Analyzing Online Schema Extraction Approaches for Linked Data Knowledge Bases

Tobias Zeimetz
zeimetz@uni-trier.de
Trier University
Trier, Germany

Ralf Schenkel
schenkel@uni-trier.de
Trier University
Trier, Germany

ABSTRACT

In order to help data curators, data scientists, and other users in the domain of Linked Data to identify potentially new data sources, it is important to understand the corresponding data schema. The schema can help to determine the domain of the data (e.g. bibliographic data, geospatial data, etc.), its structure and more. We analyzed several strategies and systems which extract schemaa in an online approach. Established systems using SPARQL endpoints for online schema extraction are limited when knowledge bases are very large or complex. Due to the growth of Linked Data, the knowledge bases has become larger and their structure more complex. Therefore, this paper will discuss some limitations of current strategies.

CCS CONCEPTS

• **Information systems → Database utilities and tools**; **Database performance evaluation**; **Database utilities and tools**; **Database performance evaluation**.

KEYWORDS

Data Summarization, Schema Inference, Schema Extraction, Knowledge Extraction

ACM Reference Format:
Tobias Zeimetz and Ralf Schenkel. 2019. Analyzing Online Schema Extraction Approaches for Linked Data Knowledge Bases. In *The International Workshop on Semantic Big Data (SBD'19), July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3323878.3325808

1 INTRODUCTION

If a user tries to familiarize himself with an unknown database and understand its structure, it is practical to have access to the schema, as in the case of relational databases. In case of Linked Data a developer is not bound to provide a schema in form of an RDFS [4] or OWL [1] ontology. Linked Data is stored in RDF [3] format, which is a standard model to transfer data on the Web. In order to discover the schema of a database, a user would have to formulate multiple queries. The query language to request data from a knowledge base is called SPARQL [2].

In order to extract the schema of a Linked Data knowledge base some complex SPARQL queries are needed; depending on the complexity and size of the explored knowledge base, this can be a time-consuming task. Especially the time required to determine the domain of a knowledge base and which kind of data is stored at the endpoint is quite a hindrance. Furthermore, the user needs to be able to formulate SPARQL queries.

In order to extract the schema of an endpoint in an acceptable time and effort, we examined a number of current systems [5, 13–22, 25]. Most approaches are so-called offline approaches, where the user needs to download a data dump and extract the schema offline. Such approaches have the disadvantage that data dumps are not up-to-date or not provided.

Only few systems extract the schema using the SPARQL endpoint of the knowledge base. This approach has the benefits that we do not need to process data dumps and that the information is as up-to-date as possible. Typical disadvantages are the high response time of endpoints or that sometimes no response is delivered.

We only analyzed systems that extract the schema in an online approach (LODeX [5, 15–17], ViziQuer [22] and LD-VOWL [21]). Thereby we have set our focuse on LODeX. It automatically extracts a set of indices containing representative information regarding the structure (schema) of a chosen SPARQL endpoint. LODeX only needs the URL of an endpoint to extract and create the schema of an endpoint. A schema shows which data types and classes are connected to each other and which properties are used for this purpose. It also presents information about the used properties for each class and indicates whether the properties points to a literal. Besides, it is desirable to have information about how often a class-property pair is used, in order to determine the domain (e.g. bibliographic data, spatial data, etc.) of the endpoint.

Our objectives in this paper are to determine limitations of current systems and to highlight which problems are still unsolved. We believe that this information is important in order to show the lack of sufficient solutions. We use LODeX as an example system to highlight some limitations and later on show, which limitations can be found in other systems. We chose LODeX for a more detailed analysis because it is one of the most promising systems. With many other systems, the limitations could be observed quite quickly, e.g. LD-VOWL or ViziQuer.

The remainder is structured as follows. In Section 2 we discuss several current systems. Afterwards, in Section 3, we focus on the LODeX system and explain how it works. In Section 4 we examine limitations of current systems and also in general. Lastly, we present a detailed evaluation of the discussed systems.

