Extracting URI Patterns from SPARQL Endpoints

Mathieu d'Aquin[1] Alessandro Adamou[1] Enrico Daga[1]
Nicolas Jay[2]
1.Knowledge Media Institute,
The Open University,
Milton Keynes, UK
{mathieu.daquin, alessandro.adamou, enrico.daga}@open.ac.uk
2. Université de Lorraine,
LORIA, UMR 7503,
Vandœvre-lès-Nancy, F-54506, France
nicolas.jay@loria.fr

April 10, 2014

Abstract

Understanding the structure of identifiers in a particular dataset is critical for users/applications that want to use such a dataset, and connect to it. This is especially true in Linked Data where, while benefiting from having the structure of URIs, identifiers are also designed according to specific conventions, which are rarely made explicit and documented. In this paper, we present an automatic method to extract such URI patterns which is based on adapting formal concept analysis techniques to the mining of string patterns. The result is a tool that can generate, in a few minutes, the documentation of the URI patterns employed in a SPARQL endpoint by the instances of each class in the corresponding datasets. We evaluate the approach through demonstrating its performance and efficiency on several endpoints of various origins.

1 Introduction

Identifiers are fundamental to any kind of data. Constructing the strings that compose identifiers often follows conventions that have some significance with respect to the content and use of the data. Linked Data use URIs as a structured mechanism to build globally resolvable IDs [1]. However, even using this global convention, patterns in the design of URIs may appear, which are specific to a given dataset. For example, the namespace of some entities may reflect their type (e.g., in `data.open.ac.uk`, a URI of the form `http://data.open.ac.uk/course/` followed by a code identifies an entity of type `aiiso:Module`). More sophisticated situations also appear, where significant parts of the ID appear within the name of entities, or the namespaces are also constructed to reflect some aspect of the data (again in `data.open.ac.uk`, URIs of floors of a building include in their namespaces a code representing the building itself – e.g. `http://data.open.ac.uk/location/building/r04birb/floor/02` is the URI of the second floor of the building `R04BIRB`).

Such patterns are very important to applications and users wanting to use or link to these entities. They document the way to construct their IDs, which in the case of URIs are also access points and link anchors. However, while they are the result of a conscious design process, they are rarely made explicit or documented.

In this paper, we present a method to automatically extract the most significant URI patterns attached to instances of certain classes in specific datasets (accessed through SPARQL 1.1 endpoints). This is a complicated task since, as shown in the previous example: (1) The significant parts of a URI do not necessarily follow the URI's structure (domain, namespace, fragment, etc.); (2) Just comparing the strings of all URIs to find the common part would result in both a computationally unfeasible approach, and unexploitable results; (3) The significance of patterns should be assessed through their relationship with others (e.g. how they jointly cover the data, which one is 'more general' than others, etc.)

We therefore propose an approach inspired by and adapted from formal concept analysis (FCA, [12]) to build hierarchies (or *lattices*) of simplified regular expressions (strings with wildcard characters) that form sample sets of URI strings. The advantage of relying on FCA is, first, that we can adapt incremental lattice construction algorithms, thus making the search for pattern more efficient and, second, that the results of these algorithms are structured in a hierachy of concepts, on which we can apply measures to assess the significance of a pattern, in relationship to other extracted patterns.

After discussing related work, we describe this approach and its implementation as an automatic URI Pattern Documentation tool. We evaluated this approach and the tool through measuring performance on a variety of datasets from various origins.

2 Related Work

The general problem of the meaning and usage of identifiers in data has been broadly explored [4]. In the Semantic Web and Linked Data communities there have been constant efforts towards the definition of good practices, recipes and recommendations regarding the structure of URIs. This resulted in recommendations such as [1], technical reports, discussions in blogs and mailing lists, but rarely in clear specifications of the way URIs are built in specific datasets. A recent study [7] promotes, amongst other things, best-practices relating to the naming of resources: besides applying core Linked Data principles, such as being dereferenceable HTTP URIs, they should also be short, easy to remember and stable over time. Similarly, best practice documentation initiatives in publishing linked data (such as [3]) make clear statements about URI patterns, e.g., promoting the usage of paths that reflect the hierarchical relation between entities or the belonging of an entity to a collection.

Recent research also proposed to embed formal data within URIs for specific applications, for example in linked sensor data [8]. Also, aside from Semantic Web practices, it is worth mentioning specifications such as [11], which proposes to standardise URI fragments in order to facilitate information discovery in websites, and [6] that defines the concept of 'URI template' formally as "*a compact sequence of characters for describing a range of Uniform Resource Identifiers through variable expansion*".

The documentation for RDF/OWL vocabularies is a common practice and examples of automatic approaches exist (e.g., OWLDoc[1]). However, the documentation of instances in the data, and specifically of their URI patterns, is uncommon. The VoID vocabulary provides specific properties to inform users about the features of a Linked Data endpoint, including also the structure of URIs for entity lookup (`void:uriSpace`, `void:uriRegexPattern`). Similarly, implementations of the Linked Data API (e.g., Elda[2] and Puelia[3]) are used to generate specific URI templates which can then be documented as part of the publication of the data, but they do not support the automatic discovery of such templates in the underlying RDF data.

