Quotient RDF Summaries Based on Type Hierarchies

Pawel Guzewicz, Ioana Manolescu

Quotient RDF Summaries Based on Type Hierarchies

Paweł Guzewicz
Université Paris-Saclay,
Inria and École Polytechnique, France
pawel.guzewicz@telecom-paristech.fr

Ioana Manolescu
Inria and École Polytechnique,
Université Paris-Saclay, France
ioana.manolescu@inria.fr

***Abstract*—Summarization has been applied to RDF graphs to obtain a compact representation thereof, easier to grasp by human users. We present a new brand of quotient-based RDF graph summaries, whose main novelty is to summarize together RDF nodes belonging to the same type hierarchy. We argue that such summaries bring more useful information to users about the structure and semantics of an RDF graph.**

I. Introduction

The structure of RDF graphs is often complex and heterogeneous, making them hard to understand for users who are not familiar with them. This problem has been encountered in the past in the data management community, when dealing with other semi-structured graph data formats, such as the Object Exchange Model (OEM, in short) [1].

Structural summaries for RDF graphs. To help discover and exploit such graphs, [2], [3] have proposed using *Dataguide summaries* to represent compactly a (potentially large) data graph by a smaller one, computed from it. In contrast with relational databases where the schema is fixed before it is populated with data (*a priori* schema), a summary is computed *from* the data (*a posteriori* schema). Each node from the summary graph *represents*, in some sense, a set of nodes from the input graph. Many other graph summarization proposals have been made, for OEM [4], later for XML trees with ID-IDREF links across tree nodes (thus turning an XML database into a graph) [5], [6], [7], and more recently for RDF [8], [9], [10], [11], [12], [13]; many more works have appeared in this area, some of which are presented in a recent tutorial [14]. Related areas are concerned with graph compression, e.g. [15], ontology summarization [16] (focusing more on the graph semantics than on its data) etc.

Quotient-based summaries are a particular family of summaries, computed based on a (summary-specific) notion of *equivalence* among graph nodes. Given an equivalence relation $\equiv$, for each equivalence class C (that is, maximal set of graph nodes comprising nodes all equivalent to each other), the summary has exactly one node n_C in the summary. Example of quotient-based summaries include [4], [5], [6], [7], [17], [11], [10], [12]; other summaries (including Dataguides) are not quotient-based.

This work is placed within the *quotient-based RDF summarization framework* introduced in [12]. That framework adapts the principles of quotient-based summarization to RDF graphs, in particular preserves the semantics (ontology), which may come with an RDF graph, in its summary. This is important as it guarantees that any summary defined within the framework is *representative*, that is: a query having *answers* on an RDF graph, has answers on its summary. This allows to use summaries as a first user interface with the data, guiding query formulation. Note that here, *query answers* take into account both the data explicitly present in the RDF graph, and the data implicitly present in the graph, through reasoning based on the explicit data and the graph's ontology.

Two RDF summaries introduced in [11] have been subsequently [18] redefined as quotients. They differ in their treatment of the *types* which may be attached to RDF graph nodes. One is focused on summarizing the structure (non-type triples) first and copies type information to summary nodes afterwards; this may erase the distinctions between resources of very different types, leading to confusing summaries. The other starts by separating nodes according to their *sets of types* (recall that an RDF node may have one or several types, which may or may not be related to each other). This ignores the relationships which may hold among the different classes present in an RDF graph.

Contribution and outline. To simultaneously avoid the drawbacks of the two proposals above, in this paper we introduce a novel summary based on the same framework. It features a *refined treatment of the type information* present in an RDF graph, so that *RDF graph nodes which are of related types are represented together in the summary*. We argue that such a summary is more intuitive and more informative to potential users of the RDF graph.

The paper is organized as follows. We recall the RDF graph summarization framework introduced in [12] which frames our work, as well as the two abovementioned concrete summaries. Then, we formally define our novel summary, and briefly discuss a summarization algorithm and its concrete applicability.

II. RDF Graphs and Summaries

A. RDF and RDF Schema

We view an RDF graph `G` as a set of *triples* of the form `s p o`. A triple states that its *subject* `s` has the *property* `p`, and the value of that property is the *object* `o`. We consider only well-formed triples, as per the RDF specification [19], using uniform resource identifiers (URIs), typed or untyped literals (constants) and blank nodes (unknown URIs or literals).

