ExpLOD: Summary-Based Exploration of Interlinking and RDF Usage in the Linked Open Data Cloud

Shahan Khatchadourian and Mariano P. Consens

University of Toronto
shahan@cs.toronto.edu, consens@cs.toronto.edu

Abstract. Publishing interlinked RDF datasets as links between data items identified using dereferenceable URIs on the web brings forward a number of issues. A key challenge is to understand the data, the schema, and the interlinks that are actually used both within and across linked datasets. Understanding actual *RDF usage* is critical in the increasingly common situations where terms from different vocabularies are mixed. In this paper we describe a tool, ExpLOD, that supports exploring summaries of RDF usage and interlinking among datasets from the Linked Open Data cloud. ExpLOD's summaries are based on a novel mechanism that combines text labels and bisimulation contractions. The labels assigned to RDF graphs are hierarchical, enabling summarization at different granularities. The bisimulation contractions are applied to subgraphs defined via queries, providing for summarization of arbitrary large or small graph neighbourhoods. Also, ExpLOD can generate SPARQL queries from a summary. Experimental results, using several collections from the Linked Open Data cloud, compare the two summary creation approaches implemented by ExpLOD (graph-based vs. SPARQL-based).

1 Introduction

As the web continues to evolve it incorporates new ways of publishing and interacting with information beyond simply linking documents. A promising area for extending the web is the addition of Linked Data [6] to provide a simple mechanism for sharing structured data via the creation of Resource Description Framework (RDF) [15] links between data items using dereferenceable URIs. The emergence of a web of linked open data (LOD) is being promoted by the Linking Open Data community project and is fostering the availability of many open interlinked datasets. Many interlinked datasets have been contributed to what has been referred to as the *LOD Cloud*. Some notable examples are structured datasets such as DBpedia [4] (extracted from Wikipedia), RKB Explorer [16], YAGO [25], and LinkedMDB [17]. These datasets are placed online by the community, fostering collaborative linkage of structured knowledge.

The web of interlinked data is closely intertwined with the existing web. As a dataset like DBpedia illustrates, structured data items have RDF links referencing classic web pages and the reverse is also possible. The increased presence

L. Aroyo et al. (Eds.): ESWC 2010, Part II, LNCS 6089, pp. 272–287, 2010.

of abundant linked data (see [18]) enables novel applications and sophisticated mashups. Linked data can be searched using semantic web search engines which leverage both page and data links, such as URI-based semantic web search engines Swoogle [13] and Sindice [26], or term-based search engines such as Falcons [8], or by web search engines returning enhanced results such as Yahoo! SearchMonkey [1].

A key challenge is to understand the data, the schema, and the interlinks that are actually used both within and across linked datasets. Using an RDF browser to explore linked data can be a tedious and time-consuming exercise for large datasets. A trial-and-error approach on a schema-conformant dataset means testing for structures permitted by the schema which becomes complicated if the dataset does not use the full schema or uses multiple schemas. In the LOD cloud, this is even more pronounced due to the possible heterogeneity of ontologies used in each dataset, so another approach is needed. Inspiring early work on *RDF usage* presented in [12] enumerates several RDF usages involving classes, predicates, schema resources, and data resources. Understanding actual RDF usage is critical for developers in the increasingly common situations where terms from many different RDFS and OWL vocabularies are mixed. This is because developers need to know a dataset's structure in order to contribute interlinks between datasets, or for scenarios involving resources described within a dataset or across datasets. We propose that, instead of examining RDF usages separately, summaries that show how different RDF usages interact with one another can be used to describe an interlinked dataset.

Related Work. Describing and understanding large collections is enabled by summaries. One type of summary which does this by grouping common substructures is a *structural summary*. Prior work has shown the usability of XML path summaries for a variety of scenarios within semi-structured XML collections such as XPath query answering [9] and information retrieval [3]. Recently, structural summaries have been proposed in the context of RDF data in [21], but for frequency estimation applied to query evaluation (also the most common use for summaries in the context of XML), but not for describing usage in a flexible manner as done in this work. Our work is based on the work in structural summaries that has been developed in the context of semi-structured data (XML in particular) over the last decade. We extend and generalize our earlier work in XML summaries [10] in significant ways, in particular, we do not rely on the tree nature of XML data or on acyclicity assumptions that do not hold for RDF datasets.

Several methods have been proposed to enumerate the interlinks between datasets. [20] describes a system that allows a user to visualize the namespaces and classes present in a dataset and the linkages between them. Semantic sitemaps [11] use *slices* as a way to specify how interlinked datasets may be structured, such as indicating that each document contains a single resource description. A slice can also be expressed as a Concise Bounded Description (CBDs) [24] which may traverse a fixed length path to or from the described resource. An alternative to CBDs is the Minimum Self-Contained Graph (MSG)

[27] which includes a resource's blank node closure. The voiD vocabulary [2] portrays a restricted view into the number and type of predicates that contribute to the actual interlinks between datasets. Silk [29] is a mechanism that can create interlinks between datasets declaratively and can use heterogeneous vocabularies independently of any schema.

