Online Index Extraction from Linked Open Data Sources

Fabio Benedetti[1,2], Sonia Bergamaschi[2], Laura Po[2]

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

Abstract. The production of machine-readable data in the form of RDF datasets belonging to the Linked Open Data (LOD) Cloud is growing very fast. However, selecting relevant knowledge sources from the Cloud, assessing the quality and extracting synthetical information from a LOD source are all tasks that require a strong human effort. This paper proposes an approach for the automatic extraction of the more representative information from a LOD source and the creation of a set of indexes that enhance the description of the dataset. These indexes collect statistical information regarding the size and the complexity of the dataset (e.g. the number of instances), but also depict all the instantiated classes and the properties among them, supplying user with a synthetical view of the LOD source. The technique is fully implemented in LODeX, a tool able to deal with the performance issues of systems that expose SPARQL endpoints and to cope with the heterogeneity on the knowledge representation of RDF data. An evaluation on LODeX on a large number of endpoints (244) belonging to the LOD Cloud has been performed and the effectiveness of the index extraction process has been presented.

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* [6] is one of the main strengths of Linked Open Data. 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 allow users to perform keyword search over the metadata associated to their list of LOD sources, but they do not provide search techniques based on structural information of these sources. As mention in [14], there is still a “lack of conceptual description of datasets". The documentation of a LOD source is produced by who published the data and, in many cases, it is incomplete or absent. Therefore, usage of LOD datasets requires a human being to identify the domain of the datasets and discriminate if they are relevant for his/her needs (usually by performing SPARQL queries).

This work has been accomplished in the framework of a PhD program organized by the Global Grant Spinner 2013 and funded by the European Social Fund and the Emilia Romagna Region.

[3] http://datahub.io

To overcome the above problems, we devise a new method and a tool, called LODeX. LODeX is able to automatically extract a set of indexes containing the most representative information about the structure of a LOD dataset, enhancing the understanding and, therefore, the exploitation of LOD sources. These indexes collect statistical information regarding the intensional and extensional knowledge of a LOD source. The intensional knowledge contains the RDFS/OWL triples used to define a vocabulary or an ontology. The extensional knowledge is characterized by the instantiated classes, the properties between them and some graph patterns (ingoing and outgoing properties from instances of a specific class). Usually, the intensional knowledge is defined as the terminology used for characterizing the assertions, i.e. the extensional knowledge. As reported in [2,9], the structure of an RDF source is implicitly defined by its set of triples; information about the schema resides explicitly in the instantiation of the classes and implicitly in the use of the properties among these.

The indexes can have different uses, but primarily they represent a good documentation of the dataset to which they refer. In fact, a knowledge engineer might be interested to describe a specific environment by reusing available vocabularies or ontologies if he easily understands their intensional contents. Otherwise, a data scientist can explore the lists of elements characterizing the extensional knowledge (e.g. occurrence of outgoing properties from a particular type of instances) of a specific dataset to easily build the SPARQL queries he/she needs to extract the data he/she is looking for. The indexes can also be used with other purposes: to support search engines (e.g DataHub) for dataset selection according to the ingoing or outgoing properties from instances of a class ranked according to the number of occurrences; to support tools able to generate queries. The indexes can be used for the schema generation and summarization of LOD sources as done in [5].

LODeX only deals with SPARQL endpoints, differently from other approaches that work with a dump of the RDF Database stored locally (e.g. [15], [3] and [8]). This choice has been made in order to provide a tool able to work as an online service. LODeX takes as input just the URL of a SPARQL endpoint and produces the necessary queries to extract the needed information and generate the statistical indexes. One of the main problems we encounter when dealing with SPARQL endpoints is the heterogeneity on the performance of the different implementations of endpoints. In fact, it happens that several SPARQL aggregation queries may trigger timeout errors. LODeX handles the problem of long running queries, that are usually going in timeout, by generating an higher number of low-complexity queries able to return the same information into smaller chunks of data. We called this mechanisms *pattern strategy*, and it will be described in Section 5.1.