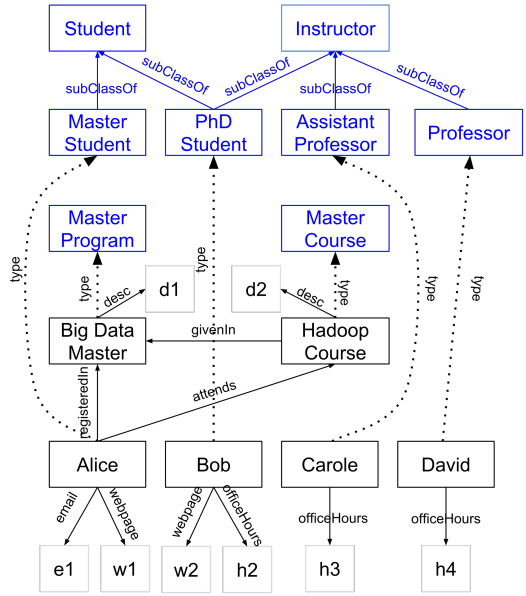

Figure 1: Sample RDF graph.

The RDF standard [19] includes the property rdf:type (τ in short), which allows specifying the type(s) or class(es), of a resource. Each resource can have zero, one or several types, which may or may not be related. We call the set of G triples, whose property is τ, the *type triples of* G, denoted TG.

RDF Schema and entailment. G may include a *schema (ontology)*, denoted SG, and expressed through RDF Schema (RDFS) triples using one of the following standard properties: *subclass*, *subproperty*, *domain* and *range*, which we denote by the symbols $\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_d$ and $\hookrightarrow_r$, respectively. Our proposal is beneficial in the presence of $\prec_{sc}$ schema statements; we do not constrain SG in any way.

RDF entailment is the mechanism through which *implicit* RDF triples are derived from explicit triples and schema information. In this work, we consider four *entailment rules*, each based on one of the four properties above: (i) $c_1 \prec_{sc} c_2$ means any resource of type c_1 is also of type c_2; (ii) $p_1 \prec_{sp} p_2$ means that as long as a triple $s\ p_1\ o$ belongs to G, the triple $s\ p_2\ o$ also holds in G; (iii) $p \hookleftarrow_d c$ means that any resource s having the property p in G is of type c, that is, $s\ \tau\ c$ holds in G; finally (iv) $p \hookrightarrow_r c$ means that any resource that is a value of the property p in G, is also of type c.

The fixpoint obtained by applying entailment rules on the triples of G and the schema rules in SG until no new triple is entailed, is termed *saturation* (or closure) of G and denoted G^∞. The saturation of an RDF graph is unique (up to blank node renaming), and does not contain implicit triples (they have all been made explicit by saturation).

We view an RDF graph G as: $G = SG \cup TG \cup DG$, where the schema SG and the type triples TG have been defined above; DG contains all the remaining triples, whose property is neither τ nor $\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_d$ or $\hookrightarrow_r$. We call DG the *data triples* of G.

In the presence of an RDFS ontology, **the semantics of an RDF graph is its saturation**; in particular, *the answers to a query posed on* G *must take into account all triples in* G^∞ [19].

Figure 1 shows an RDF graph we will use for illustration in the paper. Schema nodes and triples are shown in blue; type triples are shown in dotted lines; boxed nodes denote URIs of classes and instances, while d1, d2, e1 etc. denote literal nodes; "desc" stands for "description".

B. Quotient RDF summaries

We recall the summarization framework introduced in [12]. In a graph G, a *class node* is an URI appearing as subject or object in a $\prec_{sc}$ triple, as object in a $\hookleftarrow_d$ or $\hookrightarrow_r$ triple, or as object in a τ triple. A *property node* is an URI appearing as subject or object in a $\prec_{sp}$ triple, or as subject in a $\hookleftarrow_d$ or $\hookrightarrow_r$ triple. The framework brings a generic notion of *equivalence* among RDF nodes:

Definition 1: (RDF EQUIVALENCE) Let $\equiv$ be a binary relation between the nodes of an RDF graph. We say $\equiv$ is an *RDF equivalence relation* iff (i) $\equiv$ is reflexive, symmetric and transitive, (ii) any class node is equivalent w.r.t. $\equiv$ only to itself, and (iii) any property node is equivalent w.r.t. $\equiv$ only to itself.

Graph nodes which are equivalent will be summarized (or *represented*) by the same node in the summary. The reason behind class and property nodes being only equivalent to themselves in every RDF equivalence relation, is to ensure that each such node is preserved in the summary, as they appear in the schema and carry important information for the graph's semantics. A summary is defined as follows:

Definition 2: (RDF SUMMARY) Given an RDF graph G and an RDF node equivalence relation $\equiv$, the *summary of* G *by* $\equiv$ is an RDF graph denoted $G_{/\equiv}$ and defined as follows:

- $G_{/\equiv}$ contains exactly one node for each equivalence class of G nodes through $\equiv$; each such node has a distinct, "fresh" URI (that does not appear in G).
- For each triple $s\ p\ o \in G$ such that $s_\equiv$, $o_\equiv$ are the $G_{/\equiv}$ nodes corresponding to the equivalence classes of s and o, the triple $s_\equiv\ p\ o_\equiv$ belongs to $G_{/\equiv}$.

