumer perspective. However, uptake of VoID is lacking[5]. Additionally, from a data consumer perspective, the issue of findability is not resolved.

A number of observations and statistics related to Linked Data publishing best practices are presented in [4] and by the W3C Linked Data best practices working group[6]. The former have analyzed over a billion triples from 4 million crawled RDF/XML documents. This analysis shows that on average 15.7% of the RDF nodes (excluding literals) are blank nodes; even though, in many situations, the use of blank nodes is discouraged. Furthermore, their analysis shows that most Linked Data is not fully standards compliant yet, corroborating the need for sanitizing Linked Data. However, note that this study is purely observational, and the accessed data is not made available in cleaned form.

2.2 Data Crawlers

Sindice [7] presents itself as a Semantic Web indexer. The main question Sindice tries to address, is how and where 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

[5] See `http://sparqles.okfn.com/`

[6] `http://www.w3.org/TR/ld-bp/`

via a user interface and API. Public access to the raw data Sindice crawls is not possible, nor is access via SPARQL possible, restricting the usefulness of Sindice for Semantic Web and Big Data research.

Built on top of Sindice, Sig.ma [9] is an explorative interactive tool, enabling Linked Data discovery. Similarly to Sindice, they provide an extensive user interface, as well as API access. The explorative nature of this service is very useful. However, just as Sindice, the actual raw data are not accessible.

Contrary to Sig.ma and Sindice, data from the Billion Triple Challenge[7] (BTC) 2012 are publicly available. This dataset is crawled from the LOD cloud[8], and consists of 1.4 billion triples. It includes large RDF datasets, as well as RDFa and Microformats. However, this dataset is not a complete crawl of the Linked Open Data cloud (nor does it aim to be), as datasets from several catalogs are missing from the BTC. Additionally, the latest version of this dataset dates from 2012.

Freebase [1] publishes 1.9 billion triples, taken from manual user input and existing RDF and Microformat datasets. Access to freebase is possible via an API, via a (non-SPARQL) structured query language, and as a complete dump of N-Triples. However, these dumps include many non-conformant, syntactically incorrect triples. For instance, the data file that is the dereference of the Freebase concept 'Monkey' visually appears to have hundreds of triples, but a standards-conformant parsers such as Rapper only extracts 30. Additionally, knowing *which* datasets are included in Freebase, and finding these particular datasets, is not trivial.

Similarly, LODCache[9], provided by OpenLink, takes a similar crawling approach as FreeBase does, but does not make a data dump available, making actual re-use of the data difficult. However, LODCache does have a SPARQL endpoint, as well as features such as entity URI and label lookup.

The Open Data Communities service [10] is the UK Department for Communities and Local Government's official Linked Open Data site. These datasets are published as data dumps, and are accessible via SPARQL and API calls. And although this service supports a broad selection of protocols for accessing the data, the number of datasets is limited and restricted to a particular domain.

Finally, DyLDO [5] 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-Quad 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 covered. Another form of incompleteness stems from the fact that the crawl is based on URI dereferences, not guaranteeing that a dataset is included in its entirety.

---

[7] See `http://km.aifb.kit.edu/projects/btc-2012/`

[8] `http://lod-cloud.net/`

[9] `http://lod.openlinksw.com/`

[10] `http://opendatacommunities.org/`

2.3 Portals

Several Linked Data portals exist, attempting the improve the findability of Linked Datasets.

First, the Datahub[11] lists a large set of RDF datasets and SPARQL endpoints, including the famous collection of datasets that is called the LOD cloud. Datasets that are missing from the BTC collection are present in the datahub catalog, and vice-versa. 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 stale dataset descriptions (e.g. DBpedia links to version 3.5.1 instead of 3.9 [12]) and missing or incorrect metadata.

Second, vocab.cc [8] 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 via 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.

3 Context

Due to the points mentioned above, the poor data quality on the LOD cloud poses great challenges to Big Data and SW researchers, as well as to the developers of Web-scale applications and services. 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 3.1). We then compare three approaches towards collecting LOD, and evaluate each with respect to the completeness of their results (section 3.2).