SBD'19, July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6766-0/19/07…$15.00
https://doi.org/10.1145/3323878.3325808

2 RELATED WORK

As already mentioned in Section 1, schema extraction can be divided into two groups. The first group of algorithms works on data dumps and can therefore ignore server problems (offline approach). The second group extracts the schema using the endpoint (online approach) of a knowledge base. As already stated by Weise et al. in [21] there are only few works in the domain of online schema extraction of knowledge bases since most approaches are offline approaches. Some papers like ViziQuer [22] deal with the topic of visual query languages at schema level. They do not describe how they derive the schema because the focus is on the query language. Exploring the data of an endpoint can also help to get an understanding of the schema. Such approaches are pursued by linked data explorers such as LodLive [14]. They have the disadvantages that a user actually needs to get an understanding of the knowledge base, by exploring trough the graph. Besides, the exploration of large endpoints can also take a lot of time and efforts. For the sake of completeness, we will briefly discuss several systems that extract the schema in an offline approach.

SchemEx [19] is a system for real-time indexing and schema extraction of knowledge bases. Konrath et al. follow a stream-based approach to crawl the data from the web. A limitation of this system is that it does not consider class instances and therefore cannot retrieve class properties.

LODatio [25] is a semantic search system that uses the schema-level index of SchemEx to find relevant Linked Data sources. It provides a ranked result list of (possibly) relevant data sources. In addition, it provides meta information that allows a quick judgement on the importance of data sources.

Pham et al. [20] describe techniques that allow to extract the structure from a knowledge base, denoted as emergent relational schema. The described techniques use characteristic sets [26] for extracting a schema. Neumann et al. describe in [26] that in RDF multiple triple are used describe the same object. They concluded, that it is possible to characterize an entity by its properties. To extract the characteristic set, Pham et al. use a bulk loader and furthermore, count the frequency for each characterization. Although such a set may be well suited to describe the structure of a knowledge base, it is not the actual structure. Information such as class hierarchies are hidden to the user.

The first online approaches are made by systems that help a user to understand the structure of a knowledge base via an exploration tool (e.g. LodLive [14] or RelFinder [23]) or by providing the user with statistics (e.g. RDFStats [13]).

LodLive [14] can be used to browse RDF resources, link resources stored in different endpoints and to discover new connections. RelFinder [23] presents an approach that extracts graph covering relationships between two data objects. A major disadvantage of both systems is that the user must already have a rough understanding of the structure. Both systems only provide a deeper understanding of the structure instead of a schema.

Zviedris et al. present in ViziQuer [22] a tool that extracts the schema from a SPARQL endpoint and allow users to graphically inspect the extracted schema. ViziQuer only needs the address of an endpoint to start with the extraction process. A limitation is that it only supports typed data and it is needed that an SPARQL endpoint does not have any limit to answer size. In addition, Zviedris et al. disadvise to test their approach on large knowledge bases like DBpedia since such large schemas are not easy to explore. Because their work is based on the exploration and querying part of knowledge bases, they do not provide much information on how they extracted the schema.

Figure 1: Schema of a Knowledge Base

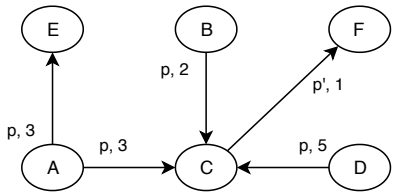


Table 1: Sc and Oc Index

Sc	P	No.
A	p	6
B	p	2
C	p'	1
D	p	5

Oc	P	No.
E	p	3
C	p	10
F	p'	1

Kellou-Menouer et al. present in SchemaDecrypt [18] an approach for discovering a versioned schema for SPARQL endpoints. SchemaDecrypt enables the discovery of different structures of existing classes in a knowledge base. It shows which versions of classes and types exist whereby a version of a class is characterized by the combination of its properties.