The above definition can also be stated "$G_{/\equiv}$ is the quotient graph of G by the equivalence relation $\equiv$", based on the classical notion of quotient graph[1]. We make two observations:

- Regardless of the chosen $\equiv$, all SG triples are also part of $G_{/\equiv}$, as class and property nodes are represented by themselves, and thanks to the way $G_{/\equiv}$ edges are defined; indeed, G and $G_{/\equiv}$ *have the same schema*;
- No particular treatment is given to type triples: how to take them into account is left to each individual $\equiv$.

Different RDF equivalence relations lead to different summaries. At one extreme, if all data nodes are equivalent, the summary has a single data node; on the contrary, if $\equiv$ is "empty" (each node is equivalent only to itself), the summary degenerates into G itself. Well-studied equivalence relations for graph quotient summaries are based on the so-called forward, backward, or forward and backward (FB) bisimulation [4], [5], [6], [7]. It has been noted though, e.g. in [20], that RDF graphs

[1] https://en.wikipedia.org/wiki/Quotient_graph

exhibit so much structural heterogeneity that bisimulation-based summaries are very large, almost of the size of G, thus not very useful. In contrast, [11], [17] introduced ≡ relations which lead to compact summaries, many orders of magnitude smaller than the original graphs.

C. Types in summarization: first or last?

Let us consider how *type* triples can be used in quotient RDF summaries. Two approaches have been studied in the literature, and in particular in quotient summaries. The approach we will call *data-first* focuses on summarizing the *data (or structure)* of G, and then carries (or copies) the possible types of G nodes, to the summary nodes representing them. Conversely, *type-first* approaches summarize graph nodes first (or only) by their types. Below, we recall two quotient summaries described in [11], [18], which are the starting point of this work; they are both very compact, and illustrate the data-first and type-first approaches respectively. They both rely on the notion of property cliques:

Definition 3: (PROPERTY RELATIONS AND CLIQUES) Let p_1, p_2 be two data properties in DG:

1) $p_1, p_2 \in$ G are *source-related* iff either: (i) a data node in DG is the subject of both p_1 and p_2, or (ii) DG holds a data node r and a data property p_3 such that r is the subject of p_1 and p_3, with p_3 and p_2 being source-related.
2) $p_1, p_2 \in$ G are *target-related* iff either: (i) a data node in DG is the object of both p_1 and p_2, or (ii) DG holds a data node r and a data property p_3 such that r is the object of p_1 and p_3, with p_3 and p_2 being target-related.

A maximal set of properties in DG which are pairwise source-related (respectively, target-related) is called a *source* (respectively, *target*) *property clique*.

For example, in Figure 1, the properties *email* and *webpage* are source-related since Alice is the subject of both; *webpage* and *officeHours* are source-related due to Bob; also due to Alice, *registeredIn* and *attends* are source-related to the above properties, leading to a source clique SC_1 = {*attends, email, webpage, officeHours, registeredIn*}. Another source clique is SC_2 = {*desc, givenIn*}.

It is easy to see that the set of non-empty source (or target) property cliques is a partition over the data properties of DG. Further, all data properties of a resource $r \in$ G are all in the same source clique, which we denote $SC(r)$; similarly, all the properties of which r is a value are in the same target clique, denoted $TC(r)$. If r is not the value of any property (respectively, has no property), we consider its target (respectively, source) is $\varnothing$. For instance, in our example, SC_1 is the source clique of Alice, Bob, Carole and David, while SC_2 is the source clique of the BigDataMaster and of the HadoopCourse.

Definition 4: (WEAK EQUIVALENCE) Two data nodes are *weakly equivalent*, denoted $n_1 \equiv_W n_2$, iff: (i) they have the same non-empty source or non-empty target clique, or (ii) they both have empty source and empty target cliques, or (iii) they are both weakly equivalent to another node of G.

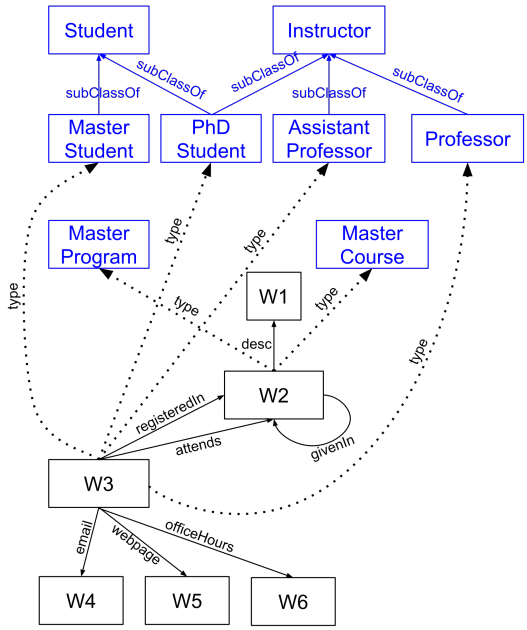

Figure 2: Weak summary of the sample RDF graph in Figure 1.