3 An FCA-based approach to extracting URI patterns from SPARQL endpoints

In this section, we describe our approach to the extraction of URI patterns from SPARQL endpoints. We describe how we adapt techniques from FCA to extract hierarchies (lattices) of common patterns (strings with wildcards) from as set strings, and how we prune this hierarchy to keep only the most significant patterns, which we expect to be in small numbers. As explained in Section 4, these two steps will be performed on samples of URIs that are randomly selected from

[1]OWLDoc, `http://www.co-ode.org/downloads/owldoc/`
[2]Elda, `https://code.google.com/p/elda/`
[3]Puelia, `https://code.google.com/p/puelia-php/`

the SPARQL endpoint at hand. It is noteworthy, however, that this approach could be employed on other types of identifiers than just URIs.

We first give a quick introduction to relevant FCA-related notions.

3.1 Overview of basic FCA notions

FCA [12] is a formal, generic framework, generally associated with the fields of data mining and knowledge discovery. In broad terms, it is concerned with identifying from raw data patterns of objects characteristics that form formal concepts. Such concepts are characterised both by their *intent* (a set of attributes), and their *extent* (the set of objects in the data that share these attributes).

More formally, FCA relies on the notion of a *formal context*, which represents the raw data. A formal context $C = (G, M, I)$ is made of a set of objects G, a set of attributes M and a binary relation $I \subseteq G \times M$. In simpler terms, a formal context is a binary matrix where the rows represent objects, and columns represent attributes of these objects. Given O a set of objects of G, we note O' the set of attributes of M which are shared by all the objects of O. In the same way, given $A \subseteq M$, $A' \subseteq G$ is the set of objects that share all the attributes in A. The double application of $(.)'$ is said to represent the closure of a set of objects or attributes. In other terms, O'' and A'' are said to be closed.

A formal concept of a context $C = (G, M, I)$ is characterised by a pair (O, A), where $O \subseteq G$ and $A \subseteq M$. O is called the *extent* and represents the objects that share the attributes of A, i.e., $O = A'$. A is called the *intent* and represents the attributes that are shared by the objects of O, i.e., $A = O'$. Note that this implies that $O = O''$ and $A = A''$, i.e., the concept (O, A) is equivalently defined both by its set of objects, and by its set of attributes.

The set of all concepts that can be derived from a formal context form a *lattice* that relies on the subconcept relation (denoted by $\leq$). Indeed, we say that a concept (O_1, A_1) is a subconcept of another concept (O_2, A_2) – i.e., $(O_1, A_1) \leq (O_2, A_2)$ – if $O_1 \subseteq O_2$ and (equivalently) $A_2 \subseteq A_1$. This concept lattice has an upper-bound and a lower-bound (which are often the concept with an empty extent and the one with an empty intent respectively).

An interesting aspect of FCA, considering our problem, is that it helps identifying groups of objects according to their common properties and it gives a partial order to these groups. In other words, it organises them in a hierarchy, based on how general/specific they are. We show below how we adapt this same approach to groups of strings matching common pseudo-regular expressions, rather than objects with common attributes.

3.2 Adaptation to building hierarchies of string patterns

Adapting the notions described above from FCA to our problem involves replacing the extent of concepts with strings, and the intent with string patterns. More precisely, we do not anymore work with a formal context made of binary attributes, but simply with a set of strings C.

A *pseudo-regular expression* here is a string which might include wildcard characters (represented by the * sign) that can be replaced by any (possibly empty) sub-string, e.g., "$aa*bb*cc$". The $(.)'$ function is therefore defined from the point of view of a pseudo-regular expression to represent the 'matching' operation; i.e., for e a pseudo-regular expression, $e' \subseteq C$ is the set of strings in the corpus that match e. For example, with e the pattern "$aa * bb$", and $C = \{$"$aaccbb$", "$accbb$", "$aabb$"$\}$, $e' = \{$"$aaccbb$", "$aabb$"$\}$.

One major difference with typical FCA however is that we do not have initially a finite and constructable set of attributes, but a potentially infinite set of all the pseudo-regular expressions E that might apply on each of the strings of C. The $(.)'$ function from the point of view of the strings in the corpus is therefore defined so that it generates the minimal set of pseudo-regular expressions that are necessary to characterise the strings of C. Considering S a set of strings, $S' \subseteq E$ is the set of the most specific pseudo-regular expressions matched by all the strings in S. The most specific pseudo-regular expressions are the ones for which none of the $*$-wildcards can be replaced, prefixed or suffixed by a concrete sub-string and still match all the strings of S.

Thanks to these definitions, the notion of closure applies to string patterns similarly as to objects and attributes of typical FCA. We can therefore define a string pattern-concept as a pair (S, P), where P is a set of pseudo-regular expressions and S is the set of strings of C that match all of P, such that $S' = P$ and $P' = S$. The subconcept relation applies to string patterns in a similar way as in typical FCA and complies with the natural intuition of the notion of a regular expression being more specific than another, i.e. that it is matched by a subset of the matching strings of the more general regular expression. More formally, $(S_1, P_1) \leq (S_2, P_2)$ iff $S_1 \subseteq S_2$.

3.3 Lattice building for string patterns

The objective of adapting FCA to string-patterns as defined above is that we can then reuse the techniques employed by FCA, especially concept lattice construction, to the extraction of string (URI) patterns. Here, we take inspiration from well-known incremental lattice construction algorithms [5], which have shown to represent an useful trade-off between efficiency and flexibility. Indeed, the incremental nature of these algorithms means that we can extend the set of strings C (here, a sample set of URIs) without having to rebuild the entire lattice.

