A Visual Summary for Linked Open Data sources

Fabio Benedetti¹, Sonia Bergamaschi², Laura Po²

¹ ICT PhD School - Università di Modena e Reggio Emilia - Italy
  fabio.benedetti@unimore.it
² Università di Modena e Reggio Emilia - Dipartimento di Ingegneria "Enzo Ferrari" - Italy
  firstname.lastname@unimore.it

Abstract. In recent years, there has been an increase in the production of machine-readable data in the form of RDF Datasets belonging to the Linked Open Data (LOD) Cloud, however, this growth has not being followed by an analogous improvement and specialization of tools to search, analyze and visualize LOD sources. As a matter of fact, when dealing with LOD sources, users waste a lot of effort trying to understand which kind of knowledge is stored and how the data are organized within a specific Dataset.

In this paper, we propose LODeX, a tool that produces a highly representative summary of a LOD source. Starting from the URL of a SPARQL Endpoint, the tool launches a set of predefined SPARQL queries and automatically generates a summary of the source. The summary reports statistical and structural information regarding the LOD Dataset; moreover, it permits to be browsed to focus on particular classes, by exploring their properties and their use (e.g. number of instances in the dataset).

Thanks to this high-level view of a LOD Dataset, LODeX significantly helps users in understanding the contents of a source starting from scratch. To the best of our knowledge, this is the first tool able to integrate the automatic extraction of structural and statistical information with its visualization and providing a summary of the contents of a LOD source (an online demo of LODeX is available at http://dbgroup.unimo.it/lodex).

1 Introduction

The possibility to expose any sort of data on the Web by exploiting a consolidated group of technologies of the Semantic Web Stack [5] is one of the main strengths of Linked Open Data. The RDF Data Model plays a key role in the birth and continuous expansion of the Web of data, since it allows to represent structured and semi-structured data. While the LOD cloud is still growing, we assist to a lack of tools able to produce a meaningful, high level representation of these datasets. Thus, discovering and identifying LOD sources of interest is a complex task for a user.

There are several portals that catalog datasets that are available as LOD on the Web, one of the main globally available Open Data catalogues is The Data Hub (formerly CKAN)³. All of these portals permit users to perform keyword search over their list of

³ http://datahub.io
LOD sources. Nevertheless, the result does not explain why some matches are found, so that, the user has to verify, for each source in the result, if it is compliant with his/her interest.

Assuming that a user has been lucky enough to identify one (or more) datasets of interest with an available SPARQL endpoint, he/she still has to figure out how data are structured (for instance, what types of resources are available (classes) and what properties relate the resources of each type). When a user starts exploring in details an unknown LOD source, several issues arise: (1) the difficulty in finding documentation, often poor and sometimes missing; (2) the complexity of understanding the schema of the source (since there are no fixed modeling rules); (3) the effort to explore a source with a high number of instances. Some LOD providers make a high level description of classes and properties of the dataset directly available in a web page, but this is not ensured for each LOD dataset. In all other cases, the user is asked to write specific SPARQL queries in order to start the analysis of the LOD Dataset. At this point, a user with no SPARQL skills could find himself stuck in the process of selecting the most suitable source for his/her needs. He/she might try to use some available visualization tool. Usually, LOD visualization tools focus on representing instances of a dataset by using a graph. Considering that LOD sources may contain hundreds or thousands of class instances, the graph generated might be difficult to be read. To overcome the above problems, we devise a new method and a tool, LODeX, able to automatically provide a high level summarization of a LOD dataset, including its inferred schema.

LODeX neither requires any apriori knowledge of a dataset, nor asks users to set any parameter. Starting from the URL of a SPARQL endpoint, the tool extracts statistical and structural information that gives a high-level view of a LOD Dataset. LODeX extracts the triples which represent the ontological constraints of the LOD Dataset and builds a schema inferred from the properties between class instances. It exploits the query computation power of the endpoint, without requiring any local materialization of the dataset and optimizing its performance (all the information are extracted through queries executed on the SPARQL endpoint).

