How is your Knowledge Graph Used: Content-Centric Analysis of SPARQL Query Logs*

Luigi Asprino[1] [0000−0003−1907−0677] and Miguel Ceriani[2,3] [0000−0002−5074−2112]

[1] University of Bologna, Via Zamboni 33, Bologna, Italy
[2] University of Bari Aldo Moro, Via Orabona 4, Bari, Italy
[3] ISTC-CNR, Via S. Martino della Battaglia 44, Roma, Italy
luigi.asprino@unibo.it
miguel.ceriani@uniba.it

Abstract. Knowledge graphs (KGs) are used to integrate and persist information useful to organisations, communities, or the general public. It is essential to understand how KGs are used so as to evaluate the strengths and shortcomings of semantic web standards, data modelling choices formalised in ontologies, deployment settings of triple stores etc. One source of information on the usage of the KGs is the query logs, but making sense of hundreds of thousands of log entries is not trivial. Previous works that studied available logs from public SPARQL endpoints mainly focused on the general syntactic properties of the queries disregarding the semantics and their intent. We introduce a novel, content-centric, approach that we call *query log summarisation*, in which we group the queries that can be derived from some common pattern. The type of patterns considered in this work is *query templates*, i.e. common blueprints from which multiple queries can be generated by the replacement of parameters with constants. Moreover, we present an algorithm able to summarise a query log as a list of templates whose time and space complexity is linear with respect to the size of the input (number and dimension of queries). We experimented with the algorithm on the query logs of the Linked SPARQL Queries dataset showing promising results.

Keywords: SPARQL · Query log summarisation · Linked SPARQL queries.

1 Introduction

Knowledge Graphs (KGs) are pervasive assets used by organisations and communities to share information with other stakeholders. For knowledge engineers, it is essential to understand how KGs are used so as to assess their strengths and shortcomings, but, neither established methodologies nor tools are available. We observe that it is customary to make KGs accessible via SPARQL endpoints,

* An extended version of this paper (pre-print) is available at https://doi.org/10.6084/m9.figshare.23751243

therefore their query logs, i.e. the list of queries evaluated by the endpoint, are a valuable source from which the use of the KGs can be pictured. Compared to logs of "traditional" (centralised) databases (both relational and NoSQL), logs of public SPARQL endpoints bear much more information because they show usage of a dataset by multiple agents (human or robotic), for multiple applications, in different ways, and even in the context of multiple domains (especially if the dataset is generic).

Several works had already analysed the available SPARQL logs [29,2,32,16,37,11,10,38,9]. Most of them centred the analysis on the general structure of the queries (usage of specific SPARQL clauses, the shape of the basic graph patterns). The output of the analyses is mostly quantitative, possibly coupled by some examples. Relatively less focus has been so far given to aspects that go beyond the general query syntactic structure and relate to the actual content, such as aspects ranging from the usage of specific RDF terms (both classes, properties, and individuals), to specific (sub)query patterns, to inference of template usage and query evolution. Analysis of the actual content of queries can lead to further quantitative results, but most importantly can be used as a tool for qualitative analysis of one or multiple query logs: different levels of abstractions on the queries enable a meaningful exploration of the given data set.

The potential usage contexts for such analysis are manifold. For example, maintainers of SPARQL endpoints could optimise the execution of common queries by caching results or indexing predicates; designers of ontologies could assess what predicates are actually used thus allowing reshaping the model with shortcuts or removing unused predicates; designers of semantic web standards could introduce new constructs and operators in order to address common query patterns; and, researchers of the field could design benchmark to assess the performance of SPARQL endpoints.

The present work introduces a novel general approach to analyse query logs with a focus on query content and qualitative information. Specifically, we frame the *query log summarisation* as the problem of finding a list of templates modelling a query log. We introduce an algorithm able to solve the problem whose time and space complexity is linear in the size of the input. Finally, we experiment with the algorithm on the logs available in the LSQ dataset [37] to evaluate its usefulness. The analysis of the results shows that the method is able to provide more concise representations of the logs and novel insights on the usage of 28 public SPARQL endpoints.

The rest of the paper is organised as follows. Section 2 gives an overview of the existing work on query logs analysis. Section 3 lays the theoretical foundation of the work and introduces the problem of query log summarisation. The proposed algorithm to address the problem is presented in Section 4. Section 5 describes the experimental evaluation and its results, discussing strengths and opportunities enabled by the proposed approach. Section 6 concludes and outlines the ongoing and future work.

2 Related Work

Query logs are insightful sources for profiling the access to datasets. Although there are no approaches that aim to summarise SPARQL query logs as a list of query templates, an overview of the main approaches to analysing query logs is worthwhile. We classify the approaches according to the target query language.

Approaches targeting SQL query logs. Even if not directly applicable to assess the usage of knowledge graphs, techniques analysing query logs of relational databases may be adapted as SQL and SPARQL have syntactic similarities. These techniques have been used for detecting anomalous access patterns [21], preventing insider attacks [26] and optimising the workload of database management systems [14] thus becoming standard features for automatic indexing in commercial relational databases [28,31]. All the approaches can be generalised as feature extraction methods needed for clustering queries and profiling user behaviour. In most cases, the features extracted are basic, such as the SQL command used (e.g. SELECT, INSERT), the list of relations queried, and the operators used. Nevertheless, similarly to our approach, query templates and structural features are also used for computing query similarity [22,44], albeit still in a clustering approach. Some issues of such feature-based clustering approaches are that finding a useful way to convey the meaning of the clusters is not trivial, that scalability can be a problem as the worst-case cost is quadratic, and that some aspects of the query are scraped since the beginning for performance reasons, while they may be a relevant facet of a common pattern. Specifically, some methods [22,44,43] replace all the constants in the query with placeholders as a pre-processing step, which for SPARQL would hide the intent of most of the queries. Our method also replaces the constants with placeholders in an initial phase but, crucially, keeps the mapping with the original constants and puts them back if they have always the same value in a group of queries.