LD-VOWL [21] is a system to extract and visualize the top k classes of a knowledge base. Not all classes of an endpoint are needed in order to determine the domain of the knowledge base, only the top k classes are extracted. LD-VOWL is based on a class-centric perspective, i.e. the classes are extracted first and afterwards connected by properties and enriched by data types. A big advantage of this approach is that only the most used schema information is extracted. This way the user is not flooded with information. The major disadvantage of this approach is that operators like ORDER BY must be used. Especially weak servers or servers with large amounts of data are quickly brought to their limits.

Benedetti et al. [5, 15–17] proposed LODeX, an approach that creates a set of indices that enables schema extraction of a knowledge base. They state that these indices collect statistical information regarding the size and complexity of the knowledge base (e.g. number of instances, etc.), but also present all the instantiated classes and the properties among them. The indices, extracted from an endpoint will later be used to generate the schema. The schema is generated offline by only using the previous extracted indices. However, LODeX has (nowadays) two major problems: (1) depending on the complexity of the knowledge base, it will extract an erroneous schema (with missing or additinal connections between classes) and (2) it does not work on large endpoints (e.g. Wikidata [6] ord DBpedia [9]).

As stated by Normey et al. in [24] there is no comprehensive literature survey/review that summarizes current systems, their limitations and soundness. They have reviewed 31 systems, 7 of which can be classified in the domain of Linked Data. Since the paper is not oriented in one specific domain, they cannot describe current problems and limitations (which is a mayor shortcoming) but rather features that are promised by the reviewed systems. In addition, they checked if the systems are scalable, support parallelization, support schema evolution and more. This literature review also shows weaknesses, because it does not sufficiently describe the problems and limitations of current systems.

Query 1: Query Sc Index

```
select ?sc ?p (count(?p) as ?no)
where { ?s a ?sc . ?s ?p ?o .
filter(!isLiteral(?o))}
group by ?sc ?p
```

Query 2: Query Oc Index

```
select ?oc ?p (count(?p) as ?n)
where { ?s ?p ?o . ?o a ?oc . }
group by ?oc ?p
```

3 LODEX

LODeX only needs the URL of an SPARQL endpoint as input and extracts afterwards several indices to later on create a schema for the corresponding endpoint. First, we will focus on the extraction and afterwards on the schema generation algorithm.

3.1 General Approach

As described in [16], the entire set of RDF triples (of an endpoint) can be split into two categories, denoted as *intensional knowledge* (**IK**) and *extensional knowledge* (**EK**). Triples contained in the IK set define the terminology. Benedetti et al. state that triples belonging to the EK set usually cover most of the structures of a data set and contain the real world entities of a knowledge base.

The *extensional knowledge index* (**EI**) contains information about class connections, what kind of properties are used and how often they are instantiated. The EI can be split into three subsets: subject class index (**Sc**), subject to literal index (**Scl**) and object class index (**Oc**). The Sc index contains pairs of (source) classes and properties that are connected to other classes. The Scl index contains pairs of (source) classes and properties that are connected with literals. The last index (Oc) contains pairs of object classes and the properties that connect a source class with the corresponding object class. In addition, all three indices also contain the number of instances for each class-property-pair contained in the data set. The core aspect of these indices is to determine how often entities from two classes are connected by a specific property.

To extract the mentioned indices Benedetti et al. use a so called *pattern strategy* [16]. The first step is to query the knowledge base with Query 1. If an endpoint does not respond to this query, they use several low-complexity queries and iterate class-wise over the knowledge base. The query presented in Query 1 only extracts the Sc index. To extract the Scl and Oc index, LODeX uses the same strategy just with different filters and slightly different queries. To extract the Scl index, we only have to remove the negation in the filter part and to extract the Oc index, Query 2 is used.

Benedetti et al. evaluated that for most endpoints an index extraction is possible. Only knowledge bases such as WikiData [6] or DBpedia [9] are way too large and results in a timeout.

3.2 Schema Generation

LODeX uses only the Sc and Oc indices to create the schema of an endpoint. The system checks for elements $s = (sc, p, n) \in Sc$ and $o = (oc, p, n) \in Oc$ that have the same property p and infers that the classes $s.sc$ (source class of s) and $o.oc$ (object class of o) are connected via p.