The central task of these algorithms is to find common patterns (the closed itemsets, i.e., combinations of attributes/objects that form concepts) in the considered objects, and to classify them within the hierarchy being constructed. A core operation to achieve this is the one that, given two string patterns (which might be concrete strings from the set of individuals), generates the most specific common string pattern generalising (or matching) both. To achieve that, we adapt an algorithm to calculate the Levenshtein distance [10] between strings, not to calculate their so-called *edit distance*, but to identify the minimal *set* of addition, suppression and substitution operations necessary to transform one string into another. We then replace the original characters where these operations occur by $*$-wildcards.

Using this method for example, the URIs `http://data.open.ac.uk/course/m366` and `http://data.open.ac.uk/course/presentation/aa100-2006b` are generalised into the pattern `http://data.open.ac.uk/course/*6*`. Applying it again between `http://data.open.ac.uk/course/*6*` and `http://data.open.ac.uk/course/aa100` gives the pattern `http://data.open.ac.uk/course/*`[4]. We use this generalisation function in the lattice building algorithm to create the most specific subsuming concepts of two existing concepts, as well as to test subsumption between concepts: if the generalisation of two strings a and b is equal to a, then a is a more general pseudo-regular expression than b.

Based on these elementary operations, the lattice building algorithm generates hierarchies of pseudo-regular expressions. For example, using as corpus of strings C the set of URIs {`http://data.open.ac.uk/course/aa100`, `http://data.open.ac.uk/course/m366`, `http://data.open.ac.uk/course/presentation/m366-2013b`, `http://data.open.ac.uk/course/presentation/aa100-2013b`}, we obtain the simple hierarchy of patterns (excluding concepts only covering the individual URIs from C) shown in Figure 1.

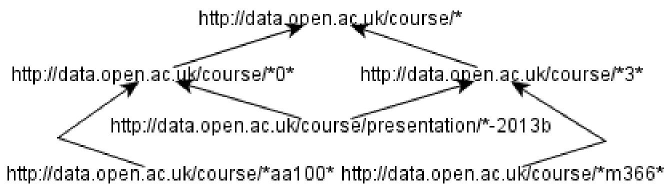

http://data.open.ac.uk/course/*
http://data.open.ac.uk/course/*0*
http://data.open.ac.uk/course/*3*
http://data.open.ac.uk/course/presentation/*-2013b
http://data.open.ac.uk/course/*aa100*
http://data.open.ac.uk/course/*m366*

Figure 1: Simple example of a lattice of URI string patterns.

3.4 Pruning the hierarchy

As obvious from the example in Figure 1, not all string patterns found in a set of identifiers/URI would be relevant to the documentation of the design of these URIs, as some of them are only shared by a small sub-part of the considered strings. It is therefore necessary to prune the hierarchy so that it only contains the (generally small set of) patterns that are significant.[5]

[4]Note that the result of this generalisation function is, in theory, not necessarily unique. However, to simplify, we implement a deterministic way to only return one of the generalisations. While this affects the construction of the lattice in theory, it has little effect in practice on the results of the application to URI pattern documentation and drastically reduces the complexity of the process.

[5]It is worth noting that, while most of the patterns in the lattice will therefore be filtered out as irrelevant, they still play a role in the construction of the selected ones, and in assessing their relevance.

We therefore devise a method to filter out patterns based on three basic principles: (1) **Coverage**: a set of patterns/concepts selected should cover all/most of the set of strings C; (2) **Connectivity**: other than patterns/concepts at the top of the hierarchy, a pattern/concept can only be selected if at least one of its parents is selected; (3) **Complementarity**: selected patterns/concepts that are not hiearchically related should not cover the same strings/individuals.

To address principles 1 and 3, we devise a method based on calculating how a set of sibling concepts represent a partition of C. We define $parents(c)$ as the set of parents (subsuming concepts) of the concept $c = (S_c, P_c)$ in the lattice, and $children(p)$ as the set of children (subsumed concepts) of $p = (S_p, C_p)$. Given a concept c, the following formula therefore gives a score $score(x)$ to a concept x, depending on how it is included in a set of siblings which partitions C.

$$score(x) = \max_{p \in parents(x)} \left(\alpha \frac{\left| \bigcap_{c \in children(p)} S_c \right|}{|C|} + \beta \frac{1}{1 + \left| \bigcup_{c \in children(p)} S_c \right|} \right) \quad (1)$$

In this equation, α and β are weights that can be used to give more or less importance to principle 1 (coverage) or 3 (non overlap) respectively. In our implementation we set them at 0.5 so that both aspects have the same importance. To comply with principle 2, we apply this function with a threshold (we use 0.99 to only keep the patterns which represent a partition of the set of URIs being analysed) following the hierarchy from top to bottom, i.e. only assessing for selection the children of selected concepts.

As mentioned above, in most cases only a few patterns/concepts will be selected by following this approach. In the example of Figure 1 for instance, only the top pattern `http://data.open.ac.uk/course/*` is selected.

4 Implementation and Evaluation

We implemented the approach above in a PHP-based framework which, given a set of strings, builds the corresponding string pattern lattice, and applies the pruning mechanism described in the previous section. On this basis, we developed a tool[6] which, given a SPARQL endpoint, extracts samples of URIs associated to the classes used in the graphs (sub-datasets) of the endpoint, finds the relevant URI patterns, and generates a Web interface to document them. In this section, we describe the key points of this implementation and give examples. We also evaluate the behaviour of the tool on several datasets or various sizes, complexity and origins, focusing on the key performance factor, i.e. the size of samples.

