A SPARQL Endpoint Profiler for Efficient Question Answering Systems

Yasunori YAMAMOTO

Database Center for Life Science,
Univ. of Tokyo Kashiwa-no-ha Campus Station Satellite 6F.
178-4-4 Wakashiba, Kashiwa-shi, Chiba 277-0871, JAPAN
yy@dbcls.rois.ac.jp

Abstract. A question answering (QA) system querying SPARQL endpoints must know which endpoints have data to match a triple pattern of a given question. To develop a system that locates endpoints more efficiently, we propose a tool to collect several metadata of a SPARQL endpoint using SPARQL queries only. The metadata include IRIs of explicitly declared classes (*i.e.*, objects of the rdf:type predicate) as well as the number of triples whose subject and object belong to the respective classes. In addition, our tool counts triples whose objects are literals or any IRIs not appearing as the subject of the rdf:type predicate. The result is in RDF format using Vocabulary of Interlinked Datasets (VoID), Service Description (SD), and a newly developed vocabulary. We have collected data from various SPARQL endpoints and have provided them from our SPARQL endpoint. QA systems can easily utilize these data to improve their efficiency in finding endpoints by obtaining endpoints preemptively.

Keywords. SPARQL, Question Answering, Source Selection

1 Background

With the advent of triple data stores with the provision of SPARQL APIs and openly available datasets in Resource Description Framework (RDF) format, such as DBpedia [1] and UniProt [2], querying over multiple SPARQL endpoints is becoming a practical method to gather data to answer a question. There have been workshops of shared tasks that have reported this situation. Question Answering over Linked Data (QALD) has taken place three times[1]. The latest QALD (QALD-4) set up three tasks, and their target datasets were DBpedia, SIDER, Diseasome, and Drugbank. Other than DBpedia, the three datasets are in the life science domain, which has a relatively longer history of involvement in the RDF and the Semantic Web, and a certain number of RDF datasets, such as UniProt and Bio2RDF[2], are openly available in addition to those used in QALD.

To address the source selection issue, that is, how to find or choose an endpoint that provides datasets relevant to a given question effectively and efficiently, the in-

[1] http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/
[2] http://lod-cloud.net/state/

dustry held a workshop called PROFILES (1st International Workshop on Dataset PROFIling & fEderated Search for Linked Data). This workshop focused on data source contextualization for searching and exploring linked data, responding to the lack of scalable and usable methods for formulating and distributing semantic and keyword queries across the Web of Data [3].

We, the Database Center for Life Science (DBCLS), share this issue: additional datasets in Life Sciences have been recently released in RDF format. Examples include ChEMBL [4], Reactome [5], LinkDB[3], and Allie [6]. For this situation, we have developed a tool to obtain the profiles of the endpoints that helps a QA system find relevant datasets efficiently and effectively. In this context, a QA system accepts a question in natural language, converts it into a SPARQL query or set of sub-queries, determines the remote SPARQL endpoints, issues queries to them, and shows the result to the user. Our primary interest in the profiling issue is the lack of knowledge of the relationships among classes and literals as well as their statistics. Although some endpoints provide their ontologies, statistics, such as the number of triples where subjects and objects belong to certain classes, cannot be obtained. These metadata are used by a QA system to determine a SPARQL endpoint to obtain a dataset and answer a given query. For example, the question "What genes are associated with Alzheimer disease?" can be translated to a pseudo SPARQL query as follows[4]:

```
SELECT ?t1
  WHERE {
    ?t1 [:isa] [genes] .
    ?t2 [:isa] [alzheimer disease] .
    ?t1 [associated with] ?t2 .
}
```

In this query, terms within brackets are mapped to existing IRIs or literals by a QA system needing to find endpoints that provide datasets to answer the above question. More specifically, the first QA system task is to search for vocabularies that have terms mapped to each bracketed term. The second task is to find endpoints that have datasets corresponding to the vocabularies. In this study, we focus on the second task. Several groups [7-9] have studied this topic and demonstrated that VoID [10] datasets can be used. Although VoID is useful because its dataset can be obtained using the HTTP, the system cannot learn a number of elements useful for QA. It does not provide the relationships among classes and literals with respect to predicates along with these statistics. Using the above question as an example, a QA system attempts to determine which endpoints can provide triples that connect instances of [genes] and [alzheimer disease] with meaningful predicates related to [associated with]. VoID does provide the number of instances of association to each class but does not provide the statistics for such relationships.