In this paper, we address the specific challenge of creating RDF summaries that describe structures within and among interlinked datasets in a flexible manner. In particular, our work can answer the following question - *how do data, metadata, and interlinking contribute to the structure of RDF datasets from the LOD cloud?* The contributions of our work are: (1) A novel bisimulation contraction framework to explore the LOD cloud; (2) A tool, Explod, that creates RDF usage summaries based on bisimulation contractions; (3) A flexible mechanism to compute RDF usage summaries at different granularities using text labels and different subgraphs of RDF data (that can be specified using a query); (4) A way to generate a SPARQL query that returns RDF data having a particular bisimulation contraction; (5) A SPARQL-based approach to computing RDF usage summaries; and (6) A performance comparison of the SPARQL-based and graph-based approaches.

This paper is structured as follows. In Section 2, we define the summaries that ExpLOD creates. Section 3 we showcase how ExpLOD can be used to explore the LOD cloud. In Section 4, the two summary creation approaches implemented in ExpLOD are described, and a performance comparison is provided in Section 5. We conclude in Section 6.

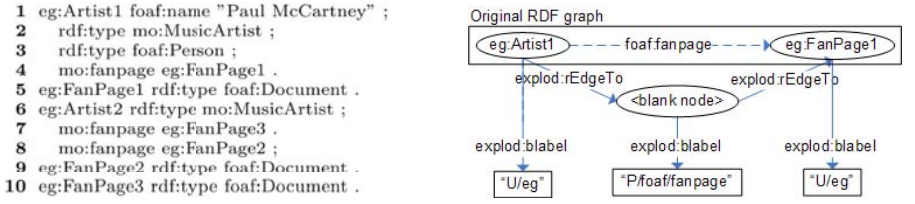
2 ExpLOD Bisimulation Contraction Framework

This section defines the summaries that are generated by ExpLOD. In Section 2.1, a labeled graph (with unlabeled edges) is constructed from an RDF dataset using hierarchical labels. In Section 2.2, we describe how summaries are defined by applying bisimulation contractions to neighbourhoods of the labeled graph.

2.1 Applying Bisimulation Labels to RDF

ExpLOD's RDF data model is based on quads. A quad consists of: a context, such as a graph name URI [7]; and a triple, a 3-tuple consisting of a *subject* URI, a *predicate* URI, and an *object* that can either be a URI or any valid XML data type [5]. A URI is represented as in [15], a pair of labels called the *prefixed name* composed of a *prefix label* (also known as a namespace prefix in XML) and a *local part* that are concatenated with a colon.

Figure 1(a) shows the triples of a sample dataset. The dataset gives information about two music artists, their fan pages, and one artist's name using the FOAF ontology (with prefix label *foaf*) to describe people and documents, and the Music Ontology (*mo*) vocabulary to describe music artists and their work. The triple on line 1, *(eg:Artist1, foaf:name, 'Paul McCartney')*, indicates that the name of the web resource given by the URI *eg:Artist1* is the string value



(a) RDF dataset (b) ExpLOD graph

Fig. 1. Simple example

'Paul McCartney'. The object of a triple whose predicate is *rdf:type* is a *class* and the subject is an *instance* of the class. The triple on line 6, *(eg:Artist2, rdf:type, mo:MusicArtist)*, instantiates *eg:Artist2* as a music artist. The context of the statements is excluded for brevity.

Summaries of RDF data are computed over a graph using each node's *bisimulation label* (BL); we describe the construction of the labeled graph in more detail later in this section. The BL of each node in the labeled graph is based on *RDF usage*. In this work, we define an RDF resource's usage that is dependent on its *neighbourhood*, a subgraph of the labeled graph. Different neighbourhoods of an RDF resource describes its usage in different ways. We consider four RDF usages that describe the interaction of data and meta-data: (i) *class instantiation*, the number of instances that are typed as a particular class; (ii) *predicate instantiation*, the number of times a predicate is used to describe all instances; (iii) *class usage*, the *sets* of classes to which instance belongs; and (iv) *predicate usage*, the *sets* of predicates used to describe an instance. For example, in Figure 1(a), the class usage of *eg:Artist2* is the singleton *mo:MusicArtist*, which is different from the class usage of *eg:Artist1*, the set of classes {*mo:MusicArtist*, *foaf:Person*}. The predicate usage of the two music artists also differs because the predicate *foaf:name* is used to describe *eg:Artist1*, but not *eg:Artist2*.

We use hierarchical BLs, similar to dimension hierarchies common to most OLAP data models [28]. In the current implementation we are using string literals for BL values due to limitations in creating URIs in SPARQL; however, it would make sense use dereferenceable URIs for BLs since this would allow us to fetch additional information about the bisimulation labeling hierarchies that are used. BL hierarchies allow describing datasets at a range of granularities. The BL hierarchy to describe RDF usage is of the following form: RDF usage prefix, context, URI prefix label, URI local part; parts of the hierarchy are concatenated with a forward slash ('/'). The RDF usage prefix is 'P' for predicates, 'C' for classes, 'I' for instances, and 'L' for literals. The BL of a URI used as a predicate or class includes the full hierarchy. The BL of an instance URI excludes the local part of its prefixed name, and the BL of literals shows the content instead of the prefix label and local part (since it does not have a URI). In the above example, the BL of instances *eg:Artist1* and *eg:FanPage1* is 'I/eg', and the BL of the predicate *foaf:fanpage* is 'P/foaf/fanpage'.

```
1  # predicates and edges
2    CONSTRUCT {
3      ?s explod:rEdgeTo [
4        explod:blabel concat("P/", str(?g), "/", namespace(?p), "/", localname(?p)) ;
5        explod:rEdgeTo ?o . ] .
6    } WHERE { GRAPH ?g { ?s ?p ?o. FILTER (!isLiteral(?o))}.}
7  # classes
8    CONSTRUCT { ?c explod:blabel concat("C/", namespace(?c), "/", localname(?c)) }
9    WHERE { ?u rdf:type ?c . }
10 # instances (everything else except literals)
11   CONSTRUCT { ?s explod:blabel concat("I/",namespace(?s))}
12   WHERE { OPTIONAL { ?s explod:blabel ?blabel }
13     FILTER (!bound(?blabel) && !isLiteral(?s)) . }
```