Approaches targeting SPARQL query logs. Analyses of SPARQL query logs have been performed since the early years of the Semantic Web. These studies fall into a more general line of research adopting empirical methods for observing typical characteristics of data [4,6], identifying common patterns in data [5], assessing the usage and identifying shortcomings of data [24,3] and using the obtained insights for developing better tools [20]. This kind of analysis has been also promoted by international workshops, such as USEWOD[4] which from 2011 to 2016 fostered research on mining the usage of the Web of Data [25]. Most of the existing work focus on quantitative and syntactic characteristics, such as the types of clients requesting semantic data [29] (including analyses of the characteristics of queries issues by humans, called organic, and those sent by artificial agents, robotic queries [9,36]), the user profile [19], the number of triple patterns per query [29,2,32,41,16], the use of predicates [29,2,32], the use of SPARQL operators [2,32,41,16] or a specific function (e.g. REGEX [1]), the

[4] `http://usewod.org/workshops.html`

structure of the Basic Graph Patterns (e.g. the out-degree of nodes, the number of join vertices) [2,41], the monotonicity of the queries [16], the probabilistic safeness [38], and the presence of non-conjunctive queries [32]. However, the analysis is limited at the triple-pattern level by paying less attention to the structural and semantic characteristics of the queries, thus making it difficult to figure out what the prototypical queries submitted to the endpoints look like. A noteworthy exception is [35], in which the author, while analysing queries at the triple pattern level, attempts to extract generic query patterns.

Bonifati et al. [11] investigate the structural characteristics related to the graph and hypergraph representation of queries by outlining the most common shapes. Moreover, they analyse the evolution of queries over time, by introducing the notion of the streak, i.e., a sequence of queries that appear as subsequent modifications of a seed query. By grouping queries based on similarity, this aspect of their work is akin to the approach presented in this work.

The existing studies are valuable for assessing the usage of SPARQL as a standard query language or for benchmarking and optimising the query engines. However, none of the existing approaches provides any insight into how KG is actually queried in terms of KG patterns queried by the users, and, therefore are of little help in designing the KGs. This paper investigates an alternative approach aiming at extracting query templates from SPARQL logs that may help designers to characterise the prototypical queries submitted by the users.

3 Preliminaries

This Section lays the theoretical foundation of this work.

RDF and SPARQL. For the sake of completeness, we introduce the basic notions of RDF [13] and SPARQL [17] needed to understand the methods and analysis described in this work. We defer the reader to the corresponding documentation for a complete description of these standards. Formally, let I, B, and L be infinite sets of IRIs, blank nodes, and literals. The sets are assumed to be pairwise disjoint and we will collectively refer to them as *RDF terms*. A tuple $(s, p, o) \in (I \cup B) \times (I) \times (I \cup B \cup L)$ is called *(RDF) triple* and we say s is the *subject* of the triple, p the *predicate*, and o the *object*. An *RDF graph* is a set of RDF triples, whereas an *RDF dataset* is a collection of named RDF graphs, each one identified by IRI, and a default RDF graph.

SPARQL is based on the idea of defining patterns to be matched against an input RDF dataset. Formally, considering the set of variables V, disjoint from the previously defined I, B, and L, a *triple pattern* is a tuple of the form $(s, p, o) \in (I \cup B \times V) \times (I \times V) \times (I \cup B \cup L \times V)$. A *basic graph pattern (BGP)* is a set of triple patterns. A SPARQL query Q is composed of the following components: *(i)* the query type (i.e. SELECT, ASK, DESCRIBE, CONSTRUCT); *(ii)* the dataset clause; *(iii)* the graph pattern (recursively defined as being a BGP or the result of the composition of one or more graph patterns through one of several SPARQL operators that modify and combine the obtained results); *(iv)* the solution modifiers (i.e. LIMIT, GROUP BY, OFFSET).

3.1 Query templates

Intuitively, a query template is a SPARQL query containing a set of placeholders which are meant to be substituted with RDF terms. The placeholders are called *parameters* of the query template and will be represented in queries using variable names starting with "`$_`"[5]. For example, consider the following queries 1.1 and 1.2[6]. The intent of both queries is to retrieve the types of a given entity. Such intent can be expressed via the Template 1.1.

Query 1.1:

```
SELECT ?type WHERE {
 dbr:Barack_Obama rdf:type ?type
}
```

Query 1.2:

```
SELECT ?type WHERE {
 dbr:Interstellar_(film) rdf:type ?type
}
```

Template 1.1:

```
SELECT ?type WHERE {
  $_1 rdf:type ?type
}
```

Query 1.3:

```
SELECT ?p WHERE {
 ?p rdf:type foaf:Person
}
```

Template 1.2:

```
SELECT ?type WHERE {
 $_1 $_2 ?type
}
```

We say that a query template q^t *models* a query q, indicated as $q^t \prec q$, if there exists a partial bijective function m^t, called *mapping*, that maps parameters P^t in q^t onto RDF terms of q such that applying m^t onto q^t gives q, i.e. $m^t : P^t \to (I \cup L)$ and $m(q^t) = q$. For example, the following mappings m_1 and m_2 transform the Template 1.1 into the queries 1.1 and 1.2 respectively: m_1`($_1) := dbr:Barack_Obama` and m_2`($_1) := dbr:Interstellar_(film)`.

It is worth noticing that, to preserve the intent of the query, templates do not substitute variables and blank nodes (as they are considered non-distinguished variables) with parameters, reduce the number of triple patterns, or replace SPARQL operators. As a result, a template for modelling a set of queries does not always exist (e.g. a single template modelling queries 1.1, 1.2, and 1.3 can not exist). Moreover, multiple templates may model the same set of queries. For example, the Template 1.2 models the queries 1.1 and 1.2 (in this case m_1 and m_2 must also map `$_2` onto `rdf:type`, i.e. m_1`($_2) := rdf:type` and m_2`($_2) := rdf:type`). In fact, the number of parameters of a template allows us to formalise the intuition of more specific/generic template. We say that the Template 1.2 is more generic (or, less specific) of Template 1.1 as it maps a higher number of parameters. As a result, given a query q, the *most generic*

[5] Using the initial underscore in the variable name to identify parameters matches with existing practice [27], while using "$" visually helps distinguish the parameters from query variables that often start with "?"