Our objective is to collect these statistics from SPARQL endpoints and provide them through a public SPARQL endpoint, and to our knowledge, this is the first at-

[3] http://www.genome.jp/linkdb/linkdb_rdf.html

[4] http://lodqa.dbcls.jp

tempt. Our approach requires only the URLs of endpoints of interest. By using the metadata from the endpoints, a QA system can choose relevant endpoints before issuing specific SPARQL queries to answer a given question. In addition, our tool stores obtained data in RDF format, allowing everyone to share the data easily. We used VoID, SPARQL 1.1 Service Description (SD) vocabularies, and a vocabulary developed by our project called SPARQL Builder Metadata (SBM)[5].

2 Our approach

To collect the statistics of a SPARQL endpoint, our tool obtains all combinations of subject and object relationships with respect to classes and literals in the dataset provided. There are six triple types, that is, combinations of two and three cases for subject and object, respectively. Any subject can be a resource (*i.e.*, IRI or blank node) of a locally declared or undeclared class instance, and any object can be literal or identical to the case of the subject. A *locally declared class* includes resources that are objects of the rdf:type predicate in the dataset. For example, Tree is a locally declared class if there is a following triple in the dataset:

```
:sycamore rdf:type :Tree .
```

Accordingly, sycamore is a locally declared class instance. In addition, we call a resource a *locally undeclared class* (LUC) instance if it does not appear as a subject of the rdf:type predicate in the dataset. Then, our tool counts predicates with respect to each triple type mentioned above and each subject and object classes, respectively.

Figure 1 shows our proposed algorithm. In the algorithm, LOOP0 corresponds to a process for each graph, and LOOP1 is an inner loop of LOOP0 that addresses each class in the graph. The inner loop corresponds to the following cases:

1. Both the subject and the object of a triple belong to respective locally declared classes (*i.e.*, combinations of locally declared classes).
2. The subject of a triple belongs to a locally declared class, and its object is literal.
3. Either the subject or the object of a triple belongs to a locally declared class.

Here, process number 1 of the inner loop meets the first case, process number 2 meets the second case, and process numbers 3 and 4 meet the third case. As for process numbers 2 and 3 of LOOP0, the former corresponds to the case where the subject and the object of a triple are an LUC instance and a literal, respectively. The latter corresponds to the case where both the subject and object of a triple are LUC instances.

[5] http://www.sparqlbuilder.org/doc/?page_id=20

```
 Obtain a graph set G.
LOOP0:
 for each g in G
   1: obtain a locally declared class set C in g.
  LOOP1:
   for each c in C
     1: count the number of triples w.r.t each predicate
        whose subject and object belong to c and c'
        where c' is a locally declared class.
     2: count the number of triples w.r.t each predicate
        whose subject belongs to c,
        and whose object is literal.
     3: count the number of triples w.r.t each predicate
        whose subject belongs to c,
        and whose object is an LUC instance.
     4: count the number of triples w.r.t each predicate
        whose subject is an LUC instance,
        and whose object belongs to c.
   end for (LOOP1)
   2: count the number of triples w.r.t each predicate
      whose subject is an LUC instance,
      and whose object is literal.
   3: count the number of triples w.r.t each predicate
      whose subject and object are both LUC instances.
 end for (LOOP0)
```

Fig. 1. An algorithm to obtain statistics of a SPARQL endpoint.

After obtaining the data, our tool stores them in a triple store. We use three vocabularies to create an RDF dataset: VoID[6], SD[7], and SBM[8]. SBM was developed to express relationships between classes where a triple connects instances of classes as its subject and object. For example, an instance of the sbm:ClassRelation class has the properties sbm:subjectClass and sbm:objectClass, and the instance assumes to be an object of a void:Dataset instance with the predicate sbm:classRelation. Therefore, SBM can be used to extend the VoID data.