Fig. 2. SPARQL queries that construct a labeled graph

We now describe the construction of the labeled graph; the computation of bisimulation contractions using the labeled graph is shown in Section 2.2. A *labeled graph*, $R = (V, E, Label, \lambda)$, is a graph with nodes V, unlabeled directed edges $E \subseteq V \times V$, a set of BLs $Label$, and a function λ that assigns each node in V a BL in $Label$. $\langle v_1, v_2 \rangle \in E$ denotes a directed edge from v_1 to v_2, where $v_1, v_2 \in V$. BLs are assigned based on an adaptation of the common RDF graph representation, in which edges between nodes are labeled with a predicate, into a graph with labeled nodes and unlabeled edges. A node is created in the labeled graph for each predicate as a way to convert the predicate-labeled edge to an unlabeled edge. The creation of a labeled graph from the triple on line 4 of Figure 1(a) is depicted in Figure 1(b) above. The *ExpLOD RDF graph* is a set of RDF statements (within ExpLOD's application context) which describes the labeled graph. Oval nodes represent nodes in the ExpLOD RDF graph and each node's BL values is in a square node. Notice that representing the predicate with a node (a blank node in this case) is a form of reification [22]. Solid edges represent statements in the ExpLOD RDF graph, and dashed edges represent statements in the original RDF graph. The underlying label graph can be recovered from the ExpLOD graph.

Labeled graphs can be constructed using an ordered set of SPARQL CONSTRUCT queries that assign the BLs to RDF data; 3 example queries are shown in Figure 2. Each query generates a BL using the extension function *concat* (similar to the *fn:concat* function in XQuery) to concatenate strings. The BL of each node is ascribed as the string value of an *explod:blabel* predicate. A *explod:rEdgeTo* RDF property is used to represent directed edges between nodes.

2.2 Bisimulation Contractions

Bisimulation contractions are computed on sets of neighbourhoods. A *labeled bisimulation* between two labeled graphs X, Y is a symmetric equivalence relation $\approx \subseteq V_X \times V_Y$ such that, for $x \in V_X$, $y \in V_Y$: if $x \approx y$, then $\lambda(x) = \lambda(y)$; if $x \approx y$, and $\langle x, x' \rangle \in E_X$, then $\lambda(x) = \lambda(y)$, $\langle y, y' \rangle \in E_Y$, and $x' \approx y'$. A partition of a set of nodes is a set of pairwise disjoint subsets whose union is the set of nodes; each subset is called a *partition block*. The *coarsest stable partition* (CSP) of a set of nodes is a partition such that, for each pair of partition blocks B_1, B_2, either: each node in B_1 has an edge to a node in B_2, or no nodes in B_1

(a) Nested View (b) ExpLOD view

Fig. 3. Example: Class usage summary

have an edge to a node in B_2. According to [14], computing the CSP produces the bisimulation contraction, also known as the *maximum bisimulation*. That is, all nodes in a partition block are equivalent according to bisimulation, and no pair of nodes from different partition blocks are equivalent.

An RDF summary is a graph with labeled nodes and unlabeled edges. To distinguish between a labeled graph node and a summary node, we refer to a summary node as a *summary block* (or just 'block'). There is a block in the RDF summary for each partition block in the CSP, and each block in the summary has an *extent* that represent its associated partition block. There is an edge in the summary from block B_1 to block B_2 if each node in the extent of block B_1 has an edge to a node in the extent of block B_2.

Figure 3(a) is a class usage summary of the RDF datasets in Figure 1(a). It is a CSP of blocks whose extents contain nodes having the same bisimulation contraction. There is an instance block (a block whose label starts with 'I/eg', the common BL of the bisimilar nodes in its extent) that contains an instance that is both music artist and a person, i.e., *eg:Artist1*. There is also an instance block that is just a music artist (*eg:Artist2*), and there is an instance block containing 3 instances in its extent that are documents (the three fanpage instances). The statement identifier is included for reference to the statements in Figure 1(a). The same class usage summary that ExpLOD displays is shown in Figure 3(b), with the extent size shown between parentheses, and a unique block identifier for instance blocks within square brackets.

We have shown the construction of each node's BL based on RDF usage, and how bisimulation contractions are computed based on BLs. In the next section, we showcase how using different BL labels and RDF usage neighbourhoods can be used to explore the LOD cloud.

3 Exploring the LOD Cloud Using ExpLOD

In this section we showcase examples of RDF usage summaries to describe datasets in the LOD cloud. Section 3.1 shows the flexibility of modifying the BL scheme to obtain a coarser or more detailed summary. Section 3.2 demonstrates how a larger RDF usage neighbourhood can be employed to describe interlinked datasets. Finally, Section 3.3 describes one of the main features of

ExpLOD, which is the ability to return to the user, for any one block in a summary, the SPARQL query that returns the resources grouped in the extent of a block.