The paper is structured as follows. Next Section outlines some relevant works connected to this topic or papers that have inspired the development of this tool. Section 3 defines intensional and extensional knowledge in the scenario of LOD sources. An overview of LODeX and its architecture is depicted in Section 4. Section 5 details the extraction of the set of indexes. In section 6 some tests are reported and finally, conclusions and some ideas for future work are described in Section 7.

2 Related Work

In the 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, *triples-level* and *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 type of these triples. SchemEx [15] 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.

RDFstats [17], instead, defines a vocabulary and an algorithm able to collects statistics about the triples belonging to an RDF dataset that can be used to build histograms and document the source. The information extracted by RDFstats has a low granularity and it can not be used as documentation of a RDF dataset, because it does not contain any sort of information about the structure of the RDF source. Another valuable example of these works is LODStats [3]. Here, the RDF graphs are scanned at triple level and in the post-processing phase, structural information such as the class and property hierarchies are discovered.

In the instance-level approach the RDF Graph is inspected by 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 [12, 20] in which the proposed instance-level summary can support efficient and federated query evaluation. Another valuable example of these group is [7] in which an approach that allows to enrich knowledge bases with OWL2 axioms is described. The information extracted by [7] overlaps the intensional knowledge that can be present within the triples of a dataset; LODeX is able to extract the triples belonging the intensional knowledge through an ad hoc algorithm reducing the time and the complexity of this task.

All these techniques have been tested using a small number of datasets and taking as input the dump of an RDF graph; instead, LODeX has been designed to be used with a wide number of SPARQL endpoints in an online environment.

The most important example of LOD indexes are Void descriptors [1]. The Void descriptors are a W3C standard used for expressing metadata about RDF datasets. In particular, they are primarily used to describe links among different datasets rather than the structure of the dataset itself. Despite they report valuable information, their definition is demanded to the producer of the LOD dataset, thus, not all the datasets are equipped with VOID descriptors. LODeX makes use of some of the VOID descriptors and adds further indexes to supply more detailed information about the structure of the dataset.

3 Intensional and Extensional Knowledge in LOD sources

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 [18] [10].

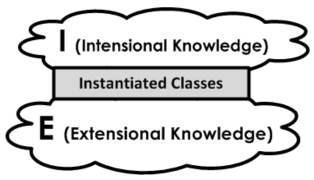


Fig. 1: Data structure of a generic LOD dataset

We can think the entire set of RDF triples partitioned between *Intensional Knowledge (I)* and *Extensional Knowledge (E)*. The triples belonging to the (**I**) group define the terminology used in the dataset. They are expressed in RDF, however they can be usually interpreted through RDFS or OWL and seen, with some restrictions, as the T-Box component of the knowledge base described in the endpoint [16]. The (**E**) group of triples usually covers most of the datasets 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 [16].

These two kinds of knowledge are distinct, according to their semantic meaning, but they form a unique RDF graph. The triples belonging to (**E**) and (**I**) are connected by particular classes, that we will call instantiated classes, defined in (**I**) and instantiated in (**E**) that act as a bridge among the two resources (as represented in Figure 1).

A well designed dataset should contain both intensional and extensional knowledge, however by analyzing a large number of endpoints we have observed that this is not generally true. Sometime, LOD datasets are biased toward a kind of knowledge (intensional or extensional). For example, there are LOD sources that contain ontologies in which instances are not present, on the other hand some datasets include only one class (*owl:Class*) and a large number of instances that are improperly used as such. In other cases, LOD datasets define 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 as RDF sources). 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 RDB2RDF[4].

4 Architectural Overview

LODeX aims to be totally automatic in the information extraction and in the production of the indexes. Therefore, it does not require any kind of a priori knowledge about the dataset on which it works.