3 Scenario

Taking the above question as an example, we show how an obtained dataset can be used to determine an endpoint. There are three triple patterns as follows:

```
1: ?t1 [:isa] [genes] .
```

[6] http://rdfs.org/ns/void#
[7] http://www.w3.org/ns/sparql-service-description#
[8] http://www.sparqlbuilder.org/2014/05/rdf-metadata-schema#

```
2: ?t2 [:isa] [alzheimer disease] .
3: ?t1 [associated with] ?t2 .
```

As for the first pattern, a QA system needs to know that something is a gene, and Bio2RDF endpoints[9] can be used. Bio2RDF provides a set of SPARQL endpoints from life science datasets that include GenBank, Gene Ontology, and OMIM. Therefore, we assume that mapping from a pseudo class to a specific IRI is done based on vocabularies used in Bio2RDF. We use the prefix *bio2rdf:* to denote http://bio2rdf.org/. The question of interest is related to genes and disease, and there are the following classes:

bio2rdf:omim_vocabulary:Gene, and
bio2rdf:umls_vocabulary:Resource .

The latter class holds 29,602 instances[10], each of which corresponds to an UMLS Metathesaurus concept identifier (CUI). The UMLS Metathesaurus [11] is a very large multi-lingual vocabulary database in the life science domain, and CUIs are used to identify concepts and meanings in the Metathesaurus. The term Alzheimer's disease is represented as C0002395 using its CUI, and therefore the term can be expressed as bio2rdf:umls:C0002395 in the Bio2RDF dataset. At this point, the system must find relationships between bio2rdf:umls:C0002395 and instances that belong to bio2rdf:omim_vocabulary:Gene (we refer to this as the *Gene class* hereafter). As previously mentioned, mapping a pseudo class to a specific URI is a significant task, but not the focus of this study.

Once the mapping is completed, the next task is to identify a path between terms. By looking up the statistics obtained using our approach, the system can discover two groups of triples. Those of one group have a subject belonging to the Gene class, and those of the other have a subject belonging to the class of bio2rdf:umls_vocabulary:Resource (we refer to this as the *UMLS class* hereafter)[11]. However, triples of the latter group fall into one of the following three subgroups[12]:

1. Taking rdf[13]:type as its predicate
2. Taking void:inDataset as its predicate
3. Taking a literal as its object

The first and second groups indicate that those triples are not statements relevant to the question in the life science domain. The last group indicates that those triples do

---

[9] http://download.bio2rdf.org/release/3/release.html
[10] SELECT ?n { <http://bio2rdf.org/umls_vocabulary:Resource> ^void:class/void:entities ?n }
[11] SELECT (count(distinct ?c1) as ?cc1) (count(distinct ?c2) as ?cc2) {
?c1 sbm:subjectClass <http://bio2rdf.org/omim_vocabulary:Gene> .
?c2 sbm:subjectClass <http://bio2rdf.org/umls_vocabulary:Resource> . }
[12] SELECT ?p ?o {
[] sbm:subjectClass <http://bio2rdf.org/umls_vocabulary:Resource> ;
sbm:objectClass ?o ;
^sbm:classRelation/void:property ?p }
[13] http://www.w3.org/1999/02/22-rdf-syntax-ns#

not connect instances in the UMLS class to instances in another class. Therefore, the system can determine that it must search for triples whose subject belongs to the Gene class first.

The system then looks up the statistics of the triples whose subject belongs to the Gene class. Table 1 shows the top five predicates and object classes of triples whose subject belongs to the Gene class, in the order of their counts[14]. Next, the system looks up the statistics of each from the top down. Similar to the case of the UMLS class, triples whose subject belongs to the top class have literals as their objects or are not relevant to the life science domain, such as rdfs:Resource and dcterms:Dataset[15]. As for the second class, 33,659 triples connect their instances to ones belonging to the class bio2rdf:umls_vocabulary:Resource with the predicate of bio2rdf:omim_vocabulary:x-umls[16]. In this way, the system discovers the possibilities of connections from the Gene class to the UMLS class. The obtainable statistics are merely binary relations, and there may be no designated path. However, the system can determine whether there is a possibility or not, and this decision can be made without further inquiry to remote SPARQL endpoints. Therefore, the system will choose only promising endpoints to answer a given (sub) question.

Table 1. Top five predicates and object classes of triples whose subject belongs to the Gene class. The prefix omim denotes http://bio2rdf.org/omim_vocabulary:.