3.1 Class and Predicate Usage

A class and predicate RDF usage summary shown in Figure 4 is used to describe Jamendo, a dataset from the LOD cloud containing information about music artists and their productions. Our goal is to understand how tracks and records are described by examining the interaction of class and predicate instantiation, and class and predicate usage, of records and tracks in Jamendo.

Class instantiation is reported in the extent size of each class block. For example, the extent size of class block 5 (with BL prefix 'C/') shows that there are 5,786 instances that are typed as records. The class usage of instance block 1430 is the singleton *mo:Track* since there is only one such path to the class block with BL 'C/mo/Track' . Predicate instantiation is reported in the extent size of each predicate block (with BL prefix 'P/'). For example, the predicate *mo:license* (with BL 'P/mo/license') has been instantiated 45,634 times. The predicate usage of all tracks in the extent of instance block 1430 is the set of predicates {*mo:license*, *mo:track_number*, *mo:available_as*, *dc:title*, *rdf:type*}, visible by the edges from the instance block to each of those predicate blocks. Notice that each instance block has a unique class and predicate usage. Amongst records, there is variation in the extent size of predicate usages - the extent size of instance block 1361 is 4,509 compared to the extent size of instance block 1388 that contains only 1 record instance.

An RDF usage summary with many blocks can be difficult to understand. Using a reduced portion of the BL hierarchy can sometimes produce a summary with fewer blocks, and as many blocks as before (in the worst case). For example, excluding the local part of each predicate's BL groups predicates by their namespace, reducing the 8 instance blocks in Figure 4 to 3 in Figure 5.

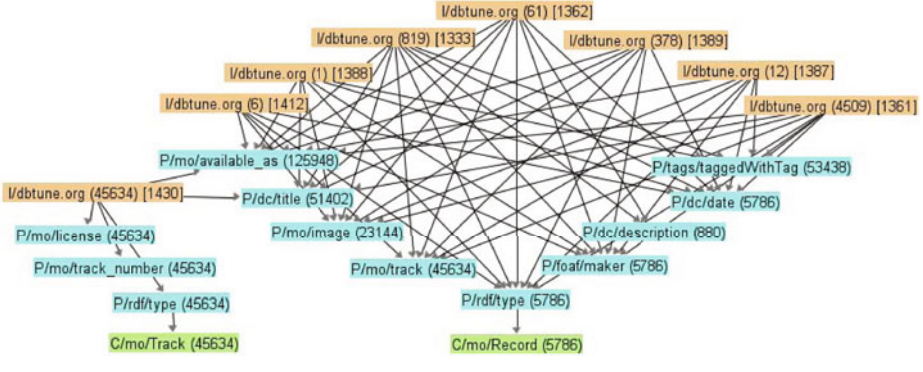


Fig. 4. Jamendo: RDF usage summary of class and predicate instantiation, and class and predicate usage

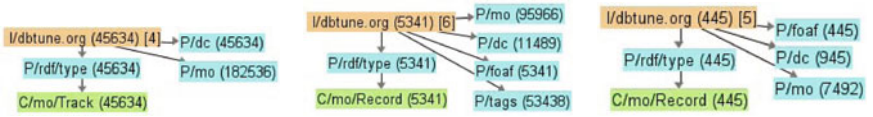


Fig. 5. Jamendo: predicate usage summary of records and tracks grouped by namespace

3.2 Interlinking

Two interlinked RDF datasets in the LOD cloud may contain information about the same real-world entity, but it is possible that each dataset uses its own unique URI to represent it. A triple with an *owl:sameAs* predicate captures the information that the subject and object URIs refer to the same real-world entity, referred to as *URI-equivalence*; such statements can be found readily within many datasets in the LOD cloud. A resource's description in one dataset may not match the description of its URI-equivalent resource in another dataset.

An *interlink usage* summary, which we define as a bisimulation contraction of URI-equivalent resources, is used to understand each dataset's contribution to a real-world entity's description. In our work, an interlink usage neighbourhood is defined as the subgraph that includes the nodes and edges on the path linking the subject of a triple whose predicate is *owl:sameAs* to the object node. Additionally, the incoming edge of each *owl:sameAs* predicate node is reversed, both in the labeled graph and the ExpLOD graph, so that it points to the subject. Since we are interested in the description of instances, we consider only statements in which the subject and object are instances. In addition to creating a summary solely based on interlink usage neighbourhoods, considering class and predicate usage neighbourhoods within each dataset provides additional information about URI-equivalent resource descriptions.

Figure 6 shows the blocks of instances that are interlinked, have the same class and predicate usage in one dataset, and have the same class usage in a second dataset. This example covers two datasets, Southampton and Newcastle, from the RKB Explorer collection. This image shows that there are 57 instances in Southampton with the same class and predicate usage whose equivalent instances in Newcastle have the same class usage. It also shows that both namespaces use the class *aktors:Publication-Reference* to the class usage of these resources and that the predicate usage of the instances from Southampton includes the predicates *aktors:portal#has-title* and *aktors:portal#has-publication-reference*. Thus,

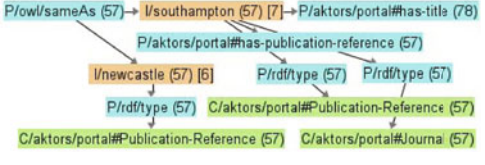


Fig. 6. Southampton and Newcastle: Interlink usage summary

we have shown how interlinking in the LOD cloud can be explored by computing bisimulation contractions of interlink usage neighbourhoods.