As described in [5], an element $s \in Sc$ is selected first, followed by a matching element $o \in Oc$, such that $s.p = o.p$. The indices are iterated step by step to find a matching p. Here, sc is the source class that uses the property p. The use of sc with p occurs exactly n times. If there is a matching p, the two classes $s.sc$ and $o.oc$ are inferred to be connected via p. The last step also determines how often the two classes are connected via p. For this the minimum n of s and o is determined and assumed as connection frequency.

In Figure 1 shows an example of a schema with only two properties, p and p'. Table 1 presents the extracted indices. The first step is to choose the first $s \in Sc$ and the first matching $o \in Oc$, so that $s.p = o.p$. In this example the classes A and E are connected by the property p. It is evident that the minimum of n is three and therefore the classes A and E are connected three times via property p. Thereafter the number n for the class E can be set to zero because all connections for E got determined. In a last step LODeX reduces the number n of class A by the number of connections between A and E. This results in a new number of instantiated occurrences for class A and p.

Because now the number n of class E is set to zero, this object class will be ignored in the remainder of the connection process. Afterwards, LODeX will check for another matching element in Oc for A. Since A and C have the same property the algorithm will match both classes in the same manner as before. Because A has the lower number ($n = 3$), LODeX will assume, that the connection frequency between A and C via p is three. Afterwards, the number n of C will be decremented by three and the whole process goes on until all connections are calculated.

4 LIMITATIONS

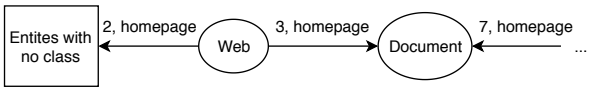
We will now focus on the limitations we have found in LODeX and other systems. Even if LODeX is able to extract the indices for most endpoints in a short time, the system suffers from some undetected errors that lead to an erroneous schema being extracted. Besides, we will show some general limitations of online approaches that will possibly lead to an erroneous schemas.

4.1 Limitations of LODeX

We can group the limitations of LODeX in two groups: (1) *non-typed entitiy errors* and (2) *conflicting property errors*.

All queries used for extracting the Sc index (in the pattern strategy) use the filter function *filter(!isLiteral())*. It is used to remove all properties that point to a literal such that we only get connections

Figure 2: Non-Typed Entity Example



between entities as result. Afterwards LODeX extracts the source class with the triple pattern ?*s a* ?*sclass*. If a data set is not well maintained, there may be untyped entities (entities that do not have a *rdf:type* relation). If a data set stores private homepages or links and does not mark them as literals the query cannot filter them and considers them as entities. Even if the retrieved properties are entities, they are not typed and have no class they belong to. That means, while some properties point to a typed entity and connect a given source class with an object class, several other properties will not connect a source class to another class. To understand the resulting problems in LODeX in more detail, take a look at Figure 2.

The Sc index for this example would contain one triple: the class *Web* and property *homepage* is instantiated five times. Furthermore, imagine that also other classes link via the *homepage* property to *Document*. The corresponding Oc index would result that the class *Document* is referenced ten times via *homepage*. According to the LODeX schema generation algorithm, both classes, *Web* and *Document*, would be connected via *homepage* five times. This result would be wrong because the two classes would only be connected three times. The reason for this lies in the *filter* function, which only filters literals but not untyped entities.

To fix this problem, we need to replace the *filter* function by an *optional* triple pattern: *optional*{?*o a* ?*oclass*}. This is the only way to filter untyped entities and get a correct connection frequency. However, even if we iterate over all source classes and use the pattern strategy, this kind of requests will stress most SPARQL endpoints to throw a timeout.

This limitation is not only a theoretical problem it also occurs in real world data sets. BNF [8] contains a *homepage* property[1] which also points to non-typed entities. For example, the class *InteractiveResource*[2] is connected to the class *Document*[3] via *homepage* 13 times. However, the LODeX schema generation algorithm determines that these two classes are connected 33 times via *homepage*. The reason therefore is that not every object entity of *homepage* is connected to a class/type. Besides, a homepage URL is not a literal and therefore cannot be filtered using *filter(!isLiteral(?o))*.