Count	Predicate	Class
608220	omim:x-gi	bio2rdf:gi_vocabulary:Resource
98639	omim:article	bio2rdf:omim_vocabulary:Resource
95443	omim:refers-to	bio2rdf:pubmed_vocabulary:Resource
81438	omim:refers-to	bio2rdf:omim_vocabulary:Gene
28032	omim:gene-symbol	bio2rdf:hgnc.symbol_vocabulary:Resource

Once the system learns that the endpoint of interest may have a dataset to answer the question, it issues a SPARQL query to retrieve an answer. By traversing the dataset, the system identifies the paths from instances of the Gene class to bio2rdf:umls:C0002395. Figure 2 shows the obtained paths. In Figure 2, an oval de-

[14] SELECT ?c ?p ?o {
[] sbm:subjectClass <http://bio2rdf.org/omim_vocabulary:Gene> ;
sbm:objectClass ?o ;
void:triples ?c ;
^sbm:classRelation/void:property ?p }
ORDER BY DESC(?c) LIMIT 5

[15] SELECT distinct ?o {
<http://bio2rdf.org/gi_vocabulary:Resource> ^sbm:subjectClass/sbm:objectClass ?o }

[16] SELECT ?p ?o ?c {
[] sbm:subjectClass <http://bio2rdf.org/omim_vocabulary:Resource> ;
sbm:objectClass ?o ;
void:triples ?c ;
^sbm:classRelation/void:property ?p .
FILTER(regex(str(?o),"umls","i"))}

notes a class and an arrow denotes a predicate. Note that we omit the prefix of http://bio2rdf.org/ from the class and predicate IRIs. As Figure 2 shows, there are multiple paths. For example, the omim_vocabulary:Phenotype class and the omim_vocabulary:Resource classes are connected to the Gene class by the omim_vocabulary:refers-to predicate. At the time of this writing, 170 instances of the Gene class have been retrieved, as shown in the following example:

1. AMYLOID P COMPONENT, SERUM; APCS [omim:104770]
2. PRION PROTEIN; PRNP [omim:176640]
3. REELIN; RELN [omim:600514]

The SPARQL query to obtain this result is as follows.

```
PREFIX omim: <http://bio2rdf.org/omim_vocabulary:>
PREFIX umls: <http://bio2rdf.org/umls:>

SELECT distinct (str(?l) as ?gene)
  WHERE {
    ?o a omim:Gene ;
      rdfs:label ?l ;
      ?p ?s .
    ?s omim:x-umls umls:C0002395 .

} ORDER BY ?gene
```

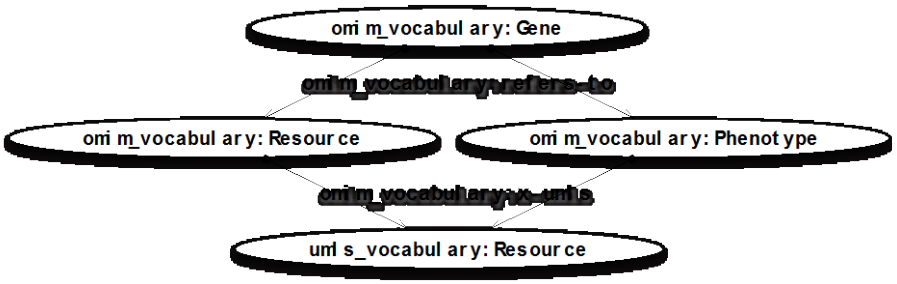

Fig. 2. Paths from the Gene class to the UMLS class.

4 Discussion and Conclusions

We have proposed a solution to the source selection issue. A QA system can choose a SPARQL endpoint efficiently and effectively by using metadata obtained by our proposed method. The efficiency is gained because a QA system issues the minimum possible number of SPARQL queries to remote endpoints. The effectiveness is gained because a QA system searches for possible paths corresponding to a graph pattern prior to issuing a query.

Currently, we note three issues concerning our approach. First, it takes time to gather endpoint data because our tool must issue multiple SPARQL queries. The more classes an endpoint graph has, the more queries the tool must issue. Letting C denote the number of classes, the tool issues approximately $4C+3$ queries. To alleviate the load on an endpoint, we added a 1/3 second default wait time between queries. Although collecting data requires time, our approach is not fatally impractical because datasets provided by currently accessible endpoints normally have small numbers of classes. According to the statistics provided by LODStats[17], representing 365 datasets, the median number of classes per dataset is 5.0. We have already demonstrated data collection from 22 endpoints and 39 datasets in the life science domain, and our approach performs as expected. The subject datasets are readily accessible through the SPARQL interface at http://tm.dbcls.jp/tdp/ .