[6] For brevity, the queries omit prefix declarations:

- `dbr: <http://dbpedia.org/resource/>`
- `rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>`
- `foaf: <http://xmlns.com/foaf/0.1/>`
- `dbo: <https://dbpedia.org/ontology/>`

template modelling q is the template in which all the IRIs and literals of q are substituted by parameters. Therefore, it is easy to see that given a set of queries (that can be modelled by a single template) it is always possible to derive the most generic template by substituting all literals and IRIs by parameters.

We characterised templates as queries having placeholders that are to be replaced by IRIs or literals. However, there are two extensions to this rule which are needed to capture very common patterns for paginating the results and injecting values into a query. One is the usage of placeholders in `LIMIT` and `OFFSET` clauses, which are the solution modifiers used to get a specific slice of all the results. Both clauses are always followed by an integer, specifying respectively the number and initial position of the query solutions. By allowing this, integers to be replaced by parameters, multiple versions of the same query in which only one or both are changed (e.g., changing the `OFFSET` to perform pagination) can be represented by the same template.

The second extension to the rule has been defined for another specific clause: `VALUES`. This clause is used to bind one or more variables with a multiset of RDF terms. It is thus a way to give constraints to a query with multi-valued data that could come from previous computations, possibly also other queries[7]. In the case of a `VALUES` clause, rather than replacing single RDF terms, a placeholder either replace the whole corresponding multiset of terms or none.

Even if they are not explicitly mentioned, all the SPARQL clauses and operators (FILTERs, OPTIONALs, UNIONs etc.) can be part of a query template. We only mentioned the VALUE, LIMIT, and OFFSET operators as they deserve special treatment.

One of the main intuitions behind the usage of query templates to study a log is that it can help to "reverse engineer" the methods and processes used to generate the queries. In order to discuss this aspect, we define a *query-source* as a specific and unique piece of code (which could nevertheless span multiple software components in complex cases) that is responsible for the generation (possibly based on parameters) and execution of a query. A template that models many queries in a log may capture a common usage pattern that spans multiple query sources or a broadly used single query source. Both cases can be of interest in the analysis of a query log.

3.2 Query log summarisation problem

We formally describe a query log and frame its summarisation as a theoretical problem. A SPARQL Query Log $l = [e_1, e_2, .., e_n]$ is a list of entries $e_i = (q, t, s, m)$ each representing the execution at a certain time t of a query q by a SPARQL endpoint s with associated metadata m. For the purpose of the algorithm presented below, the information of the SPARQL endpoint executing the query is only used to group together queries evaluated over the same KG, and we do not consider time and metadata. Therefore, for brevity, SPARQL query logs reduce to a sequence of queries $l = [q_1, q_2, .., q_n]$. Note that queries can be

[7] It is for example a recommended way to perform query federation [34].

repeated in a log, so for convenience, we define an operator Q to get them as a set (hence without repetitions): $Q(l) = \{q_i | q_i \in l\}$.

Given a query log $l = [q_1, q_2.., q_n]$, the *SPARQL log summarisation* is the problem of finding a set of query templates $Q_t = \{q_1^t, q_2^t, .., q_m^t\}$ (with $m \leq n$), called *log model*, such that for each query $q_i \in l$, there exists a query template q_j^t such that $q_j^t \prec q_i$. It is worth noticing that, since each query is the template of itself (in this case the mapping from placeholders to RDF terms is empty), a trivial solution to the problem is $Q_t = Q(l)$. Therefore, we have that the size of log model Q_t may range from 1, in the case that all the queries in the log are modelled by a single template, to $|Q(l)|$, when a common template for any pair of queries does not exist.

It is worth noticing that summarising a query log differs from evaluating the containment/equivalence of a pair of queries [12,33]. In fact, given a query q and its template q^t (i.e. $q^t \prec q$), q and q^t are (except for constants and parameters) the exact same query. Whereas evaluating the query containment/equivalence requires deciding if the result set of one query is always (i.e. for any dataset) contained into/equivalent to the result set of a *different* query. Of course, the two approaches, log summarisation and query containment/equivalence can be potentially combined to derive more succinct log models, but this is outside the scope of this paper.

Metrics. The aim of summarising a query log is to assist KG engineers in understanding how their KGs are queried. To do so, a KG engineer has ideally to go through the list of all the queries. Obviously, the shorter the list of queries to examine, the less effort from the KG engineer is required for the analysis. Intuitively, the benefit of using a log model instead of a full query log is to reduce the list of queries to examine. This benefit is proportional to the difference between the size of the log model and the size of the query log. However, one must consider that not all the query templates have the same informational value. In fact, we can consider that the more log entries a template models, the more informative it is (in other words, it allows the KG engineer to have an indication of a larger portion of the query log). Therefore, if the templates are ordered according to their informational value, the KG engineer would be able to analyse a large portion of the log by going only through the most informative templates.

To measure the impact of this on the informational value of a model we employ the concept of *entropy*. The entropy over a discrete random variable X, taking values in the alphabet $\mathcal{X}$ and distributed according to $p : \mathcal{X} \to [0, 1]$, is defined as follows [39]:

$$\mathrm{H}(X) := -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

Given a query log l and a model Q_t over it, we consider a random variable T taking values over the "alphabet" Q_t and distributed as the templates of Q_t are distributed over the log l. That is, with probability distribution p_{Q_t} defined as follows:

$$p_{Q_t}(q_i^t) = \frac{|\{q_j | q_j \in l, q_i^t \prec q_j\}|}{|l|}$$

We can thus measure the entropy of this distribution, which depends both on the log l and the model Q_t. The entropy corresponds to the average number of bits (considering base 2 for log) used to encode an item, which in our case is a template, in an optimal encoding. For a uniform distribution over n values, the entropy is $log(n)$, which is the number of bits required for a simple encoding of n values. If the values are not uniformly distributed a more efficient representation (as in a lossless compression) can be used, where more frequent values are represented with shorter encodings.

Recalling that the set of queries $Q(l)$ is already a model of l, the one created by simply taking all the queries as they are, we can compute the entropy for this model. The aim of another computed model Q_t of l is to achieve a more concise representation of the log and thus lower entropy. In the experiments with a dataset of logs (cf. Section 5), we measure the entropy of $Q(l)$ (indicated as $H(Q)$) as opposed to that of a derived log model Q_t (indicated as $H(T)$). The difference between $H(Q)$ and $H(T)$ indicates how much less information needs to be screened by the KG engineer to examine the log.

