Creating voiD Descriptions for Web-scale Data

Christoph Böhm, Johannes Lorey, Felix Naumann

Hasso Plattner Institute, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
firstname.lastname@hpi.uni-potsdam.de

Abstract

When working with large amounts of crawled semantic data as provided by the Billion Triple Challenge (BTC), it is desirable to present the data in a manner best suited for end users. This includes conceiving and presenting explanatory metainformation. The Vocabulary of Interlinked Data (voiD) has been proposed as a means to annotate sets of RDF resources to facilitate not only human understanding, but also query optimization.

In this article we introduce tools that automatically generate voiD descriptions for large datasets. Our approach comprises different means to identify (sub)datasets and annotate the derived subsets according to the voiD specification. Due to the complexity of Web-scale Linked Data, all algorithms used for partitioning and augmenting are implemented in a cloud environment utilizing the MapReduce paradigm. We employed the Billion Triple Challenge 2010 dataset [6] to evaluate our approach, and present the results in this article. We have released a tool named voiDgen to the public that allows the generation of metainformation for such large datasets.

Keywords: Semantic Web; Vocabulary of Interlinked Data; Semantic Data Profiling; RDF Metadata Generation; Cloud Computing

1. Introduction

Open data emerges from a variety of sources, e.g., government agencies, bio-science institutes, social networks, or community-driven knowledge bases. As of January 2011, ckan.net states to contain over 1,600 data packages. Often, such data is published as Linked Open Data (LOD) – data that adheres to a set of guidelines to allow easy reuse and semantic integration [4]. Specifically, ckan.net provides approximately 250 LOD sources. Due to the wealth of information available, descriptive metadata is essential for every open dataset. Furthermore, we believe that metainformation concerning relationships between distinct datasets represents valuable inter-domain knowledge and therefore provides additional insight.

Metadata is useful in a multitude of scenarios: the most obvious case is when data engineers search for information about a specific topic. How do they know what a dataset at hand is about and how can they quickly discover connections to other open sources that they already work with? A data source should provide this information in a standardized way. A second application is crawling the LOD cloud: here, raw statistics, e.g., the number of triples, resources, links, etc., are of interest for scheduling crawling tasks and provisioning resources. Also, semantic information, such as considered types or related resources, can facilitate useful segmentation of the data. Query answering for Linked Data is another scenario where statistics and metainformation can support decision making and help achieve better results more efficiently. We believe that a wide availability of well-defined metadata expedites the causes of data interconnectivity and semantic integration. The Vocabulary of Interlinked Datasets (voiD) addresses this need.

VoiD. The Vocabulary of Interlinked Datasets is an RDF-based schema to describe linked datasets [2, 8]. By providing a standardized vocabulary, it aims at facilitating the discovery of linked datasets as well as their usage. VoiD offers two main classes: a void:Dataset describes collections of data published and maintained by a single provider. A void:Linkset on the other hand is a subclass of void:Dataset, which describes entities linking to other sources. For linksets, interlinking predicates or link directions can be stored. Additionally, a number of properties defined in voidD describe technical or statistical features of datasets.

VoiD was introduced in 2009 and is already used by a number of projects. For instance, the authors of Zemánek and Schenk. [15] plan to leverage voiD dataset statistics for query optimization. The voiD browser [11] allows to use URIs to search for datasets. These approaches require existing voiD annotations, which have been created for a number of sources, e.g., data.gov.uk [14], the OECD Glossary [13], or just recently DBpedia [7]. Though it is considered to be fairly simple to produce voiD descriptions by hand, there are numerous sources in today’s LOD cloud that do not provide them [1]. In our opinion, this is

\footnote{For example, at the time of writing the prominent New York Times dataset did not provide a voiD description.}
due to the in fact substantial manual effort required to create them. Also, we often find that this metadata is either incomplete or does not reflect the entire contents of a dataset. For this reason we developed a set of algorithms and heuristics to create void descriptions automatically.