The paper is structured as follows. Our definition of summarization of a LOD source and an overview of the LODeX architecture are depicted in section 2. Section 3 is devoted to describe some technical insight regarding the process of Schema Extraction. In Section 4 some tests are reported while, in section 5, we describe some relevant works that are related to this topic or that have inspired the development of this tool. Conclusions and some ideas for future work are described in section 6.

2 LODeX - Overview

The LOD Cloud consists of an huge number of SPARQL endpoints, each aiming to describe a knowledge base of a specific domain. The language used to describe data is RDF, while RDFS and OWL are used to represent intensional knowledge [19] [11].

A SPARQL endpoint exposes data that can have different uses and purposes. For example, an organization can model a specific environment and produce a complex OWL ontology and then publishes it as an RDF graph. Even a public institution can use the same paradigm to share its Open Data, maybe, automatically translating a traditional
Relational Database into an RDF graph. It is clear that the most valuable information is contained in the intensional knowledge in the first case, while the relevant data can be found within the instances and their properties in the second one.

Generally speaking, all the endpoints contain intensional and extensional knowledge in different proportions. An abstract representation of this is shown in Figure 1:

- **Intensional Knowledge (I)** - The triples belonging to this group define the terminology used in the Dataset. They are expressed in RDF, however I can be usually interpreted through RDFS or OWL. I can be seen, with some restriction, as the T-Box component of the knowledge base described in the endpoint [17].

- **Extensional Knowledge (E)** - This group of triples usually cover most of the Dataset and contains the entities of the real world described in the Dataset. Usually, the knowledge contained in E is described through RDF instances which compose the A-box of the knowledge base [17].

These two kinds of knowledge are logically distinct, according to their semantic meaning, but they form an unique RDF graph. The triples belonging to E and I are not connected each other randomly; indeed, there are particular classes, that we called Instantiated Classes, defined in I and instantiated in E that act as a bridge among the two resources.

A well designed Dataset should contain both intensional and extensional knowledge, but by analyzing a large number of endpoints we have observed that this is not generally true. Sometime, LOD datasets contain ontologies in which instances are not present or otherwise, in which only one Class (owl:Class) is defined and a large number of instances are improperly used as such. In other cases, LOD datasets defines only the extensional knowledge, thus they do not include the description of the used vocabulary within instances (i.e. this happens quite often when Open Data are published). These design issues are mainly caused by a large use of automated translation tools. For example, there are plenty of techniques to produce an OWL version of an Ontology expressed with other standards as DAML+OIL or RRF and also the W3C consortium is spending many efforts in defining technologies able to translate RDB in RDF, called RDB2RDF4.

4 [http://www.w3.org/2001/sw/rdb2rdf](http://www.w3.org/2001/sw/rdb2rdf)
The RDF Data Model recalls the Graph Data Model where the peculiarity is that the data structure is implicitly defined by the set of triples [2]. Taking into considerations some resources belonging to the LOD Cloud, it can be seen that information about the schema resides explicitly in the instantiation of the classes and implicitly in the use of the properties among these instances [10]. LODeX takes into consideration all these assumptions to gather the Schema Summary from the distribution of the instances.

2.1 Schema Summary

The Schema Summary is representative of the extensional knowledge contained in a LOD dataset. Its formal definition is shown in the following:

**Definition 1** A Schema Summary S, derived from a LOD endpoint, is a tuple: \( S = \langle C, A, P, L_c, L_p \rangle \), where:

- \( C \) contains a set of pairs \( (c,n) \), where \( c \) is a Class of the LOD Dataset and \( n \) is the number of its instances
- \( A \) contains the attributes of Classes of the LOD Dataset and it is composed of a set of triple \( (c,a,n) \), where \( c \in C \), \( a \) is a property direct from \( c \) to a literal and \( n \) is the number of times this attribute appears between the instances of \( c \)
- \( P \) contains the properties between Classes of the LOD Dataset, and it is composed of a set of \( (s,o,n) \), where \( s \) and \( o \in C \) and \( n \) is the number of times this property is used among \( s \) and \( o \)
- \( L_c \) contains a set of \( (c,l) \), where \( c \in C \) and \( l \) is the label associated with the Class \( c \)
- \( L_p \) contains a set of \( (p,l) \), where \( p \in P \) and \( l \) is label associated with the property \( p \)

2.2 Architectural Overview

LODeX aims to be totally automatic in the information extraction and in the production of the Schema Summary. Therefore, it does not require any kind of a priori knowledge about the dataset on which it works. In Figure 2, you can see a representation of the two main subprocesses which are executed sequentially and led to the definition of the schema summary starting from the URL of SPARQL endpoint. The first is called Index Extraction (IE) and produces the Statistical Index that describes a LOD source, the second is called Post-processing (PP) and brings to the definition of the Schema Summary of a LOD source.

Since IE can be time consuming, we have chosen to materialize the partial results coming from this process in a NoSQL Database, so that PP can be performed once IE has completed its task. We have chosen a NoSQL document database, MongoDB\(^5\) [4], because it allows a flexible representation of the index extracted by IE and, in particular, it can easily manage lists of elements (the components of the index will be described in section 3.2).

\(^5\) https://www.mongodb.org
Index Extraction The Index Extraction is the core process of LODeX. It takes as input just the URL of a SPARQL endpoint and generates the queries able to extract structural and statistical information about the source. The IE process has been designed in order to maximize the compatibility with LOD sources and minimize the costs in terms of time and computational complexity. Two different algorithms, based on a set of SPARQL patterns, have been developed to extract the most relevant intensional and extensional information from heterogeneous LOD datasets.

We have chosen to deal only with SPARQL endpoints, differently from other approaches that prefer to work with a dump of the RDF Database loaded in local (eg. [16], [3] and [8]). This choice is due to three main reasons:

- The dump of a Dataset is not always available and when it is available it can happen that it is not up to date.
- It is necessary to download the dump files before starting to extract the needed information.
- Exploiting SPARQL endpoints ensures that just the remote server scans the whole RDF Dataset, thus moving a significant part of the computation cost on it. In addition, this means that just the needed information are extracted from the endpoint.

LODeX is designed to be able to manage the parallel execution of several instances of the IE process on a single machine, exploiting the idle times caused by response-time delays. Moving part of the computational cost of the extraction process on the endpoint can improve the performance, but it brings some drawbacks. First of all, a portion of the queries generated by IE uses some operators introduced with SPARQL 1.1 [12], thus the endpoint must be compatible with this standard. Another issue regards the heterogeneity of the implementation of SPARQL endpoints that affects their performance. Some endpoints are not able to answer to some queries before the timeout expires. To avoid this problem, we have defined a strategy to scale the complexity of the queries when a timeout error occurs. This strategy will be described in section 3.
Post-processing  The PP\(^6\) process is performed when the Statistical Index has been materialized and its primary goal is to combine the information extracted by the IE process to produce the Schema Summary of a specific Dataset. The Schema Summary is induced from the distribution of the instance in the Dataset. Along with the Schema Summary, PP collects Synthetic Information regarding the endpoint; these are statistics about both the Intensional and the Extensional knowledge.

Summary Visualization  The visualization of the Schema Summary is performed through a Web Application written in python that uses MongoDB as database to store the Statistical Index. We use D3 (Data Driven Documents\(^7\) [6]) to create a visual representation of the dataset with which the user can interact and discover the information that he/she is looking for. A demo of LODeX Web Application is available at http://dbgroup.unimo.it/lodex.

In the home page of the Web App you can find the lists of the datasets and their Synthetic Information. It is possible to access to the Schema Summary just clicking on the name of dataset. Synthetic Information reports the size and the complexity of the dataset: number of triples, number of instances, number of classes, number of properties, number of triples belonging to the intensional knowledge etc.