[6]`http://linkedu.eu/muse`

4.1 URI Sampling

Given a class and (optionally) a graph from a SPARQL endpoint, querying the endpoint to obtain a subset of URIs of instances of this class is rather trivial. However, for this subset to be a representative sample of these URIs, they need to be selected at random. We therefore employ the following SPARQL 1.1 query, given a graph [g] and a class [c] obtained from prior SPARQL queries, as well as a sample size [n]:

```
select distinct ?uri from [g]
where {?uri a [c]}
order by rand() limit [n]
```

This naturally limits the use of the tool to SPARQL 1.1 endpoints that support the `rand()` function. Alternative sampling methods, which we will explore in later versions of the tool, can help relax this limitation. A key parameter affecting both the performance – as lattice construction complexity is highly dependent on the size of the population – and the accuracy of the results – i.e. whether the right patterns are found and selected – is therefore the size of the sample [n]. We assess this impact in Section 4.4, whilst exploring the trade-off between these two aspects.

4.2 Documentation generation

The documentation of the URI patterns employed in an endpoint is divided according to the endpoint's graphs (if the endpoint is divided in graphs) and the classes that each graph instantiates. The endpoint is therefore first queried to retrieve the list of graphs, and for each graph, the list of classes used as types of entities within the graph. The sampling query described above is then applied to generate the base population for lattice construction (the set of strings C). An example of the generated documentation interface is shown in Figure 2, as integrated within the interface of the LinkedUp catalogue of educational datasets[7].

As will be shown in the evaluation section, depending on the number of graphs and classes, on the sample size, on the complexity of the URIs and on the response time of the SPARQL endpoint, the tool can take up to several minutes to generate the entire documentation. We therefore also implemented a series of caching mechanisms, by storing the results first of SPARQL queries and then of the lattice construction process. As a result, while it might take some time to generate the documentation initially, the extracted patterns are persisted and can be retrieved in negligible times, unless the caches are reset.

4.3 Examples and Usage

One of the main motivations for building this tool was to provide the generated documentation as part of the LinkedUp catalogue of educational datasets. This

[7]The LinkedUp catalogue `http://data.linkededucation.org/linkedup/catalog/`

catalogue (see [2] for a preliminary analysis of its content) gathers SPARQL endpoints from various origins that might, individually or in combination, be useful to developers of educational applications. We therefore integrated the tool described above (with a sample size of 10) in the interface that describes each endpoint on the LinkedUp catalogue (see Figure 2). Besides the more formal evaluation of the next section, this provides us with a view of the results of the tool in different situations.

As expected for example, many classes in many of the graphs are represented with a unique and very simple pattern. Course modules (`aiiso:Module`) in `data.open.ac.uk` (following our earlier examples) are, for instance, associated with the pattern `http://data.open.ac.uk/course/*`. More interestingly however, there is a specific type of course (online courses - `ou:OnlineCourse`) whose detected pattern is `http://data.open.ac.uk/course/*zx*`: the codes of online courses, used in the last part of the URIs, are indeed built by taking the code of non-online courses and adding 'zx' in addition to another first letter. Also, this example shows how the way URI patterns are hierarchically organised by the lattice construction technique can sometimes help understanding the space of URIs covered. Indeed, two significant patterns are found as children of the one for online courses: One showing that most online courses have codes starting with 'bzx' (`.../bzx*`) and another showing that exceptions exist where it starts with 'dzx'. The `.../*zx*` pattern therefore represents the general 'rule' for building such identifiers, while the two others are sub-rules that form a partition of the URI space (it is either a 'bzx' online course or a 'dzx' online course).

Many other similar cases clearly show how URIs are constructed, sometimes in a sophisticated manner. In `linked.opendata.cz` for example, one graph contains the class `frbr:Expression` with the URI pattern

`http://linked.opendata.cz/resource/legislation/cz/act/1993/15-1993/version/cz/201*-*-01/section/*`

where we can clearly recognise that one part of the URI is constructed to represent a date (supposedly of publication). In the same endpoint, but another graph, we can find a class `Product` with the pattern

`http://linkedopendata.cz/resource/fda-spl/product/*`

and another class `ProductIngredient` with the pattern

`http://linked.opendata.cz/resource/fda-spl/product/*/product-ingredient/*`.

We can therefore expect instances of the former to represent ingredients of the products which identifiers are represented in the first part of the URI. It is interesting to note that, while some URIs in this endpoint might appear very sophisticated, the pattern detection reveals that the part which is actually significant is not necessarily complex. For example, in one of the graphs, `Dosage` is associated with the pattern

`http://linked.opendata.cz/resource/drugbank/dosage/*-A1E*-11E2-dce1-09173F13E4C5`.

No doubt that codes which are therefore constant at the end of the URIs would have some significance to a domain expert or the originator of the data.

Interestingly, the tool for URI pattern documentation can also be used to detect errors, inconsistencies or mistakes in the way URIs are built. For example, in `data.cnr.it` the class `foaf:Person` is associated with the pattern `http://vocab.deri.ie/void#*` (i.e., the namespace of the VoID vocabulary). According to one of the creators of this endpoint, also the third author of this paper, this was due to the actual data in the endpoint using another class (`http://www.cnr.it/ontology/cnr/persone.owl#Persona`) for people. However, the RDF of the VoID vocabulary, which includes URIs for its authors as instances of `foaf:Person`, is also imported in the dataset.