4 Approach

We describe the procedure for query log summarisation. Appendix A of the extended version of the paper contains the complete pseudo-code for the algorithm, the sketch of the proof of soundness, the detailed complexity analysis, and other formal considerations on the output of the algorithm. To convey the intuition, we use the following log as a running example l =[Query 1.1, Query 1.2, Query 1.1, Query 1.4, Query 1.2, Query 1.5] where Queries 1.1 and 1.2 are defined above and Queries 1.4 and 1.5 follow.

Query 1.4:

```
SELECT ?director ?starring WHERE {
   dbr:Pulp_Fiction dbo:director ?
        director .
   dbr:Pulp_Fiction dbo:starring ?
        starring .
}
```

Query 1.5:

```
SELECT ?director ?starring WHERE {
   dbr:Django_Unchained dbo:director
        ?director .
   dbr:Django_Unchained dbo:starring
        ?starring .
}
```

Template 1.3:

```
SELECT ?director ?starring WHERE {
   $_1 $_2  ?director .
   $_3 $_4  ?starring .
}
```

Template 1.4:

```
SELECT ?director ?starring WHERE {
   $_1 dbo:director  ?director .
   $_1 dbo:starring ?starring .
}
```

Intuitively, the algorithm performs two steps, called GENERALISE() and SPECIALISE(). The function GENERALISE() creates a generic template for a

query, replacing each occurrence of IRIs and literals with a different new parameter. Therefore, the generated template is the most generic that models the query. At the same time a mapping is created, associating each parameter with the RDF term that was replaced. For example, GENERALISE(Query 1.1) returns the Template 1.2 and the mapping m_1 defined as follows: m_1(**$_1**) := `dbr:Barack_Obama`, m_1(**$_2**) := `rdf:type`; GENERALISE(Query 1.2) returns the Template 1.2 and the mapping m_2 defined as follows: m_2(**$_1**) := `dbr:Interstellar_(film)`, m_2(**$_2**) := `rdf:type`; GENERALISE(Query 1.4) returns the Template 1.3 and the mapping m_4 defined as follows: m_4(**$_1**) := `dbr:Pulp_Fiction`, m_4(**$_2**) := `dbo:director`, m_4(**$_3**) := `dbr:Pulp_Fiction`, m_4(**$_4**) := `dbo:starring`; GENERALISE(Query 1.5) returns the Template 1.3 and the mapping m_5 defined as follows: m_5(**$_1**) := `dbr:Django_Unchained`, m_5(**$_2**) := `dbo:director`, m_5(**$_3**) := `dbr:Django_Unchained`, m_5(**$_4**) := `dbo:starring`.

The function SPECIALISE() takes as input a template and an associated set of mappings and, by just analysing the set of mappings, it establishes if the number of parameters can be reduced. There are two interesting cases for this purpose: *(i)* for a parameter, all the mappings in the set map it to the same RDF term (it is thus a constant); *(ii)* for a pair of parameters of a template, each mappings in the set maps them to a common RDF term (one parameter is actually a duplicate of the other). For each instance of these cases, the template and the mappings are updated accordingly: *(i)* in the first case (the parameter is constant), the parameter in the template is replaced by the constant and removed from the mappings; *(ii)* in the second case (two parameters mapped to the same RDF terms), one parameter in the template is replaced by the other and removed from the mappings. For example, both m_1 and m_2 map **$_2** to `rdf:type` which can be considered as a constant (i.e. m_1(**$_2**) = m_2(**$_2**) = `rdf:type`), therefore the Template 1.2 can be specialised as Template 1.1 and the parameter **$_2** replaced with `rdf:type`. Concerning the Template 1.3 and the mappings m_4 and m_5, the SPECIALISE function replaces **$_2** and **$_4** with two constants (`dbo:director` and `dbo:starring`) and unifies **$_1** and **$_3** in both mappings as they map to the same RDF term (`dbr:Pulp_Fiction` and `dbr:Django_Unchained` respectively for m_4 and m_5). The function returns the Template 1.4 and m_4 and m_5 updated.

The main function DISCOVERTEMPLATES(): *(i)* takes a set of queries; *(ii)* extracts a pair (template, mapping) for each query by invoking GENERALISE; *(iii)* accumulates the mappings associated with the same template into a dictionary (the dictionary uses the templates as keys and mapping sets as values); *(iv)* then, for each pair (template, mapping set), calls SPECIALISE() and, possibly, replaces the pair with a specialised one.

Furthermore, along with the mappings, the algorithm maintains the original query ids, which in turn allows to find the data of each corresponding execution in the log. Keeping track of this relationship is crucial so that is later possible to derive statistics based on their usage or explore the detail of specific executions.

Properties of the extracted log model. It is worth noticing that, given a query log, the algorithm first maximizes the number of queries a single template can represent, by grouping each query under its most generic template. Then, the algorithm minimizes the number of parameters of each template, by returning the most specific template modelling that group of queries (in other words, it keeps a minimal set of parameters needed to represent the set of queries). This ensures that for any pair queries of the log, if a single template can model the queries, then, the template is in the log model and the template is the most specific one.

Moreover, since the algorithm does not perform any normalisation of the input queries, syntactic differences affect the templates, e.g. two queries having the same triple patterns in a different order result in two different templates. This implies that the extracted templates generalise over fewer input queries (hence the algorithm tends to extract more templates) in respect to what could be if some normalisation was adopted, but the extracted templates are closer to the queries sent by the clients (which is desirable for identifying queries sent from the same process). Some form query normalisation can then be included as a preliminary step for different perspectives, but this is left to future work.

Implementation of the algorithm. The algorithm has been implemented in Javascript, relying on the SPARQL.js library[8] for SPARQL parsing. Both the LSQ dataset in input and the discovered templates are represented as RDF in a local triple store, namely Apache Jena Fuseki[9]. The code is freely available on GitHub[10]

5 Experimentation

The LSQ dataset, already briefly introduced in Section 2, is the de-facto state-of-the-art collection of SPARQL query logs. We tested our method by using it to analyse all the logs available in the latest version of the LSQ dataset. In this section, we describe and discuss the dataset, its analysis, and the findings, focusing on the high level view and the details that can be useful to discuss the algorithm. For the detailed description of the results obtained for each endpoint and the full code of all the templates we refer the reader respectively to Appendix B and C of the extended version of the paper.