As described in Section 3.2, the whole schema generation process rests on the extracted indices. The idea to crawl in a first step, with a minimum of requests, all information and afterwards reuse the extracted information to generate a schema in order not to stress the SPARQL endpoint too much is a good approach. Unfortunately, it makes to a few hidden assumptions which do not always apply.

The first assumption is that the information of the extracted indices is enough in order to generate a sound schema. The second assumption is that the schema generation process is independent of the order of elements in the indices. We could already see in the previous part, that the information stored in the indices can be misleading and leads to errors in the class connection frequency.

In the following we will discuss the problem of *conflicting properties*. A property p is denoted as a conflicting property iff it is used more than once in both indices (Sc and Oc). Using the indices shown in Table 1 we can see that p is a conflicting property since it appears more than once in both indices. If we remove the element containing class C in this example, the property p would only appear once in the Oc index and would not yield as conflicting anymore.

To illustrate the problem of conflicting properties, assume we would just switch the order of the elements in Sc in Table 1, so that class B is now in the first position and class A is the second entry in Sc. The order of the remaining elements will not be changed.

The first step of the schema generation algorithm is to take the first entry $s \in Sc$ and search a matching element $o \in Oc$ so that $s.p = o.p$. After changing the order of Sc, the first element s contains class B. The first matching o contains class E with property p. According to the schema generation algorithm, a connection between B and E would be calculated. The result would be that class B and E will be connected by two instances in the data set via property p. However, if we check the original schema in Figure 1, we can see that this connection never existed.

If we perform the example to the end we can observe that connections between classes are missing too. We later present in Section 5 how many conflicting properties in real world data sets are existing. To work around this problem we have to send another query to the endpoint for all possible combinations of source and object classes for the corresponding conflicting property. In this query we would request whether there is a connection between two classes with the corresponding conflicting property and how often this occurs. This means that we cannot generate the schema with only the indices. Also, depending on the number of conflicting properties, a lot of additional requests would flood the SPARQL endpoint.

Even though the SPARQL query presented can be evaluated relatively quickly for most endpoints, we will show in Section 5 that the time for generating schemas is extended from a few minutes (7-10 mins.) to several hours (80-100 mins.), since many conflicting properties exist for HAL and BNF. This solution is not applicable for knowledge bases like Europeana, because they contain too many classes and too many conflicting properties.

4.2 Limitations in other Systems

As already mentioned, there are only a few papers dealing with schema extraction of SPARQL endpoints. Only the systems LD-VOWL, ViziQuer and the solution of Kellou-Menouer et al. are concerned with online approaches. However, only LD-VOWL and ViziQuer derive the actual schema of an endpoint. Unfortunately, the authors of ViziQuer do not provide information about how they extract the schema. The only statement is that a requirement for the system is that the server must not have a limit on the answer size. This is a strong assumption since most endpoints have a limitation on the size of the answer. Because of the lack of description it is not possible to evaluate ViziQuer.

Weise et al. provide a much more detailed description and present the used queries. Four different queries are used in order to extract the schema. The first is used to retrieve the top k classes. The second

[1] http://xmlns.com/foaf/0.1/homepage
[2] http://purl.org/dc/dcmitype/InteractiveResource
[3] http://xmlns.com/foaf/0.1/Document

to extract the top k properties that connected the retrieved classes. The third retrieves the top k data types of every class and afterwards the last query is used to retrieve the properties which are connected with the found data types. Especially the third step is a problem, because it needs the ORDER BY function. This function can stress the server heavily and may end in an timeout since it is needed to load all results to order them. In Section 5 we will present a detailed evaluation.