Finally, cases of specifically constructed URIs show how the pattern detection mechanism is flexible in the way it can find significant invariants in URIs through not imposing assumptions on their structure. A typical example is the one of the `vcard:Voice` class of one of the graphs of `data.ox.ac.uk`, which has for top URI pattern `tel:+441865*`, with +441865 being the prefix of phone numbers in Oxford, UK. In another graph of the same endpoint, dedicated to the IT services of the university, the same class is associated with an even more specific pattern: `tel:+44186527320*`, which hints at how IT support lines can be reached.

4.4 Evaluation

There are two key aspects that need to be considered in evaluating the approach and the tool proposed in this paper:

1. *Stability of the results*: The results of each pattern extraction task are by nature uncertain as they rely on random samples to find significant patterns. Understanding how stable the results are depending on the sample size gives us a confidence indicator on whether or not the pattern found is significant.

2. *Performance*: The techniques employed in the tool are complex (see e.g. [9] on the complexity of lattice construction in FCA). Besides the number of graphs and classes in an endpoint (which will affect the number of times URIs are sampled and a lattice is built), the time to construct a lattice is highly dependent on the size of the initial population, represented by the sample size here.

Of course, we do not expect the tool to process an entire endpoint in under a few minutes, as it will in most cases have to generate several dozen lattices and will anyway only be required to do so once (as explained above, the resulting documentation does not require to be computed more than once if there are no significant changes to the data in the endpoint). It is expected, however, that an increased size of samples will affect positively the stability of the results, while impacting negatively on performance. One of the goal here is therefore also to assess what is the right trade-off between the two aspects, i.e., what size of sample is large enough to give acceptable results while keeping computing time reasonably low.

As expressed in the previous section, one of our main motivations for this work was to provide an interface for users of the LinkedUp catalogue of educational datasets to have an overview of the type and design of identifiers in these

Table 1: Overview of the test SPARQL endpoints.

Endpoint	**Description**	**Graphs**	**Classes**	**Triples**
http://data.open.ac.uk/query	The Open University	25	82	2,733,194
http://data.linkedu.eu/slidewiki/query	SlideWiki RDF export	1	6	53,300
http://linked.opendata.cz/sparql	Open Data Czech Republic	45	188	71,085,064
http://data.aalto.fi/sparql	Aalto University	8	22	1,547,552
https://data.ox.ac.uk/sparql/	(OxPoints) Oxford University	99	131	296,881
http://data.uni-muenster.de/sparql	Muenster University	27	85	1,947,320
http://data.linkedu.eu/hud/query	Hud. Uni. Book Borrowings Stats	4	6	3,553,343
http://data.linkededucation.org/request/lak-conference/sparql	LAK Data Challenge	13	14	37,194

datasets. We therefore focus our evaluation on a sub-set of the endpoints in this catalogue. These endpoints were selected because they support the `rand()` function in SPARQL 1.1, they include named graphs and they were stable enough to sustain the series of queries our experiments required. The selected endpoints are summarised in Table 1.

To assess the performance of the pattern extraction technique, we ran it (with the caching mechanisms disabled) on a virtualised server with PHP 5.3, Apache HTTPD, 4 cores and 8GB of RAM using varying sample sizes (5, 10, 15, 25, 30, 35 and 40), and logged the time taken specifically to extract patterns (i.e., excluding the query time to the endpoint, since it is dependent on factors out of our control). To assess stability, we ran each of these tests 5 times on different samples and logged the results for comparison.

Figure 3 shows the results of the stability test in terms of the average stability (in percentage) for each of the tested sample sizes. Here, percentage of stability refers to the average size of the stable part of the 5 tests, i.e., if 4 out of 5 results were the same for a given endpoint, graph and class, then the assessment of stability is 80%.

As can be seen from this figure, while large variations can be observed between endpoints (some never reaching more than an average of 80% stability, and others reaching 100% from the smallest sample size), most endpoints on average reach a plateau of stability at a sample size of 10 or 15, reaching in most cases between 80% and 90%. This shows that such sample sizes are sufficient to reach acceptably accurate results. Also, while it cannot be easily assessed formally, often the erroneous patterns (the unstable ones) are similar to the ones found to be stable.

The variations between endpoints can basically be explained by the variation in the complexity of URIs in these endpoints, and in their significance within the overall URI population. Reaching very good results at very small sample sizes happens in datasets where the URI patterns are very simple (mostly, a namespace and a single, simple code for each type of entities) and are applied consistently. More unstable endpoints have more complex URIs (with several variable parts, made of longer, more random codes), that might not be as consistently applied (e.g., where there are exceptions, or even many different conventions applied to the same type of entities).

Figure 4 shows the results of the performance evaluation, showing for each endpoint the average time in seconds to build a lattice (in log scale) for each of the tested sample sizes, as well as some indication of the maximum computation times. As was expected, computation time increases steadily with the size of the samples. It shows that for sample sizes below 20, each computation will take only a few seconds on average.