**Contribution.** We have created a set of algorithms to automatically generate void descriptions for Web-scale datasets. Note that we do not mean to replace manual creation of metadata — we rather target large, crawled datasets without full void descriptions. In addition, we propose extensions of the void standard, i.e., novel approaches to distinguish datasets (see Sec. 3) as well as fuzzy linksets (see Sec. 2.2). Due to the large volumes of data we tackle, we employ the MapReduce paradigm to efficiently compute void content. To demonstrate feasibility and scalability of our approaches, we present results for the Billion Triple Challenge Dataset 2010 [6] at [https://www.hpi.uni-potsdam.de/naumann/sites/btc2010](https://www.hpi.uni-potsdam.de/naumann/sites/btc2010). On this site, we also offer a user-friendly tool as well as all sources and comprehensive documentation for download. For this article and our implementation, we use void version 1 of May 07, 2010 (the most current at the time of writing). To avoid namespace squatting and ambiguity we use voidgen as namespace prefix for the void extensions we propose. For ease of reading, we reuse void properties such as void:feature but note that, of course, property values would require a proper definition as void:TechnicalFeature.

**Related Work.** Related work includes tools meant to create any sort of metainformation. Because such tools mainly originate from traditional database vendors and are not suitable for graph-structured RDF data, we do not discuss them here. However, in our group we are developing ProLOD [5] for iterative RDF data profiling. ProLOD aims at determining data quality issues instead of descriptive metadata and does thus not address void descriptions.

Besides tools there are libraries such as NXParse [9], which reads files in Nx format and is capable of dumping simple statistics about the data. RDFStats[7] computes statistics and outputs them using the so-called RDFStats statistics vocabulary. Finally, many developers use hand-crafted scripts to perform metadata extraction. On Grimm’s Blog [8] one can find interesting results of such an approach. Others use high-end hardware to perform statistics computation for web-scale datasets [10].

For automatic generation of void properties, Virtuoso’s database provides a function called RDF VOID STORE that creates descriptions for RDF graphs. For this tool, we are unable to state how well it performs for Web-scale datasets and heterogeneous data from multiple sources in a single graph. Further, Virtuoso offers additional functions that create metainformation.

Also, there are tools for manual curation of void descriptions, e.g., ve2 [9], which is a Web-based application. It allows manual input for dataset characteristics such as categories, interlinking, as well as technical features and creates RDF output in an on-the-fly manner.

Last, there is a notable project by the RKB Explorer team that collects existing void descriptions, stores them, and provides query and browsing functionality.

**Structure of this article.** First, we introduce a basic partitioning algorithm that outputs dataset information according to the void definition (Sec. 2.1). We then illustrate the computation of linksets, using both the original approach and a new fuzzy version (Sec. 2.2), followed by a description of dataset metadata generation (Sec. 2.3). Next, we propose three new ideas for dataset partitioning and the corresponding algorithms (Sec. 3) before concluding this article (Sec. 4).

2. Generating void annotations

As mentioned above, void is centered around datasets and linksets. Datasets group subjects according to a specific property, i.e., the data publisher in the original void definition. In contrast, linksets contain links, i.e., triples among datasets. We first provide a way to compute both concepts in their original form before discussing metadata generation and possible void extensions. Note that while in the next sections we assume the semantic data at hand to be in a triple format (subject, predicate, object), our approaches also apply to quadruple format (with additional context information) as presented in the BTC 2010 dump.

2.1. Basic Datasets

The void standard associates a dataset with a single publisher, e.g., through a dereferencable HTTP URI or a SPARQL endpoint [8]. Typically, this means that all URLs described in a dataset are similar. Thus, we base our clustering on the dataset notation of Def. 1.

**Definition 1.** Two triples belong to the same dataset, iff the subjects’ URIs start with the same pattern. The length of the pattern is determined by the longest common prefix of all subjects in a dataset ending in one of the characters \( :/\) or \( #\).

For example, the two subjects [http://dbpedia.org/resource/Category:Germany](http://dbpedia.org/resource/Category:Germany)

[^1]: http://www.v3.org/wiki/NamespaceSquatting
[^2]: Please note that we do not plan to formally allocate our own namespace but rather suggest to incorporate our extensions in void.
[^3]: http://www.hpi.uni-potsdam.de/naumann/sites/prolod
[^4]: http://sw.deri.org/2006/08/nxparser
[^5]: http://rdfsstats.sourceforge.net
[^6]: http://prologui.net/blog/rf=btc
[^7]: http://virtuoso.openlinksw.com/
[^8]: http://lab.linkeddata.deri.ie/ve2
[^9]: http://lab.linkeddata.deri.ie/ve2
[^10]: http://www.w3.org/wiki/NamespaceSquatting
[^11]: http://void.rkbexplorer.com
and http://dbpedia.org/resource/Tim_Berners-Lee
belong to the same dataset—in this case http://dbpedia.org/resource/. For simplicity and
readability, we identify a dataset by its URI endpoint, i.e., the void:uriLookupEndpoint. The generation of the
remaining void:Dataset attributes is described in Sec. 2.3.