Definition 5: (WEAK SUMMARY) The *weak summary* of the graph G, denoted G_W, is the RDF summary obtained from the weak equivalence $\equiv_W$.

Figure 2 shows the weak summary of our sample RDF graph. The URIs W_1 to W_6 are "new" summary nodes, representing literals and/or URIs from G. Thus, W_3 represents Alice, Bob, Carole and David together, due to their common source clique SC_1. W_3 represents the course *and* the master program, due to their common source clique SC_2. Note that the *givenIn* edge from G leads to a summary edge from W_2 to itself; also, W_1 carries over the types of the nodes it represents, thus it is both of type *MasterProgram* and *MasterCourse*. This example shows that *data-first summarization may represent together* G *resources whose types clearly indicate their different meaning*; this may be confusing.

In contrast, the *typed weak* [18] summary recalled below illustrates the type-first approach:

Let $\equiv_T$ be an RDF equivalence relation which holds on two nodes iff they have the *exact same set of types*.

Let $\equiv_{UW}$ be an RDF equivalence relation which holds on two nodes iff (i) they have *no* type, *and* (ii) they are weakly equivalent.

Definition 6: (TYPED WEAK SUMMARY) The typed weak summary of an RDF graph G, denoted G_{TW}, is the summary through $\equiv_{UW}$ of the summary through $\equiv_T$ of G:

$$G_{TW} = (G_{/\equiv_T})_{/\equiv_{UW}}$$

This double-quotient summarization acts as follows. First, nodes are grouped by their *sets of types* (inner quotient through $\equiv_T$); second, *untyped nodes only* are grouped according to weak (structural) equivalence.

For instance, our sample G has six typed data nodes (*Alice* to *David*, *BigDataMaster* and *HadoopCourse*), each of which has a set of exactly one type; all these types are different. Thus, $\equiv_T$ is empty, and G_{TW} (drawing omitted) has eight typed

nodes UTW_1 to UTW_8, each with a distinct type and the property(ies) of one of these nodes. We now consider G's eight untyped data nodes. We have d1 $\equiv_W$ d2 due to their common target clique $\{desc\}$, and similarly w1 $\equiv_W$ w2 and h2 $\equiv_W$ h3 $\equiv_W$ h4. Thus, G_{TW} has four untyped nodes, each of which is an object of *desc*, *email*, *webpage* and respectively *officeHours* triples.

The typed weak summary, as well as other type-first summaries, e.g. [17], also have limitations:

- They are defined based on the type triples of G, which may change through saturation, leading to different G_{TW} summaries for *conceptually the same graph* (as all G leading to the same G^∞ are equivalent). Thus, for a type-first summary to be most meaningful, one should build it on the saturated graph G^∞. Note the reason for saturation at Figure 8 in [18].
- They do not exploit the relationships which the ontology may state among the types. For instance, *AssistantProfessor* nodes like Carole are summarized separately from *Professors* like David, although they are all instructors.

III. Summarization aware of type hierarchies

A. Novel type-based RDF equivalence

Our first goal is to *define an RDF equivalence relation* which:

1) takes type information into account, thus belongs to the "types-first" approach;
2) leads (through Definition 2) to a summary which represents together, to the extent possible (see below), nodes that have *the same most general type*.

Formally, let $\mathcal{C} = \{c_1, c_2, \ldots, \}$ be the set of class nodes present in G (that is, in SG and/or in TG). We can view these nodes as organized in a *directed graph* where there is an edge $c_1 \rightarrow c_2$ as long as G's saturated schema SG^∞ states that c_1 is a subclass of c_2. By a slight abuse of notation, we use $\mathcal{C}$ to also refer to this graph[2]. In principle, $\mathcal{C}$ could have cycles, but this does not appear to correspond to meaningful schema designs. Therefore, we assume without loss of generality that $\mathcal{C}$ is a directed acyclic graph (DAG, in short)[3]. In Figure 1, $\mathcal{C}$ is the DAG comprising the eight (blue) class nodes and edges between them; this DAG has four roots.

First, assume that $\mathcal{C}$ is a tree, e.g., with *Instructor* as a root type and *PhDStudent*, *AssistantProfessor* as its subclasses. In such a case, we would like instances of all the abovementioned types to be represented together, because they are all instances of the top type *Instructor*. This extends easily to the case when $\mathcal{C}$ is a forest, e.g., a second type hierarchy in $\mathcal{C}$ could feature a root type *Paper* whose subclasses are *ConferencePaper*, *JournalPaper* etc. In this case, we aim to represent all authors together because they are instances of *Paper*.