As an example, we used the "Linked Clean Energy Data" dataset. This dataset is produced and maintained by reegle\(^8\) and it contains information about actors and stakeholders in the field of the green energy. Table 1 shows the main characteristics of this source.

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triples</td>
<td>60140</td>
</tr>
<tr>
<td>Classes</td>
<td>20</td>
</tr>
<tr>
<td>Instances</td>
<td>5099</td>
</tr>
<tr>
<td>Properties</td>
<td>68</td>
</tr>
</tbody>
</table>

In Figure 3, a screenshot of the summary for the Linked Clean Energy dataset is shown. The Schema Summary is represented through a graph where the nodes represent the classes (elements of C) and the edges represent the existence of one or more property between these classes (elements of P). The legends on the right contain static information; the name of the dataset is shown on the bottom right and the list of vocabularies used is written on the top right. A color is assigned to each vocabulary and all these colors are used to indicate to which vocabulary each class, property or attribute belongs to.

---

\(^6\) The PP process is not the topic of this paper, a detailed description of this process and the pseudocode of the algorithm used to generate the Schema Summary from the Statistical Index, can be find at http://www.dbgroup.unimo.it/lodex_schemasummary

\(^7\) http://d3js.org/

\(^8\) http://www.reegle.info/
The left part of the screen changes dynamically when the user goes with the mouse over a node in the graph. On the top left, the class name and the number of its instances are depicted. Below the class name, the ingoing and outgoing properties are represented; the value on the right of each row indicates the average number of properties of that type defined for each instance of the class. Clicking on one of those rows a subset of the triples belonging to the intensional knowledge regarding the Subject Class, the Object Class and the property is provided to user in a new window (not shown in this Figure). In the end, in the left down portion of the screen you can find the attributes referring to the current class; as in the case of the properties the value associated with each row indicate the average number of times that an attribute is used for each instance.

Fig. 3. Example of Schema Summary for the Linked Clean Energy Data dataset

3 LODeX - Technical insights

In this section will be explained the techniques and the algorithms behind the IE process, the elements of the Statistical Index and the way they are extracted.

3.1 SPARQL Patterns

LODeX exploits different patterns, through these patterns it can produce queries of different complexity. With the introduction of the version 1.1 of SPARQL it would be possible to collect aggregate information about a specific Basic Graph Patterns (BGP) [14]
over an RDF Dataset using the operator GROUP BY, but in many cases a large Dataset is not capable of providing this response before the timeout expires. The operator GROUP BY is very expensive in terms of performance, but it is necessary in order to know how many times a BGP is used. We have chosen to use a restricted group of operators, apart GROUP BY, as: FILTER, COUNT, DISTINCT and AND to minimize the complexity of the queries generated (e.g. the evaluation of an expression containing AND and FILTER can be solved in linear time [20]).

We also handled the problem of long running queries, that are usually going in timeout, by generating an higher number of queries able to return the same information into smaller chunks of data. We called this mechanisms *Pattern Strategy* and it will be deeply describe in section 3.4.

We have divided the typology of the queries used by LODeX in three distinct category:

- **General Queries** - This group contains some atomic queries used to extract general information about the Dataset. The information extracted are: number of triples, number of classes, number of properties, number of instances, list of classes and their number of instances, list of properties and their number of times they are used. Some of these queries refer to those used to extract the Void Descriptor\(^9\) of a Dataset [1].

- **Ontology Patterns** - These patterns are used to extract the Intensional knowledge of a Dataset. They are based on a simple triple pattern (subject, predicate and object) in which the subject is replaced iteratively with the URIs representing the constraints of the ontology, in order to traverse the RDF graph and extract the Intensional knowledge. The patterns differ each other according to the filter condition (literal, blank node or uri; see Section 3.5).