The second issue is attributed to the implementations of target endpoints. There are implementations that do not support all operations conforming to SPARQL 1.1. For example, to obtain the number of triples whose subject belongs to an LUC, a query must contain the MINUS keyword, which was newly introduced in SPARQL 1.1. Fortunately, newly released implementations are expected to support these operations and will address this issue in the near future.

The third issue is that not all the datasets accessible through their SPARQL endpoints have domain-specific classes like Bio2RDF. For example, there is a SPARQL endpoint provided by the BioPortal project [12]. We can also obtain OMIM data from the dataset, but the only declared classes are owl:Class[18]. Therefore, the tool cannot obtain any useful statistics when using the relationships solely among the locally declared classes.

We continue to obtain the statistics of several endpoints and utilize them to develop a QA system in the life science domain. Beyond our proposed approach, we will study how to effectively map a given pseudo class to existing obtained classes. In addition, we hope that many endpoints provide the proposed statistics under the conditions that they can be openly accessed and shared with others to minimize the number of gathering actions at these endpoints. The tool is implemented in Java and distributed from https://bitbucket.org/yayamamo/tripledataprofiler under the terms of the MIT license.

Acknowledgments

This work has been supported by the National Bioscience Database Center (NBDC) of the Japan Science and Technology Agency (JST). We thank the SparqlBuilder team for providing us the SBM vocabulary before making it public.

[17] http://stats.lod2.eu/

[18] http://www.w3.org/2002/07/owl#Class

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, C., Ives, Z.: DBpedia: A Nucleus for a Web of Open Data. Proceedings of the 6th International Semantic Web Conference (ISWC2007) (2007)
2. Jain, E., Bairoch, A., Duvaud, S., Phan, I., Redaschi, N., Suzek, B. E., Martin, M. J., McGarvey, P., Gasteiger, E.: Infrastructure for the life sciences: design and implementation of the UniProt website. BMC Bioinformatics. 10:136. (2009)
3. Demidova, E., Dietze, S., Szymanski, J., Breslin, J.: Preface. Proceedings of the 1st International Workshop on Dataset PROFIling & fEderated Search for Linked Data. (2014)
4. Willighagen, E. L., Waagmeester, A., Spjuth, O., Ansell, P., Williams, A. J., Tkachenko, V., Hastings, J., Chen, B., Wild, D. J.: The ChEMBL database as linked open data. J Cheminform. 5(1):23. (2013)
5. Jupp, S., Malone, J., Bolleman, J., Brandizi, M., Davies, M., Garcia, L., Gaulton, A., Gehant, S., Laibe, C., Redaschi, N., Wimalaratne, S. M., Martin, M., Le Novère, N., Parkinson, H., Birney, E., Jenkinson, A. M.: The EBI RDF platform: linked open data for the life sciences. Bioinformatics. 30(9):1338-9. (2014)
6. Yamamoto, Y., Yamaguchi, A., Yonezawa, A.: Building Linked Open Data towards integration of biomedical scientific literature with DBpedia. J Biomed Semantics. 4(1):8. (2013)
7. Rakhmawati, N. A., Umbrich, J., Karnstedt, M., Hasnain, A., Hausenblas, M.: A Comparison of Federation over SPARQL Endpoints Frameworks. In Knowledge Engineering and the Semantic Web (pp. 132-146). Springer Berlin Heidelberg. (2013)
8. Görlitz, O., Steffen S.: SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In: Proc. of the 2nd Int. Workshop on Consuming Linked Data (2011)
9. Nikolov, A., Schwarte, A., Hütter, C.: FedSearch: Efficiently Combining Structured Queries and Full-Text Search in a SPARQL Federation. In The Semantic Web–ISWC 2013 (pp. 427-443). Springer Berlin Heidelberg. (2013)
10. Describing Linked Datasets with the VoID Vocabulary, http://www.w3.org/TR/void/
11. UMLS® Reference Manual [Internet], http://www.ncbi.nlm.nih.gov/books/NBK9684/
12. Whetzel PL., Noy, NF., Shah, NH., Alexander, PR., Nyulas, C., Tudorache, T., Musen, MA.: BioPortal: enhanced functionality via new Web services from the National Center for Biomedical Ontology to access and use ontologies in software applications. Nucleic Acids Res. 39(Web Server issue):W541-5. (2011)