University of Zurich
Department of Informatics

Quality Estimation and Provider Selection Mechanism

Dynamic and Distributed Information Systems

Bachelor Thesis Nov 22, 2012

András Heé
of Zurich, Switzerland

Student-ID: 08-982-142
andras.hee@uzh.ch

Advisor: **Mengia Zollinger**

Prof. Abraham Bernstein, PhD
Department of Informatics
University of Zurich
http://www.ifi.uzh.ch/ddis

Acknowledgements

I would like to thank the following persons for their support in writing this thesis:

First and foremost my advisor Mengia Zollinger for her time, efforts and encouragements during the thesis. Her helpful and qualified support was very motivating. Professor Abraham Bernstein for giving me the opportunity to write the thesis at the Dynamic and Distributed Information Systems Group at the University of Zurich. And last but not least my family and all my friends who supported me.

Abstract

This paper documents the algorithms and key aspects of a *Quality Estimation and Provider Selection Mechanism (QEPSM)* for SPARQL endpoints. The prototype implements a mechanism that crawls the Web for SPARQL endpoints and then collects metadata about the data providers to estimate the quality of their provided data. This data quality is determined by an assessment of the data providers and their SPARQL endpoints using three different algorithms. They rank the reputation analysing the relationships between the datasets similar to Google's PageRank, the availability of the SPARQL endpoints, the support of SPARQL functionalities and the quality of the used vocabularies. With this information the tool offers a list of data providers ordered by decreasing data quality, which can support other metrics to elicit an optimal allocation of federated queries. A web interface visualises the data and ranks.

Zusammenfassung

Dieses Dokument beschreibt die Architektur und die Umsetzung eines *Quality Estimation and Provider Selection Mechanism (QEPSM)* für SPARQL Endpoints. Das Tool implementiert einen Crawler, welcher das Web nach SPARQL Endpoints absucht und danach mit Hilfe von zusätzlich gesammelten Metadaten die Qualität der Daten schätzt. Diese Qualität wird aufgrund einer Analyse des Datenanbieters und dessen SPARQL Endpoints anhand von drei verschiedenen Algorithmen ermittelt. Diese bewerten die Reputation anhand einer Analyse der Verbindungen zwischen den Datensätzen ähnlich wie der PageRank von Google, die Verfügbarkeit der SPARQL Endpoints, die Unterstützung von SPARQL Funktionen und die Qualität des gebrauchten Vokabulars. Mit diesen Informationen kann das Tool eine Liste mit Datenanbietern generieren, sortiert nach absteigender Datenqualität. Diese erlaubt die Unterstützung von anderen Metriken für eine optimale Anbieterzuweisung bei SPARQL Anfragen. Ein Webinterface bereitet die Daten grafisch auf und erlaubt einen schnellen Überblick über die Datenanbieter.

Table of Contents

1

Introduction

One of the main problems of the current web is that it connects only unstructured websites. The semantics of web pages is hardly interpretable by algorithms. This makes it very difficult to extract and query the data in a structured way. The *Linked Open Data Project (LOD)*[1] addresses this problem by using the Web to connect related data that were not previously linked and to generate processable data. The *Resource Description Framework (RDF)*[2] allows to describe information in the formal form of subject-predicate-object triples. Each of them is represented by a URI, but objects can be string literals as well. A collection of RDF triples represents a labelled, directed multigraph. Subject and object are the nodes and the predicate is represented as an edge between the two nodes. RDF data is often persisted in repositories called *triple stores*. However, most triple stores also store other information with each triple, which makes the name confusing. *SPARQL*[3] is a query language for RDF. It can be used to express queries across diverse data sources, whether the data is part of a Semantic Web project or a relational database is viewed as RDF via middleware. We call the web service that allows to send SPARQL queries a *SPARQL endpoint*. RDF and SPARQL are both part of the *W3C Semantic Web Activity*[4].

The goal of the LOD community is to extend the existing data on the Web with additional metadata. They publish various open datasets in RDF format on the Web and define RDF links between the different data entities. Tim Berners-Lee [Berners-Lee, 2006], famous for being the inventor of the World Wide Web, outlines a set of principles for publishing linked data on the Web:

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using standards like RDF or SPARQL.
- Include links to other URIs, so that they can discover more things.

[1] http://linkeddata.org
[2] http://www.w3.org/RDF/
[3] http://www.w3.org/TR/rdf-sparql-query/
[4] http://www.w3.org/2001/sw/

1.1 QueryManager

This tool is part of the project *QueryManager* developed by Mengia Zollinger at the *Dynamic and Distributed Information Systems Group (DDIS)*[5] at the University of Zurich. The Linked Data Project is mainly funded by non-profit organisations and several governments. Since the data are published for free, data publishers cannot charge consumers for their services even though it takes much effort. This leads to a market breakdown when government subventions are omitted.

The QueryManager solves this problem by introducing a pricing model for data providers. It presents an intermediate auction mechanism to elicit an optimal allocation and pricing of queries. Along with the offered price, the quality of the data provider should be considered as well to suggest which data provider should be invited to a forthcoming auction given a data requester's query. This is where this tool comes into play.

1.2 Work Description

Figure 1.1 presents the process of the mechanism. In a first step the tool crawls the web to find datasets providing a SPARQL endpoint service. It collects further metadata about the data providers. It estimates then the data providers' quality by analysing all available data. We assess the quality of the datasets and their SPARQL endpoints *globally*, as opposed to a local assessment. This means that the quality is estimated query independently. The quality metrics and algorithms are based on provider reputation, data accessibility and degree of integration of the RDF triples. In the third and last step we retrieve the crawled data and calculated ranks.

The tool implementing the presented mechanism is built modularly and can therefore be used as a standalone framework for the SPARQL endpoint discovery and ranking. The crawled datasets and their ranking values can be accessed either using SPARQL queries or a wrapper library coded in Python. The tool further provides a web interface that allows to browse the datasets. Interactive line chart plots visualise the distribution of the ranking values.

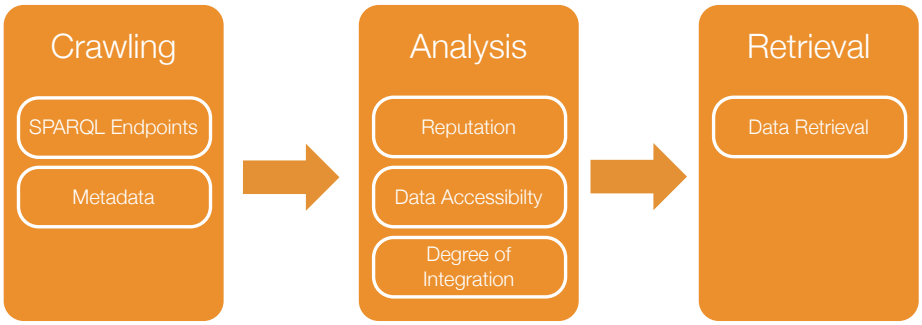

Figure 1.1: Process Flow of QEPSM.

[5] http://www.ifi.uzh.ch/ddis.html

1.3 Thesis Outline

We start with an overview of the related works. Chapter 3 specifies the framework used to describe the datasets, their metadata and the resulting ranking values. The following chapter 4 discusses the techniques we have applied to crawl the SPARQL endpoints and the additional metadata. After having the required information we discuss in chapter 5 the algorithms which estimate the global data quality of a data provider. Chapter 6 illustrates the architecture and implementation details of the tool. In chapter 7 and 8 we discuss some limitations of the tool and subsequent future works. This leads to the final chapter 9 which summarises the thesis.

2

Related Works

In this chapter we shortly present some related works. The existing works for ranking entities in the Semantic Web are mainly based on link analysis. In chapter 5, the existing mechanisms and algorithms are discussed in more detail.

Swoogle[1] [Ding et al., 2004; Finin et al., 2005] is an indexing and retrieval system for the Semantic Web. It discovers Semantic Web documents, extracts metadata for each discovered document and calculates relations among them. The authors introduce with *OntoRank* a metric for the importance of a Semantic Web document. *TripleRank* [Franz et al., 2009] presents another mechanism and prototype which ranks RDF triples. *Spring* [Mulay and Kumar, 2011] follows a similar setup to rank the results of SPARQL queries. In contrast to the works mentioned, we operate on the granularity of datasets rather than documents or RDF triples. *DING!* [Ding et al., 2005] uses formal descriptions of datasets to rank datasets and works therefore on the same granularity level. Its ranking algorithm uses also the relationships of the datasets as input.

In our mechanism we extended the ranking mechanism to consider as well the data accessibility and the degree of integration of the datasets in addition to the reputation calculated by the link analysis.

[1] http://swoogle.umbc.edu

3

Describing Datasets

We need an appropriate framework to save and to retrieve the information we crawl and analyse. Every dataset should be associated with metadata, which allows to estimate its quality. For the sake of consistency we use also RDF for describing our analyses. The three main vocabularies (*VoID*, *vRank*, *EndS*) used are described in detail in the next sections. VoID is the standard to describe datasets defined by W3C, vRank is a vocabulary for algorithm values and EndS is useful to track availabilities of endpoints. We use always the same vocabulary regardless of the crawling methods and algorithms applied. Figure 3.1 gives an overview of all classes and properties. The orange-coloured entities are used for the final retrieval of the SPARQL endpoints, while the white ones are used in intermediate steps. The rectangular boxes represent RDF classes and the oval ones properties.

3.1 VoID Vocabulary

The *Vocabulary of Interlinked Datasets (VoID)*[1] is a vocabulary that allows to formally describe RDF datasets, providing a set of instructions. It acts as a bridge between the providers and consumers of data.

The fundamental entity in VoID is the class *void:Dataset*[2]. In the W3C Interest Group Note a dataset is defined as "set of RDF triples that are published, maintained or aggregated by single provider" [Alexander et al., 2011]. In contrast to *RDF graphs*[3], which consist also of a set of RDF triples, but are purely mathematical constructs, a dataset has a more semantic dimension. A dataset is a *meaningful* collection of triples, which deal for example with a certain topic. For our purposed triples form a dataset if they are accessible by the same SPARQL endpoint.

Every dataset can be associated with general metadata like its name, its creator or a link to its homepage. We are especially interested in information about the access and the structure.

Access metadata is used to describe methods of accessing the actual RDF triples of the dataset. Since our goal is to rank SPARQL endpoints we use the *void:sparqlEndpoint*[4] property to establish a link between the dataset and the url of the SPARQL access. Furthermore, we can filter out all datasets that do not provide a SPARQL endpoint since they are not relevant for us.