3.1 Requirements

We enumerate the following requirements for such datasets, in order to be practically useful for researchers and developers:

Easy grammar We want to support further processing of the data for non-RDF tools, e.g. Pig [6], grep, sed, and the like. These tools do not come with full-blown SW support and would benefit from a uniform data format that is easy to parse.

Speed We want to allow preprocessing tools to process LOD in a speedy way. Parsing of data documents may be slow due to inefficient serialization formats (e.g. RDF/XML, RDFa), or due to 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 (thousands) and triples (billions), to cover a large parts of the LOD cloud.

[11] `http://datahub.io/`

[12] See `http://datahub.io/dataset/dbpedia` (12 May 2014)

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 *e.g.* load job balancing in large-scale processing, since the distribution of triples across data documents is very uneven (see section 6).

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 at which the original dataset is standards-compliant.

3.2 Dataset completeness

The first problem that we come accross when collecting large amounts of LOD, is that it is difficult to claim completeness while collecting LOD. 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 exits for collecting large volumes of LOD:

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

Resource crawlers use the dereferencability of IRIs in order to find LOD. This approach has the following deficiencies:

1. Datasets that do not contain dereferenceable IRIs are ignored. In [3], 7.2% of the crawled IRIs were not dereferenceable.
2. Datasets that contain some dereferenceable IRIs may have non-dereferenceable parts that will never be reached by the crawler.
3. There is no definition of completeness, since even for datasets that consist of dereferencable IRIs exclusively, the crawler can never be certain that the entire dataset is crawled.
4. Not all dereferenceable IRIs contain back-links [4], making it more difficult to reach most IRIs in a dataset.

Querying endpoints provides another way of collecting 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 exists today[13] , there are thousands of Linked Datasets.
2. Datasets that have a custom API and/or that require an API key in order to pose questions, are not generally accessible and require either the appropriation to a specific API or the creation of an account in order to receive a custom key.

[13] See Feeling The Pulse of Linked Data (ISWC 2014), under submission

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[14]. This results in incomplete datasets retrieval.
4. Existing LOD observatories show that SPARQL endpoint availability is quite low [15]. 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.

Downloading data dumps is the third approach to collecting large volumes of LOD. Its disadvantages are:

1. Datasets that are not available as datadump are ignored.
2. Datasets that have only part of their documents available for download are incomplete.

In terms of completeness, downloading datadumps seems to be the best approach towards collecting large amounts of LOD for further processing, but it comes with various other errors that we qualify in the next section.

4 LOD Washing Machine

In the previous section we have describe our requirements to which Linked Datasets should conform in order to be more useful. We also explained why the downloading of datadumps is the best approach towards this goal from a completeness point of view. Here, we will make the requirements concrete in such a way that they can be automatically applied to dirty Linked Datasets, so as to produce clean data that can be more easily and widely used. The part of the LOD Laundromat that performs the automated data cleaning is called the LOD Washing Machine.[16]

Step A: Collect URLs that denote dataset dumps Before we start laundrying data, we need some dirty data to fill our LOD Laundry Basket with. At the moment, Linked Datasets can be found at various locations on the Web, but despite the efforts of LOD search engines mentioned in section 2, it is not easy to search for all – or even most – data sources from within a single location.

The LOD Washing Machine does not completely automate the search for the initial seed points for collecting LOD. We believe that with current technologies this step cannot be easily automated, and we therefore rely on catalog-specific scripts that collect such seed URLs for washing. At the moment we use the CKAN API[17] for this, which for instance gives us access to the datasets described

[14] E.g., Virtuoso, an often used triple store, limits both the result set size and the number of rows within a sort operation by default

[15] `http://sparqles.okfn.org/`

[16] Code available at `https://github.com/LODLaundry/LOD-Washing-Machine`.

[17] `http://ckan.org/`

in Datahub[18], including the datasets in the original LOD cloud[19]. This means that URLs that are not included in a LOD catalog or portal are unlikely to be thrown in the LOD Washing Machine, although everyone can queue washing jobs by adding dataset URLs to the LOD Laundry Basket[20].