In other words, our basic dataset grouping partitions
a data corpus using the individual subjects’ uniform re-
source identifiers if present. According to the W3C RDF
format specification, a subject has to be either referenced
by a URI or a blank node [12]. In the latter case, the asso-
ciated triples are ignored for dataset grouping, as there is
no publisher associated with them. Malformed URIs are
disregarded as well. As the majority of subject URIs in our
corpus adhere to the HTTP scheme, all other schemes each
form one individual dataset, such as the set of phone
numbers (with its scheme identifier tel). However, for non-
HTTP schemes it is also possible to use domain-specific
properties to distinguish subsets, e.g., the country code in
case of phone numbers.

Discovering datasets based on the longest common URI
prefix is conceptually straightforward and requires two
MapReduce runs. The first run determines all possible
non-trivial prefixes for every subject. For example, for
http://dbpedia.org/resource/Category:Germany the pre-
fixes http://dbpedia.org/, http://dbpedia.org/resource/
and http://dbpedia.org/resource/Category: are discov-
ered. In the second step, the most suitable prefix for a
dataset is determined, i.e., the longest one common to all
subjects in the dataset. Consider the sample structure
given in Fig. 1. Here, datasets identified by this basic par-
tioning approach are indicated by the color of the sub-
jects that belong to them. They are also listed accordingly
in the first column of Tab. 1.

To avoid ambiguity, for the remainder of this article the
original, unpartitioned BTC 2010 dump is referred to as
‘corpus’, whereas a ‘dataset’ describes any logical subset
of the corpus determined by one of the introduced parti-
tioning approaches.

After partitioning the data corpus, we compute differ-
ent void:statistics of the newly discovered datasets, such as
void:numberOfPredicates, void:numberOfSubjects, etc.
Notice that these attributes each refer to unique entities
within a dataset only. These void:attributes already pro-
vide interesting insight into the structure of the datasets:
for example, a low number of void:numberOfPredicates rel-
ative to void:numberOfSubjects suggest that the included
entities all belong to the same type and thus share most of
their attributes. For a heterogeneous dataset like DBpedia
on the other hand, this relation is very different. Listing 1
presents a sample void: description for the example.com/
dataset from Fig. 1 in the Turtle RDF serialization for-
mat 3.

2.2. Linksets

Crisp Linksets. Besides datasets, the void standard also
introduces the notion of linksets. A void:Linkset contains

\[
\text{Listing 1: A void:Dataset description}
\]

\[
\text{Listing 2: A (crisp) void:Linkset description}
\]

all links from one dataset to another, where links are iden-
tified by triples in which the subject belongs to a different
dataset than the object. In our implementation, a linkset
may also be reflexive, i.e., it can describe the connections
within a single dataset. Linksets are not symmetric, but
rather directed from one dataset to another. For example,
we discovered 4,042 links from DBpedia to GeoNames, but
6,956 links in the other direction.

Once datasets have been determined, linksets among
them can be obtained in one MapReduce run. In the Map
phase, all triples in which both subject and object are
members of previously identified datasets are extracted.
The emitted tuple then contains the subject and object
dataset identifier as key. The Reduce phase subsumes all
tuples identified by the same key. Listing 2 presents the
void: description for the linkset from bar.net to foo.org.