3.3 SPARQL Block-Extent Queries

To find which instances are in a block's extent, the extent needs to be materialized. A *SPARQL block-extent query* (SBE) is used to materialize the extent of a block. An SBE is a SELECT query that returns the instances in a block extent by using negation-by-failure (with keywords OPTIONAL, FILTER, and !BOUND) as a way to retrieve instances with a specific bisimulation contraction. The bisimulation contraction is specified in an SBE by including the portions of the neighbourhood that should be matched and using negation-by-failure to exclude the parts of the neighbourhood should not be present in the neighbourhood. Futher details of how the SBE graph pattern is generated is described in Section 4.2. The SBE of Figure 7 returns the 45,634 instances in the extent of instance block 1430 in Figure 4. The SBE specifies the class usage and predicate usage on lines 2 and 3, respectively; these are the required portions of the class usage neighbourhood. Negation-by-failure is used to exclude portions of the class usage on lines 4-6, and similarly on lines 7-9 for excluding portions of the predicate usage.

```
1 SELECT ?U WHERE {
2   ?U rdf:type mo:Track ;
3     mo:track_number [] ; dc:title [] ; mo:license [] ; mo:available_as [] .
4   OPTIONAL { ?U rdf:type ?o0 .
5     FILTER (?o0 != mo:Track) .
6   } FILTER ( !bound(?o0) ) .
7   OPTIONAL { ?U ?prop ?o .
8     FILTER (?prop != mo:available_as && ?prop != mo:track_number && ?prop != mo:license && ?prop
      != dc:title && ?prop != rdf:type ) .
9   } FILTER ( !bound(?prop) ) . }
```

Fig. 7. SBE to materialize extent of block 1430 in Figure 4

4 ExpLOD Implementation

In this section, we describe the two techniques used by ExpLOD to create RDF summaries based on bisimulation contractions. Section 4.1 references an existing partition-refinement approach. Section 4.2 focuses on a main contribution of this work, a SPARQL-based approach to compute bisimulation contractions from SPARQL-generated BLs and neighbourhoods of interest specified in SPARQL.

4.1 Partition-Refinement Approach

The PRAIG implementation is covered only briefly here, details are given in [19]. Neighbourhoods to summarize are obtained using an intersection-automaton construction. The set of neighbourhoods is obtained from intersecting the automaton representation of a path regular expression that selects neighbourhoods in the labeled graph, and the automaton representation of a labeled graph. From

[14], the CSP representing an equivalence relation based on bisimulation contractions is obtained by using a partition-refinement algorithm. PRAIG uses a partition refinement algorithm from [23] to create the CSP. A side-effect of partition-refinement is that each block's extent is *materialized*, that is, pointers are created originating from each block and directed to the resources in its extent. The next section shows how it is possible to create a partition of blocks without materializing the extents, while retaining the ability to materialize the extent of individual blocks at a later time.

4.2 SPARQL Approach

In order to compute the blocks of the summary using SPARQL, the BLs need to be known before-hand. A *SPARQL label retrieval query* (SLRQ) is used to retrieve the distinct labels of a neighbourhood over which bisimulation contractions are to be computed.

We define an SLRQ as a SPARQL graph pattern as in [15], with variables and terms (such as URIs and literals) that returns subgraphs with each variable bound to a value. The SLRQ of Figure 8(a) picks the class usage neighbourhoods, i.e., it returns neighbourhoods where instances are connected to classes via *rdf:type* predicates. The neighbourhood is based on the RDF representation of the labeled graph shown in Figure 1(b). Lines 3 and 4 specify a path from a node $n1$ to a node $n3$ via a node $n2$, where edges are represented by the predicate *explod:rEdgeTo*. Each node's BL is bound to a variable in lines 5-7, and the labels of these variables are then filtered in lines 8-10 according to a class usage neighbourhood, where an instance (with BL prefix 'I/') has an edge in the labeled graph to a predicate *rdf:type* (having a BL of 'P/rdf/type') followed by an edge to a class (with BL prefix 'C/'). Thus, an SLRQ returns the BLs that are present in a neighbourhood of interest, and next, we will show how to use the BLs returned by an SLRQ to compute the bisimulation contraction of that neighbourhood, also in SPARQL.

A SPARQL block query (SB) is a way of returning the blocks of an RDF summary without materializing any block extents. An SB query obtains the unique bisimulation contractions using the labels from the SLRQ and a *SPARQL block pattern* (SBP), a graph pattern that selects each possible bisimulation contraction for the neighbourhoods of interest. The SBP is generated from an SLRQ as follows. The SLRQ specifies a neighbourhood of interest and, after obtaining the

```
1  SELECT DISTINCT ?n1_bl ?n2_bl ?n3_bl
2  WHERE {
3    ?n1 explod:rEdgeTo ?n2 .
4    ?n2 explod:rEdgeTo ?n3 .
5    ?n1 explod:blabel ?n1_bl .
6    ?n2 explod:blabel ?n2_bl .
7    ?n3 explod:blabel ?n3_bl .
8    FILTER (regex(?n1_bl, "I/.*")) .
9    FILTER (?n2_bl = "P/rdf/type") .
10   FILTER (regex(?n3_bl, "C/.*")) . }
```