Some URL strings – e.g., values for the "URL" property in a catalog – do not parse according to RFC 3986 grammar[21]. Some URL strings parse as IRIs but not as URLs, mostly because of unescaped spaces. Some URL strings parse per RFC 3986 but have no IANA-registered scheme[22], 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, when needed) 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 carefull 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 not 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, the LOD Washing Machine sends an HTTP request with SSL verification (for Secure HTTP) and an accept header that includes a preference for LOD content types. This includes standardized content types (e.g. `text/turtle`) and content types that occur in practice (e.g. `application/rdf+turtle`).

Some requests are unsuccessful, receiving either a server (500-range), existence, or permission error. Some requests are unsuccessful due to redirection 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 library libarchive[23]. The LOD Washing Machine accesses archives in a stream and opens additional streams for every archive entry it contains. Since archive entries can themselves be archives, this procedure is nested, resulting in

[18] `http://datahub.io/`
[19] `http://lod-cloud.net/`
[20] `http://lodlaundry.github.io/laundryBasket.html`
[21] `http://tools.ietf.org/html/rfc3986`
[22] `http://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml`
[23] `https://code.google.com/p/libarchive/`

a tree of streams. The root node of the tree is the stream to the original archive file, the child nodes are streams to non-archived files, and the other non-leaf files are streams to intermediate archived files.

Some archives cannot be read by libarchive and throw 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 un such archived files, but does report the exception that was thrown.

Stef F: Guess serialization format In order to parse the contents of the textual data that resides in the leaf nodes of the tree stream, we need to know the grammar of that data. The LOD Washing Machine uses content types to denote the grammar used for parsing, as content types are often included in the header of an HTTP response.

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 occurs 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 files (disk space) or multiple downloads of the same file (bandwidth).

We make the observation that the standardized RDF serialization formats occur in two families: XML-like (XML/RDF, RDFa) and Turtle-like (Turtle, Trig, N-Triples, N-Quads). The distinction between these two families can be reliably made by looking only 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.[24] Moreover, the LOD Washing Machine is able to recognize different kinds of syntax errors and recover from them during parsing.

There are many kinds of syntax errors in RDF. We enumerate some of the most common ones here:

- Bad encoding sequences (e.g. non-UTF-8).
- Undefined IRI prefixes.
- Missing end of statement characters.
- Non-escaped, illegal characters in IRIs.
- Multi-line literals in serialization formats that do not support them (e.g. N-Triples, N-Quads).
- Missing or non-matching end tags (e.g. RDF/XML).
- End of file occurs within the last triple (indicating a mistake made in file splitting).
- IRI refs or prefixes not occurring 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 out 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 an example. Most parsers that come accross the syntax in XXX, and, specifically, the dollar sign in the first line, will look for the next end-of-triple statement (i.e., the dot at the end of the first line), and will resume parsing from thereon. This results in "recovering" the collection of triples starting with $\langle$`rdf:c1`, `rdf:c2`, `ex:c3`$\rangle$ and ending with $\langle$`rdf:z1`, `rdf:z2`, `ex:z3`$\rangle$. However, using another heuristic can produce very different results. For instance, by using minumum error distance, the syntax error can also be recovered by replacing the dollar sign with a double quote sign, resulting is a single triple (with an unusually long literal term).

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

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

[24] The SemWeb library[10] that is used by the LOD Washing Machine passes the RDF 1.1 test cases, and is actively used in SW research and applications.

triple store, we perform deduplication on RDF statements, not (syntactic) surface forms. Deduplication cannot be performed on the syntax level, because the same RDF statement can be written in different ways, e.g. by using character escaping, the use of extra white space and/or newlines, and interspersed comments. Another source of the many-to-one mapping between syntax and semantics occurs for RDF data types, for which multiple lexical expressions can map onto the same value.[25] E.g., 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 different 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 3.1.

The LOD Washing Machine produces data in a canonical format that enforces a one-to-one mapping between triples and 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 split without splitting triples. Furthermore, the number of triples in a graph can be easily and reliably determined by counting the number of lines in a file describing that graph.