5.1 The Dataset

The LSQ 2.0 dataset[11] contains information about approximately 46M query executions and is composed of logs extracted from 28 public SPARQL endpoints. 24 of the endpoints are part of **Bio2RDF**, a project aimed at converting to

[8] `https://github.com/RubenVerborgh/SPARQL.js`
[9] `https://jena.apache.org/documentation/fuseki2`
[10] `https://github.com/miguel76/sparql-clustering`
[11] `http://lsq.aksw.org/`

RDF different collections of heterogeneously formatted structured biomedical data [8]. The other four endpoints are the following ones: **DBpedia**, a well-known knowledge base automatically extracted from Wikipedia [7]; **Wikidata**, an encyclopedic knowledge graph built collaboratively [42]; **Semantic Web Dog Food (SWDF)**, a dataset describing research in the area of the semantic web [30]; **LinkedGeoData** [40], an RDF mapping of OpenStreetMap, which is, in turn, a user-curated geographical knowledge base [15].

The LSQ project provides the collection of these SPARQL logs and their conversion to a common (RDF-based) format. In the process of conversion, the LSQ software performs also some filtering (e.g., only successful queries are considered) and anonymisation (e.g., client host information is hidden). The main information items offered by LSQ from each entry of a query log are the following ones: the endpoint against which the query was executed; the actual *SPARQL query*, the *timestamp* of execution, and an anonymised identifier of the client *host* which sent the query.

Dataset	Execs	Hosts	Queries	H(Q)	Templ.s	H(T)	ΔH
Bio2RDF	**33 829 184**	**2 306**	**1 899 027**	**15.22**	**12 296**	**3.73**	**11.49**
DBpedia	**6 999 815**	**37 056**	**4 257 903**	**21.16**	**17 715**	**5.58**	**15.59**
DBpedia-2010	518 717	1 649	358 955	17.99	2 223	5.66	12.33
DBpedia-2015/6	6 481 098	35 407	3 903 734	21.01	15 808	5.21	15.80
Wikidata	**3 298 254**	**-**	**844 260**	**12.26**	**167 578**	**7.47**	**4.80**
LinkedGeoData	**501 197**	**25 431**	**173 043**	**14.24**	**2 748**	**4.78**	**9.46**
SWDF	**1 415 568**	**921**	**101 422**	**14.54**	**1 826**	**1.03**	**13.51**

Table 1: Statistics on the LSQ 2.0 dataset before/after summarisation.

Table 1 shows some statistics about the data in the LSQ dataset, organised by endpoints[12]. The column *Execs* indicates the number of query executions contained in the log. Column *Hosts* is the total number of client hosts and *Queries* is the number of unique queries. The column H(Q) is the entropy of the unique queries distribution across the executions.

5.2 Methodology of Analysis

The aforementioned templates-mining algorithm was applied separately on each query log in the LSQ 2.0 dataset, with the corresponding set of queries as input.

[12] In the table, for conciseness, the statistics of the Bio2RDF endpoints are shown only aggregated for the whole project. In Appendix B of the extended version of the paper there is a more detailed version of the table showing the statistics endpoint by endpoint.

Furthermore, the queries of Bio2RDF were also considered as a whole, on top of analysing each specific endpoint[13]

The templates obtained with our method can be analysed in a variety of ways. Different statistics can be computed on top of this summarised representation of the original data. Furthermore, the templates can be explored in several ways to have a content-based insight of how an endpoint has been used. In this study we will focus on two main aspects:

- a quantitative analysis of the effectiveness of the summarisation by measuring for each log 1) the number of templates in comparison with the number of queries and 2) the entropy of the templates distribution in comparison with the entropy of the query distribution;
- a qualitative analysis of the templates obtained, choosing for each log the ten most executed ones and discussing the possible intent of the queries, what they say about the usage of the endpoint, which ones probably come from a single code source, which ones instead probably correspond to common usage patterns, if and how some of them are related between each other.

It should be noted many other perspectives are possible (some of them will be sketched among the future work in Section 6).

5.3 Results

The execution of the algorithm overall took approximately nine hours on consumer hardware. Statistics about the results for each log or set of logs are shown in Table 1, alongside the previously described information. The column *Templ.s* corresponds to the number of templates generated, while the column $\mathrm{H}(T)$ is the entropy of the templates distribution across the log and $\Delta\mathrm{H}$ is the difference between the entropy according to the unique queries and the one according to the templates ($\Delta\mathrm{H} = \mathrm{H}(Q) - \mathrm{H}(T)$).

For all the logs the number of templates is significantly smaller than the number of unique queries, with a reduction amounting to around two orders of magnitude (the ratio going from ∼56 to ∼240) for all cases but Wikidata (for which the reduction is smaller, namely five-fold). The reduction in entropy considering the distribution using templates shows even more strongly the effectiveness of the summarisation, as the value is in all the cases greater than $log_2 \frac{|Q|}{|T|}$, which would be the reduction in entropy in case of uniform distributions, showing that the algorithm is able to merge the most relevant (in terms of executions) queries.

Furthermore, it is worth noticing that, regarding the DBpedia log, while there is a significant difference in the query entropy from the data of 2010 (17.99) to the ones of 2015/6 (21.01), in line with a ten-fold increase in both executions and unique queries, the respective entropies measured on templates distribution

[13] This choice is motivated by the fact that the Bio2RDF endpoints are part of the same project, the collected logs refer roughly to the same period, and there is considerable overlap in the clients querying the endpoints.

are much closer, actually sightly decreasing from 2010 (5.66) to 2015/6 (5.21). This is interesting because it shows that the template diversity remains stable, while the number and diversity of specific queries increase roughly as the volume of the executions. In our opinion this case also manifests the importance of using the entropy as an index of diversity, rather than just counting the total number templates (which is instead quite different between the two datasets, ∼2.2K against ∼16K).