In addition, to the described limitations and known problems with SPARQL endpoints (server timeouts in case of too complex queries), there is another problem regarding Virtuoso endpoints. Virtuoso [11] is one of the systems that can be used to provide an SPARQL endpoint. It is widely spread and provides a solution for data access, virtualization and integration for SQL and RDF data sets. Since version 6.0 of OpenLink Virtuoso [7] they introduced a new feature denoted as *anytime queries*. This feature guarantees answers to arbitrary queries within a fixed time but possibly only a subset of the true result will be returned. They argue in [7], that this enforces a finite duration to all queries while returning meaningful partial results and is more user-friendly than a regular timeout. In case of aggregated results, it is possible that only a subset of the true result set will be counted and therefore an incomplete result set will be aggregated. This feature is not part of the current standard of the SPARQL protocol and therefore the standard does not provide any support for partial results. A Virtuoso endpoint can indicate that it returns only partial results by adding a S1TAT SQL state in the HTTP response header of the result set. If the query was not interrupted and successful the S1TAT SQL state will not be sent. Since this feature is not part of the standard we cannot use a Linked Data framework, e.g., Apache Jena (https://jena.apache.org/), to determine if an endpoint has returned partial results. Once we have a list or aggregation in the result set, we cannot be sure if we get a correct result. This is a major problem, since we rely on correctness. Since Virtuoso is one of the most used systems for SPARQL endpoints, this a problem that tackles all online schema inference systems and can lead to erroneous schema extraction.

5 EVALUATION

First we discuss whether and how frequently the described limitations occur in real world data sets. Afterwards we will examine the general soundness of the schemas derived using LODeX. In the last part we will evaluate how many timout problems really occur when using LD-VOWL.

5.1 Conflicting Properties

We will first examine, how many conflicting properties exist in real world data sets, e.g., BNF, HAL and Europeana [10]. We searched for properties in the extracted Sc and Oc indices that occur in both more than once. As presented in Table 2 with the exception of BNF not that many conflicting properties exists. The ratio of conflicting properties to normal properties is relatively small. Only BNF has more than nine percent conflicting properties.

We can see in Table 3 that BNF has an average of 2.63 possible source classes per conflicting property that can be connected with 2.05 target classes. If we examine HAL and Europeana, we can see that both use their conflicting properties in a more generic way because they have more possible source and target classes per property. Even though HAL and Europeana have less conflicting properties than BNF, the number of possible source and target classes per property is significantly larger. To calculate the number of additional queries we can simply multiply the average number of source classes (ASC) with the average number of object classes (AOC) and with the number of conflicting properties in a data set.

As we can see in Table 3, the number of additional queries is in case of HAL and Europeana higher than the number for BNF, even though they have less conflicting properties. Because of the additional requests the run time of the schema generation algorithm extends from approximately 8-10 minutes (for HAL and BNF) up to approximately 90 minutes. Note, that the problem of partial results is still unsolved and we cannot be sure, if we get a correct result. For this reason we conclude that this fix is not usable and scalable in a real world scenario.

Table 2: Conflicting Property Occurrences

Endpoint	Conflicting Properties	All Properties	Percentage
BNF [8]	272	797	34.13
HAL [12]	18	219	8.22
Europeana [10]	25	1038	2.41

Table 3: Average Source Classes (ASC) and Average Object Classes (AOC) per conflicting property and Number of Additional Queries (NAQ)

Endpoint	ASC	AOC	NAQ
BNF [8]	2.63	2.05	~ 1467
HAL [12]	11.84	9.61	~ 2048
Europeana [10]	34.82	24.13	~ 21006

Table 4: Correct Edges (CE), Erroneous Counted Edges (EC), Missing Edges (ME), Additional Edges (AE), Erroneous Edges (EE), Number of Edges in LODeX (LE), Original Number of Edges (OE)

Endpoint	CE	EC	ME	EE	LE	OE
BNF	437	346	1360	64	783	2162
HAL	82	221	572	139	303	875
Europeana	-	-	-	397	-	-

Soundness of LODeX

We will discuss now the soundness of the schemas generated by LODeX. We applied the algorithm to the endpoints BNF, HAL and Europeana. Here we have to take into account that due to the size and complexity of Europeana only some details could be evaluated. A more complex and comprehensive analysis of the schema generation algorithm was therefore performed using BNF and HAL.