(a) SLRQ

```
1  SELECT DISTINCT count(?s) ?c0 ?c1 ?c2 WHERE {
2  OPTIONAL { ?s rdf:type ?c0 FILTER(?c0 = foaf:Person)}
3  OPTIONAL { ?s rdf:type ?c1 FILTER(?c1 = mo:MusicArtist)}
4  OPTIONAL { ?s rdf:type ?c2 FILTER(?c2 = foaf:Document)}}
```

(b) SPARQL block query

Fig. 8. Example: SPARQL approach to compute class usage summary

BL values that are possible for each possible block, an OPTIONAL code block is created for each of its possible BL values. The query then displays the blocks that are instantiated by each bisimulation contraction. It is also possible to use a shortcut that uses the URI directly instead of the BL, such for classes. The SBP is also used to generate an SBE such as the example shown in Section 3.3.

Figure 8(b) shows an example SB in which a shortcut is used to group instances from the RDF dataset in Figure 1(a) having the same bisimulation contraction of their class usage neighbourhoods. Line 2 specifies the neighbourhood of the SBP. Each possible distinct class BL is specified within an OPTIONAL code block (lines 2 through 4) and the keyword DISTINCT on line 1 causes the SB to return only unique bisimulation contractions based on the optional code blocks that are matched. That is, the set of possible BL values that were returned by the SLRQ for node $n3$ in Figure 8(b) are *mo:MusicArtist*, *foaf:Person*, and *foaf:Document*. The BL values for nodes $n1$ and $n2$ are excluded for brevity since their BL values are singletons, 'I/eg' and 'P/rdf/type', respectively, and their exclusion does not modify the results of using the DISTINCT keyword. Notice that the BL value of $n2$ is specified directly within the class usage neighbourhoods of lines 2 through 4 as *rdf:type*. Optionally, since some SPARQL implementations support aggregation, it is also possible to return a block's extent size as part of the SB, and this is shown on line 1 of the query as *count(?s)*. The set of results returned by running the SB would be {*(1, foaf:Person, mo:MusicArtist, null), (1, null, mo:MusicArtist, null), (3, null, null, foaf:Document)*} and this matches the class usage summary visualized in Figure 3(b).

Thus far, we have shown how each resource is assigned a BL that captures each resource's RDF usage. Then, using the BLs, bisimulation contractions are computed using one of two implementations in ExpLOD, a graph-based approach, or a SPARQL-based approach that takes advantage of SPARQL-generated BLs. We have also shown how using different BLs can help to control the size of a summary (to reduce its complexity), and how considering different neighbourhoods of interest can help analyze interlinking in the LOD cloud. This completes our study of the RDF usage summaries generated using our ExpLOD tool. In the next section, we compare the performance of graph-based and SPARQL-based approaches that are implemented in ExpLOD.

5 Experimental Study

In this section, we compare the performance of the two techniques used by ExpLOD and covered in the preceding section.

ExpLOD creates summaries based on the following input: the dataset, the BL scheme, and the neighbourhoods to consider. The summaries produced can be viewed and explored in an interactive graphical environment and they can also be exported in a variety of formats (including RDF). ExpLOD is a Java application developed within the Eclipse environment with support for plug-ins. Custom code was developed for all graph and automaton data structures used by PRAIG and the Jena toolkit (`jena.sourceforge.net`) is used to manage RDF data. Explod can also invoke the Virtuoso RDF store (`www.openlinksw.com/virtuoso`).

In the experiments, we use Virtuoso's default configuration, except that checkpoint logs are disabled.

Section 5.1 describes the datasets considered in the experiments. Section 5.2 gives a performance evaluation of summary creation using ExpLOD. All the experiments used a dedicated AMD Opteron 2.4 GHz server hosting a Windows XP virtual machine with 2 GB of RAM (of which 1.5 GB was assigned to the PRAIG JVM).

5.1 Datasets

The first 7 columns of Table 1 shows the datasets from the LOD cloud that are considered for the experiments. The table columns show the following information about each dataset: its name, the number of triples it contains, and the number of instance (whose BL starts with 'I/'), class ('C/'), and predicate ('P/') nodes in its labeled graph. The next two columns show the number of instance blocks, blocks whose label starts with 'I/', that are in the class usage summary ('#S1'), and in the class and predicate usage summary ('#S2'). The instance blocks are counted as we are interested in exploring the interaction of classes and predicates in instance descriptions, and the number of instance blocks reported is from PRAIG, except in the case of LinkedCT and LinkedMDB. These two dataset did not fit in virtual-memory and SB was used instead; however, due to query limitations in Virtuoso's SPARQL engine (described further in Section 5.2), the exact number of instance blocks in their class and predicate usage summary (column '#S2') could not be determined exactly and is a lower-bound (shown in italics).