[1] http://www.w3.org/TR/void/
[2] http://rdfs.org/ns/void#Dataset
[3] http://www.w3.org/TR/rdf-concepts/#section-rdf-graph
[4] http://vocab.deri.ie/void#sparqlEndpoint

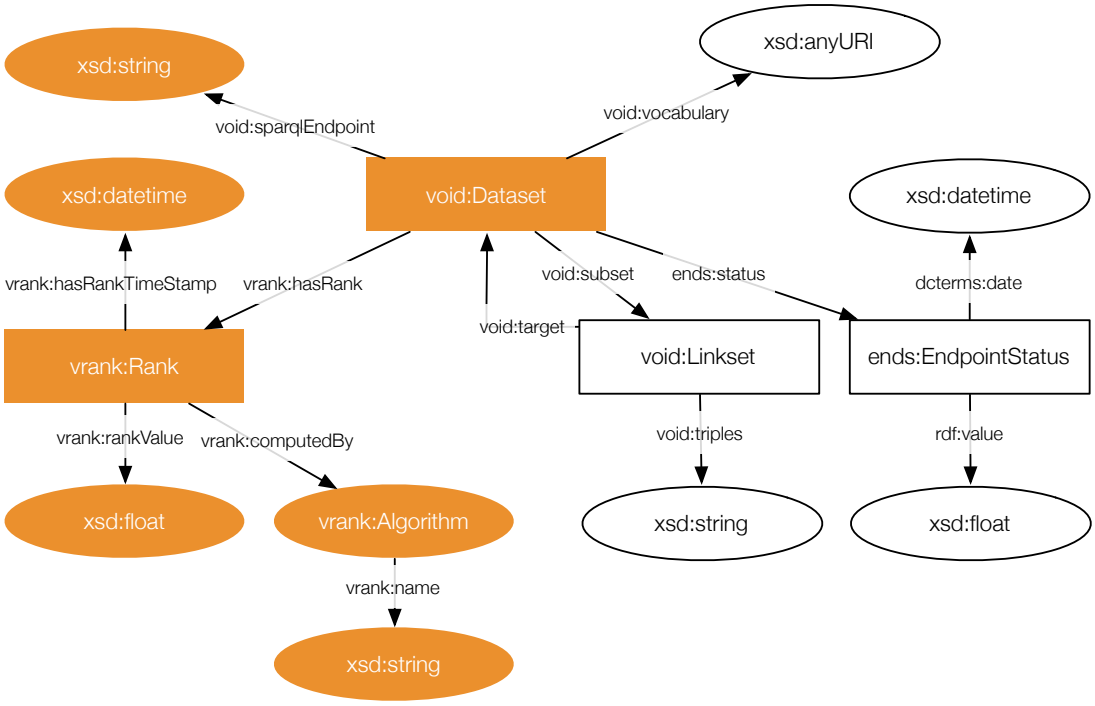

Figure 3.1: Used RDF Vocabulary.

Structural metadata provides high-level information about the schema and internal structure of a dataset. This includes statistics about the size of the dataset, examples of typical resources in the dataset and information about which vocabularies are used. The last point is of special interest for our tool, analysing the reputation of the used vocabularies given in the *void:vocabulary*[5] property.

Link metadata deals with the interlinking of datasets. The *void:Linkset*[6] class inherits from *void:Dataset*, hence all elements for describing datasets can be used for linksets as well. Linksets are collections of *RDF links* between two distinct datasets. An RDF link is an RDF triple whose subject and object are described in different datasets. The *void:target*[7] predicate is used to express the link.

A linkset allows to express two different viewpoints of directionality for RDF links: "Which dataset provides the subjects of the triples, and which the objects?" [Alexander et al., 2011] We can use the properties *void:subjectsTarget*[8] and *void:objectsTarget*[9], both inherit from the target property. This describes the subject-object direction of the links explicitly. Secondly, we can ask which dataset does contain the links. This is expressible by declaring the linkset a *void:susbet*[10] of the respective dataset.

For our link analysis we are mainly interested in describing outgoing links from one dataset

[5] http://vocab.deri.ie/void#vocabulary
[6] http://vocab.deri.ie/void#Linkset
[7] http://vocab.deri.ie/void#target
[8] http://vocab.deri.ie/void#subjectsTarget
[9] http://vocab.deri.ie/void#objectsTarget
[10] http://vocab.deri.ie/void#subset

to another. The link analysis discussed in section 5.2 uses relationships to estimate quality. We are able to represent this by using the subset notation. To further differentiate between the datasets providing the subjects and the objects is not relevant for our purposes. The statistics property *void:triples*[11] quantifies the relationship between two datasets based on the number of RDF links.

3.2 vRank Vocabulary

The *Vocabulary for Ranking (vRank)*[12] provides a standardised and formal way of representing ranking computations. Its structure allows to describe the result of rankings which are produced by different algorithms. The entity *vrank:Rank*[13] consists of the properties *vrank:rankValue* containing the value of the rank and *vrank:hasRankTimeStamp*[14] describing the time of the computation. It is connected with the analysed dataset using the *vrank:hasRank*[15] property. Every rank is associated with an instance of the *vrank:Algorithm*[16] class. [Roa-Valverde et al., 2012]

3.3 EndS Vocabulary

The *Endpoint Status (EndS)*[17] vocabulary is an extension for the VoID vocabulary. It defines specific elements to periodically monitor the availability of an endpoint. We are using a subset of the available vocabulary. A dataset is connected to multiple *ends:EndpointStatus*[18] classes. For our purposes the class contains two properties: *dcterms:date*[19] saves the time of the sample and *rdf:value*[20] the result value.

[11] http://vocab.deri.ie/void#triples
[12] http://vocab.sti2.at/vrank
[13] http://vocab.sti2.at/vrank#Rank
[14] http://vocab.sti2.at/vrank#hasRankTimeStamp
[15] http://vocab.sti2.at/vrank#hasRank
[16] http://vocab.sti2.at/vrank#Algorithm
[17] http://labs.mondeca.com/vocab/endpointStatus/
[18] http://labs.mondeca.com/vocab/endpointStatus#EndpointStatus
[19] http://dublincore.org/usage/terms/history/#dateT-001
[20] http://www.w3.org/TR/rdf-schema/#ch_value

4

Data Crawling

4.1 Introduction

In this chapter we discuss our approaches to crawl SPARQL endpoints and relevant information we can use for the quality assessment. As entry point we use catalogues collecting datasets. The most frequent used framework to build a catalogue is *CKAN*[1] and discussed in the next subsection. We distinguish two different methods to crawl datasets and metadata about them. We can use the catalogues to find links to raw VoID files created by the data providers or we can use the metadata given directly in the catalogues. The first method will be discussed in section 4.2 and the second one in 4.3.

4.1.1 CKAN Catalogues

The *Comprehensive Knowledge Archive Network (CKAN)* is an open-source data cataloguing platform. It is written and maintained by the *Open Knowledge Foundation (OKFN)*[2]. The software is a web-based system for the publishing and searching of data. CKAN powers a large number of data catalogues like the one of the public data of most European countries[3].

Rufus Pollock (co-founder and director of the OKFN) argues that componentization is a crucial factor for the distribution of data. This is the process of atomising given resources into smaller, more reusable packages. The software provides a platform for this package management and is intended to be the "apt-get of Debian for data" [Dietrich and Pollock, 2009].

The quantity and quality of the information available in CKAN records depends heavily on the catalogue using the software. It usually contains a description of the data, a list of formats and access points available, it states who is the provider, whether it is freely available and what subject areas the data is about.

The CKAN is not only reachable using a web browser, but the data is fully accessible through an API[4]. Further, there are code modules for the most frequently used programming languages, including Python. These provide convenient wrappers around many functionalities of the CKAN API.

[1] http://ckan.org
[2] http://okfn.org
[3] http://publicdata.eu
[4] http://docs.ckan.org/en/latest/api.html

4.2 Crawling VoID Descriptions

One attempt to crawl metadata is to use catalogues to find links to raw VoID files. These files can be parsed and the aggregated information relevant for the quality estimation can be saved in a central database. The implemented VoID discovery mechanism consists of two main steps:

1. The mechanism crawls catalogues to find as many links to VoID files as possible, regardless of whether they provide a SPARQL endpoint or not. These are saved temporarily in a central RDF database.

2. An indexer tries to connect the datasets and to create a directed graph. All VoID descriptions of datasets without a SPARQL endpoint and no influence to others providing one are removed from the graph. This decreases dramatically the amount of datasets to be investigated by the analysis algorithms.

The process is illustrated in figure 4.1.

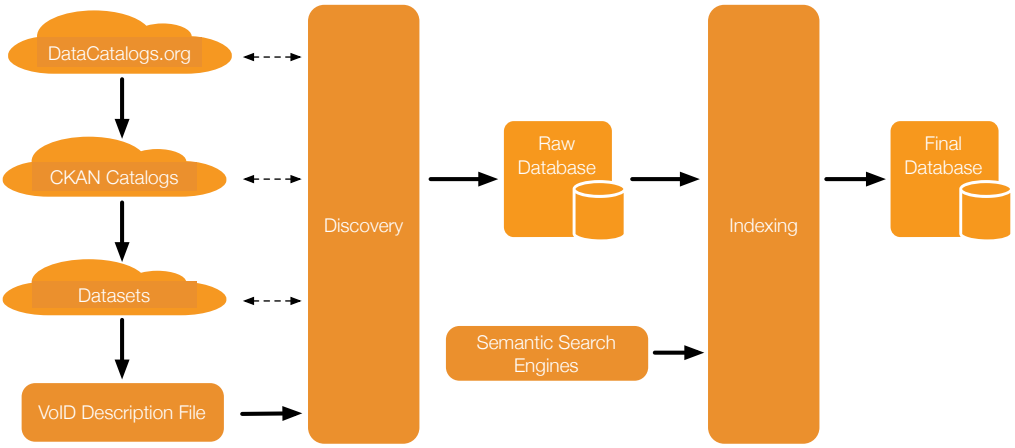

Figure 4.1: Architecture VoID Crawling and Indexing.

4.2.1 Discovery

We use the meta-catalogue DataCatalogs.org[5] as an entry point to find the VoID files published by the data provider itself. It lists currently 270 catalogues containing records to data providers. Amongst others there is a group of catalogues powered by the CKAN software - there are currently 36 available. DataCatalogs.org itself runs also on an CKAN instance. We are concentrating on catalogues using CKAN because the software allows to associate various kinds of resources with a *package*. A package in the CKAN language is the smallest entity in the catalogue. In our case the term package is identical to a dataset.

We retrieve every package in each of these catalogues looking for a URL to a VoID description file. We then copy the RDF triples of the VoID description into the intermediate triple store.

[5]http://datacatalogs.org

4.2.2 Indexing

The description of datasets using the VoID framework is a relative new possibility for expressing metadata about RDF datasets. The first version of the W3C note was published by its authors starting in 2008. For this reason, not for every dataset a corresponding VoID description is published by the data provider as well.