Figure 2 illustrates the architecture of LODeX. The tool takes as input just the URL of a SPARQL endpoint and perform the Index Extraction (IE) process where the

[4] http://www.w3.org/2001/sw/rdb2rdf

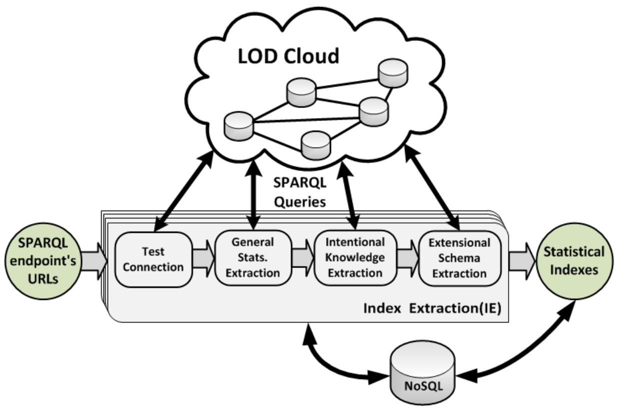


Fig. 2: LODeX Architecture

queries, able to extract structural and statistical information about the source, are generated. 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. Finally, the indexes are stored into a NoSQL Database. We have chosen a NoSQL document database server, MongoDB[5] [4], because it allows a flexible representation of the indexes and, in particular, it can easily manage heterogeneous lists of elements.

The architecture has been designed to parallel the processing of multiple endpoints, thus exploiting the idle times caused by response-time delays of single endpoints. 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 needs some operators introduced with SPARQL 1.1 [11], 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 these problems, we have limited the use of SPARQL 1.1 operators and have defined particular *pattern strategies* to scale the complexity of the queries.

5 Index Extraction Process

The indexes extracted by the IE process through SPARQL queries[6] can be grouped in three categories according to the kind of knowledge they stored: *Generic*, *Intensional*, *Extensional* (see also Table 1).

[5] https://www.mongodb.org
[6] A complete list of the query patterns is available at http://dbgroup.unimo.it/lodexQueries

Table 1: LODeX statistical indexes. Legend: Cn: class name; Pn: property name; s: subject; p: property; o: object; n: number of times a path (or a propery) exists; nI: number of instances

Name	Description	Structure	Category
t	Number of Triples	Integer	Generic
c	Number of Instantiated Classes	Integer	
i	Number of Instances	Integer	
Cl	Instantiated Class list	List(Cn,nI)	
Pl	Property list	List(Pn,n)	
IK	Intensional Knowledge Triples	List(s,p,o)	Intensional
Sc	Subject Class	List (s,p,n)	Extensional
Scl	Subject Class to Literal	List (s,p,n)	
Oc	Object Class	List (o,p,n)	

In the *Generic* group all the information regarding the size and the complexity of the dataset are reported. In particular, the first three elements (**t**, **c**, **i**) give an insight of the dimension of the RDF graph; while, the last two components (**Cl** and **Pl**) contain information about the instantiated classes and the properties usage. The queries used to extract these values refer to those used to create the Void Descriptors[7] of a dataset [1]. As mentioned before in Section 2, not all the LOD sources are provided of these indexes, therefore we added them in our list.

The *Intensional* group contains only the **IK** index, i.e. the list of all the triples that characterize the intensional knowledge of the dataset. The queries used to extract these triples are based on a simple triple pattern (subject, predicate and object), in which the subject is iteratively replaced with the URIs representing the constraints of the ontology, this in order to traverse the RDF graph and extract the intensional knowledge.

The *Extensional* group contains the information regarding the distribution of instances within the source. To extract the three indexes, the Subject and Object paths are inspected (see the paths on Figure 3). The first two indexes (**Sc**, **Scl**) refer to Subject paths (the first has an URI as object and the second a literal), while the last (**Oc**) regards the Object path. Each of these lists is described by three elements: **s/o** that is a *Subject Class/Object Class*, **p** that refers to a property and **n** that represent the number of times the path is used in the dataset.