- **Subject & Object Patterns** - These patterns are used to extract information regarding the schema implicit in the distribution of the instances. In Figure 4 you can see a schematic representation of two patterns called Object and Subject Path. LODeX actually uses many variations of these patterns, adding filtering condition on the rounded nodes or replacing the nodes Classes or ?p) with some value previously extracted. The idea is to make implicit the primitive of instantiation (rdf:type) and inspect the properties incoming (Object Path) or outgoing (Subject path) of a specific Class through its instances.

### Table 2. LODeX Index components. Legend: Cn: class name; Pn: property name; s: subject; p: property; o: object; n: number of times a path (or a property) exists; nI: number of instances

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Structure</th>
<th>Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Number of Triples</td>
<td>Integer</td>
<td>General Statistics</td>
</tr>
<tr>
<td>c</td>
<td>Number of Classes</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>Number of Instances</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>CI</td>
<td>Class list</td>
<td>List(Cn,nI)</td>
<td></td>
</tr>
<tr>
<td>PI</td>
<td>Property list</td>
<td>List(Pn,n)</td>
<td></td>
</tr>
<tr>
<td>IK</td>
<td>Intensional Knowledge Triples</td>
<td>List(s,p,o)</td>
<td>Intensional Knowledge</td>
</tr>
<tr>
<td>Sc</td>
<td>Subject Class</td>
<td>List (s,p,n)</td>
<td></td>
</tr>
<tr>
<td>Scl</td>
<td>Subject Class to Literal</td>
<td>List (s,p,n)</td>
<td>Subject and Object Path</td>
</tr>
<tr>
<td>Oc</td>
<td>Object Class</td>
<td>List (o,p,n)</td>
<td></td>
</tr>
</tbody>
</table>

### 3.2 Index components

In this Section we describe the information extracted by the IE process to build the Statistical Index materialized in the NoSQL Database. We divided the information collected in the index in three category: General Statistics, Intensional Knowledge, Inferred Extensional Schema. The list of the index elements is presented in Table 2; below we will describe the categories introduced above and their components.

**General Statistics** - In this group some generical information regarding the size and the complexity of the Dataset are collected. In particular, the first three elements (t, c, i) can give an insight of the dimension of the RDF Graph. Instead, the last two components (CI and PI) are two lists containing information about the Class instances and the Properties usage.

**Intensional Knowledge** - This group contains all the triples related to the intensional knowledge. These triples are extracted through an iterative algorithm (presented in Section 3.5).

**Subject and Object Path** - This group contains the schema information, regarding the distribution of instances, extracted by the IE process projecting the Subject and Object Paths on the Dataset. As you can see in Table 2, it is composed by three lists; the first two refer to Subject Path, while the last regards the Object Path. Each element of these lists contains three elements: s (or o) contains a Subject Class (or an Object Class), p contains the property (?p in Figure 4) and n contains the number of times this path is used.

### 3.3 Statistical Index extraction

In Figure 5 you can see a graphical representation of the steps performed by the IE process briefly described below:

- **Test Connection** - First of all, a first test of connection with the endpoint is performed because in many cases the Datasets can be out of service. Secondly, the compatibility of the endpoint with the operator used in the queries provided by LODeX is tested.
– **General Statistics Extraction** - In this step the components of the Index belonging to General Statistics (see Table 2) are extracted. The first three elements ($t, c$ and $i$) are extracted through simple SPARQL queries. Instead, $Cl$ and $Pl$ cannot be always extracted using a single query, because of performance problems of endpoints. For this reason, $Cl$ and $Pl$ are extracted using a Pattern Strategy (see section 3.4) in order to increase the success rate of this operation.

– **Intensional Knowledge Extraction** - In this step the extraction of the Intensional Knowledge through an iterative algorithm which will be described in Section 3.5 is performed.

– **Extensional Schema Extraction** - This is the last step performed in the IE process. In particular the elements extracted in this step are $Sc$, $Scl$ and $Oc$. This step is the most critical as complexity; in the next section is explained the Pattern Strategy used to complete this phase.