These datasets were chosen as they vary in the amount and type of information they describe. The largest dataset (counting the number of triples) examined by PRAIG is Jamendo with over 1 million triples and its labeled graph contains 410,784 instance, 11 class, and 25 predicate nodes. The largest dataset examined using SB and SBE is LinkedCT that contains over 7 million statements and whose labeled graph contains 860,510 instance, 13 class, and 91 predicate nodes. Notice that, even though LinkedCT is the dataset with the most number of state-

Table 1. LOD dataset graphs and performance (in ms) of PRAIG, SB, and SBE

							Class usage			Class and predicate usage		
Dataset	Triples	I/	C/	P/	#S1	#S2	PRAIG	SB	SBE	PRAIG	SB	SBE
Dailymed	116,992	9,623	2	26	4	440	7,269	178	168	184,169	307,913	5,874
Diseasome	69,639	20,165	2	18	5	13	4,556	248	241	8,847	148,608	655
EPSRC	340,064	40,913	13	33	14	31	27,413	6,123	1,877	47,412	56,962	4,644
ESWC2007	7,256	1,315	35	48	35	74	1,844	867	63	14,981	2,113	915
ESWC2008	4,628	1,052	23	66	25	53	662	220	42	6,075	126,111	637
Jamendo	1,047,837	410,784	11	25	12	35	158,600	30,652	16,876	165,625	6,868,293	1,700,131
LinkedCT	7,025,488	860,510	13	91	13	*52*	—	91,715	34,972	—	*361,075*	1,314,477
LinkedMDB	3,579,594	757,878	41	148	39	*49*	—	218,091	19,631	—	*452,232*	150,998
Magnatune	169,004	41,199	7	24	8	8	12,769	2,028	1,026	20,875	9,633	2,973
Peel	271,369	76,894	9	25	10	28	23,638	5,523	3,338	35,806	30,460	31,852
Southampton	219,019	30,721	21	31	45	151	17,894	6,583	1,241	54,516	50,628	62,068

ments, there are datasets with more classes or predicates, such as LinkedMDB with 41 classes and 148 predicates. Despite the large number of variations possible in the interaction of classes and predicates in these large dataset, it is the Dailymed dataset that contains the most number of instance blocks in its class and predicate usage summary (it has 440) when it only has 2 classes. Jamendo's C-summary has 12 instance blocks and its class usage summary and 35 instance blocks in its class and predicate usage summary, and LinkedCT has 39 instance blocks in its class usage summary and 49 instance blocks in its class and predicate usage summary. Furthermore, aside from Dailymed and Diseasome, none of the datasets have as many instance blocks in their class usage summary as their combinatorial potential, showing limited variation in how classes and predicates are used.

5.2 Performance Results

To show the efficiency of the two techniques shown in the previous section, we conduct a performance evaluation. Performance is measured as the time taken to compute an RDF usage summary, assuming that the labeled graph has already been created. Performance of computing class usage summaries, and class and predicate usage is explored for datasets enumerated in Section 5.1.

PRAIG performance includes materializing extents (as it is part of its algorithm). We report the time of using SBE to materialize the extent of all instance blocks computed by SB. In Virtuoso, computing SB queries containing more than 7 distinct predicates sometimes took hours to complete. Additionally, SB queries could not contain more than 63 distinct predicates, disallowing generating SB queries for datasets such as LinkedMDB since it contains 148 predicates. To reduce the number of predicates an SB contained, an SB query was created for each class usage and a predicate was included in an SB only if it described at least one instance of that class usage. The workaround was used with all datasets (even if they did not violate the predicate limitation) and was implemented as follows. First, all distinct class labels present in the labeled graph we used to generate an SB to compute a class usage summary. Then, for each distinct class usage, the distinct predicate labels that described some instance having that class usage were used to generate a class and predicate usage SB whose class usage was bound. All the blocks returned from the predicate usage SBs form the final class and predicate usage summary. Despite this workaround, some modified SBs still contained more than 7 predicates, and these modified SBs are not included in the SB and SBE performance times reported.

The last 6 columns of Table 1 shows the performance (in ms) of PRAIG, SB, and SBE. PRAIG performance times are the average of 3 runs. SB and SBE performance times are the average of 8 runs. Although 10 runs were performed for SB and SBE, the first run of SB was sometimes twice as slow as the remaining runs so 8 runs were average, excluding the first run and a performance outlier,

the time with the highest absolute deviation from the average of 9 runs (that excludes the first run). PRAIG required around 7 s to compute the class usage summary of Dailymed, 178 ms using SB, and 168 ms to materialize the 4 instance blocks using SBE.

PRAIG took around 184 s to compute the class and predicate usage summary of Dailymed, 307 s using SB, and almost 6 s to materialize the 440 instance blocks using SBE. Because PRAIG cannot be used for datasets larger than virtual-memory, it was not used to compute the RDF usage summaries for the LinkedMDB and LinkedCT datasets, and is shown as '—' in the table.

We now discuss our findings on the datasets considered. SB is faster than PRAIG for computing class usage summaries. PRAIG consistently computes class and predicate usage summaries faster than SB; however, SB can be used on datasets that do not fit in virtual-memory (with some limitations). Although using SB to compute class and predicate usage summaries for Peel and Southampton is comparable to PRAIG's performance, adding SBE's time more than doubles the SPARQL-based approach's time. The cost of SBE relative to SB ranges from less than 1 percent for ESWC2008 to more than 3 times for LinkedCT.

Since no PRAIG run deviated more than a second from the times reported, the standard deviation of PRAIG's performance is not reported. By excluding the first run, the relative standard deviation of computing class usage summaries with SB decreased from around 60% to 14%, with a further reduction to around 7% by excluding the performance outlier. The relative standard deviation of computing class and predicate usage summaries with SB decreased from around 36% to 9%, then to 5% without the performance outlier. The relative standard deviation of computing SBE for the class usage summary changed from 60% to 34% then to 19%. The relative standard deviation of computing SBE for the class and predicate usage summary changed from 10% to 9% then to 4%.