A slightly unexpected aspect of this evaluation, which appears clearly in this figure, is the large variations appearing in computation time for the same sample sizes not only between endpoints, but also within endpoints (difference between min – always 0 – and max). Very low computational times (0, i.e. under a second even for large sample sizes) can be easily explained as being due to exceptional cases of classes having very small number of instances (therefore not reaching the limit in the size of the sample). Longer times are of course more interesting. Indeed, it shows that the assumption that the time to compute the lattice is dependent purely on the size of the population is actually naive. One key aspect that affects computational time is the complexity or lack of regularity in the patterns. Indeed, some of the most difficult cases are the ones where one of the variable parts of the pattern (the ones changed into a wildcard) is in individual URIs a long string of random characters (such as a hashcode). Such cases naturally generate more intermediary (non-significant) patterns, and therefore lead to larger lattices, which take more time to construct for the incremental algorithm.

5 Conclusion

In this paper, we presented an FCA-based technique to extract URI patterns from SPARQL 1.1 endpoints. This technique is used to create a tool that can automatically generate an interface documenting how URIs are constructed for each class of each graph of a dataset exposed through SPARQL. Sampling of URIs is a necessary part of this approach. We therefore evaluated how the size of the samples of URIs used to extract patterns affected both the stability of the results and the performance of the tool on several datasets of various origins, sizes and richness. Besides providing many examples showing the value of the approach in terms of its flexibility and of the information provided to the users (and even the developers) of datasets, this evaluation demonstrated that, in most cases, small sample sizes of around 15 URIs picked randomly were enough to obtain acceptable results. Such sample sizes lead to the URI pattern extraction algorithm requiring only a few seconds for each graph and class, meaning that even in datasets with a large number of such graphs and classes, generating the whole documentation will take a few minutes at most.

While the value of documenting the content and structure of linked datasets is evident, and the design of URI patterns is a core aspect of the creation of a linked dataset, it is an area that has not received much attention so far. The approach proposed here therefore constitutes a first contribution in this area, which has the advantage to be both formally grounded (using FCA as a base foundation) and concretely implemented (in a usable tool, integrated with the LinkedUp data catalogue). There are however limitations that we are planning to address in our future work. One of them is the reliance on SPARQL 1.1. The reason for this is that it allows us to efficiently achieve random sampling.[8] Indeed, the rest of the technique is independent from the way sample URIs are obtained (it could even be used with other types of identifiers, or even other kinds of strings). An efficient way to sample URIs randomly from a SPARQL 1.0 endpoint or other data access methods would therefore broaden the applicability of our tool.

Besides the technical implementation of sampling, another aspect that can be explored is how sampling can be performed in a smarter way, through dynamically assessing the results and extending the sample as needed. Indeed, one approach would be to iteratively increase the size of the sample and incrementally build the lattice until an acceptable level of stability has been reached. Another complementary approach would be to interrogate the endpoint to assess the completeness of the current results, checking whether URIs exist that do not comply with the top-level patterns that were obtained, adding samples of such noncompliant URIs to the lattice being built. The possibility of implementing such strategies is actually the reason why we used an incremental lattice building algorithm.

Performance is obviously one aspect that can be improved from this first version of the technique/tool. While, as discussed above, the advantage of an incremental lattice construction algorithm is that it makes it possible to iteratively increase the size of the sample, it is also less efficient than an algorithm that builds the lattice using the whole population at the same time. Implementing such an approach as a way to reduce at least the time to build the initial lattice would therefore reduce the time needed to generate the documentation. Considering the type of results we obtain, one could argue that other, possibly simpler ways (not based on FCA) to generalise URI strings into patterns might be more efficient. However, FCA-based algorithms are explicitly designed to efficiently achieve such tasks, using the hierarchical structure of the lattice to avoid unnecessary comparisons. This hierarchical organisation of the results, while not strongly featuring in the final documentation presented to the user, is also a key aspect in our approach to filter URI patterns.

Finally, an interesting aspect to explore is the way in which the extracted patterns could be used beyond providing potential users with an overview documentation of an endpoint. We have already discussed use cases where the creators of datasets could detect peculiarities in their URIs and use this information to improve their consistency. Another interesting use case is to support data linking, through providing information about the way to construct the target URIs for the links. An interesting further extension of our approach related to this use case would be to enable more complex patterns that not only replace the variable part of the URIs with a wildcard character, but with some

[8]This is a strong requirement, as, while it is not in theory significant, the order in which results of SPARQL queries are returned by endpoints tends to introduce a very significant bias with respect to URI patterns

indications of its relationship to the data associated with the entity (e.g., that the URI of a book is formed using its date of publication and a number, or its ISBN).

References

[1] D. Ayers and M. Völkel. Cool URIs for the Semantic Web. W3C Interest Group Note, `http://www.w3.org/TR/cooluris/`, 2008.

[2] M. d'Aquin, A. Adamou, and S. Dietze. Assessing the educational Linked Data landscape. In *ACM Web Science*, 2013.

[3] L. Dodds and I. Davis. Linked Data patterns. Online book - `http://patterns.dataincubator.org/book/`, 2012.

[4] O. Eriksson and P. Ågerfalk. Rethinking the meaning of identifiers in information infrastructures. *Journal of the Association for Information Systems*, 11(8), 2010.

[5] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational intelligence*, 11(2), 1995.

[6] J. Gregorio, R. Fielding, M. Hadley, M. Nottingham, and D. Orchard. URI template. IETF RFC 6570, `http://tools.ietf.org/html/rfc6570`, 2012.

[7] A. Hogan, J. Umbrich, A. Harth, R. Cyganiak, A. Polleres, and S. Decker. An empirical survey of Linked Data conformance. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2012.