Secondly, the LOD Washing Machine leaves out any header information. This 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-specifc header declarations (e.g., XML, RDFa) and namespace definitions (e.g., XML, Turtle).

Thirdly, the LOD Washing Machine replaces all occurrences of blank nodes with well-known IRIs[26], in line with the RDF 1.1 specification.[27] 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.[28] This makes it easy to append and split

[25] `http://www.w3.org/TR/xmlschema11-2/`

[26] `https://tools.ietf.org/html/rfc5785`

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

[28] 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.

data documents, without the need to standardize apart blank nodes originating 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 7), we use a canonical form of N-Triples that excludes superfluous white space (only between the RDF terms in a triple and before the end-of-triple character), superfluous newlines (only after the end-of-triple character), and comments. Newlines in literals are escaped (as stated by the N-Triples 1.1 specification) and simple literals do not occur, explicitly including the XML Schema string datatype.

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 perform 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[29]. We store these triples in the global log file for later retrieval. For each dataset that is described in VoID triples, we follow links to datadumps (if present) and download those datadumps as 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 creators to improve their adherence to guidelines, we keep track of all the mistakes that were discussed in this section. The mistakes (if any) are enumerated in the JSON description for each dataset. For syntax errors we denote 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 JSON descriptions are automatically updated at future executions of the LOD Washing Machine.

5 LOD Wardrobe

The datasets that are cleaned by LOD Washing Machine, are ironed and folded into the LOD Wardrobe[30] (See Figure 1).

Here, we publish the datasets coming out of the LOD Washing Machine and their corresponding metadata via a JSON API. Additionally, we publish metadata for those datasets we failed to crawl. This metadata includes information such as the number of triples in a dataset, the number of removed duplicates, the original serialization format, syntax errors, and more. The metadata that the LOD Wardrobe publishes is continuously updated whenever new cleaned laundry comes in. We present a live updated Web UI for easy human access, as well as a JSON API for easy machine access. Other than *access* to these dataset, the LOD Wardrobe provides realtime *visualizations* of the crawled datasets as well, using our JSON API.

[29] `http://www.w3.org/TR/void/`
[30] See `http://lodlaundry.github.io/wardrobe.html`

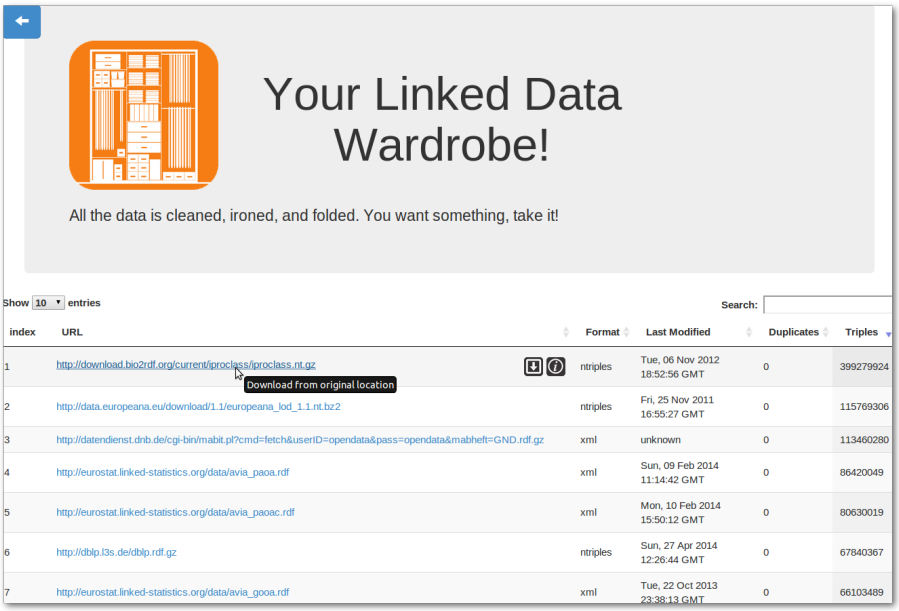


Fig. 1: Wardrobe

Additionally, to extend our collection of datasets in the LOD Laundry Basket, we allow users to add seed locations via the Web UI. Such locations can either be URLs pointing to VoID description or to data dumps.