To enrich the quantity of the information gathered, we utilise semantic search engines which are indexing VoID descriptions. In the current configuration of the tool we are using *Sindice*[6] and the *VoID Store of rkbexplorer.com*[7]. Sindice is a general lookup index of resources crawled on the Semantic Web. It provides an interweaving framework for the decentralised Semantic Web. [Oren et al., 2008; Dietrich and Pollock, 2009] The VoID Store of rkbexplorer.com gathers VoID documents and stores them in a SPARQL repository. The available datasets are focused on VoID descriptions published in the RKBExplorer[8] domain. The explorer provides a framework for publishing and aggregating Linked Data, focused in the domain of scientific endeavour. [Glaser et al., 2009]

Of all available information, we are only interested in a VoID description of a dataset if it fulfils *one* of following criteria:

- the dataset provides a SPARQL endpoint
- an outgoing link of the datasets leads in a finite number of steps to a SPARQL endpoint

Hence we remove all datasets which do not provide a SPARQL endpoint and their outgoing links lead to a subgraph which consists only of datasets without a SPARQL endpoint. We keep the datasets without a SPARQL endpoint but with relevant connections to other datasets in this intermediate step to have a more complete graph for the link analysis in a later step.

Algorithm 1 illustrates the basic outline of the implemented algorithm in pseudocode. Programmatically we start with querying all available sources (raw VoID database, Sindice, VoID Store of RKBExplorer) to retrieve a distinct list of datasets which provide a SPARQL endpoint. From this point on, we use the url to the SPARQL endpoint as an unique identifier and subject for the RDF triple, rather than the subject of the original sources. This avoids inconsistency and duplicates because of different naming used in the different sources. We then follow recursively all incoming links of this datasets until we do not detect new datasets. In this step we consider also endpoints which do not provide SPARQL access, because their outgoing links to SPARQL datasets are relevant for the link analysis.

Listing 4.1 shows the query used to retrieve the datasets which link to a given one. The variable $\{dataset\}$ is given and replaced in the query. If our dataset is a dataset without a SPARQL endpoint we retrieve all datasets which have a *void:subset* which directly links to the given dataset using the *void:objectsTarget* or *void:target* property. Otherwise, we check the target links in the linkset using the *void:sparqlEndpoint* property. In the optional sections we collect the information we want to copy to the final database.

From the remaining datasets we copy only the properties which are relevant for our analysis algorithms. This is for example the list of used vocabularies and the subset properties. To normalise all link information we use always the *void:target* property, rather than the more specific *void:objectsTarget* or *void:subjectsTarget*.

At the moment of writing this thesis, there are overall 4203 distinct datasets registered in the three sources (raw VoID database, Sindice, VoID Store of RKBExplorer). After the elimination only 397 ($< 10\%$) relevant datasets remain to be investigated. This not very relevant given the current amount of datasets, but allows a better scaling in the future.

[6] http://sindice.com
[7] http://void.rkbexplorer.com
[8] http://www.rkbexplorer.com

Algorithm 1 Indexing SPARQL Endpoints.

```
for all sources do
    retrieve all SPARQL endpoints
end for
for all sparql_endpoints do
    CRAWL(sparql_endpoint)
end for
procedure CRAWL(dataset)
    if dataset is not yet crawled then
        for all sources do
            retrieve all incoming links from dataset
        end for
        for all incoming_links do
            CRAWL(incoming_link)
        end for
    end if
end procedure
```

Listing 4.1: SPARQL Query to Retrieve Incoming Links.

```
1  PREFIX void: <http://rdfs.org/ns/void#>
2  SELECT DISTINCT ?origin ?sparql
3  WHERE {
4      ?origin void:subset ?linkset .
5      { ?linkset void:objectsTarget | void:target <{dataset}> . }
6      UNION {
7          ?linkset void:objectsTarget|void:target ?target .
8          ?target void:sparqlEndpoint <{dataset}>  .
9      }
10     OPTIONAL { ?origin void:sparqlEndpoint ?sparql . }
11     OPTIONAL { ?origin void:vocabulary ?vocabulary . }
12 }
```

4.3 Metadata from Catalogues

To crawl the metadata directly from a catalogue we use the platform *Data Hub*[9], powered also by the CKAN software. It is an openly editable community-run catalogue listing public datasets. In contrast to other catalogues it provides an additional information: quantified links between the datasets. Part of this data is also the source for the *Linking Open Data cloud diagram*[10]. This image displays relationships between datasets with at least 1000 triples. We convert the information available on Data Hub into the RDF vocabulary plotted in figure 3.1.

4.4 Comparison

A deeper analysis of the different crawling approaches is shown in the table 4.1. Crawling the VoID description as described in the previous chapter has not yet a very high coverage. Using the

[9] http://datahub.io/
[10] http://richard.cyganiak.de/2007/10/lod/

	VoID	VoID + search engines	Data Hub
Nr of datasets	131	397	4635
Nr of SPARQL endpoints	86	278	446
Nr of links	552	1279	14173
Nr of SPARQL endpoints with incoming links	48 (56%)	59 (21%)	164 (37%)
Nr of SPARQL endpoints with outgoing links	12 (14%)	120 (43%)	291 (65%)
Avg outgoing links / SPARQL endpoint	0.24	1.47	3.37

Table 4.1: Data Sources Comparison.

raw VoID files without extending it with the data of semantic search engines the crawler finds 86 SPARQL endpoints. Adding the searching engines we can increase it to 278. Data Hub however lists 446. More important for the link analysis discussed in the next chapter is the number of SPARQL endpoints which have been linked at least *once*. The algorithm used can only rank this subset of the endpoints. In the raw VoID files this a set of only 48 entries. In the Data Hub data there are at least 164 SPARQL endpoints with an incoming link.

Using the metadata collected by the catalogues leads to the questioning which sources they take. For example how does the community of Data Hub define which dataset links to another and how does it estimate the quantity of the RDF links? The platform just allows you to enter the relationships in plain text. It does not include a mechanism to crawl the links. So it is up to the contributor to do this analysis. The problem is that the metrics to calculate or estimate the relationships are not standardised. Every user can apply its own algorithm. The question is if the power of the crowd can flatten these inconsistencies.

The community-based approach has several positive properties. Firstly, the quantity of described links is very high in comparison to the VoID descriptions. Secondly, the misleading values are controlled by the community, hence it is more difficult for the data provider to manipulate the values. The system has a revision feature which allows to track all changes made to a dataset. Looking at the users list[11] there are currently 36680 users registered. 8656 (24%) have made at least one edit to a dataset, 1851 (5%) have made two ore more edits. The Data Hub publishes some statistics[12] of their datasets and the usage. Figure 4.2 illustrates that the total number of datasets increases exponentially. Figure 4.3 shows that the effort put into the creation of new datasets and the one put into updating existing ones balance each other approximately.

[11] http://thedatahub.org/user
[12] http://thedatahub.org/stats

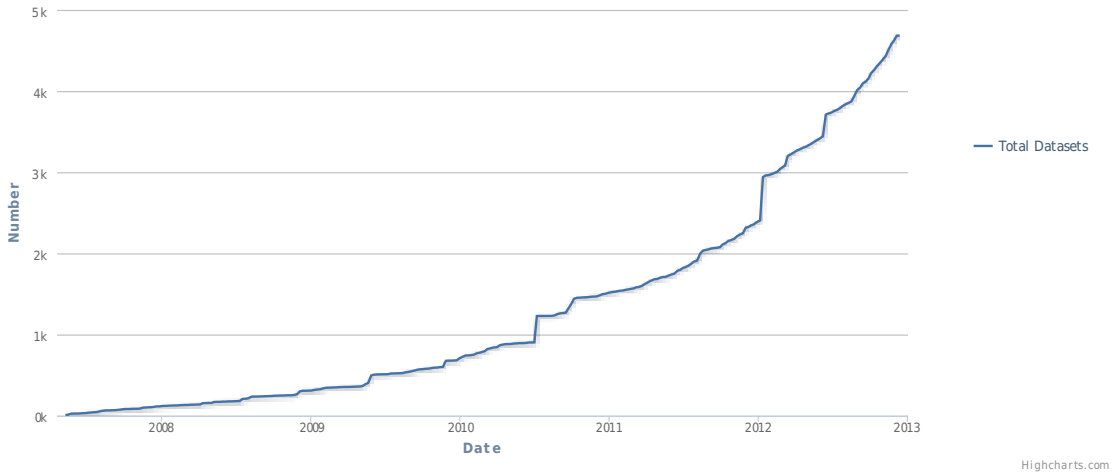

Figure 4.2: Data Hub Statistics: Number of Datasets.

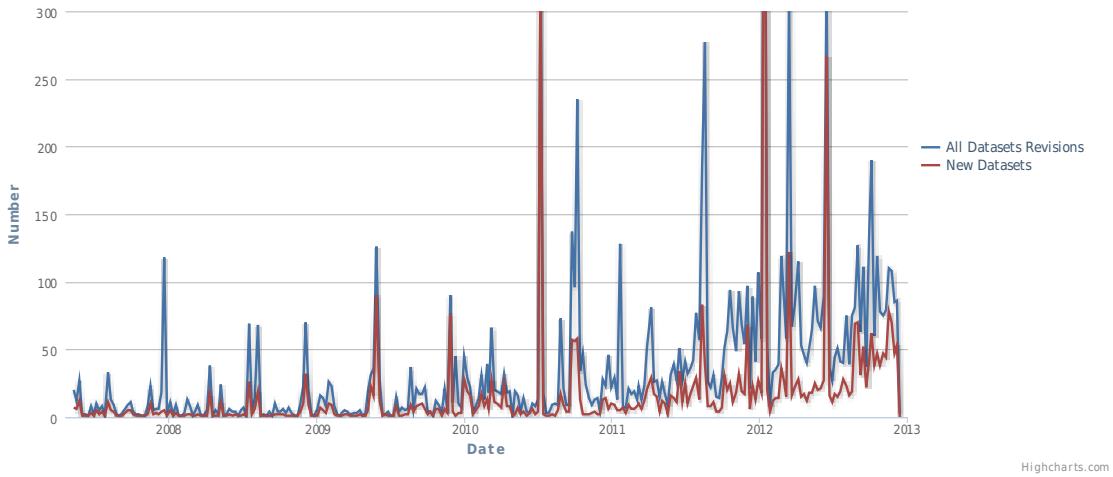

Figure 4.3: Data Hub Statistics: Revisions.

5

Data Analysis

Based on the formal and high-level description of a dataset's content and interlinking provided by the vocabulary defined in section 3.1, the tool should rank datasets in an efficient way.

5.1 Introduction

We are using only algorithms which do not require manually-created categorisations. Our mechanism should work without any maintenance effort. This includes for example a manual grading of all currently available license types. As the licenses are not fixed and new ones can be created in the future, without a constant maintenance effort there is a possibility of unmapped licenses.