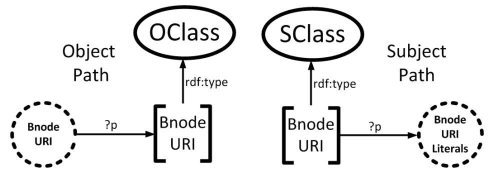


Fig. 3: Subject and Object Path

[7] https://code.google.com/p/void-impl/wiki/SPARQLQueriesForStatistics

The IE process starts by testing the connection to the endpoint (as shown in Figure 2) and the compatibility of the endpoint with the operators used in the queries. After this, the first three indexes (**t**, **c** and **i**) are extracted through simple SPARQL queries. The **Cl** and **Pl** indexes, instead, are extracted using the pattern strategies that deal with possible failure of the endpoint. The extraction of the intensional knowledge, such as **IK**, makes use of an iterative algorithm. In the end, the **Sc**, **Scl** and **Oc** indexes exploit pattern strategies to increase the success rate of their extraction.

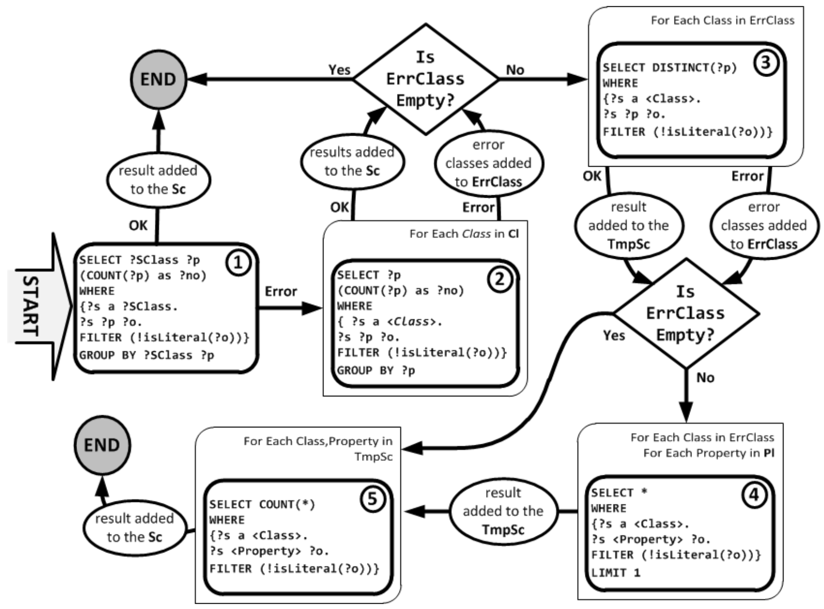
LODeX exploits different patterns through which it can produce queries of different complexity. With the introduction of the version 1.1 of SPARQL, it is possible to collect aggregated information about a specific Basic Graph Patterns (BGP) [13] over an RDF dataset using the operator GROUP BY. However, in many cases endpoints hosting large RDF datasets are not capable of providing a response before the timeout expires. We have chosen to use a restricted group of operators to minimize the complexity of the queries generated, i.e. GROUP BY, FILTER, COUNT, DISTINCT, AND. As stated in [19], the evaluation of an expression containing AND and FILTER can be solved in linear time, while the operator GROUP BY is more expensive in terms of performance. To handle performance problems, we have designed particular pattern strategies able to extract the same information resulting from complex GROUP BY queries using a higher number of low-complexity queries.

5.1 Pattern Strategies

We often stumbled across errors triggered by endpoints due to performance issues populating some of the indexes. In particular, this problem occurred when extracting the Subject Path (**Sc** and **Scl**), the Object Path (**Oc**) and in few cases the class and property lists (**Cl** and **Pl**). 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 occurs. Hence, we have designed a *pattern strategy* able to handle this type of error and scale the complexity of the SPARQL query.