Then, for each endpoint[14], we performed the qualitative analysis of the ten most frequently executed templates. As part of the interpretation of these templates, we labelled them using a functional syntax composed of the a name given to the function (template) and a name given to each parameter. Interestingly, the most executed templates are quite vary across different endpoints and fulfil different kinds of purposes. Some templates correspond to generic, content-independent, patterns, like the template from SWDF log labelled PROPERTIESANDVALUES(*resource*) that list all properties and values associated to a resource and has been executed ∼17K times. Others are specific of some triple store software as they use specific extensions, as it is the case for as in the template COMMONSUPERCLASSANDDISTANCE(*class1*,*class2*) from Wikidata, executed ∼107K times, which employs a feature specific of Blazegraph, the software used for this dataset. Others are specific of some domain that the dataset encompasses, like CLOSEPOIS(*latitude*,*longitude*) from LinkedGeoData, executed ∼81K times, that looks for points of interest close to a geographic location. Some of them, finally, are specific of a certain application, like AIRPORTSFORCITY(*cityLabel*,*lang*) in DBpedia, executed ∼1.4M times,.

As previously mentioned, it can be of interest to understand if a template correspond to a single query-source or instead arises from a pattern which is common in the usage of an endpoint. While we do not propose a specific metric for this purpose, nor we have a general way to check the ground truth, the qualitative analysis of the most executed templates offers a chance to reason on this topic. The generality of the template, as accessed above, offers a hint: the more general the more likely that it correspond to commonly adopted pattern rather than a single query-source. But the analysis of the general-purpose templates found show that they are not necessarily simple and may not correspond to the most straightforward solution to design a certain query. The structural complexity is perhaps then a better predictor of the usage of a template. For example, the template TRIPLES(*subject*) in Bio2RDF is a `CONSTRUCT` that return all the triples for which *subject* is the subject. The query is hence functionally generic but it is peculiar for being in a form slightly more complex than necessary: it is composed of a triple pattern and a `filter` instead of using directly a triple pattern with fixed subject. This template has been executed across most of the endpoints of Bio2RDF, for a total of ∼9.3M times.

Another interesting aspect that emerges from the qualitative analysis is the evidence of relationships between different templates. For each endpoint, even considering just the most executed templates, it is possible to find one or more

[14] With the exception of the Bio2RDF endpoints, which are considered as a whole.

groups of templates that for structure, function, number of executions, hosts, period of use show many commonalities and can reasonably be conjectured to be part of a common process. For example among the most executed templates on SWDF four of them have been executed the same number of times and have the same kind of parameter (a researcher) albeit they extract different kind of data (respectively general information, affiliations, participation to events, publications). Still on SWDF, there are other two groups of templates having the same aspects in common (with a group having as common parameter an article and another having as common parameter an organisation). While in this case the grouped templates are probably part of a single process that executes multiple queries, in other cases the related templates could testify the evolution of a process. The template COMMONSUBCLASSES(*class1*, *class2*) from the LinkedGeoData log is executed ∼17K times across a span of ∼7 hours, then it is "replaced" by the template COMMONSUBCLASSES(*class1*, *class2*, *class3*) that fulfills the same purpose but having one class more as parameter. The second version is then executed ∼17K times across a span of other ∼7 hours.

Such hypothesises about the relationship between among a group of queries are reinforced in all the cases we found by the fact that the templates are executed by a common set of hosts. In most of the cases it is a single host that execute all the templates in a group, but not necessarily: on DBpedia the templates COUNTLINKSBETWEEN(*res1*, *res2*) and COUNTCOMMONLINKS(*res1*, *res2*) have different but related functions[15] on the same kind of parameters, they are both executed ∼181K times by the same set of ∼1130 hosts.

The complete results are available online for download[16] The templates found for each endpoint are represented both as CSVs and RDF. The RDF representation of the templates is meant to be used alongside the RDF representation of LSQ and is based on the Provenance Vocabulary [18], a specialisation of the standard W3C provenance ontology (PROV-O) [23] dealing with web data and in particular SPARQL queries and query templates.

5.4 Discussion

The aim of the analysis of the LSQ dataset was to prove that our method is able to effectively summarise the given logs, that the inferred templates often correspond to broadly used patterns or single query-sources, and that their analysis can give new insights on the usage of the considered endpoints. We quantitatively measured the efficacy of the summarisation through the ratio of original queries per template and the reduction in entropy when considering each log entry as an instance of a template, rather than as an instance of a query. Both measures show that the summarisation had a noteworthy impact on all the considered logs. Moreover, the qualitative analysis of a selected sample of templates (specifically the most executed) shows how their function may be appropriately analysed and discussed, without the need to check the thousands of corresponding queries.

[15] One counts the triples in which one resource is subject and the other object, the other counts the triples in which they replace each other or have symmetric role.

[16] `https://doi.org/110.6084/m9.figshare.23751138`

Regarding the accuracy of the predicted templates in identifying a single source for a set of queries, there is no gold standard or previous attempt to compare with. Thus the qualitative analysis resorts to educated guesses, where we decide if an inferred template corresponds plausibly to a single source based on the syntactic distinctness and relationship with other templates and data from the log. For many of the described templates, it is possible to reasonably infer a single origin. In terms of the usefulness of the inferred templates to gain insights, the qualitative analysis has shown multiple ways in which the analysis of the templates gives direct access to information that was previously not straightforward and stimulates further study.

Finally, another finding has been that this template-based analysis paves the way to the analysis of another level of relationships between queries, namely when different queries are applied to the same (or related) data items as part of a (possibly automatic) process. Evidence of such relationships has been found in the qualitative analysis of all the considered logs.

6 Conclusions

In this work, we address the *query log summarisation* problem, i.e. identifying a set of *query templates* (i.e. queries with placeholder meant to be replaced with RDF terms) describing the queries of a log. We designed and implemented a method to perform the summarisation of a query log in linear time, based on the use of a hash table to group sets of queries that can be derived from a common query template. The approach has been experimented with the available logs of the LSQ dataset. The representation of the logs using templates has been shown to be significantly more concise. A qualitative analysis performed on the most executed templates enabled the characterisation of the log in ways that would not have been directly possible by analysing just the single queries.