All of the algorithms that will be discussed generate ranks in the range from 0 to 1. However the algorithms do not always use the entire bandwidth. Hence the maximum rank value can be much smaller than 1 or the smallest value can be higher than 0. To avoid this, we add a normalisation step after calculating the ranks to transform the distribution in such a way that the highest rank value equals 1 and the lowest 0. Mathematically this is computed as shown in equation 5.1.

$$R_i = R_i \times \frac{1}{max(R) - min(R)} \tag{5.1}$$

5.2 Link Analysis

5.2.1 Transitive Trust Mechanism

Using the *void:Linkset* class and *void:subset* property we can build a graph with the datasets as nodes and the directed links as edges. Our goal is to calculate a rank for every dataset in order to estimate how *trustful* its quality is. We assume that the network of datasets fulfils the *transitivity* property. Transitivity means that if node i trusts node j and node j trusts node k, then node i also trusts node k to some degree. [Seuken and Parkes, 2012]

Transitive-trust algorithms are very widely used for estimating the quality of a website. The most famous link analysis algorithms which have been successfully applied to measure the influence of a website are *Hyperlink-Induced Topic Search (HITS)* [Kleinberg et al., 1999] and *PageRank* [Page et al., 1999]. HITS which is also known as Hubs and Authorities algorithm is the forerunner

of PageRank. It distinguishes two different types of nodes in the web graph: *hubs* are nodes which point to many other nodes and *authorities* represent pages that are linked by many different hubs. The algorithm assigns this two scores to every node. The authority score stands for the quality of the content of the page and the hub value for the value of its links to other pages. This bipartite graph structure is not possible or reasonable to calculate for every domain.

The PageRank algorithm does not separate the graph into hubs and authorities, but calculates one single rank for every node. It is named after Larry Page, one of the founders of Google and was invented in 1998. It was the core of Google's webpage ranking method. It starts with counting the number of incoming links and then it applies the principle of repeated improvement. The current PageRank score of a node corresponds to the current trustfulness of the node. In every iteration it passes its rank forward to other nodes via its outgoing links. Hence a node with a high rank can endorse other nodes more strongly. A one of the problems with this basic definition of the algorithm is that nodes without outgoing links ultimately end up with all of the PageRank. We can avoid this by introducing a *damping factor*. This is the probability that the user will jump to a random node by directly accessing it, rather than by following a link. There are various studies testing different damping factors, but the original suggestion of 0.85 by Brin and Page is the most used one.

Equation 5.2 captures the calculation of the PageRank for one iteration. The PageRank i is the sum of the probability d that a user calls a given node either by directly addressing it or by following one of the links pointing to it. The second probability is dependent on the PageRank values for each page j contained in the set I_i consisting of all pages linking to page i, divided by the number of outgoing links $freq(j)$ of page j. Initialising all nodes with a PageRank value of the inverse of the number of all nodes, the overall PageRank sums always to 1. It is a property of the PageRank algorithm that the ranks of the nodes converges after a finite step of iterations.

$$\begin{aligned} PR(i) &= PR_{direct}(i) + PR_{link}(i) \\ PR_{direct}(i) &= (1-d) \\ PR_{link}(i) &= d(\textstyle\sum_{j \in I_i} \dfrac{PR(j)}{freq(j)}) \end{aligned} \tag{5.2}$$

5.2.2 Related Works

DING! [Ding et al., 2005] has a very similar setup to ours. The authors describe a ranking method using as data VoID descriptions and a modified PageRank algorithm. In contrast to our implementation, they use the PageRank version without the damping factor. VoID offers with the *void:linkPredicate*[1] predicate a possibility to describe the type of the link between two datasets. DING! presents an algorithm based on *TF-IDF*, a well known algorithm from the information retrieval domain. They use the principle of the *Inverse Document Frequency (IDF)* to calculate the weight of a link predicate. If a link predicate appears in many VoID descriptions its weight is less than a predicate appearing in a fews only. In our opinion, there is no correlation between the frequency of a link predicate and the weight to be associated with it. IDF is useful for a classic information retrieval task. If a query consists of several terms it is reasonable to weight a rarer and hence more specific term higher than a very general one. In the PageRank algorithm, however, the weight represents the probability of a user following the link. From our point of view, this is not dependent on how often the link predicate is used in other relationships, rather than how strong the type of the connection is.

The *Swoogle* search engine[2] proposes with *OntoRank* [Ding et al., 2004; Finin et al., 2005] an

[1] http://vocab.deri.ie/void#linkPredicate
[2] http://swoogle.umbc.edu

adaption of PageRank for semantic web resources. They differentiate three levels of granularity in their computation: documents, terms and RDF graphs. Our goal to rank datasets using VoID descriptions operates on a coarser granularity. They distinguish between different types of relations among semantic web resources. We could create such a classification also for classes and properties used in the link property of a linkset. However, the set of possible values is not restricted. Without continuous human administration effort this would lead to values which are not associated with a corresponding weight. Because of a missing machine learning algorithms with reasonable results, we do not consider the link predicates to calculate the weights for a link.

Spring [Mulay and Kumar, 2011] uses also link analysis algorithms to rank the results of SPARQL queries. In addition to the links between datasets they also take inter-dataset links into consideration.

5.2.3 Implementation

The adaption from the ranking web documents to datasets in the semantic web is done in the following way. The web pages are now datasets defined by *void:dataset*. A hyperlink corresponds to linksets (*void:Linkset*) which are connected declaring it as subset (*void:subset*) of a dataset.

In addition to a web page link we have quantified links available. VoID supports to describe how many triples are linked. In practice almost all descriptions contain this information. This allows to modify the original PageRank algorithm to a weighted one. In our implementation we use a logarithmic distribution of the number of triples, rather than a linear one. The idea is to rank datasets with incoming links from many other datasets higher than datasets with a lot of incoming links from few datasets. Equation 5.3 shows an example of this constraint. $L_{u \to v}$ stands for a link from dataset u to v. The $freq$ function stands for the number of links from u to v.

$$\begin{aligned} freq(L_{1\to3}) &= 500 \\ freq(L_{2\to3}) &= 500 \\ freq(L_{1\to4}) &= 1000 \\ R(3) &\overset{!}{>} R(4) \end{aligned} \tag{5.3}$$

The natural logarithms discriminates links with a high quantity of triples very hard. For this reason we decided to take the logarithms to the base of 2. This leads to the final algorithm listed in 5.4. O_j represents the set of outgoing links from node j. $w(L_{u \to v})$ is the value of the link from node u to v after applying the weighting function.

$$\begin{aligned} R(i) &= (1-d) + d(\sum_{j \in I_i} \frac{R(j) \times w(L_{j\to i})}{\sum_{k \in O_j} w(L_{j\to k})}) \\ w(L_{u\to v}) &= \log_2 freq(L_{u\to v}) \end{aligned} \tag{5.4}$$

The resulting rank results in a power law distribution. This does not surprise because studies have shown that the PageRank distribution of the web graph follows a power law of exponent 2.1. [Pandurangan et al., 2002] We are not only interested in the ranking order of the dataset but also in the rank score. To have a more linear distribution of the final ranks we are taking the logarithm of the calculated rank as shown in 5.5. The constant α defines how strong the flattening impact should be.

$$R(i) = log(1 + (\alpha \times R(i))) \times \frac{1}{log(1 + (\alpha \times max(R)))} \tag{5.5}$$

5.2.4 Strategic Manipulations

We can consider two different kinds of robustness to manipulation. The first is *rank-strategyproofness*. This means in our case that a data provider cannot increase the dataset's *relative position* in the rank order with respect to the view of any other dataset. The second one is *value-strategyproofness*. This means that a data provider cannot increase the dataset's *ranking score* from the perspective of any other dataset. Our adapted PageRank algorithm is neither value-strategyproof nor rank-strategyproof. By misreporting a data provider (for example by not declaring some outgoing links), it can reduce the rank-position and rank-score of another dataset. Creating multiple sybils linked to the original dataset is another manipulation method. The sybil link to each other and form a star. So it is possible to keep the rank in this subgraph, once the random walk reaches the dataset. This leads to a higher rank for the original dataset. However, this robustness perspective is only one side. Another important property is the *informativeness* of a trust algorithm. It is easy to construct a robust mechanism like for example a random mechanism. However, the informativeness of this example is very low. There is a trade-off we need to make when designing and choosing trust algorithms. Informative mechanism tends to be less robust and otherwise around. [Seuken and Parkes, 2012]

How manipulable the algorithm is depends also what input we choose to build the graph. Using the raw VoID descriptions of the data providers is the easiest to manipulate. The data provider can create fake datasets and leave away other outgoing links. Using the data from community-based sources like the Data Hub we can use the wisdom of the crowd to control the declarations to some extent.

5.2.5 Ranking Result

Distribution

The line chart in figure 5.1 plots the distribution of the rank value versus the rank position. This plot uses the normalised rank values. As illustrated in section 4.4, the Data Hub provides much more detailed relationship information than the crawling of raw VoID files. So it is not a surprise that the outcomes in this ranking algorithm are more facetted using the Data Hub source.

Looking at the *Data Hub* data we can observe that there is one dataset with a lot of incoming links (Dbpedia.org[3]), outperforming the second one with a high difference. The rank value decreases then with increasing rank position relatively constantly until rank position 151. For the next 12 datasets the rank value remains the same. These SPARQL endpoints are all referenced by one other dataset. After rank 164 the datasets do not have any incoming links at all. Overall we do have 145 distinct rank values for totally 446 SPARQL endpoints - this is 33%.

The source of the raw *VoID* files and semantic search engines a rate only the first 59 datasets. There are 52 distinct ranking scores for totally 278 endpoints, this is 19%.

Rank Correlation

It is interesting to find out whether the ranks calculated using the VoID files and the ranks using the Data Hub catalogue lead to the same result. Rank correlation metrics measure the degree of similarity between two rankings comparing the relative ordering. The rank correlation coefficient is defined inside the interval $[-1, 1]$. A value of 1 indicates a perfect agreement, 0 complete independence and -1 perfect disagreement. Two popular metrics are *Spearman's rho* and *Kendall's Tau*. The first coefficient takes the sum of the squares of the differences for each pair of rankings. The

[3] http://dbpedia.org

Rank 1	Rank 2	Spearman's Rho	Kendall's Tau	Nr. Datasets
Data Hub	VoID Files	0.451	0.321	38
Data Hub	Google's PageRank	0.054	0.050	36
Data Hub	Alexa's Rank	0.038	0.029	159

Table 5.1: Link Analysis: Rank Correlations.

second looks for all ordered pairs if the difference of the rank positions have the same sign. [Hill and Lewicki, 2006]

Table 5.1 displays the result of the calculations. We first compare the ranks using the Data Hub and the VoID files crawling methods. The intersection of the two sources contains only 38 datasets, although a Spearman's rho of 0.451 and a Kendall's Tau of 0.321 indicate moderate correlation of the rankings for this subset. Even tough the link relationships are coming from two complete different sources, this result implies that the link analysis of the VoID files and Data Hub metadata leads to similar ranks.

We further measure the similarity of the Data Hub ranks with two well-known rankings for web pages. We checked *Google's PageRank*[4] and *Alexa's Rank*[5] of the URL linking to the SPARQL endpoint of the dataset. Surprisingly, with correlation ranks smaller than 0.06 our rankings using the RDF links of the datasets do not correlate at all with the rankings of Google and Alexa for the website hosting the SPARQL endpoints. The ranks calculated by Google are no longer based only on the calculated PageRank value of the webpage alone, but also on many other metrics. Nevertheless, the results indicate that the directed graph of the webpages and the one of RDF links differ fundamentally.