In Figure 4 you can see a representation of the pattern strategy used to complete the extraction of the **Sc** index. By using the first query, it is possible to extract all the information in one go. If the endpoint is not able to answer to the first query, the strategy switches to the second step. In this case, a query for each class in **Cl** is generated, then, each successful response returns an element of **Sc**, while, if an error occurs the current class is added to the set of **ErrClass**. At the end of the second step, if some error still exists, the strategy tries to download the two items that compose each element of **Sc** (property name, and property count) separately. Thus, in the third and fourth steps the queries are generated 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. It is worth noting that while the first and second queries exploit the GROUP BY operator, the others does not. Therefore, it is possible to compute the **Sc** index even without the GROUP BY operator.

Similar strategies (with different queries) are used to complete the extraction of **Scl**,**Oc**,**Cl** and **Pl** indexes.


START
END
result added to the Sc
OK
Error
Yes
Is ErrClass Empty?
No
results added to the Sc
OK
error classes added to ErrClass
Error
For Each Class in Cl
For Each Class in ErrClass
OK
Error
result added to the TmpSc
error classes added to ErrClass
Is ErrClass Empty?
Yes
No
For Each Class in ErrClass For Each Property in Pl
For Each Class,Property in TmpSc
result added to the TmpSc
END
result added to the Sc
1
2
3
4
5

```
SELECT ?SClass ?p
(COUNT(?p) as ?no)
WHERE
{?s a ?SClass.
?s ?p ?o.
FILTER (!isLiteral(?o))}
GROUP BY ?SClass ?p
```

```
SELECT ?p
(COUNT(?p) as ?no)
WHERE
{ ?s a <Class>.
?s ?p ?o.
FILTER (!isLiteral(?o))}
GROUP BY ?p
```

```
SELECT DISTINCT(?p)
WHERE
{?s a <Class>.
?s ?p ?o.
FILTER (!isLiteral(?o))}
```

```
SELECT *
WHERE
{?s a <Class>.
?s <Property> ?o.
FILTER (!isLiteral(?o))}
LIMIT 1
```

```
SELECT COUNT(*)
WHERE
{?s a <Class>.
?s <Property> ?o.
FILTER (!isLiteral(?o))}
```

Fig. 4: Pattern strategy for extraction of the *Sc* index.

5.2 Intensional Knowledge Extraction Algorithm

The intensional knowledge contained in a generic dataset usually consists in few triples within the dataset with an 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 extract the IK index.

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, the algorithm has to avoid to traverse and download the entire graph and instances, since this could cause the deadlock of the IE process. Fortunately, we can take advantage of the structure induced by RDF. 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 pattern 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. Easy to do as the URIs that relate properties, are used as subject only in the intensional knowledge. The pseudo-code of the IK Extraction Algorithm is shown in the following.

```
Data: Cl, Pl
Result: Ik
Qn=∅, Fn=Cl.cn ∪ Pl.pn;
while |Qn| < |Fn| do
    forall the node in Fn - Qn do
        results←generate query for node and query the endpoint;
        add node to Qn;
        forall the r in results do
            add r to Ik;
            if r.o is not a Literal then
                add r.o to Fn;
        end
    end
end
```

We have tested the algorithm on several datasets (an analysis of this evaluation is described in Section 6) and it always stops once it has downloaded the entire intensional knowledge without traversing the whole RDF graph. The number of iterations can give an estimation of the ontology deepness and complexity.