Table 4 shows only results evaluated on the extensional knowledge of the corresponding SPARQL endpoints. Therefore, no *subClassOf* connections or other intentional knowledge relations between classes are considered. Furthermore, only relations between

Table 5: Evaluation of step three of LD-VOWL. Top-k classes, properties and data types are retrieved. The top five classes lead to the most timeouts, but are also the most interesting information for a user.

Endpoint	Step	k=5	k=10	k=20	k=30
BNF	3	5/5	5/10	5/20	5/30
HAL	3	4/5	4/10	4/20	4/30
Europeana	3	4/5	4/10	4/20	4/30

classes were examined, i.e. we did not evaluate connections between classes and literals. We discovered that LODeX generated in case of BNF 437 correct relations between classes (see Table 4). A correct relation means that the connection between two classes via a property actually occurs in the data set. In addition, the frequency of the connection must have been determined correctly.

We evaluated that 346 generated relations had erroneous frequency numbers, i.e. the relation between two classes actually existed but with a different frequency. Furthermore, the schema generated by LODeX missed 1360 relations, compared to a correct generated schema. In addition, LODeX created 64 relations between classes that does not exist in a valid schema.

As presented in Table 4, LODeX created 783 relations between classes but in reality BNF contains more than 2000 relations. The number relations generated by LODeX (LE) is the sum of the number of correct relations (CE) and the number of erroneous counted relations (EC). Because Europeana was too large and complex, we only validated how many relations (397) LODeX created that do not exist in the real data set. This result shows clearly that approaches as LODeX do not perform very well on real world data sets.

LD-VOWL Timeout Limitations

As already mentioned in Section 4.2, especially step three can stress an SPARQL endpoint and lead to a timeout. Step three extracts the top-k data types for each class.

As shown in Table 5 the endpoint of BNF had five timeouts for five sent queries ($k = 5$). Also, HAL and Europeana could not answer to all sent queries and had four timeouts. With a higher value of k we can see, that we have a less ratio of timeouts (e.g. $k = 10$, $k = 20$, etc.). The table shows clearly that the most commonly used classes (e.g. top five) can stress the endpoints most regarding step three. This means that LD-VOWL is not able to retrieve information about the most important classes of a knowledge base.

6 CONCLUSION

Virtuoso's anytime query feature, which also allows partial results, challenges all current systems. Since this feature does not belong to the SPARQL standard, it is not possible with current Linked Data frameworks to determine whether the received response is correct. Furthermore, we could show that LODeX cannot handle the complexity of current knowledge bases and therefore derives erroneous schemas. The idea of extracting indices in a first step and trying to derive a schema from them is a good approach to prevent SPARQL endpoints from being overwhelmed with too many (complex) queries. Unfortunately we were able to proof in our evaluation that by using the indices and when extracting them some sources of error crept in even with the presented fixes.

The authors of ViziQuer do not provide information about how they extract the schema and a requirement for the system is that the server must not have a limit on the answer size.

Also, LD-VOWL is not sufficient, because information about the most used classes of an endpoint cannot be retrieved. Therefore, we come to the conclusion that all current systems are not able to derive a correct schema. Even LD-VOWL, which extracts an approximate schema regarding the most used information, cannot solve this problem.