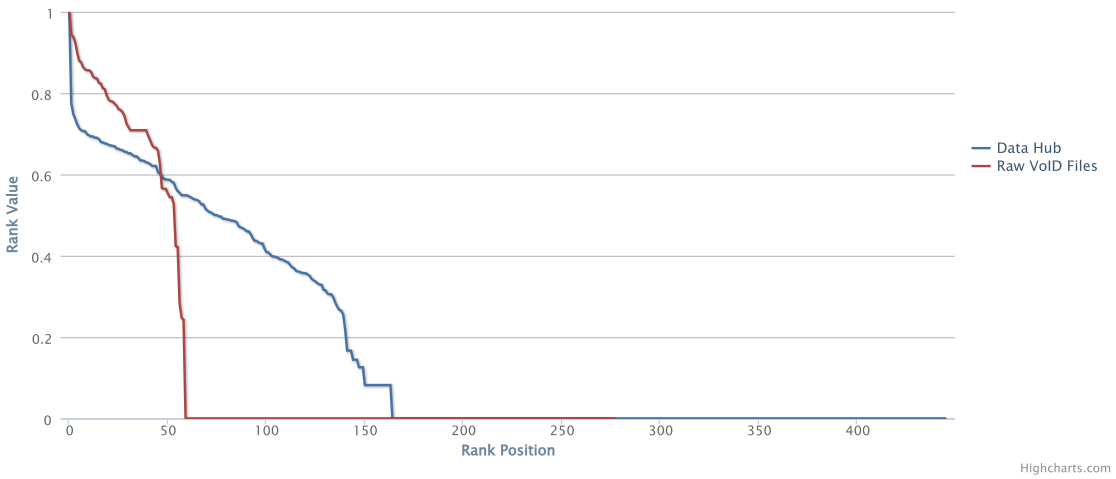

Figure 5.1: Link Analysis: Distribution of Rank Values.

[4]http://www.google.com/competition/howgooglesearchworks.html
[5]http://www.alexa.com

5.3 Availability & SPARQL Support

5.3.1 Introduction

The degree of availability is crucial for the quality of a SPARQL endpoint. If the server is down at the moment of the execution of a query, it is completely useless. Another important aspect is how well the endpoint implements the SPARQL language constructs. The first *W3C Working Draft* of *SPARQL 1.1* was released in October 2009 and the last version was published in November 2012[6]. This illustrates that the new version 1.1 is still very new. Many of the used SPARQL engines do not yet support all features.

5.3.2 Related Works

Mondeca[7] is a French software company. Their core products are located in the semantic web domain. Their tool *SPARQL Endpoints Status*[8] monitors SPARQL endpoint availability. As source, they use the CKAN repository *Data Hub* as we did to crawl datasets. It tests the server accessibility and response time every hour and publishes the results on the web. All data generated is also available through it SPARQL endpoint[9]. In order to express the collected data they extend the VoID vocabulary and define specific elements for availability status definition. The vocabulary is published under the name *Endpoint Status Ontology (EndS)*[10] and is described in section 3.3.

5.3.3 Implementation

Tests made by us have shown that calling an unavailable SPARQL endpoint often returns a success HTTP status code of *2XX*, rather than an error like *4XX* or *5XX*. This is often caused by a wrongly configured web proxy server of the data provider. For this reason, we do no just ping the server to check if the SPARQL endpoint functionalities are available, but request always real queries. This is a crucial difference with the implementation of the Mondeca service. Mondeca expresses the availability as a binary function - available if the server answers positively and offline if a time-out occurs or an error returns. We are defining a set of SPARQL queries which, in addition to the availability, test also the capabilities of a SPARQL endpoint. If the endpoint does not answer for all queries, we know that it is offline; if it answers 1 of 10 queries, we know that it is available, but it supports only a few of the SPARQL functions. This allows a more finely graded assessment of the endpoints. The queries are defined as independent from each other as possible to avoid interferences in the results. This is not possible for all keywords, basic constructs like *LIMIT* are used in several queries. Table 5.2 lists the topic and SPARQL constructs tested in the currently 14 queries. The queries are formulated in a way that the number of results is constant independent of the SPARQL endpoint to be checked. It is important to avoid queries which possibly return an empty result set, because some SPARQL endpoint also return an empty set when they cannot handle the query. This requires a careful design of the queries. In our implementation the only generality constraint is that we assume that the dataset has at least one entry.

[6] http://www.w3.org/TR/2012/PR-sparql11-query-20121108/
[7] http://mondeca.com
[8] http://labs.mondeca.com/sparqlEndpointsStatus/index.html
[9] http://labs.mondeca.com/endpoint/ends
[10] http://labs.mondeca.com/vocab/endpointStatus/index.html

Description	SPARQL Constructs
Retrieve one triple	*SELECT, WHERE, LIMIT*
Eliminate duplicate results	*DISTINCT*
Assign value to variable, condition	*BIND, IF*
Concat lexical values	*CONCAT*
Filter results with regex	*FILTER, REGEX*
Set operations	*UNION, MINUS*
Alternative property path	|
Sequence property path	/
Count aggregation	*COUNT*
Complex aggregation	*COUNT, GROUP BY, HAVING*
Sort the results	*ORDER BY*
Subqueries	*SELECT* {*SELECT*}
String manipulations	*UCASE, SUBSTR*
Type testing	*isBlank, isLiteral, isNumeric, isIri, isUri*

Table 5.2: Query Testing Set.

Ranking One Sample

For every query in the set we get a boolean value if the request was successful or not. We combine the availability and SPARQL functionality check in *one* ranking. Equation 5.6 shows how we calculate the rank for one sample. It consists of two different ranking functions: the first one evaluates the response latency and the second one, how many queries are answered correctly. w is a constant that defines the weight how much the response time of a query is considered. In the current implementation we set a value of 0.1. We attach more importance that the query is answered, the influence of the speed performance is small. τ_{time_out} stands for the maximum time within which the query has to be answered by the data provider for it to be considered as successful. The functionality rank is simply calculated by taking the percentage of answered queries $n_{successes}$ and total number of queries tested $n_{total\_queries}$.

$$\begin{gathered} Y = w * \frac{\tau_{time\_out} - avg(\tau_{response\_time})}{time\_out} + (1-w)\frac{n_{successes}}{n_{total\_queries}}, \\ w \in [0,1], n_{total\_queries} > 0, avg(\tau_{response\_time}) > 0, \tau_{time\_out} > 0 \end{gathered} \tag{5.6}$$

Ranking Multiple Samples

These samples are collected continuously. Our final rank considers all data samples over the whole time period. The more up-to-date of the collected samples should be given more weight than the old samples. It is more important how the SPARQL endpoint performs at the current time than in the past. Equation 5.7 illustrates this constraint. $Y_t(i)$ stands for the rank sampled at time t for the SPARQL endpoint i. $R(i)$ is the final rank for endpoint i.

$$\begin{gathered} Y_2(1) = 1,\ Y_1(1) = 0.5,\ Y_0(1) = 0.5 \\ Y_2(2) = 0.5,\ Y_1(2) = 1,\ Y_0(2) = 0.5 \\ R(1) \overset{!}{>} R(2) \end{gathered} \tag{5.7}$$

The *exponential moving average (EMA)*[11] applies weighting factors to the samples which decrease exponentially. Equation 5.8 displays the basic formula to calculate the average. n stands for the number of available samples - if we have only one the average is the value itself. a is a coefficient to represent the degree of weighting decrease. If the value is higher, older samples are discounted more quickly. Y_t stands for the value of the sample at time t and R_t is the exponential moving average at time t.

$$R = \begin{cases} Y_0 & \text{if } n = 1 \\ \sum_{t=1}^{t=n-1} \alpha \times Y_t + (1-\alpha) \times R_{t-1} & \text{otherwise} \end{cases}, \quad a \in [0,1], Y_t \in [0,1], R_t \in [0,1], n > 0 \tag{5.8}$$

We can transform the equation to isolate the coefficient a as shown in equation 5.9. This allows us to code the algorithm using the $+=$ operator. With this formula we have implemented the initial constraint given in 5.7.

$$R = \begin{cases} Y_0 & \text{if } n = 1 \\ \sum_{t=1}^{t=n-1} R_{t-1} + \alpha(Y_t - R_{t-1}) & \text{otherwise} \end{cases}, \quad a \in [0,1], Y_t \in [0,1], R_t \in [0,1], n > 0 \tag{5.9}$$

The exponential moving average with a constant coefficient a has one drawback. It discounts the samples by the number of samples, but not by time. If the samples are collected at varying times the average can be distorted. Equation 5.10 shows this additional constraint. The SPARQL endpoint 1 is sampled at time $0, 2, 3$, where at time 2 it has a lower rank. Endpoint 2 is sampled at time $0, 1, 3$, and its lower rank is at time 1. Because the bad rank of endpoint 1 is more current, it should have more weight. Hence the final average R should be lower for endpoint 1.

$$\begin{aligned} Y_3(1) = 1,\ Y_2(1) = 0.5,\ Y_0(1) = 1 \\ Y_3(2) = 1,\ Y_1(2) = 0.5,\ Y_0(2) = 1 \\ R(2) \overset{!}{>} R(1) \end{aligned} \tag{5.10}$$

We can achieve this by replacing the constant coefficient with a time varying one as shown in equation 5.11. This the final algorithm implemented in our tool. τ is the overall time period from the first sample till the last. Δ represents the time difference in regard to the previous sample.

$$R = \begin{cases} Y_0 & \text{if } n = 1 \\ \sum_{t=1}^{t=n-1} R_{t-1} + e^{-\Delta/\tau}(Y_t - R_{t-1}) & \text{otherwise} \end{cases}, \quad \Delta > 0, \tau > 0, Y_t \in [0,1], R_t \in [0,1], n > 0 \tag{5.11}$$

5.3.4 Strategic Manipulations

For the availability of the server there is no manipulation possible other than to have a reliable server. Of course the queries requested by our mechanism can be analysed and the SPARQL endpoint could send just the correct amount of results back rather than the correct answers, but this is rather improbable.

[11] http://lorien.ncl.ac.uk/ming/filter/filewma.htm

5.3.5 Ranking Result

After one iteration we get a resulting ranking distribution as shown in figure 5.2. The ranking values form a staircase-shaped function. SPARQL endpoints coming from the same domain and institution are often grouped together and have very similar ranks. We are guessing that they use the same SPARQL engine. The small differences result from the consideration of the response time which of course varies always in a certain degree. The constant rank value around 0.27 is so because there are a lot of SPARQL endpoints hosted at RKBExplorer[12].

It is rather surprising that the rank has many different values. There is not a binary behaviour like engine supports SPARQL 1.1 or not. The degree of the support of the SPARQL functionalities are manifold.

Using the Data Hub source only 217 of total 446 SPARQL endpoints are available. This is only 48.7%. In VoID file source there are 144 of 278 available, which is 51.8%. This illustrates that the availability check is very important. The reliability and life-time of a SPARQL endpoint is not comparable with other web services. They are often built for research and testing purposes, rather than to provide a constant service. With more samples the information will be more detailed. A continuously refreshed web interface is discussed in section 6.5.