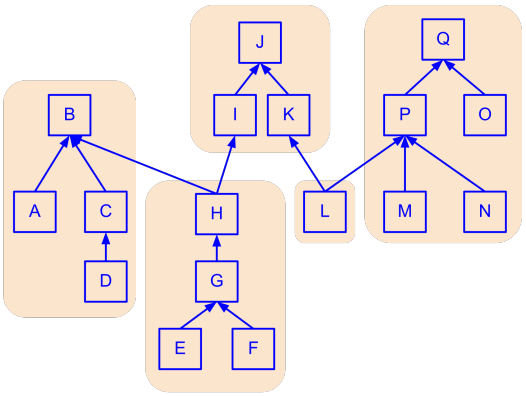

Figure 3: Sample RDF schema and min-size cover of the corresponding $\mathcal{C}$.

In general, though, $\mathcal{C}$ may not be a forest, but instead it may be a graph where some classes have multiple superclasses, potentially unrelated. For instance, in Figure 1, *PhDStudent* has two superclasses, *Student* and *Instructor*. Therefore, it is not possible to represent G nodes of type *PhDStudent* based on their most general type, because they have more than one such type. Representing them twice (once as *Instructor*, once as *Student*) would violate the framework (Definition 2), in which any summary is a quotient and thus, each G node must be represented by exactly one summary node.

To represent resources as much as possible according to their most general type, we proceed as follows.

Definition 7: (Tree cover) Given a DAG $\mathcal{C}$, we call a *tree cover* of $\mathcal{C}$ a set of trees such that: (i) each node in $\mathcal{C}$ appears in exactly one tree; (ii) together, they contain all the nodes of $\mathcal{C}$; and (iii) each $\mathcal{C}$ edge appears either in one tree or connects the root of one tree to a node in another.

Given $\mathcal{C}$ admits many tree covers, however, it can be shown that there exists a tree cover with the least possible number of trees, which we will call **min-size cover**. This cover can be computed in a single traversal of the graph by *creating a tree root exactly from each $\mathcal{C}$ node having two supertypes such that none is a supertype of the other*, and attaching to it all its descendants which are not themselves roots of another tree. For instance, the RDF schema from Figure 1 leads to a min-size cover of five trees:

- A tree rooted at *Instructor* and the edges connecting it to its children *AssistantProfessor* and *Professor*;
- A single-node tree rooted at *PhDStudent*;
- A tree rooted at *Student* with its child *MasterStudent*;
- A single-node tree for *MasterProgram* and another for *MasterCourse*.

Figure 3 illustrates min-size covers on a more complex RDF schema, consisting of the types A to Q. Every arrow goes from a type to one of its supertypes (for readability, the figure does not include all the implicit subclass relationships, e.g., that E is also a subclass of H, I, J etc.). The pink areas each denote

[2]Ontology languages such as RDF Schema or OWL feature a top type, that is a supertype of any other type, such as rdfs:Resource. We do not include such a generic, top type in $\mathcal{C}$.

[3]If $\mathcal{C}$ has cycles, the types in each cycle can all be seen as equivalent, as each is a specialization of all the other, and could be replaced by a single (new) type in a simplified ontology. The process can be repeated until $\mathcal{C}$ becomes a DAG, then the approach below can be applied, following which the simplified types can be restored, replacing the ones we introduced. We omit the details.

a tree in the corresponding min-size cover. H and L are tree roots because they have multiple, unrelated supertypes.

To complete our proposal, we need to make an extra hypothesis on G:

(†) Whenever a data node n is of two distinct types c_1, c_2 which are *not in the same tree in the min-size tree cover* of $\mathcal{C}$, then (i) c_1 and c_2 have some common subclasses, (ii) among these, there exists a class $c_{1,2}$ that is a superclass of all the others, and (iii) n is of type $c_{1,2}$.

For instance, in our example, hypothesis (†) states that if a node n is an *Instructor* and a *Student*, these two types must have a common subclass (in our case, this is *PhDStudent*), and n must be of type *PhDStudent*. The hypothesis would be violated if there was another common subclass of *Instructor* and *Student*, say *MusicLover*[4], that was neither a subclass of *PhDStudent* nor a superclass of it.

(†) may be checked by a SPARQL query on G. While it may not hold, we have not found such counter-examples in a set of RDF graphs we have examined (see Section IV). In particular, (†) immediately holds in the frequent case when $\mathcal{C}$ is a tree (taxonomy) or, more generally, a forest: in such cases, the min-size cover of $\mathcal{C}$ is exactly its set of trees, and any types c_1, c_2 of a data node n are in the same tree.