Fuzzy Linksets. In addition to these ‘crisp’ linksets, we
also examine links between different datasets that are not
explicitly stated. We introduce the notion of k-similarity,
where two subjects are k-similar, if k of their predi-
cate/object combinations are exact matches. Def. 2 pro-
vides a formal definition that helps identify ‘fuzzy’ linksets
among datasets.

\[
\text{Definition 2. For a fixed subject } s_1, \text{ and a number of}
\text{ associated predicates } p_{i,1} \text{ with objects } o_{i,1}, etc., \text{ for the}
\text{triples } (s_1 p_{1,1} o_{1,1}), (s_1 p_{1,2} o_{1,2}), \ldots, \text{ the set } C_1 \text{ denotes all}
\]

3
of the predicate/object combinations at hand. For $k > 0$, two subjects $s_1$ and $s_2$ are $k$-similar iff

$$C_1 = \{(p_{1,1}, o_{1,1}), \ldots, (p_{1,n}, o_{1,n})\},$$

$$C_2 = \{(p_{2,1}, o_{2,1}), \ldots, (p_{2,m}, o_{2,m})\},$$

and

$$|C_1 \cap C_2| = k,$$

with

$$(p_{i,j}, o_{i,j}) = (p_{k,l}, o_{k,l}) \iff p_{i,j} = p_{k,l} \land o_{i,j} = o_{k,l}.$$

The intuition of $k$-similarity is that two subjects are to some degree similar if they share a common set of attribute values, and might therefore be relatable, e.g., using one of the methods introduced in [10]. These fuzzy linksets connect similar entities (and thereby datasets) that are not explicitly referenced by one another. Instead of $k$-similarity one could use any other notion of similarity among subjects. With $k$-similarity, however, we chose a strict starting point for fuzzy linksets that is generic, simple, and easy to parameterize (using $k$). It should be noted that in contrast to crisp linksets, fuzzy linksets using $k$-similarity are always symmetrical.

In Listing 3 a description for a fuzzy linkset between example.com and foo.org is denoted in Turtle format. The listing refers to Fig. 1 where there are three 1-similar links with predicate rdf:type and object :city connecting example.com/Munich, example.com/Berlin, and foo.org/place#Lyon, thus indicating a fuzzy linkset. As fuzzy linksets are not yet defined in the Vocabulary of Interlinked Data, we propose a new class voidgen:FuzzyLinkset and a new attribute voidgen:kSimilarity to specify them properly. The latter may also be represented as a technical feature as is the case in Listing 3.

A number of factors influence the ‘interestingness’ of fuzzy links. On the one hand, a higher value for $k$ indicates that the two subjects have a high number of predicate/object combinations in common. Hence, $k$ (calcul-
2. Dataset Metadata

The void standard introduces a number of properties that characterize a dataset. However, some of the properties are meant to be augmented manually by the data provider and cannot be derived automatically, e.g., the license of a dataset or the date of its creation. Hence, we limit ourselves to the subset of properties that can be deduced from the resources within the dataset, but still provide interesting insights for data consumers.

The attribute void:exampleResource provides a link to a representative entity within the dataset. In our approach, we filtered the subject that provided the most

lated absolute or relative) can indicate similarity of subjects and thus a new relationship between the associated entities is disclosed. To illustrate this effect, we analyzed a 10% sample of the BTC 2010 corpus, containing 317 million quadruples. Of these, around 122 million had one predicate/object combination in common with at least one other quadruple. By setting $k$ to 2, the number of associated quadruples drastically decreased to 25.

On the other hand, some predicate/object combinations appear very often, and are therefore not insightful. In the BTC 2010 corpus for example, the rdf:type predicate occurs quite frequently in conjunction with the rdf:Resource object, rendering a small value for $k$ unsuitable for our approach and this specific combination. In general, more specific predicates, i.e., predicates that do not appear very often themselves, provided a better lead for detecting dataset similarity. Overall, $k$-similar linksets can be considered an extension to the void:Linkset class, revealing implicit, fuzzy connections between two datasets.

The computation of $k$-similarity is implemented as two MapReduce runs. The first run clusters subjects by a hash value obtained from respective predicate/object pairs. The second run performs a Map-Side (Self-)Join on the hash values and determines $k$-similarity values in the Reducer.

3. Extending void Content

Detecting individual datasets provides interesting insights into the contents of large data corpora. In its original void definition, a dataset identifies a set of data provided by a specific publisher. We define semantic datasets, i.e., partitions of resources that share certain semantic features. Specifically, we provide means to identify connected sets of resources or sets of conceptually similar resources.
Given two such semantic datasets and respective linksets, one can, for instance, observe the connectivity among concepts.

3.1. Connected Datasets

Two resources reside within the same connected dataset, iff there is a link of a specific type between them. Hence, we compute connected components of a predicate-based subgraph $H$ of the original RDF graph $G$. Ding et al. provide a formal definition of a predicate-based subgraph filter $psf$ and the resulting subgraph $H$ \[9\].