### 3.4 Pattern Strategy

We often stumbled across errors triggered by endpoints due to performance issues populating some parts of the Statistical Index. In particular, this problem occurred when extracting the Subject Path ($Sc$, $Scl$), the Object Path ($Oc$) and in few cases the Class and Property lists: the subgraphs matching these patterns could be extracted using just one query for pattern, but this operation has an high cost for the endpoint and in most cases a timeout error is throw. Hence, we have designed a Pattern Strategy able to handle this type of errors and scale the complexity of the SPARQL generated queries.

In Figure 6 you can see a representation of the Pattern Strategy used to complete the extraction of $Sc$. By using the first query it is possible to extract all $Sc$ in one go, but this query has a high failure rate. If the endpoint is not able to answer to the first query, the strategy switches to the second step and a query for each element of $Cl$ (class) is generated; each successful response returns an element of $Sc$, while when an error occurs the current class is added to $ErrClass$. If some error still exist after the second step, the Strategy tries to download the two items that compose each element of $Sc$ (property name, and property count) separately. In particular, in step 3 and 4 the queries are generated in order for discovering the properties related to the current class by a Subject Path; this information is temporarily stored in a list called $TmpSc$ which is taken as input by the last step, where the queries are generated for completing the partial results contained in $TmpSc$ with the information regarding the number of times these paths are present in the Dataset.
A similar strategy (the queries used are changed) is used to complete the extraction of ScI, Oc, Cl and Pl. The success rate of the extraction of those elements of the Statistical Index is increased from the 22% to the 78% using this Pattern Strategy.

### 3.5 Intensional knowledge Extraction

The Intensional knowledge contained in a generic Dataset usually consists in few triples within the dataset with a high information load. It is therefore important to pull out all these triples; to achieve this goal we have designed an iterative algorithm able to traverse the RDF Graph and pull out the Intensional knowledge.

An iterative algorithm is well suited for traversing any sort of graph, but we have to properly choose the starting point and the condition of traversing, in order to make the algorithm efficient and to be sure that only the triples desired are extracted. Moreover, a issue could lead to traverse and download the entire graph and hence the instances, this will cause the deadlock of the IE process given the high number of data that it should download. Fortunately, we can take advantage of the structure induced by RDF to avoid this issue. In fact, the instantiation primitive of the RDF language is a recognizable triple in which the subject can be a URI or a Blank node, the predicate is `rdf:type` and the object is an URI (representing a class). We can create a group of SPARQL queries using the Class list (Cl). Each of these queries will be composed by a simple triple in which we bind the subject with each element of Cl, and use them to start traversing
the portion of the graph containing the Intensional knowledge. Moreover, in order to include the hierarchy of properties and their RDFS or OWL definition, it is necessary to include the Property list (Pl) and generate a second group of SPARQL queries. This does not imply particular issues, because the URIs, that relate properties, are used as subject only in the Intensional knowledge.

Algorithm 1 shows the pseudo-code of the Intensional knowledge Extraction (IkE).

```
Data: Cl, Pl
Result: Ik
1 Qn=∅, Fn=Cl.cn ∪ Pl.pn;
2 while |Qn| < |Fn| do
3    forall the node in Fn - Qn do
4       results←generate query for node and query the endpoint;
5       add node to Qn;
6       forall the r in results do
7           add r to Ik;
8           if r.o is not a Literal then
9               add r.o to Fn;
10          end
11    end
12 end

Algorithm 1: Intensional knowledge Extraction Algorithm
```

We have tested IkE on several Datasets (an analysis of this evaluation is described in Section 4) and it always stops once it has downloaded the entire intenstional knowledge, without traversing the whole RDF graph. The number of iterations can give an estimation of the ontology deepness and complexity. Usually Ike stops after around 5 iterations and it reaches a maximum of 22 iterations.