When (†) holds, we can state:

Lemma 1 (Lowest branching type): Let G be an RDF graph satisfying (†), n be a data node in G, cs_n be the set of types of n in G, and cs_n^∞ be the classes from cs_n together with all their superclasses (according to the saturated schema of G). Assume that $cs_n^\infty \neq \emptyset$.

Then there exists a type lbt_n, called *lowest branching type*, such that:

- $cs_n^\infty = cs'_n \uplus cs''_n$, where $\{lbt_n\} \in cs'_n$ and cs''_n may be empty;
- the types in cs'_n (if any) can be arranged in a tree according to $\prec_{sc}$ relation between them, and the most general one is lbt_n;
- if cs''_n is not empty, it is at least of size two, and all its types are superclasses of lbt_n.

Proof: Let's assume to the contrary that there exists an RDF graph $\mathtt{G}_1$ satisfying (†), a node n in $\mathtt{G}_1$, cs_n the set of types of n, $cs_n^\infty \neq \emptyset$ is the set of types of n with all their supertypes (according to saturated schema of $\mathtt{G}_1$) and there is no *lowest branching type* for n.

Let $\mathcal{G}$ be the set of all such RDF graphs and let G be the $\mathcal{G}$ graph containing a node n that violates the lemma and such that $|cs_n^\infty|$ is the smallest, across all such lemma-violating nodes n in any graph from $\mathcal{G}$.

Let $k = |cs_n^\infty|$. Note that $k > 0$ by definition. Let's consider the cases:

1) $k = 1$ In this case, the lemma trivially holds.
2) $k \geq 2$ In this case, let $t_1, \ldots, t_k$ be the types of node n (their order not important). Let's consider graph $\mathtt{G}'$ which is the same as G but without node n having type t_k. From the way we chose G and $\mathtt{G}'$, $\mathtt{G}'$ satisfies the lemma, thus there exists a *lowest branching type* lbt_n for n in $\mathtt{G}'$. Now, let's add t_k to the types of n in $\mathtt{G}'$. There are 3 possibilities:
   a) t_k is a subclass of lbt_n. Then lbt_n is also *lowest branching type* after this addition.
   b) t_k is a superclass of lbt_n. If it's the only superclass of lbt_n then t_k is a new *lowest branching type*, else n still admits the lowest branching type lbt_n.
   c) t_k is neither a sub- nor a superclass of lbt_n. Then it is in another tree in min-size cover of G, thus by (†) it follows that t_k and some other type between $t_1, \ldots, t_{k-1}$ have a common subtype which serves as a lowest branching type for n.

From the above discussion we conclude that the node n for which $k = |cs_n^\infty|$ is not the lemma counterexample with the smallest k, which contradicts the assumption we made when picking it! Therefore no graph exists in $\mathcal{G}$, thus all Gs satisfy the lemma. □

For instance, let n be Bob in Figure 1, then cs_n is {*PhDStudent*}, thus cs_n^∞ is {*PhDStudent, Student, Instructor*}. In this case, lbt_n is *PhDStudent*, cs'_n is {*PhDStudent*} and cs''_n is {*Student, Instructor*}.

If we take n to be Carole, cs_n^∞ is {*AssistantProfessor, Instructor*}; no type from this set has two distinct superclasses, thus cs''_n must be empty, lbt_{Carole} is *Instructor*, and cs'_n is {*AssistantProfessor*, *Instructor*}. By a similar reasoning, lbt_{David} is *Instructor*, and lbt_{Alice} is *Student*. When n has a type without subclasses or superclasses, such as BigData-Master, it leads to cs''_n being empty, and cs'_n is lbt_n, the only type of n. Thus, $lbt_{\text{BigDataMaster}}$ is *MasterProgram* and $lbt_{\text{HadoopCourse}}$ is *MasterCourse*.

For a more complex example, recall the RDF schema in Figure 3, and let n be a node of type E in an RDF graph having this schema. In this case, cs_n is $\{E, G, H, B, I, J\}$, lbt_n is H, cs'_n is $\{E, G, H\}$ while cs''_n is $\{B, I, J\}$. Based on Lemma 1, we define our novel notion of equivalence, reflecting the hierarchy among the types of G data nodes:

Definition 8: (TYPE-HIERARCHY EQUIVALENCE) *Type-hierarchy equivalence*, denoted $\equiv_{\mathtt{TH}}$, is an RDF node equivalence relation defined as follows: two data nodes n_1 and n_2 are type-hierarchy equivalent, noted $n_1 \equiv_{\mathtt{TH}} n_2$, iff $lbt_{n_1} = lbt_{n_2}$.