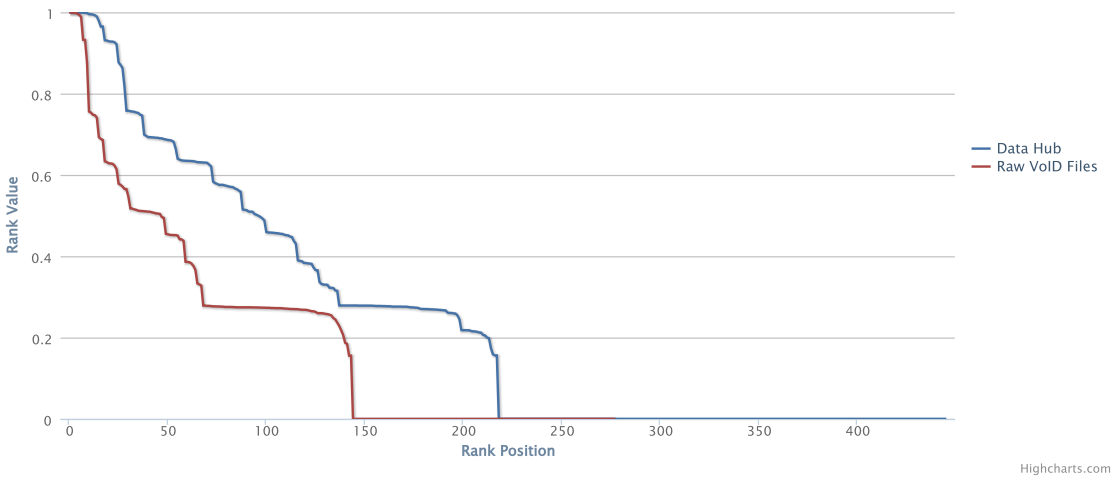

Figure 5.2: Availability & SPARQL Support: Distribution of Rank Values.

5.4 Used RDF Vocabulary

5.4.1 Introduction

The Linked Data project attemps to connect semantic related data that was not previously linked in the heterogenous and unstructured Web. The basic idea of the Linked Data is not only to enable clients to discover new data sources by following RDF links, but also to help them to integrate data from these sources. One approach to dealing with heterogeneous data representation is to advocate the reusage of terms from widely used vocabularies. There exists a set of vocabularies which has emerged in the Linked Data community. The more these vocabularies are used, the

[12]http://www.rkbexplorer.com

more the data are structured and consistent. Heath and Bizer [Heath and Bizer, 2011] propose the following workflow: First search for terms from widely used vocabularies that could be reused. Create only a new vocabulary if no existing one fits the requirements. Wherever possible, the data provider should publish the new vocabulary and seek wider adoption for it from others with related data.

With VoID descriptions for a dataset, the *void:vocabulary* property can be used to list vocabularies contained in a dataset. We investigate this list for every dataset using a similar approach to the *term frequency - inverse document frequency (tf–idf)*, a well-known method coming from the information retrieval area. [Baeza-Yates and Ribeiro-Neto, 1999]

5.4.2 Related Works

To the best of our knowledge, there is no other work which ranks datasets using as data input the used vocabularies.

5.4.3 Implementation

The tf-idf method is often used as a weighting factor to reflect how important a word is for a document. Its value increased proportionally to the number of times a word appears in the document, but it is offset by the frequency of the word in the corpus. The *inverse document frequency* assumes that a word has less value if it is very common. For our case we assume exactly the opposite. In our implementation, often-used vocabularies should have a higher weight than proprietary ones.

We define the *dataset frequency* df_v of a vocabulary v as the fraction of the number of documents in which the vocabulary is used f_v and the total number of documents N_D as shown in 5.12. At best, when all documents use the vocabulary this value is 1. At worst, it is the reciprocal of the number of documents. This is still strictly greater than 0. We use this distribution to encourage the declaration of vocabularies. A declared proprietary vocabulary is still better than declaring no vocabulary at all.

$$\begin{aligned} df_v &= \left\{ \frac{f_v}{N_D} \right. , \\ N_D &\geq f_v > 0 \end{aligned} \tag{5.12}$$

The distribution of the rank should not be linear. The difference between a vocabulary used by 1 or by 2 of 10 documents should be much higher than the difference between one used by 9 or 10 documents. This leads to the *logarithmic dataset frequency* ldf_v as shown in equation 5.13. The constant α is used to adjust the importance of changes in the lower range of the ranks even more.

$$\begin{aligned} ldf_v = log(1 + \alpha \times df_v), \\ df_v \in [0, 1] \end{aligned} \tag{5.13}$$

The *maximum logarithmic dataset frequency* $mldf$ represents the transformed value of the highest rank, if all documents implement a vocabulary.5.14

$$mldf = log(1 + \alpha \times 1) \tag{5.14}$$

This leads to a distribution of the ranks as plotted in figure 5.3. The *x-axis* represents the input rank as calculated in df_v and the *y-axis* is the transformed value. Analog to the *term frequency*

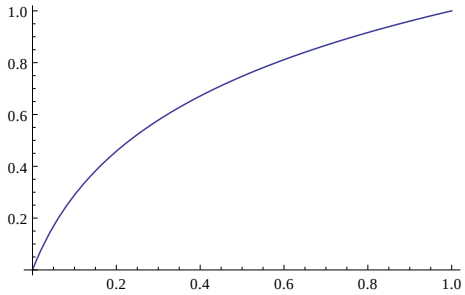

Figure 5.3: Used Vocabulary: Distribution of the ldf_v Function.

in the information retrieval we define the *vocabulary frequency* $vf_{v,d}$. In our case this is a trivial function: if a vocabulary v occurs in a dataset d the weight is 1 otherwise 0.

$$vf_{v,d} = \begin{cases} 1 & \text{if } v \in d \\ 0 & \text{otherwise} \end{cases} \tag{5.15}$$

The *vocabulary frequency - dataset frequency* for a document d is calculated by iterating through all vocabularies and getting the product of vf_v, d and ldf_v.

$$vfdf_d = \sum_{v \in V} vf_{v,d} \times ldf_v \tag{5.16}$$

Using our binary *vocabulary frequency* function this can be simplified to the equation 5.17.

$$vfdf_d = \sum_{v \in V_d} ldf_v \tag{5.17}$$

Because our rank should be normalised in the range from 0 to 1 we divide it through the maximum possible rank for the given number of vocabularies. This leads to the final rank algorithm shown in equation 5.18.

$$R_d = \frac{\sum_{v \in V_d} ldf_v}{\sum_{v \in V_d} mldf}, \tag{5.18}$$

5.4.4 Strategic Manipulations

Using the declarations of the raw VoID files this algorithm is not very robust. Proprietary vocabularies which reduce the rank of the dataset can be ignored and just one frequent vocabulary can be declared. However, if the data is generated by analysing the used vocabularies at RDF triple level, the manipulations are much more difficult. The only way to increase the rank is to reduce the proprietary elements and to increase the amount of frequently used vocabularies. This leads to the desired higher connectivity.

In our implementation we do not use self-generated statistics about the vocabularies. For this reason we do not consider this algorithm in our configuration of the tool and should be considered as a theoretical idea.

5.4.5 Ranking Result

The Data Hub source does not contain information about the vocabularies used. Therefore we have only data from the VoID descriptions. 135 (49%) datasets of total 278 have declared at least

one vocabulary used and can be ranked. All of the first 52 (19%) datasets declare vocabularies used with the same frequency. In the most cases this is the *void* vocabulary.

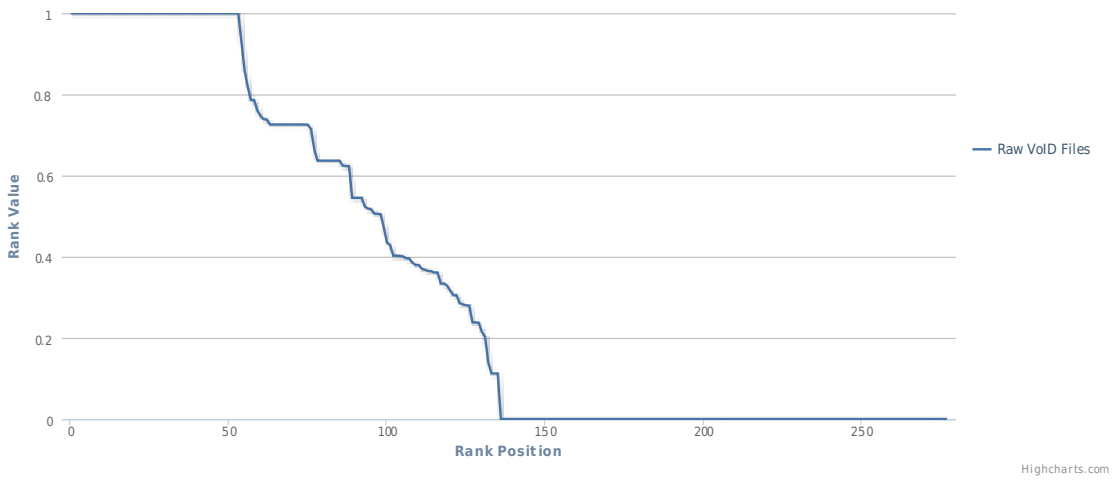

Figure 5.4: Used Vocabulary: Distribution of Rank Values.

5.5 Overall Rank

5.5.1 Introduction

The ranks calculated by the algorithms discussed before have to be aggregated to one single rank to be able to order the datasets.

5.5.2 Implementation

With a weighted average we are using a straightforward solution. Every rank is associated with a weight, all summing to 1 in total. However there is one exception. If the rank of the *Availability & SPARQL Support* analysis is zero, the overall rank should be zero as well, regardless of the result of the other algorithms. This means that the SPARQL endpoint has never been available or could never answer the most trivial SPARQL test query. In this case the endpoint is not useful and should be ranked with the minimal rank.

5.5.3 Ranking Results

A sample distribution of the ranks used in the final configuration of the tool is displayed in figure 5.5. It bases on the Data Hub source. The distribution of the values is similar. Because the overall rank is set to 0 when the SPARQL endpoint have been down for every sample, its null point is at the same point as the SPARQL Support rank.