4 Test and Performance Evaluation

LODeX has tested on a set of datasets taken from SPARQL Endpoint Status\textsuperscript{10}, a portal which collects and analyzes the SPARQL endpoints contained in DataHub\textsuperscript{11}.

Table 3 outlines the results of the first phase of Statistical Index extraction. Here, 469 endpoint URLs have been tested, but unfortunately only 244 were online when test was performed. Moreover, during the connection test phase, we checked the compliance of each endpoint with SPARQL 1.1 operators, for these reason the number of suitable endpoints decreased to 137. We highlight that since LODeX uses only a subsets of SPARQL 1.1 operators the Statistical Index extraction can be performed on 56% of the sources (137/244). Instead, the number of endpoints fully compatible with SPARQL 1.1 is much lower; they are only 14, so the 5% as reported in

\textsuperscript{10} http://sparqles.okfn.org/
\textsuperscript{11} http://datahub.io/
http://sparqles.okfn.org/interoperability on May 4th, 2014. Also the Pattern Strategy has demonstrated its effectiveness making increase the number of endpoints which have completed the extraction phase from 33 to 107.

In Table 4 statistics about the performance for the 107 datasets that have completed the extraction are proposed. The average time of extraction is 6.12 minutes (the avg size of each dataset is 32 millions of triples). Thus, we have examined 3.45 billions of triples in 11 hours using a single process. We also tested the parallelization of the process; using 9 parallel processes the extraction time decreases to just 3.35 hours. The cost refers to an implementation of LODeX on a portable machine (Operative System: Windows 7 - 64 bit, RAM: 6 GB, number of processors: 1, number of cores: 2)

If we compare LODeX with SchemEx, even if the information extracted are different, we found that our tool achieves better performances, in fact, the SchemEx tests reveal that is able to analyze 2.17 billion of triples in 15 hours [16].

The heterogeneity on the implementation of the SPARQL endpoints is one of the most critical aspects and it also dramatically affect the performances of LODeX. To highlight this issue, we have reported some of these discordant results in Table 5. Here, we took into consideration the characteristics of three datasets: KEGG Pathway (knowledge on the molecular interaction and reaction networks), Dbinary (wiktionary data for several languages) and DBLP in RDF (L3S). In terms of size and complexity the first two datasets are very similar, but the extraction time on the first dataset takes more than 10 times of the time on the second. DBLP is a borderline case, in fact, although it is much less complex than the first two datasets, it is no able to complete the extraction process.

<table>
<thead>
<tr>
<th>Dataset URLs</th>
<th>469</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachable datasets</td>
<td>244</td>
</tr>
<tr>
<td>SPARQL 1.1 compatible</td>
<td>137</td>
</tr>
<tr>
<td>Extraction completed</td>
<td>107</td>
</tr>
<tr>
<td>Extraction without Pattern Strategy</td>
<td>33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4. Performance of extraction of Statistical Index for 107 datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG time of extraction</td>
</tr>
<tr>
<td>Total time (single process)</td>
</tr>
<tr>
<td>Total time (9 processes)</td>
</tr>
<tr>
<td>Total triples</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5. Compared statistics for three datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEGG Pathway</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>Triples number</td>
</tr>
<tr>
<td>Instance number</td>
</tr>
<tr>
<td>Class number</td>
</tr>
<tr>
<td>Property number</td>
</tr>
<tr>
<td>Extraction Time</td>
</tr>
</tbody>
</table>
5 Related Work

LODeX aims to produce a summary of an RDF knowledge base, therefore his algorithms and techniques overlap with different research topics in the field of semantic web. These topics range, from issues of visualization and documentation of LOD sources, to problems of semantic index extraction and schema summarization.