6 Conclusion

RDF summaries constructed using labeled bisimulation contractions is a way to reveal the unique structure in RDF datasets from the LOD cloud. Different BL hierarchies allow flexible RDF summary construction such as to examine existing and construct new RDF usages. Performance of the graph-based approach shows a distinct advantage over SPARQL-based approach with the limitation that it works only with datasets that fit in main-memory. The SPARQL-based approach can be used for large datasets with some limitations showing that there is potential to explore the implementation of labeled bisimulation contractions in existing systems. Future work also involves moving the graph-based approach to a distributed environment.

Acknowledgements. The first author was financially supported by an IBM CAS Fellowship. We also thank the reviewers for their detailed comments.

References

1. Yahoo! SearchMonkey, http://developer.yahoo.com/searchmonkey (last Accessed: December 21, 2009)
2. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing Linked Datasets - On the Design and Usage of voiD, the Vocabulary of Interlinked Datasets. In: LDOW (2009)
3. Ali, M.S., Consens, M.P., Kazai, G., Lalmas, M.: Structural relevance: a common basis for the evaluation of structured document retrieval. In: CIKM, pp. 1153–1162 (2008)
4. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A Nucleus for a Web of Open Data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007)
5. Biron, P.V., Ashok Malhotra, W.W.W.C.: W3C Recommendation. XML Schema Part 2: Datatypes Second Edition, October 28 (2004), http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/
6. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. IJSWIS 5(3), 1–22 (2009)
7. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named Graphs, Provenance and Trust. In: WWW, pp. 613–622 (2005)
8. Cheng, G., Ge, W., Qu, Y.: Falcons: Searching and Browsing Entities on the Semantic Web. In: WWW, pp. 1101–1102 (2008)
9. Consens, M.P., Rizzolo, F.: Fast answering of XPath query workloads on web collections. In: Barbosa, D., Bonifati, A., Bellahsène, Z., Hunt, E., Unland, R. (eds.) XSym 2007. LNCS, vol. 4704, pp. 31–45. Springer, Heidelberg (2007)
10. Consens, M.P., Rizzolo, F., Vaisman, A.A.: AxPRE summaries: Exploring the (semi-)structure of XML web collections. In: ICDE, pp. 1519–1521 (2008)
11. Cyganiak, R., Stenzhorn, H., Delbru, R., Decker, S., Tummarello, G.: Semantic Sitemaps: Efficient and Flexible Access to Datasets on the Semantic Web. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 690–704. Springer, Heidelberg (2008)
12. Ding, L., Finin, T.: Characterizing the Semantic Web on the Web. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 242–257. Springer, Heidelberg (2006)
13. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: A Search and Metadata Engine for the Semantic Web. In: CIKM, pp. 652–659 (2004)
14. Dovier, A., Piazza, C., Policriti, A.: An efficient algorithm for computing bisimulation equivalence. Theor. Comput. Sci. 311(1-3), 221–256 (2004)
15. Klyne, G., Carroll, J. (eds.): Resource Description Framework (RDF): Concepts and Abstract Syntax, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
16. Glaser, H., Millard, I., Jaffri, A.: RKBExplorer.com: A Knowledge Driven Infrastructure for Linked Data Providers. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 797–801. Springer, Heidelberg (2008)
17. Hassanzadeh, O., Consens, M.P.: Linked Movie Data Base. In: I-SEMANTICS, pp. 194–196 (2008)

18. Hausenblas, M., Halb, W., Raimond, Y., Heath, T.: What is the Size of the Semantic Web? In: I-SEMANTICS, pp. 9–16 (2008)
19. Khatchadourian, S., Consens, M.P.: ExpLOD: Exploring Interlinking and RDF Usage in the Linked Open Data Cloud. Technical Report (2009), http://www.cs.toronto.edu/~shahan/tr/explodtechreport090901.pdf
20. Kinsella, S., Bojars, U., Harth, A., Breslin, J.G., Decker, S.: An interactive map of semantic web ontology usage. In: IV, pp. 179–184 (2008)
21. Maduko, A., Anyanwu, K., Sheth, A.P., Schliekelman, P.: Graph Summaries for Subgraph Frequency Estimation. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 508–523. Springer, Heidelberg (2008)
22. Manola, F., Miller, E.: RDF Primer, W3C Recommendation, February 10 (2004), http://www.w3.org/TR/REC-rdf-syntax/
23. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. 16(6), 973–989 (1987)
24. Stickler, P.: CBD - Concise Bounded Description, http://www.w3.org/Submission/2005/SUBM-CBD-20050603/
25. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A Core of Semantic Knowledge. In: WWW, pp. 697–706 (2007)
26. Tummarello, G., Delbru, R., Oren, E.: Sindice.com: Weaving the Open Linked Data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 552–565. Springer, Heidelberg (2007)
27. Tummarello, G., Morbidoni, C., Puliti, P., Piazza, F.: Signing individual fragments of an RDF graph. In: WWW, pp. 1020–1021 (2005)
28. Vassiliadis, P., Sellis, T.: A survey of logical models for OLAP databases. SIGMOD Rec. 28(4), 64–69 (1999)
29. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and Maintaining Links on the Web of Data. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 650–665. Springer, Heidelberg (2009)