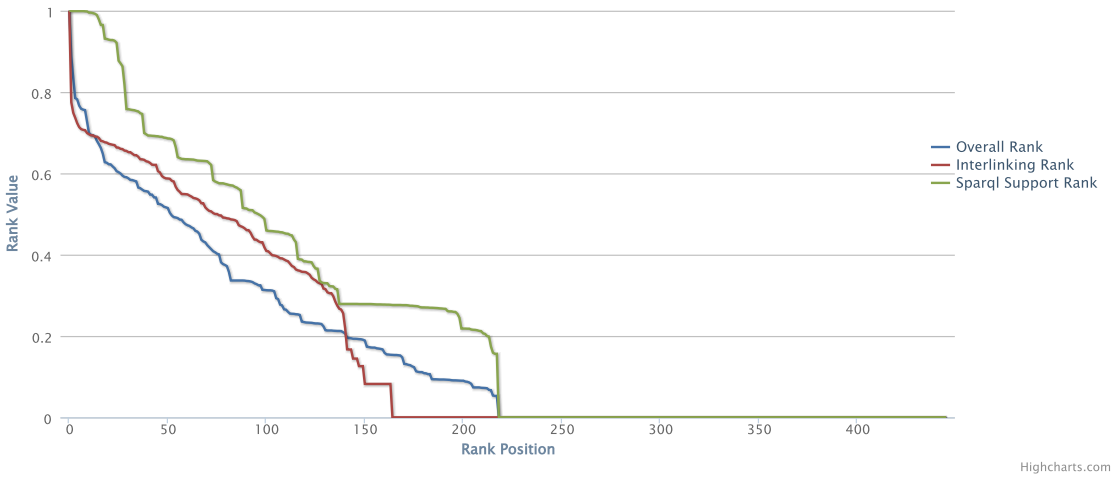

Figure 5.5: Overall Rank: Distribution of Rank Values.

6

Implementation

6.1 Architecture

In this chapter we describe some key aspects of the implementation of the tool. The program is coded in Python 2.7. All used libraries are available as open-source software and declared in a requirements file for an easy installation using the *Python Package Index (PyPI)*[1]. As RDF repository we use *Sesame2*[2] which provide also the SPARQL endpoint. Every other SPARQL engine which support *SPARQL UPDATE* statements should work as well.

The core of QEPSM consists of three main submodules: *crawling*, *analysis* and *retrieval*. Further the submodule *util* contains common used functions. Although the main modules have no code dependencies to each other they use the same triple store and RDF URIs. Every main module represents one step in the generation of a ranked list of SPARQL endpoints.

QEPSM class It is the entry point to start the program by command-line. It parses the command-line arguments using *argparse*[3] and starts the corresponding submodule. Calling it with *-h* or *–help* shows an instruction how to work with the tool. Alternatively the services of the tool can be called using the public methods of the class.

Submodules Every module has a *Manager* class which connects it to the *QEPSM* class. All public methods of the submodules are callable from the *QEPSM* class as well.

Crawling module The *CrawlerManager* class contains a list of *crawlers*. Its method *start_all_crawlers* calls iteratively the *crawl* method defined in the abstract base class *CrawlerBase*. All crawler inherit from this class and have to implement a custom *crawl* method.

Analysis module The structure is identically to the one of the *crawling module*. A manager starts the analysing process of all analysers. In addition the base class implements some utility functions to delete old rankings and save the newly calculated ones.

[1] http://pypi.python.org/pypi
[2] http://www.openrdf.org
[3] http://docs.python.org/2/library/argparse.html

Retrieval module The *RetrievalManager* allows to retrieve a list of the crawled datasets and their quality estimations. Beside the SPARQL repository itself, it is the main interface to extract the data by other tools.

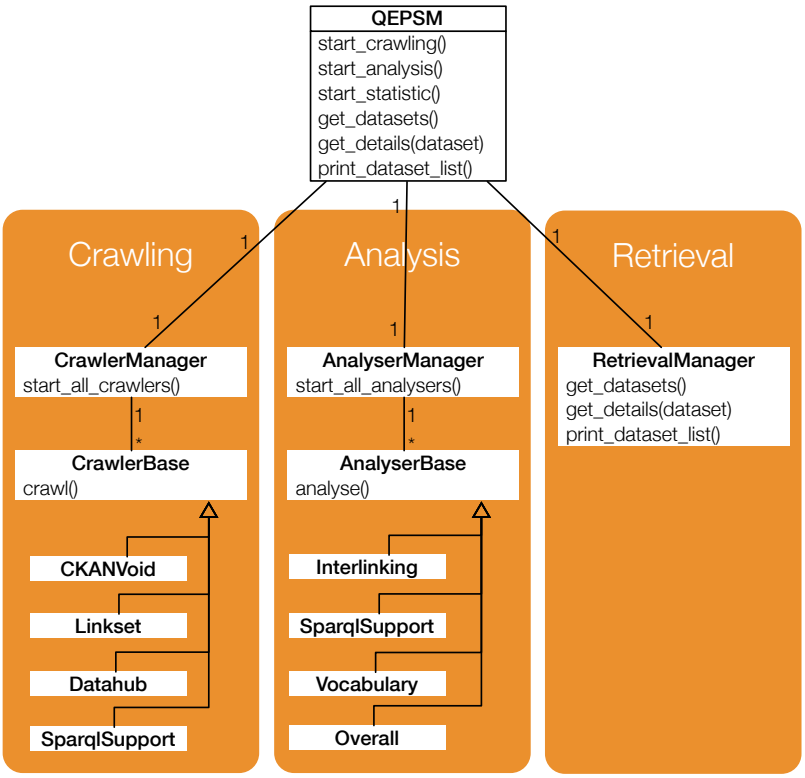

Figure 6.1: Simplified Architecture of QEPSM.

6.2 Dependency Injection

A high degree of modularity and extensibility was a primary goal developing the architecture of QEPSM. We therefore decided to use the architectural concept *Dependency Injection (DI)* - also known as *Inversion Of Control (IoC)* - to build the three submodules. The idea is to decouple classes depending on one another from inheriting other dependencies and to link them instead at interfacing level only. The instantiation of the concrete objects is done by a DI container instead of directly by the code itself. This allows to transfer the responsibility for creating and linking objects to an externally configurable element like a XML file. This shows code dependencies more clearly and allows for a comfortable configuration of the tool without touching any code.

Spring[4] is one of the most popular frameworks for Java. One key functionality is the support of DI. Although implementing DI in dynamically typed languages is much easier than in static typed and does not require necessarily a framework, with *Spring Python*[5] there exists an offshoot

[4]http://www.springsource.org
[5]http://springpython.webfactional.com

of the Java-based Framework which is very pleasant to use.

Spring Python offers multiple formats to define your objects. You can use XMLConfig, PythonConfig, PyContainerConfig or the original SpringJavaConfig. We decided to use the XML-based IoC configuration. The full documentation can be found on the official website of the tool[6].

In total QEPSM uses four configuration files. One for global settings affecting the entire tool. This includes the urls to the RDF repositories where the crawled data is saved, RDF URIs of the ranking algorithms and URIs of used namespaces. Every submodule has its own configuration file for the module specific objects. There you can define which crawler or analyser should be used and with which configurations. Listing 6.1 shows a simplified XML configuration for the *analysis* module. The first object defined is the *AnalyserManager*. The Manager has a member variable *analysers* of type *List*. The analysers which should be used are referenced as entries of this list. How they should be instantiated is defined directly after the *AnalyserObject*. You can define the constructor parameters as well as all public variables. When you want to use the AnalyserManager in the Python code you can call it using its ID. All dependent objects (like all analysers in this case) are instantiated automatically. By commenting out an entry in the list, it is possible to easily adjust which analyser respective crawler should be used without touching any Python code.

Listing 6.1: Simplified Spring XML Configuration.

```
<objects>
  <object id="AnalyserManager" class="AnalyserManager">
    <property name="analysers">
      <list>
        <ref object="AnalyserInterlinking"/>
        <ref object="AnalyserSparqlSupport"/>
        <ref object="AnalyserOverall"/>
      </list>
    </property>
  </object>
  <object id="AnalyserInterlinking" class="AnalyserInterlinking">
        <!-- All constructor parameters -->
        <!-- All public variables -->
  </object>
</objects>
```

6.3 Configuration

The process flow is determined by what input source is chosen. Figure 6.2 illustrates two possible setups. The first process takes *VoID Files* as source as described in 4.2. The *CKANVoid* crawler uses meta-catalogues to find VoID resources and saves these in a temporarily RDF repository. The *Linkset* crawler then uses this data to detect all relationships between the datasets and eliminates the irrelevant datasets.

Alternatively, the *Datahub* crawler uses directly the metadata collected by the catalogue *the Data Hub*[7]. The details are illustrated in section 4.3. Because the dataset relationships are already explicitly available in the catalogue, the *Linkset* crawler is not used when this source is chosen.

Regardless of the input used, the *SparqlSupport* queries all SPARQL endpoints periodically and takes note of how many queries have been answered successfully.

[6] http://springpython.webfactional.com/1.1.0/reference/html/objects.html
[7] http://thedatahub.org

The input has also an impact on the analysis level. The *Vocabulary* analyser cannot be applied using providers from the Data Hub. There is no information available regarding the vocabularies used by the datasets.

The intention is to decide initially to use either the *Data Hub* or the *VoID Files* data. As seen in the comparison in section 4.4, the interlinking data of *Data Hub* is of higher quality. Therefore, we recommend to take the *Data Hub* as input.

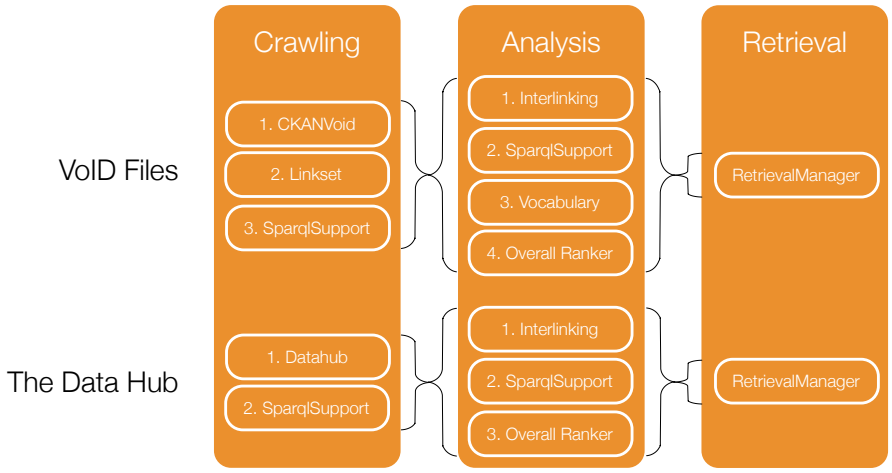

Figure 6.2: Process Flow of Two Configurations.

6.4 Code Quality & Documentation

A high code quality has been a crucial goal when developing the framework. We use the tool *pylint*[8] to analyse the source code looking for bugs and signs of poor quality. The source code is fully documented with *docstrings*[9]. In addition, there is a HTML documentation available generated by *Sphinx*[10].

6.5 Web Interface

Separated from the core tool, we implemented a web interface[11] which allows to browse the crawled datasets. It visualises the datasets with their corresponding rankings in a table. A search field allows to filter the datasets. Further, the datasets can be ordered by all ranking algorithms. This makes it easy to compare the results of the different ranking approaches.

On the same page the web interface illustrates the distribution of the values of the various ranks. The charts plot the rank position of a dataset on the x-axis and the calculated rank value, distributed between 0 and 1, on the y-axis. Figure 6.3 shows a screenshot of the overview page.