As presented in [9], the majority of the tools for data visualization is not able to provide a synthetic view of the data (instances) contained in a single source. Payola\textsuperscript{12} [15] and LOD Visualization\textsuperscript{13} [7] are two recent tools that exploits analysis functionalities for guiding the process of visualization. These tools are still under development so their behaviour is not very stable, but, apart from this, their use is highly bounded by the users knowledge about the dataset. They always need some querying parameters to start the analysis of a LOD dataset. Differently from those approaches, LODeX does not require any sort of apriori knowledge of the dataset; it focuses on extracting the schema from a LOD endpoint and producing a summarized view of the concepts contained in the dataset.

In [1], the VoID Vocabulary has been proposed to solve the scarcity of LOD documentation. VoID\textsuperscript{14} is a schema vocabulary for expressing metadata about RDF datasets. However, its descriptors are not always available since the definition of these metadata is demanded to the producer of the source (in August 2011 only 32.2% of the LOD cloud data sources contained VoID descriptor\textsuperscript{15}). The LODeX Schema Summary could also be easily translated with respect to a vocabulary and inserted into the source to enrich its documentation.

In literature, we can find several works in which a summary or a set of descriptors are extracted from a LOD source. In [8], authors divide these techniques in two groups, \textit{triples-level} and \textit{instances-level} summaries, according to the granularity with which the sources are scanned and indexed.

The triples-level techniques inspect the content of the RDF dataset scanning each triple, and then they usually build an index containing statistical information regarding the typology of these triples. SchemEx\textsuperscript{16} is an example of work belonging to this group; here, dumps of RDF Graphs are indexed in order to support user query processing. Differently from LODeX, this approach does not consider the class instances, thus it is also not able to retrieve the properties among classes. By considering a rough comparison among SchemEx and LODeX, we found that the perfomance of our tool exceeds the one of SchemEx.

In the instance-level approach the RDF Graph is inspected taking into account RDF, RDFS and OWL primitives and their semantics, in order to detect structural information pervasive in the source. In this group we can find two important works [13] [21], in which the instance-level summary can support efficient and federated query evaluation.

Several techniques for summarization have been applied on RDFS or OWL files [18, 22, 24], thus in a more static domain than the LOD Cloud. All these works show the most

\textsuperscript{12} http://live.payola.cz/
\textsuperscript{13} http://lodvisualization.appspot.com/
\textsuperscript{14} http://www.w3.org/TR/void/
\textsuperscript{15} http://lod-cloud.net/state/
important concepts identified in the ontology without producing a real representation of the content (class instances) of the source. Recently, in [23] Vocabularies coming from the Lod Cloud were taken into account as input of the summarization process. However, a limitation of this technique is that it always relies on a schema available in RDFS format.

6 Conclusions And Future Work

Starting from the URL of a SPARQL Endpoint, LODeX is able to automatically provide a high level summary of a LOD Dataset including its inferred schema. The summary reports statistical and structural information regarding the LOD Dataset, moreover, it permits to be browsed to focus on particular classes, by exploring their properties and their use (e.g. number of instances in the dataset). In this paper, we defined the architecture and algorithms that compose LODeX and showed an evaluation of the tool on a significant number (469) LOD sources available on a portal. An online demo of the behaviour of the tool is available at http://dbgroup.unimo.it/lodex. Even if the tests covers a limited set of sources, the results obtained are satisfactory and stimulate the development and optimization of LODeX in several directions.

Thanks to this high-level view of a LOD Dataset, LODeX significantly helps users in understanding the contents of a source starting from scratch. The result gained by LODeX could also be useful to enrich LOD sources’ documentation, since the schema summary can be easily translated with respect to a vocabulary and inserted into the LOD source.

We think that LODeX might become an assistance tool for LOD portals. Since the portals already provide basic search functionalities over sources, an iterative process of research and exploration of the source by using portal and LODeX functionalities, might highly improve the selection of useful LOD datasets. In addition, the graph generated from the Schema Summary can be exploited to facilitate the query definition; through the visual interface, by selecting classes and properties of interest the user can be guided in the query definition. While the composition of the SPARQL query is demanded to the system, the results of the query are shown to the user allowing a vision of the instances that meet his/her selection.

References