Besides further exploring possible extensions of the template-mining algorithm for normalising the input log (e.g. reordering triple patterns), the analysis of the discovered templates brought forward some interesting issues that we consider deserving of further research.

One aspect worth investigating is the relationships between the execution patterns of each template. In the qualitative analysis, we found groups of templates being executed by the same set of hosts, often at similar times, and many times with the same parameters. Such analysis may, for example, allow to mine the prototypical interactions (namely, processes) with data, beyond the single query or template.

Moreover, many more interesting levels of abstraction are possible beyond the query templates: e.g., a common part of the query, the usage of certain BGP, a property, and so on. The general idea of the approach and the structure of the algorithm can be still applied. Apart from computing these multiple levels, which can be done by extending the presented algorithm, it is interesting to understand if some measure may be used to select the more relevant abstractions, rather than leaving the choice entirely to the user.

Another direction worth exploring is to assess the possible benefits of combining log summarisation with strategies for bot detection (e.g. templates can help characterise the features of queries and thus favouring the classification of robotic queries) or for optimising the execution of a sequence of queries (once prototypical interaction with data is delineated, one could imagine triple stores being able to predict workload and optimise query execution).

In this work, we mainly focussed on the most frequent queries, but, future analyses may also investigate what insights can be extracted from the rare ones (for example, a long tail of rare queries may indicate a high variety of clients and data exposed by the endpoint).

Finally, the proposed method and algorithm are applicable without much change to other query languages, thus offering an approach for the analysis of logs of, e.g. relational databases.

Supplemental Material Statement. The extended version of this paper (pre-print) is publicly available (see title note), as well as the dataset with the experimentation results (see note 16). The query logs used in the experimentation can be downloaded from the LSQ website[17]. The code is available from a public git repository (see note 10).

Acknowledgements This work was partially supported by the PNRR project "Fostering Open Science in Social Science Research (FOSSR)" (CUP B83C22003950001) and by the PNRR MUR project PE0000013-FAIR.

References

1. Aljaloud, S., Luczak-Rösch, M., Chown, T., Gibbins, N.: Get all, filter details-on the use of regular expressions in sparql queries. Proceedings of the Workshop on Usage Analysis and the Web of Data (USEWOD 2014) (2014)
2. Arias, M., Fernandez, J.D., Martinez-Prieto, M.A., de la Fuente, P.: An empirical study of real-world SPARQL queries. Proceedings of Usage Analysis and the Web of Data (USEWOD 2011) (2011)
3. Asprino, L., Basile, V., Ciancarini, P., Presutti, V.: Empirical analysis of foundational distinctions in linked open data. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence and the 23rd European Conference on Artificial Intelligence (IJCAI-ECAI 2018). pp. 3962–3969 (2018). https://doi.org/10.24963/ijcai.2018/551
4. Asprino, L., Beek, W., Ciancarini, P., van Harmelen, F., Presutti, V.: Observing LOD using equivalent set graphs: It is mostly flat and sparsely linked. In: Proceedings of the 18th International Semantic Web Conference (ISWC 2019), Part I. pp. 57–74 (2019). https://doi.org/10.1007/978-3-030-30793-6_4
5. Asprino, L., Carriero, V.A., Presutti, V.: Extraction of common conceptual components from multiple ontologies. In: Proceedings of the International Conference on Knowledge Capture (K-CAP 2021). pp. 185–192 (2021). https://doi.org/10.1145/3460210.3493542

[17] http://lsq.aksw.org/

6. Asprino, L., Presutti, V.: Observing LOD: its knowledge domains and the varying behavior of ontologies across them. IEEE Access **11**, 21127–21143 (2023). https://doi.org/10.1109/ACCESS.2023.3250105
7. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A Nucleus for a Web of Open Data. In: Proceedings of the International Semantic Web Conference (ISWC 2007). pp. 722–735. Springer (2007)
8. Belleau, F., Nolin, M.A., Tourigny, N., Rigault, P., Morissette, J.: Bio2rdf: towards a mashup to build bioinformatics knowledge systems. Journal of biomedical informatics **41**(5), 706–716 (2008)
9. Bielefeldt, A., Gonsior, J., Krötzsch, M.: Practical linked data access via SPARQL: the case of wikidata. In: Proceedings of the Workshop on Linked Data on the Web co-located with The Web Conference (LDOW@WWW 2018) (2018)
10. Bonifati, A., Martens, W., Timm, T.: Navigating the maze of wikidata query logs. In: Proceedings of The Web Conference (WWW 2019). pp. 127–138 (2019). https://doi.org/10.1145/3308558.3313472
11. Bonifati, A., Martens, W., Timm, T.: An analytical study of large SPARQL query logs. VLDB Journal **29**(2-3), 655–679 (2020). https://doi.org/10.1007/s00778-019-00558-9
12. Chekol, M.W., Euzenat, J., Genevès, P., Layaïda, N.: SPARQL query containment under SHI axioms. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2012) (2012)
13. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 Concepts and Abstract Syntax, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
14. Deep, S., Gruenheid, A., Koutris, P., Viglas, S., Naughton, J.F.: Comprehensive and efficient workload summarization. Datenbank-Spektrum **22**(3), 249–256 (2022). https://doi.org/10.1007/s13222-022-00427-w
15. Haklay, M., Weber, P.: Openstreetmap: User-generated street maps. IEEE Pervasive Computing **7**(4), 12–18 (2008). https://doi.org/10.1109/MPRV.2008.80
16. Han, X., Feng, Z., Zhang, X., Wang, X., Rao, G., Jiang, S.: On the statistical analysis of practical SPARQL queries. In: Proceedings of the 19th International Workshop on Web and Databases (2016). https://doi.org/10.1145/2932194.2932196
17. Harris, S., et al.: SPARQL 1.1 Query Language, http://www.w3.org/TR/2013/REC-sparql11-query-20130321/
18. Hartig, O.: Provenance information in the web of data. In: Proceedings of the Workshop on Linked Data on the Web (LDOW 2009) (2009), http://ceur-ws.org/Vol-538/ldow2009_paper18.pdf
19. Hoxha, J., Junghans, M., Agarwal, S.: Enabling semantic analysis of user browsing patterns in the web of data. Proceedings of Usage Analysis and the Web of Data (USEWOD 2012) (2012)
20. Huelss, J., Paulheim, H.: What SPARQL query logs tell and do not tell about semantic relatedness in LOD - or: The unsuccessful attempt to improve the browsing experience of dbpedia by exploiting query logs. In: Proceedings of ESWC 2015, Revised Selected Papers. pp. 297–308 (2015). https://doi.org/10.1007/978-3-319-25639-9_44
21. Kamra, A., Terzi, E., Bertino, E.: Detecting anomalous access patterns in relational databases. VLDB Journal **17**(5), 1063–1077 (2008). https://doi.org/10.1007/s00778-007-0051-4
22. Kul, G., Luong, D., Xie, T., Coonan, P., Chandola, V., Kennedy, O., Upadhyaya, S.J.: Summarizing large query logs in ettu. CoRR (2016), http://arxiv.org/abs/1608.01013