**Definition 3.** A predicate-based subgraph filter is a function $H = psf(G, P)$, where $H$ and $G$ are RDF graphs and $P$ is a set of RDF properties. The function $psf$ returns $H$ which is a subgraph of $G$, and the predicate of any triple in $H$ is a member of $P$.

If the type of the link is undefined, i.e., $P$ is the set of all predicates in the RDF graph, any two connected resources share the same dataset and there are no linksets. In contrast, if the links among resources are fixed to one specific type, e.g., $P = \{\text{owl:sameAs}\}$ or $P = \{\text{livesIn}\}$ in Fig. 1 then one can derive meaningful linksets. Tab. 1 lists the datasets for $\text{owl:sameAs}$ and the custom-defined $\text{livesIn}$ link in the second and third column, respectively. In the first case, linksets contain $\text{livesIn}$ and $\text{worksAt}$ links whereas in the second case linksets contain $\text{owl:sameAs}$ and $\text{worksAt}$ links (for simplicity, we here disregard the other two link types).

A suggested notation for connected datasets is presented in Listing 4. It should be pointed out that besides specifying the void:linkPredicate, it is essential to indicate a pivot resource to be able to gather all entities belonging to a connected dataset. In Listing 4 the resource example.com/Berlin can be used to determine all other elements of the dataset by executing the appropriate SPARQL query using the specified void:linkPredicate attribute value. Notice that besides a new class (voidgen:ConnectedDataset), no further additions would need to be made to voidD. However, it could be argued that the improper use of void:exampleResource and void:linkPredicate should be rectified by adding new, unambiguous attributes. In addition, some of the attributes reserved for void:Dataset might be omitted for connected datasets as they do not seem to bear much information, such as void:uriLookupEndpoint.

To compute connected datasets, we have implemented a two-phase MapReduce sequence: the pseudocode in Jobs 1 and 2 exemplarily illustrate these MapReduce jobs. First, connected resources are assigned to individual clusters (Job 1 map), and then resources and ids are assigned to the respective minimum cluster id (Job 1 reduce). The second phase iteratively builds the transitive closure of clusters until all clusters are merged (Job 2).

### MapReduce Job 1 Connected Datasets Step 1

```java
function map (Object k, Text v):
1: quadruple ← parse(v)
2: if quadruple.predicate ∈ P then
3: # generates new cluster id
4: cid ← counter.get('cid').increment()
5: emit(quadruple.subject, cid)
6: emit(quadruple.object, cid)
7: end if

function reduce (StringLong k, List<Long> vs):
8: minClusterId ← min(vs)
9: # entities will be put into minClusterId cluster
10: emit(minClusterId, k)
11: # all clusters get the id of their base cluster
12: for all v ∈ vs do
13: emit(v, minClusterId)
14: end for
```

3.2. Conceptual Datasets

Two resources are contained in the same conceptual dataset, iff they are of the same or of related type. To calculate this relationship, we provide two approaches.

The **hierarchical approach** assigns any resource representing a concept and all its superconcepts (up to a certain level $d$) to an individual dataset. Consider the last column of Tab. 1 as an example: for $d = 1$, the first partition contains all resources of type :city whereas the second partition comprises entities of type :capital. By increasing $d$ to 2, the transitive closure of :capital is determined, and the aforementioned two partitions are merged into a single one. The number of MapReduce runs for this partitioning approach is variable, depending on the number of iterations $d$ allocated for detecting transitive links.

The **distinct approach** selects a single concept type per resource and assigns the resource to the respective dataset. For example, in Fig. example.com/Berlin is both a city and a capital. The previous transitive approach assigns it to both respective partitions for $d = 1$, and forms...
MapReduce Job 2: Connected Datasets Step 2

function map (Long k, StringLong v):
1: emit(key, value)

function reduce (LongString k, List<StringLong> vs):
2: if vs.getFirst() is Id and vs.getFirst() < k then
3: minClusterId ← vs.getFirst()
4: emit(k, minClusterId)
5: else
6: minClusterId ← k
7: end if
8: for all v ∈ vs do
9: if v is resource then
10: emit(minClusterId, v)
11: else
12: emit(v, minClusterId)
13: end if
14: end for

Listing 5: A conceptual dataset description (:capital)

...