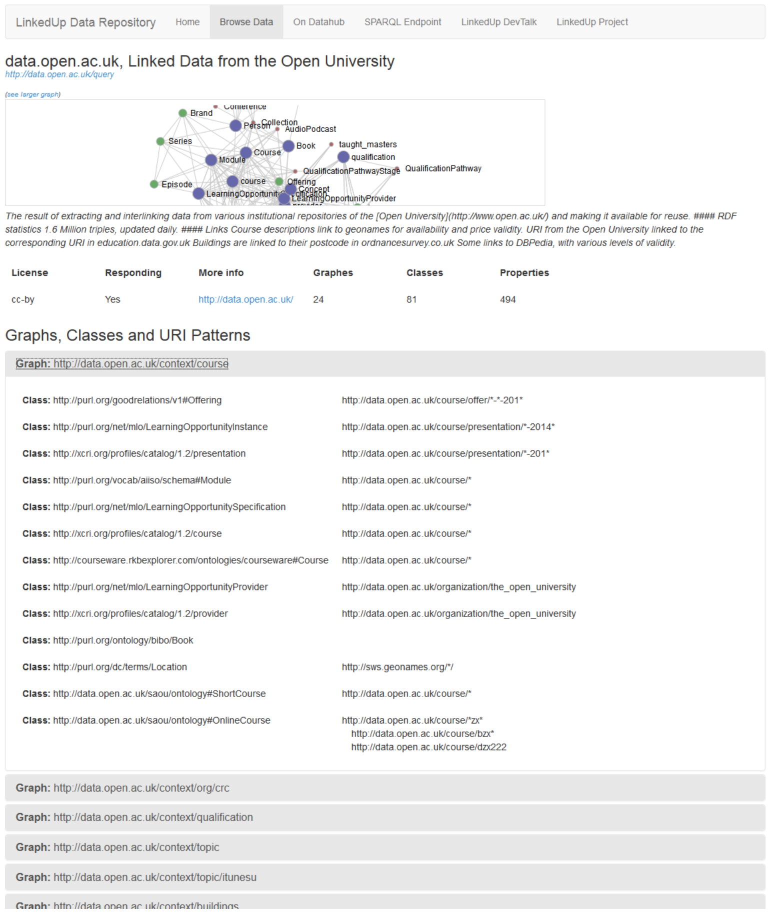
[8] K. Janowicz, A. Bröring, C. Stasch, and T. Everding. Towards meaningful uris for linked sensor data. In *Towards Digital Earth: Search, Discover and Share Geospatial Data. Workshop at Future Internet Symposium*, 2010.

[9] S. O. Kuznetsov and S. Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental and Theoretical Artificial Intelligence*, 14(2-3), 2002.

[10] V. Levenshtein. Binary codes capable of correcting deletions, insertions an reversals. *Cybernetics and Control Theory*, 10, 1966.

[11] M. Nottingham and E. Hammer-Lahav. Defining well-known uniform resource identifiers (URIs). IETF RFC 5785, `http://tools.ietf.org/html/rfc5785`, 2010.

[12] R. Wille. Why can concept lattices support knowledge discovery in databases? *J. Exp. Theor. Artif. Intell.*, 14(2-3), 2002.



LinkedUp Data Repository | Home | Browse Data | On Datahub | SPARQL Endpoint | LinkedUp DevTalk | LinkedUp Project

data.open.ac.uk, Linked Data from the Open University
http://data.open.ac.uk/query

(see larger graph)

The result of extracting and interlinking data from various institutional repositories of the [Open University](http://www.open.ac.uk/) and making it available for reuse. #### RDF statistics 1.6 Million triples, updated daily. #### Links Course descriptions link to geonames for availability and price validity. URI from the Open University linked to the corresponding URI in education.data.gov.uk Buildings are linked to their postcode in ordnancesurvey.co.uk Some links to DBPedia, with various levels of validity.

License	Responding	More info	Graphes	Classes	Properties
cc-by	Yes	http://data.open.ac.uk/	24	81	494

Graphs, Classes and URI Patterns

Graph: http://data.open.ac.uk/context/course

Class: http://purl.org/goodrelations/v1#Offering	http://data.open.ac.uk/course/offer/*-*-201*
Class: http://purl.org/net/mlo/LearningOpportunityInstance	http://data.open.ac.uk/course/presentation/*-2014*
Class: http://xcri.org/profiles/catalog/1.2/presentation	http://data.open.ac.uk/course/presentation/*-201*
Class: http://purl.org/vocab/aiiso/schema#Module	http://data.open.ac.uk/course/*
Class: http://purl.org/net/mlo/LearningOpportunitySpecification	http://data.open.ac.uk/course/*
Class: http://xcri.org/profiles/catalog/1.2/course	http://data.open.ac.uk/course/*
Class: http://courseware.rkbexplorer.com/ontologies/courseware#Course	http://data.open.ac.uk/course/*
Class: http://purl.org/net/mlo/LearningOpportunityProvider	http://data.open.ac.uk/organization/the_open_university
Class: http://xcri.org/profiles/catalog/1.2/provider	http://data.open.ac.uk/organization/the_open_university
Class: http://purl.org/ontology/bibo/Book	
Class: http://purl.org/dc/terms/Location	http://sws.geonames.org/*/
Class: http://data.open.ac.uk/saou/ontology#ShortCourse	http://data.open.ac.uk/course/*
Class: http://data.open.ac.uk/saou/ontology#OnlineCourse	http://data.open.ac.uk/course/*zx* http://data.open.ac.uk/course/bzx* http://data.open.ac.uk/course/dzx222

Graph: http://data.open.ac.uk/context/org/crc

Graph: http://data.open.ac.uk/context/qualification

Graph: http://data.open.ac.uk/context/topic

Graph: http://data.open.ac.uk/context/topic/itunesu

Graph: http://data.open.ac.uk/context/buildings

Figure 2: Screenshot of the URI pattern extraction tool (bottom) integrated with the interface of the LinkedUp data catalogue. URI patterns are show for each class of each graph of the `data.open.ac.uk` endpoint.