REFERENCES

[1] 2012. OWL 2 Web Ontology Language. https://www.w3.org/TR/owl2-syntax/. Last accessed: 15.03.2019.
[2] 2013. SPARQL 1.1 Overview. https://www.w3.org/TR/sparql11-overview/. Last accessed: 15.03.2019.
[3] 2014. RDF 1.1 Concepts and Abstract Syntax. https://www.w3.org/TR/rdf11-concepts/. Last accessed: 15.03.2019.
[4] 2014. RDF Schema 1.1. https://www.w3.org/TR/rdf-schema/. Last accessed: 15.03.2019.
[5] 2015. LODeX Model. http://dbgroup.unimo.it/lodex_model/lodex. Last accessed: 15.03.2019.
[6] 2018. WikiData. https://www.wikidata.. Last accessed: 15.03.2019.
[7] 2019. Anytime Queries. http://docs.openlinksw.com/virtuoso/anytimequeries/. Last accessed: 15.03.2019.
[8] 2019. BNF Bibliothèque nationale de France. http://www.bnf.fr/. Last accessed: 15.03.2019.
[9] 2019. DBpedia. https://wiki.dbpedia.org. Last accessed: 15.03.2019.
[10] 2019. Europeana. https://pro.europeana.eu/resources/apis/sparql. Last accessed: 15.03.2019.
[11] 2019. OpenLink Virtuoso. https://virtuoso.openlinksw.com/. Last accessed: 15.03.2019.
[12] 2019. The open archive HAL. https://hal.archives-ouvertes.fr/. Last accessed: 15.03.2019.
[13] W. Wöß A. Langegger. 2009. RDFStats - An Extensible RDF Statistics Generator and Library. (2009). https://doi.org/10.1109/DEXA.2009.25
[14] A. Antonuccio D. V. Camarda, S. Mazzini. 2012. LodLive, exploring the web of data. (2012). https://doi.org/10.1145/2362499.2362532
[15] L. Po F. Benedetti, S. Bergamaschi. 2014. A visual summary for linked open data sources. *CEUR Workshop Proceedings* 1272 (2014), 173–176.
[16] L. Po F. Benedetti, S. Bergamaschi. 2014. Online index extraction from linked open data sources. *CEUR Workshop Proceedings* 1267, January (2014), 9–20.
[17] L. Po F. Benedetti, S. Bergamaschi. 2016. Exposing the underlying schema of LOD sources. *Proceedings - 2015 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2015* 1 (2016), 301–304. https://doi.org/10.1109/WI-IAT.2015.99
[18] Z. Kedad K. Kellou-Menouer. 2017. On-line Versioned Schema Inference for Large Semantic Web Data Sources. *Proceedings of the 29th International Conference on Scientific and Statistical Database Management - SSDBM '17* (2017). https://doi.org/10.1145/3085504.3085513
[19] Mathias Konrath, Thomas Gottron, and Ansgar Scherp. 2011. Schemex–web-scale indexed schema extraction of linked open data. *Semantic Web Challenge, Submission to the Billion Triple Track* (2011), 52–58.
[20] O. Erling P. A. Boncz M. Pham, L. Passing. 2015. Deriving an Emergent Relational Schema from RDF Data. (2015). https://doi.org/10.1145/2736277.2741121
[21] F. Haag M. Weise, S. Lohmann. 2016. LD-VOWL: Extracting and Visualizing Schema Information for Linked Data Endpoints. (2016). http://ceur-ws.org/Vol-1704/paper11.pdf
[22] G. Barzdins M. Zviedris. 2011. ViziQuer: A Tool to Explore and Query SPARQL Endpoints. (2011). https://doi.org/10.1007/978-3-642-21064-8_31
[23] J. Lehmann S. Lohmann T. Stegemann P. Heim, S. Hellmann. 2009. RelFinder: Revealing Relationships in RDF Knowledge Bases. (2009). https://doi.org/10.1007/978-3-642-10543-2_21
[24] A. Marotta M. P. Consens S. Normey Gómez, L. Etcheverry. 2018. Findings from Two Decades of Research on Schema Discovery using a Systematic Literature Review. (2018). http://ceur-ws.org/Vol-2100/paper25.pdf
[25] B. Krayer A. Peters T. Gottron, A. Scherp. 2013. LODatio: using a schema-level index to support users infinding relevant sources of linked data. (2013). https://doi.org/10.1145/2479832.2479841
[26] G. Moerkotte T. Neumann. 2011. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. (2011). https://doi.org/10.1109/ICDE.2011.5767868