6 Observations

In this section we present a number of observations and statistics on the data we washed clean[31]. We provide these observations to illustrate the data we washed clean up until now, and not as a comprehensive analysis of the state of the LOD cloud (which, as Section 2 shows, is already provided for). Furthermore, these observations are only a snapshot at the time of writing, where the online version is live-updated.

We applied the LOD Washing Machine presented in section 4 to the following seed list:

- The Linked Datasets described by the Datahub catalog.[32]
- Datasets referenced from Linked Open Vocabularies.[33]
- Several URLs to Linked Datasets that we added by hand.

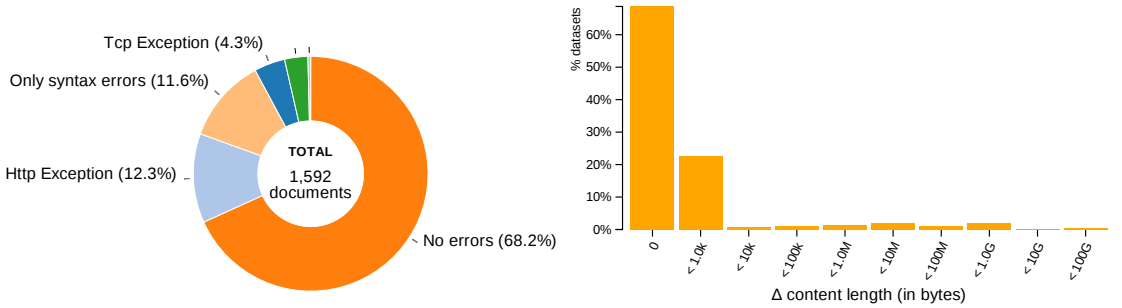
[31] Note that all plots presented in this paper are clickeable, and point to the online interactive version

[32] `http://datahub.io/`

[33] `http://lov.okfn.org/dataset/lov/`

Using our LOD Laundry Machine, we crawled 1.297 document (and counting) (See Figure 2a), containing 2 billion triples. The washing machine managed to wash 68.2% of the documents without problems. 12.3% of the crawl attempts resulted in an HTTP exception (e.g. a '500' server error) and 11.6% of the documents returned unparseable content. 4.3% of the attempts failed because of TCP errors, e.g. caused by domains which do no longer exist. Finally, for 3.2% of the documents, we managed to extract triples, despite a number of syntax errors.

Other than these errors, we encountered non-conformant HTTP headers as well, particularly the use of content-length. The proper use of content-length allows consumers retrieve data more efficiently, e.g. by load-balancing data depending on the expected size of the response. However, our results show that 32% of the documents return an invalid content length value, making efficient use of the HTTP connection difficult.



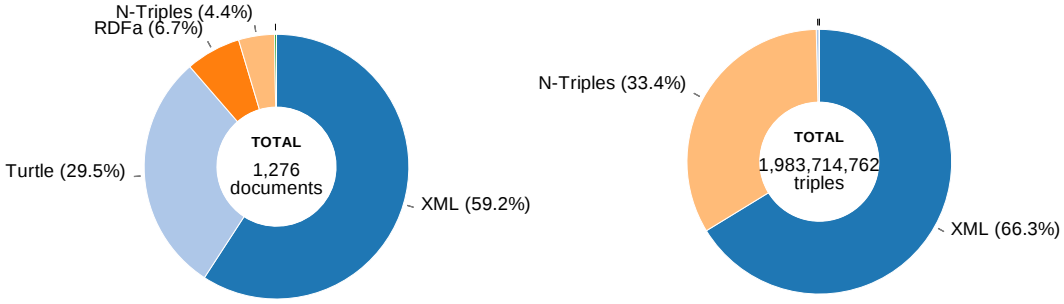
(a) Types of LOD Washing Machine errors

(b) Difference between HTTP content length and *actual* content length

Fig. 2: Types of errors

In figure 3a, we present the particular serialization formats of the 1.276 documents we were able to parse. The majority of documents, 59.2%, are represented as RDF/XML. Turtle and RDFa amount to 29.5% and 6.7% respectively. Only 4.4% of all documents are serialized as N-Triples.