23. Lebo, T., Sahoo, S., McGuinness, D.: PROV-O: The PROV Ontology, https://www.w3.org/TR/2013/REC-prov-o-20130430/
24. Luczak-Rösch, M., Bischoff, M.: Statistical analysis of web of data usage. In: Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn2011) (2011)
25. Luczak-Rösch, M., Hollink, L., Berendt, B.: Current directions for usage analysis and the web of data: The diverse ecosystem of web of data access mechanisms. In: Proceedings of the 25th International Conference on World Wide Web (WWW 2016). pp. 885–887 (2016). https://doi.org/10.1145/2872518.2891068
26. Mathew, S., Petropoulos, M., Ngo, H.Q., Upadhyaya, S.J.: A data-centric approach to insider attack detection in database systems. In: Proceedings of the 13th International Symposium on Recent Advances in Intrusion (RAID 2010). pp. 382–401 (2010). https://doi.org/10.1007/978-3-642-15512-3_20
27. Meroño-Peñuela, A., Hoekstra, R.: grlc makes github taste like linked data apis. In: Proocedings of ESWC 2016. pp. 342–353 (2016)
28. Microsoft: Automatic Tuning - Microsoft SQL Server, https://learn.microsoft.com/en-us/sql/relational-databases/automatic-tuning/automatic-tuning?view=sql-server-ver16
29. Möller, K., Hausenblas, M., Cyganiak, R., Handschuh, S.: Learning from linked open data usage: Patterns & metrics. In: Proceedings of the Web Science Conference (2010)
30. Möller, K., Heath, T., Handschuh, S., Domingue, J.: Recipes for semantic web dog food - the ESWC and ISWC metadata projects. In: Proceedings of the 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference, ISWC-ASWC 2007. pp. 802–815 (2007). https://doi.org/10.1007/978-3-540-76298-0_58
31. Oracle: Automatic Indexing - Oracle SQL Developer Web, https://docs.oracle.com/en/database/oracle/sql-developer-web/19.2.1/sdweb/automatic-indexing-page.html#GUID-8198E146-1D87-4541-8EC0-56ABBF52B438
32. Picalausa, F., Vansummeren, S.: What are real SPARQL queries like? In: Proceedings of the International Workshop on Semantic Web Information Management (SWIM 2011) (2011). https://doi.org/10.1145/1999299.1999306
33. Pichler, R., Skritek, S.: Containment and equivalence of well-designed SPARQL. In: Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 14). pp. 39–50 (2014). https://doi.org/10.1145/2594538.2594542
34. Prud'hommeaux, E., Buil-Aranda, C.: SPARQL 1.1 Federated Query, http://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/
35. Raghuveer, A.: Characterizing machine agent behavior through sparql query mining. In: Proceedings of the Workshop on Usage Analysis and the Web of Data (USEWOD 2012) (2012)
36. Rietveld, L., Hoekstra, R., et al.: Man vs. machine: Differences in sparql queries. In: Proceedings of the Workshop on Usage Analysis and the Web of Data (USEWOD 2014) (2014)
37. Saleem, M., Ali, M.I., Hogan, A., Mehmood, Q., Ngomo, A.N.: LSQ: the linked SPARQL queries dataset. In: Proceedings of the 14th International Semantic Web Conference (ISWC 2015) Part II. pp. 261–269 (2015). https://doi.org/10.1007/978-3-319-25010-6_15
38. Schoenfisch, J., Stuckenschmidt, H.: Analyzing real-world SPARQL queries and ontology-based data access in the context of probabilistic data. International Jour-

nal of Approximate Reasoning **90**, 374–388 (2017). https://doi.org/10.1016/j.ijar.2017.08.005
39. Shannon, C.E.: A mathematical theory of communication. The Bell System Technical Journal **27**(3), 379–423 (1948). https://doi.org/10.1002/j.1538-7305.1948.tb01338.x
40. Stadler, C., Lehmann, J., Höffner, K., Auer, S.: Linkedgeodata: A core for a web of spatial open data. Semantic Web **3**(4), 333–354 (2012). https://doi.org/10.3233/SW-2011-0052
41. Stadler, C., Saleem, M., Mehmood, Q., Buil-Aranda, C., Dumontier, M., Hogan, A., Ngomo, A.C.N.: Lsq 2.0: A linked dataset of sparql query logs. (Preprint) (2022), https://aidanhogan.com/docs/lsq-sparql-logs.pdf
42. Vrandečić, D.: Wikidata: A new platform for collaborative data collection. In: Proceedings of the 21st International Conference on World Wide Web (WWW 2012). p. 1063–1064 (2012). https://doi.org/10.1145/2187980.2188242
43. Wang, J., Li, T., Wang, A., Liu, X., Chen, L., Chen, J., Liu, J., Wu, J., Li, F., Gao, Y.: Real-time Workload Pattern Analysis for Large-scale Cloud Databases. arXiv e-prints arXiv:2307.02626 (Jul 2023). https://doi.org/10.48550/arXiv.2307.02626
44. Xie, T., Chandola, V., Kennedy, O.: Query log compression for workload analytics. VLDB Endowment **12**(3), 183–196 (2018). https://doi.org/10.14778/3291264.3291265