From the above discussion, it follows that Carole $\equiv_{\mathtt{TH}}$ David, matching the intuition that they are both instructors and do not belong to other type hierarchies. In contrast, PhD students (such as Bob) are only type-hierarchy equivalent to each other; they are set apart by their dual *Student* and *Instructor* status. Master students such as Alice are only type-hierarchy equivalent among themselves, as they only belong to the student type hierarchy. Every other typed node of G is only type-hierarchy equivalent to itself.

[4]*MusicLover* may be a subclass of yet another class (distinct type c_3 in third other *min-size tree*) and it would still violate the hypothesis

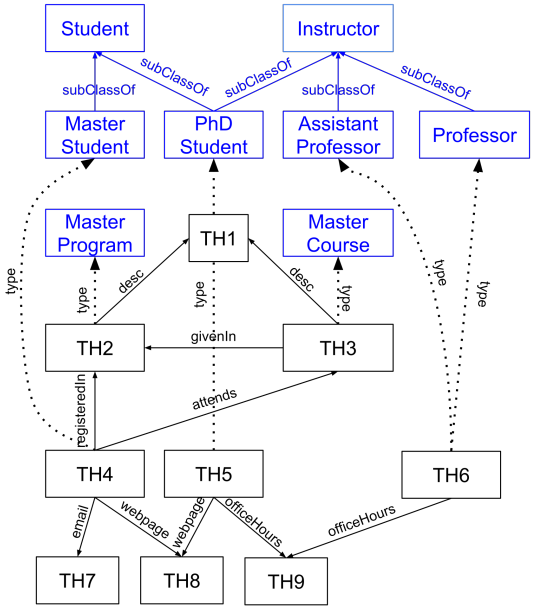

Figure 4: Weak type-hierarchy summary of the RDF graph in Figure 1. The roots of the trees in the min-size cover of $\mathcal{C}$ are underlined.

B. RDF summary based on type hierarchy equivalence

Based on $\equiv_{\mathtt{TH}}$ defined above, and the $\equiv_{\mathtt{UW}}$ structural equivalence relation (two nodes are $\equiv_{\mathtt{UW}}$ if they have no types, and are weakly equivalent), we introduce a novel summary belonging to the "type-first" approach:

Definition 9: (WEAK TYPE-HIERARCHY SUMMARY) The type hierarchy summary of $\mathtt{G}$, denoted $\mathtt{G}_{\mathtt{WTH}}$, is the summary through $\equiv_{\mathtt{UW}}$ of the summary through $\equiv_{\mathtt{TH}}$ of $\mathtt{G}$:

$$\mathtt{G}_{\mathtt{WTH}} = \left(\mathtt{G}_{/\equiv_{\mathtt{TH}}}\right)_{/\equiv_{\mathtt{UW}}}$$

Figure 4 illustrates the $\mathtt{G}_{\mathtt{WTH}}$ summary of the RDF graph in Figure 1. Different from the weak summary (Figure 2), it does not represent together nodes of unrelated types, such as BigDataMaster and HadoopCourse. At the same time, different from the typed weak summary of the same graph, it does not represent separately each individual, and instead it keeps Carole and David together as they only belong to the instructor type hierarchy.

More summaries based on $\equiv_{\mathtt{TH}}$ could be obtained by replacing $\mathtt{UW}$ with another RDF equivalence relation.

IV. ALGORITHM AND APPLICATIONS

A. Constructing the weak type-hierarchy summary

An algorithm which builds $\mathtt{G}_{\mathtt{WTH}}$ is as follows:

1) From $\mathtt{SG}$, build $\mathcal{C}$ and its min-size cover.
2) For every typed node n of $\mathtt{G}$, identify its lowest branching type lbt_n and (the first time a given lbt_n is encountered) create a new URI URI_{lbt_n}: this will be the $\mathtt{G}_{\mathtt{WTH}}$ node representing all the typed $\mathtt{G}$ nodes having the same lbt_n.
3) Build the weak summary of the untyped nodes of $\mathtt{G}$, using the algorithm described in [18]. This creates the untyped nodes in $\mathtt{G}_{\mathtt{WTH}}$ and all the triples connecting them.
4) *Add type edges:* for every triple $n\ \tau\ c$ in $\mathtt{G}$, add (unless already in the summary) the triple $URI_{lbt_n}\ \tau\ c$ to $\mathtt{G}_{\mathtt{WTH}}$.
5) *Connect the typed and untyped summary nodes:* for every triple $n_1\ p\ n_2$ in $\mathtt{G}$ such that n_1 has types in $\mathtt{G}$ and n_2 does not, add (unless already in the summary) the triple $URI_{lbt_{n_1}}\ p\ UW_{n_2}$ to $\mathtt{G}_{\mathtt{WTH}}$, where UW_{n_2} is the node representing n_2, in the weak summary of the untyped part of $\mathtt{G}$. Apply a similar procedure for the converse case (when n_1 has no types but n_2 does).