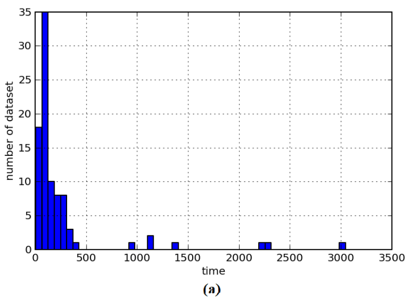
6 Test and Performance Evaluation

LODeX has been tested on the entire set of datasets taken from *SPARQL Endpoint Status*[8], a specialized application that recursively monitors the availability of public SPARQL Endpoints contained in DataHub. Table 2 reports the number of datasets that were examined. In first lines information related to the test connection are shown. Here, 469 endpoints have been tested, but unfortunately only 244 were online when tests were performed (May 2014). 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. Since LODeX uses only a subsets of SPARQL 1.1 operators, the IE process was successfully performed on 56% of the sources (137/244). At the same time, the number of endpoints fully compatible with SPARQL 1.1 was much lower; they were only 14, so the 5%[9]. Also the pattern strategy has demonstrated its effectiveness by increasing the number of endpoints which have completed the extraction phase, from 33 to 107. On these datasets, we have also evaluated the behavior of the IK Extraction Algorithm that usually stops after 5 iterations and only in few cases needs more iterations, till a maximum of 22 iterations.

In Table 3 statistics about the performance for the 107 datasets that have completed the extraction are shown. The average time of extraction is 6.12 minutes (the avg size of

[8] http://sparqles.okfn.org/

[9] as reported in http://sparqles.okfn.org/interoperability on May 4th, 2014.



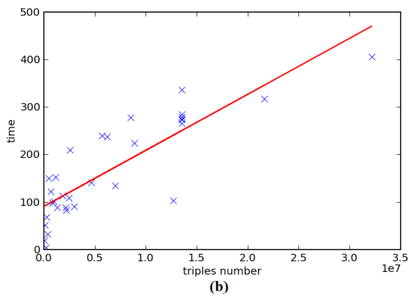


Fig. 5: (a) - Distribution of the extraction time (s) for the datasets for which the statistical indexes were successfully extracted. (b) - Extraction time (s) and number of triples

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 on multiple endpoints; using 9 parallel processes the extraction time decreases to just 3.35 hours. This is an optimal result if compared to similar tools such as SchemEx that was able to analyze 2.17 billion of triples in 15 hours [15].

Table 2: Numerical information on the evaluated sources

Dataset URLs	469
Reachable datasets	244
SPARQL 1.1 compatible	137
Extraction completed	107
Extraction without Pattern Strategy	33

Table 3: Performance of the IE process on 107 datasets

AVG time of extraction	6,12 minutes
Total time (single process)	11,15 hours
Total time (9 processes)	3,35 hours
Total triples	3,45 billions

Figure 5(a) reports the index extraction time on the 107 datasets. It can be seen that more than 90% of the datasets have completed the extraction in less than 500 seconds[10]. The correlation between the execution time and the number of instances for this 90% of the datasets is shown in Figure 5(b).

The heterogeneity on the implementation of the SPARQL endpoints is one of the most critical aspects and it also dramatically affects the performances of LODeX. To highlight this issue, in the right part of Table 4, we have compared the characteristics of three datasets: KEGG Pathway (knowledge on the molecular interaction and reaction networks), Dbnary (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 compared to the second. DBLP is a

[10] 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).

Table 4: statistical indexes comparison over three datasets and Pearson correlation between extraction time and other features for all the dataset

	KEGG Pathway	Dbnary	DBLP in RDF (L3S)	Pearson correlation
Triples number	49.859.159	39.393.237	Error	0.72
Instance number	11633810	8217804	54939	0.56
Class number	32	42	6	0.44
Property number	161	171	25	0.50
Extraction Time	19 minutes	1,5 minutes	Error	1

borderline case; although it is less complex than the first two datasets, the extraction process has not been completed.

We have also investigated which of the dataset features has the greater impact on the extraction time by using the Pearson product-moment correlation coefficient. The coefficient values are presented in the left part of Table 4. The number of triples obtains the higher value of correlation. This proves that there is a high degree of linear dependence between the extraction time and the size of the dataset as previously demonstrated in Figure 5(b).