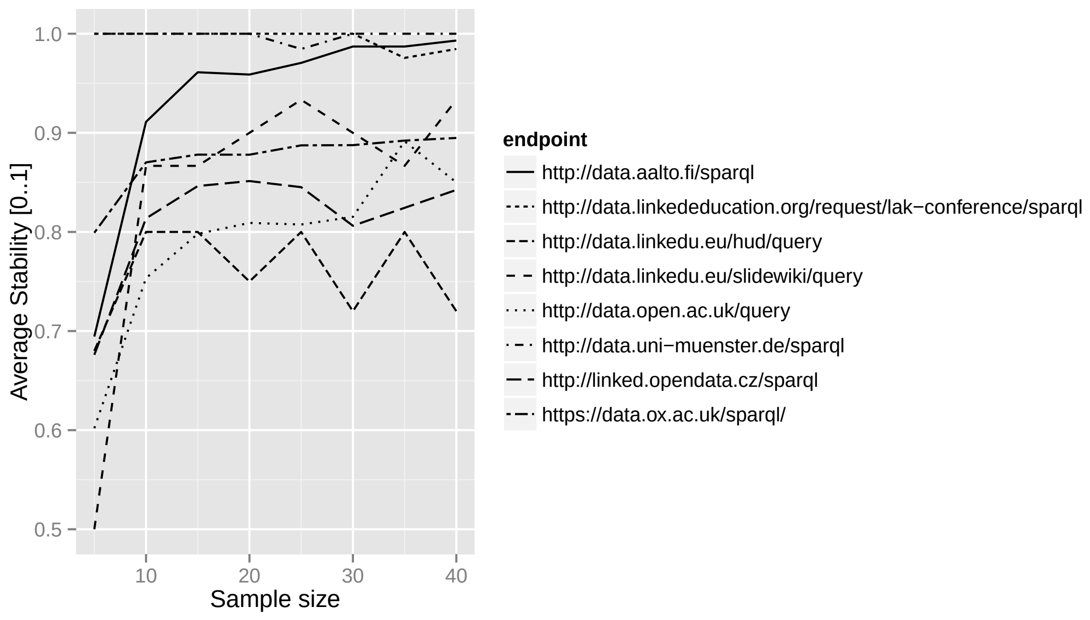


Figure 3: Average stability of results for different endpoints with different sample sizes.

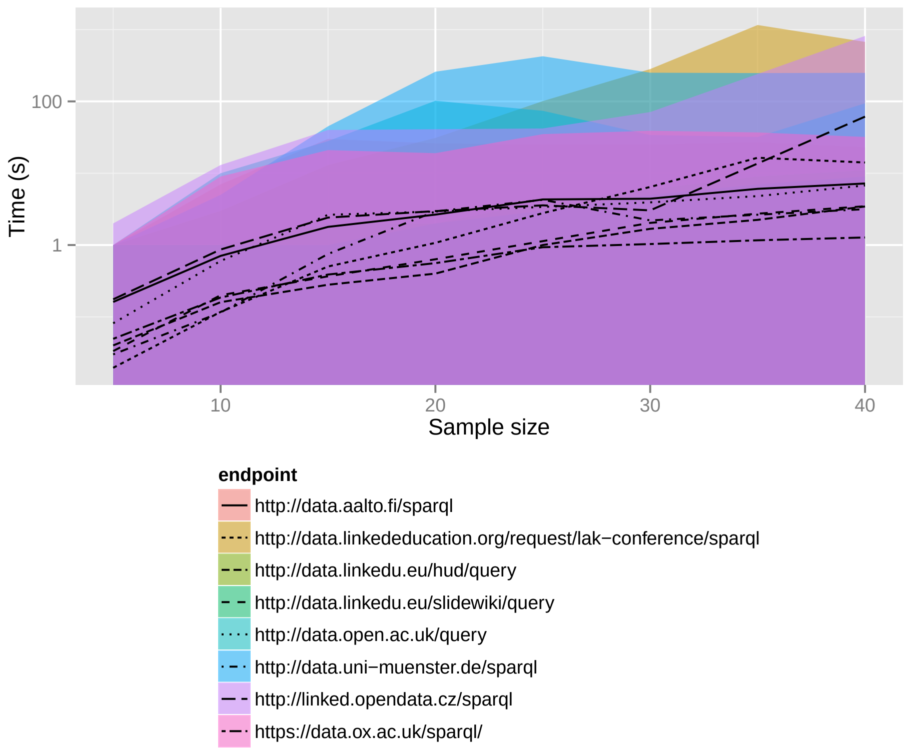


Figure 4: Average and max time in seconds for building a lattice in different endpoints with different sample sizes.