Figure 3b shows the same data related to the number of *triples*. Interestingly, the 29.5% of the *documents* which are serialized as Turtle, amounts to 0.4% of all the crawled *triples*. This large difference is probably caused by a low use of turtle over all the datasets, combined with one particular large dataset which *is* serialized as Turtle. 66.3% and 33.4% of the triples were serialized in XML and N-Triples, where only a fraction of the triples were serialized as RDFa or TRIG.



(a) Serializations per #documents

(b) Serializations per #triples

Fig. 3: Types of serializations

7 Conclusion

Related work showed how Linked Data publishing practices are not adhered by many data providers. To deal with this issue, we present LOD Laundromat, a uniform way of publishing other peoples dirty data.

Using LOD Laundromat, we publish standards- and guidelines-compliant datasets that are siblings to existing, idiosyncratic datasets. LOD Laundromat is a services which *continuously* crawls for additional datasets; the number of triples we publish already surpasses existing data collections such as the Billion Triple Challenge. Additionally, the LOD Laundromat publishes metadata for each dataset we crawled (or failed to crawl) via our site and API, providing near-realtime information and visualization as well. Because anybody can drop their dirty data in the LOD Laundry Basket, the coverage of the LOD Laundromat will increase over time.

The published datasets are a canonical form of N-Triples, increasing the ease with which to parse, stream and analyze the dataset, without the need to worry about serializations, syntax errors, encoding issues or duplications. In doing so, LOD Laundromat can act as an enabler for Big Data and SW research, as well as a provider of data for Web-scale applications.

Though the current state of LOD Laundromat offers many advantages, we aim to increase the service even more by developing a LOD alternative to the JSON API. In doing so, we will reuse existing vocabularies such as DCAT[34], VoID, and Prov-O[35].

Additionally, we will extend the dataset metadata descriptions as well. Currently, the LOD Laundromat API provides high level metadata for all the crawled datasets (and for those it failed to crawl). We will extend this metadata with more low level information. However, we cannot directly use the metadata of public LOD datasets, as these are not guaranteed to be correct (especially for

[34] `http://www.w3.org/TR/vocab-dcat/`
[35] `http://www.w3.org/TR/prov-o/`

data descriptions published via catalogs, which are often filled in by hand). Therefore, we will extend the current API to include more low level dataset meta-data, that cannot be falsified by automated means.

Finally, LOD Laundromat currently disseminates datasets in the N-Triples format. Some data documents contain multiple named graphs. Although such datasets may not be very common, such cases can be better supported by using the N-Quads format, and by scoping the deduplication of triples to graphs.

References

1. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 1247–1250. ACM (2008)
2. Callahan, A., Cruz-Toledo, J., Ansell, P., Dumontier, M.: Bio2rdf release 2: Improved coverage, interoperability and provenance of life science linked data. In: The Semantic Web: Semantics and Big Data, pp. 200–212. Springer (2013)
3. Hogan, A., Harth, A., Passant, A., Decker, S., Polleres, A.: Weaving the pedantic web (2010)
4. Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S.: An empirical survey of linked data conformance. Web Semantics: Science, Services and Agents on the World Wide Web 14, 14–44 (2012)
5. Käfer, T., Abdelrahman, A., Umbrich, J., O'Byrne, P., Hogan, A.: Observing linked data dynamics. In: The Semantic Web: Semantics and Big Data (2013)
6. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 1099–1110. ACM (2008)
7. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice. com: a document-oriented lookup index for open linked data. International Journal of Metadata, Semantics and Ontologies 3(1), 37–52 (2008)
8. Stadtmüller, S., Harth, A., Grobelnik, M.: Accessing information about linked data vocabularies with vocab. cc. In: Semantic Web and Web Science, pp. 391–396. Springer (2013)
9. Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R., Decker, S.: 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)
10. Wielemaker, J.: SWI-Prolog Semantic Web Library 3.0. Tech. rep. (2012), `http://prolog.cs.vu.nl/download/doc/semweb.pdf`