[8] http://www.logilab.org/project/pylint
[9] http://www.python.org/dev/peps/pep-0257/
[10] http://sphinx-doc.org
[11] http://qepsm.philippundhee.ch

On the detail page there are displayed further data and a link to the SPARQL endpoint. A line plot displays the history of the trend of the availability of the endpoint. The availability analysis is executed every fourth hour.

Name	Overall	Interlinking	Sparql Support
dbpedia	1.000000	1.000000	0.577494
statistics-data-gov-uk	0.907529	0.707310	0.991271
reference-data-gov-uk	0.851496	0.634698	0.997598
education-data-gov-uk	0.801194	0.675222	0.756620
geospecies	0.797877	0.748984	0.574855
revyu	0.784922	0.736902	0.565337
transport-data-gov-uk	0.781260	0.670010	0.710762
semantic-web-dog-food	0.771336	0.537205	0.991750
italian-public-schools-linkedopendata-it	0.771230	0.662718	0.698576
bio2rdf-pubchem	0.769170	0.669644	0.676419

Showing 1 to 10 of 446 entries Previous Next

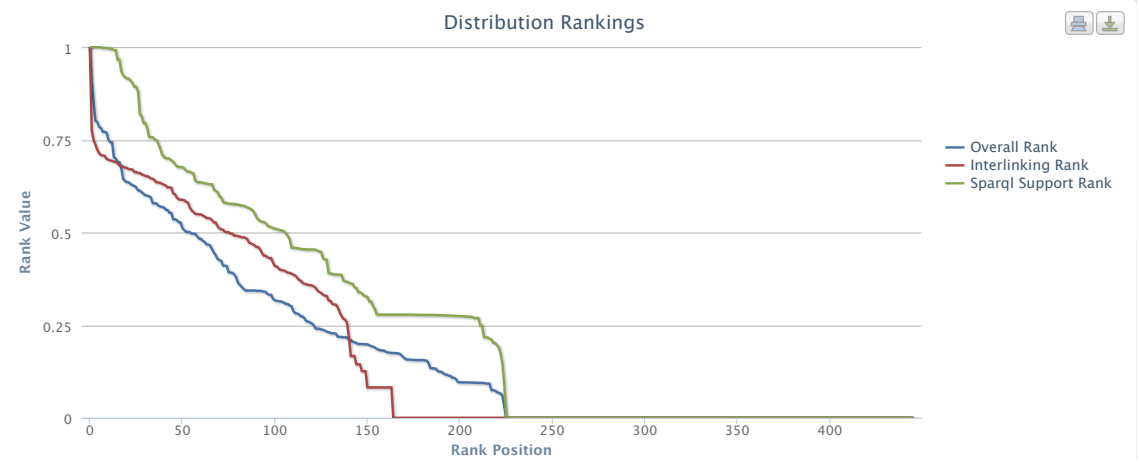

Figure 6.3: Web Interface.

7

Limitations

There exist works to generate automatically VoID descriptions for large datasets. The tool *voiDgen*[1] [Böhm et al., 2011] uses a MapReduce [Dean and Ghemawat, 2008] based approach to annotate sets of RDF resources. With LODStats [Demter et al., 2012] there exists another framework for generating statistics of RDF data. They tried to apply their tool to SPARQL endpoints available via the CKAN repository. At 71.2% of the endpoints the generation of the statistics failed. Problems most likely originate from performance restrictions of the SPARQL endpoints or from not supporting necessary SPARQL 1.1 functionalities required to do computations on the endpoint side. *RDFStats* [Langegger and Woss, 2009] suffers from the same problem. It generates statistics of RDF sources like SPARQL endpoints and RDF documents including link analysis, as long as the endpoint is very performant and the bandwidth is not restricted.

We tried to implement our own relationship crawler looking for links at triple level. If there is an object in a dataset with a link predicate like *owl:sameAs*[2] and this object is not described as a subject in the same dataset, it is very likely that this object is declared in another dataset. By finding this object defined as a subject in another dataset, it is possible to establish a directed link from the first dataset to the second one.

We know of two possible approaches to investigate the data of an SPARQL endpoint at triple level. Either we use SPARQL queries or we use the RDF dumps if they are available. Using the SPARQL queries to apply the mechanism described fails because of the same reasons described for the LODStats. Most SPARQL endpoints limit the number of triples they answer. You can try to retrieve the triples in multiple steps using *OFFSET* and *LIMIT*. However, this requires that the returned triples are sorted which increases the load of the SPARQL endpoint and leads to a blockade after just a few queries. Trying to compute more on the endpoint side instead of retrieving a lot of triples is hardly possible because of the missing support of SPARQL 1.1 functionalities at the majority of endpoints.

Despite processing challenges because of the enormous quantity, RDF dumps of the datasets are hardly offered for download.

[1] http://www.hpi.uni-potsdam.de/naumann/news/beitrag/tool-voidgen-released.html
[2] http://www.w3.org/TR/owl-ref/#sameAs-def

8

Future Work

As mentioned in the previous chapter, it would be interesting to crawl more metadata about a SPARQL endpoint by using the underlying data instead of self-descriptions and data from meta catalogues. This would improve both the quality and quantity of the available data to be analysed using the framework discussed in this thesis. This includes for example developing heuristics to query SPARQL endpoints for relationships with other datasets with simpler and less complicated queries. With the better support of SPARQL 1.1, functionalities like aggregation of the endpoints and the extraction of statistical information will be more extensive. Beyond the links between the datasets, the used vocabulary could be retrieved. Further, additional variables like the currentness of data could be interpreted and ranked.

9

Conclusions

With *QEPSM* we have presented two mechanisms to crawl RDF datasets and metadata. We have illustrated the possibilities and constraints when crawling VoID files or using meta-catalogues like Data Hub. This paper has discussed algorithms to rank the crawled datasets. The analysis of the modified PageRank algorithm showed that, with in the light of the current data available there is no correlation between the rank of the webpage of a dataset generated by Google or Alexa and the calculated ranks using the relationships on the RDF layer. The SPARQL support testing mechanism introduced a new method to evaluate the availability and functional range of SPARQL endpoints in a finely graded form. The analysis has shown that the degree of supported functionalities differs at multiple levels and is not a binary function. Applying the algorithm periodically also allows to interpret and to estimate how quickly the new SPARQL 1.1 standard will be implemented by the endpoints in the future. Testing and ranking the vocabulary of a dataset, we introduced a new theoretical metric to check how well a dataset is integrated.

Our prototype offers a extensible framework to crawl and analyse datasets. It can be used with the crawlers and analysers discussed in this paper or with custom-created ones. The included web interface allows to display the calculated data.

List of Figures

List of Tables

List of Listings

Bibliography

[Alexander et al., 2011] Alexander, K., Cyganiak, R., Hausenblas, M., and Zhao, J. (2011). Describing linked datasets with the void vocabulary. http://www.w3.org/TR/void/.

[Baeza-Yates and Ribeiro-Neto, 1999] Baeza-Yates, R. A. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Berners-Lee, 2006] Berners-Lee, T. (2006). Linked data - design issues. http://www.w3.org/DesignIssues/LinkedData.html.

[Böhm et al., 2011] Böhm, C., Lorey, J., and Naumann, F. (2011). Creating void descriptions for web-scale data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(3):339–345.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[Demter et al., 2012] Demter, J., Auer, S., Martin, M., and Lehmann, J. (2012). Lodstats – an extensible framework for high-performance dataset analytics. In *Proceedings of the EKAW 2012*, Lecture Notes in Artificial Intelligence (LNAI). Springer. To be published.

[Dietrich and Pollock, 2009] Dietrich, D. and Pollock, R. (2009). Ckan: apt-get for the debian of data. 26th Chaos Communication Congress.

[Ding et al., 2004] Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R. S., Peng, Y., Reddivari, P., Doshi, V., and Sachs, J. (2004). Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, pages 652–659, New York, NY, USA. ACM.

[Ding et al., 2005] Ding, L., Pan, R., Finin, T., Joshi, A., Peng, Y., and Kolari, P. (2005). Finding and ranking knowledge on the semantic web. In Gil, Y., Motta, E., Benjamins, V., and Musen, M., editors, *The Semantic Web – ISWC 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 156–170. Springer Berlin / Heidelberg.

[Finin et al., 2005] Finin, T., Ding, L., Pan, R., Joshi, A., Kolari, P., Java, A., and Peng, Y. (2005). Swoogle: Searching for knowledge on the semantic web. In *In AAAI 05 (intelligent systems demo*, pages 1682–1683. The MIT Press.

[Franz et al., 2009] Franz, T., Schultz, A., Sizov, S., and Staab, S. (2009). Triplerank: Ranking semantic web data by tensor decomposition. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 213–228, Berlin, Heidelberg. Springer-Verlag.

[Glaser et al., 2009] Glaser, H., Millard, I., and Carr, L. (2009). Rkbexplorer: Repositories, linked data and research support. In *Eprints User Group, Open Repositories 2009*. Event Dates: 20/05/2009.

[Heath and Bizer, 2011] Heath, T. and Bizer, C. (2011). Linked data: Evolving the web into a global data space. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 1(1):1–136.

[Hill and Lewicki, 2006] Hill, T. and Lewicki, T. (2006). *Statistics: Methods and Applications : a Comprehensive Reference for Science, Industry, and Data Mining*. StatSoft.

[Kleinberg et al., 1999] Kleinberg, J., Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. (1999). The web as a graph: Measurements, models, and methods. In Asano, T., Imai, H., Lee, D., Nakano, S.-i., and Tokuyama, T., editors, *Computing and Combinatorics*, volume 1627 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin / Heidelberg.

[Langegger and Woss, 2009] Langegger, A. and Woss, W. (2009). Rdfstats - an extensible rdf statistics generator and library. In *Proceedings of the 2009 20th International Workshop on Database and Expert Systems Application*, DEXA '09, pages 79–83, Washington, DC, USA. IEEE Computer Society.

[Mulay and Kumar, 2011] Mulay, K. and Kumar, P. (2011). Spring: Ranking the results of sparql queries on linked data.

[Oren et al., 2008] Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., and Tummarello, G. (2008). Sindice.com a document-oriented lookup index for open linked data. *Int. J. Metadata Semant. Ontologies*, 3(1):37–52.

[Page et al., 1999] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web.

[Pandurangan et al., 2002] Pandurangan, G., Raghavan, P., and Upfal, E. (2002). Using pagerank to characterize web structure. In Ibarra, O. and Zhang, L., editors, *Computing and Combinatorics*, volume 2387 of *Lecture Notes in Computer Science*, pages 330–339. Springer Berlin Heidelberg.

[Roa-Valverde et al., 2012] Roa-Valverde, A. J., Toma, I., Thalhammer, A., and Sicilia, M.-A. (2012). Towards a formal model for sharing and reusing ranking computations. In *6th International Workshop on Ranking in Databases*.

[Seuken and Parkes, 2012] Seuken, S. and Parkes, D. (2012). Economics and computation.