7 Conclusions And Future Work

Starting from the URL of a SPARQL Endpoint, LODeX is able to automatically provide a set of statistical indexes that describe the LOD datasource. In this paper, we presented the architecture and the algorithms that composed LODeX and showed an evaluation of the tool on a significant number of LOD sources available on the SPARQL Endpoint Status portal. The results obtained are satisfactory and stimulate further developments and optimizations of LODeX.

We made use of the statistical indexes for the schema generation and summarization of LOD sources developing an online tool [5]. Here, a LOD source is visually represented by a schema summary displayed by a web application[11]. Users can interact with the visual representation of the dataset and focus on the information that they are more interested in.

We envision that LODeX might become an assistance tool for LOD portals. In fact, since LOD portals already provide basic search functionalities over sources, an iterative search process by using both portal's and LODeX's functionalities, might strongly improve the selection of useful LOD datasets.

References

1. K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing linked datasets. In *LDOW*, 2009.
2. R. Angles and C. Gutierrez. Querying rdf data from a graph database perspective. In *The Semantic Web: Research and Applications*, pages 346–360. Springer, 2005.

[11] An online demo of the web application can be find at http://www.dbgroup.unimo.it/lodex

3. S. Auer, J. Demter, M. Martin, and J. Lehmann. Lodstats–an extensible framework for high-performance dataset analytics. In *Knowledge Engineering and Knowledge Management*, pages 353–362. Springer, 2012.
4. K. Banker. *MongoDB in action*. Manning Publications Co., 2011.
5. F. Benedetti, S. Bergamaschi, and L. Po. A visual summary for linked open data sources. To appear in International Semantic Web Conference (Posters & Demos), 2014.
6. C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. Linked data on the web (ldow2008). In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 1265–1266, New York, NY, USA, 2008. ACM.
7. L. Bühmann and J. Lehmann. Universal owl axiom enrichment for large knowledge bases. In *Knowledge Engineering and Knowledge Management*, pages 57–71. Springer, 2012.
8. K. Christodoulou, N. W. Paton, and A. A. Fernandes. Structure inference for linked data sources using clustering. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 60–67. ACM, 2013.
9. T. Gottron, M. Knauf, S. Scheglmann, and A. Scherp. Explicit and implicit schema information on the linked open data cloud: Joined forces or antagonists. In *the 11th International Semantic Web Conference*, 2012.
10. V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *EON*, volume 87, 2003.
11. S. Harris and A. Seaborne. Sparql 1.1 query language. *W3C working draft*, 14, 2010.
12. A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data summaries for on-demand queries over linked data. In *Proceedings of the 19th international conference on World wide web*, pages 411–420. ACM, 2010.
13. O. Hartig, C. Bizer, and J.-C. Freytag. Executing sparql queries over the web of linked data. In *The Semantic Web-ISWC 2009*, pages 293–309. Springer, 2009.
14. P. Jain, P. Hitzler, P. Z. Yeh, K. Verma, and A. P. Sheth. Linked data is merely more data. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*. AAAI, 2010.
15. M. Konrath, T. Gottron, S. Staab, and A. Scherp. Schemex−efficient construction of a data catalogue by stream-based indexing of linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 16:52–58, 2012.
16. G. Lakemeyer and B. Nebel. *Foundations of Knowledge representation and Reasoning*. Springer, 1994.
17. A. Langegger and W. Woss. Rdfstats-an extensible rdf statistics generator and library. In *Database and Expert Systems Application, 2009. DEXA'09. 20th International Workshop on*, pages 79–83. IEEE, 2009.
18. Z. Pan and I. Horrocks. *Description Logics: reasoning support for the Semantic Web*. University of Manchester, 2004.
19. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):1–45, Aug. 2009.
20. F. Prasser, A. Kemper, and K. A. Kuhn. Efficient distributed query processing for autonomous rdf databases. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 372–383. ACM, 2012.