Step 1) is the fastest as it applies on the schema, typically orders of magnitude smaller than the data. The cost of the steps 2)-4) depend on the distribution of nodes (typed or untyped) and triples (type triples; data triples between typed/untyped nodes) in $\mathtt{G}$. [18] presents an efficient, almost-linear time (in the size of $\mathtt{G}$) weak summarization algorithm (step 3). The complexity of the other steps is linear in the number of triples in $\mathtt{G}$, leading to an overall almost-linear complexity.

B. Applicability

To understand if $\mathtt{G}_{\mathtt{WTH}}$ summarization is helpful for an RDF graph, the following questions should be answered:

1) Does $\mathtt{SG}$ feature subclass hierarchies? If it does not, then $\mathtt{G}_{\mathtt{WTH}}$ reduces to the weak summary $\mathtt{G}_{\mathtt{TW}}$.
2) Does $\mathtt{SG}$ feature a class with two unrelated superclasses?
   a) No: then $\mathcal{C}$ is a tree or a forest. In this case, $\mathtt{G}_{\mathtt{WTH}}$ represents every typed node together with all the nodes whose type belong to the same type hierarchy (tree).
   b) Yes: then, does $\mathtt{G}$ satisfy (†)?
      i) Yes: one can build $\mathtt{G}_{\mathtt{WTH}}$ to obtain a refined representation of nodes according to the lowest branching type in their type hierarchy.
      ii) No: $\mathtt{G}_{\mathtt{WTH}}$ is undefined, due to the lack of a unique representative for the node(s) violating (†).

Among the RDF datasets frequently used, DBLP[5], the BSBM benchmark [21], and the real-life Slegger ontology[6] whose description has been recently published [22] exhibited subclass hierarchies. Further, BSBM graphs and the Slegger ontology feature multiple inheritance. BSBM graphs satisfy (†). On Slegger we were unable to check this, as the data is not publicly shared; our understanding of the application though as described implies that (†) holds.

An older study [23] of many concrete RDF Schemas notes a high frequence of class hierarchies, of depth going up to 12, as well as a relatively high incidence of multiple inheritance; graphs with such schema benefit from $\mathtt{G}_{\mathtt{WTH}}$ summarization when our hypothesis (†) holds.

REFERENCES

[1] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, "Object exchange across heterogeneous information sources," in *ICDE*, 1995.
[2] R. Goldman and J. Widom, "Dataguides: Enabling query formulation and optimization in semistructured databases," in *VLDB*, 1997. [Online]. Available: http://www.vldb.org/conf/1997/P436.PDF

[5]http://dblp.uni-trier.de/
[6]http://slegger.gitlab.io/

[3] S. Nestorov, J. D. Ullman, J. L. Wiener, and S. S. Chawathe, "Representative objects: Concise representations of semistructured, hierarchical data," in *ICDE*, 1997.

[4] T. Milo and D. Suciu, "Index structures for path expressions," in *ICDT*, 1999. [Online]. Available: http://dx.doi.org/10.1007/3-540-49257-7_18

[5] Q. Chen, A. Lim, and K. W. Ong, "$D(K)$-index: An adaptive structural summary for graph-structured data," in *SIGMOD*, 2003.

[6] R. Kaushik, P. Bohannon, J. F. Naughton, and H. F. Korth, "Covering indexes for branching path queries," in *SIGMOD*, 2002.

[7] Q. Li and B. Moon, "Indexing and querying XML data for regular path expressions," in *VLDB*, 2001.

[8] O. Udrea, A. Pugliese, and V. S. Subrahmanian, "GRIN: A graph based RDF index," in *AAAI*, 2007.

[9] S. Gurajada, S. Seufert, I. Miliaraki, and M. Theobald, "Using graph summarization for join-ahead pruning in a distributed RDF engine," in *SWIM*, 2014.

[10] Š. Čebirić, F. Goasdoué, and I. Manolescu, "Query-oriented summarization of RDF graphs (demonstration)," *PVLDB*, vol. 8, no. 12, 2015. [Online]. Available: http://www.vldb.org/pvldb/vol8/p2012-cebiric.pdf

[11] ——, "Query-oriented summarization of RDF graphs," in *BICOD*, 2015.

[12] ——, "A framework for efficient representative summarization of RDF graphs," in *International Semantic Web Conference (ISWC)*, 2017.

[13] M. Palmonari, A. Rula, R. Porrini, A. Maurino, B. Spahiu, and V. Ferme, "ABSTAT: linked data summaries with abstraction and statistics," in *ESWC Workshops*, 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-25639-9_25

[14] A. Khan, S. S. Bhowmick, and F. Bonchi, "Summarizing static and dynamic big graphs," *PVLDB*, vol. 10, no. 12, pp. 1981–1984, 2017.

[15] A. Sadri, F. D. Salim, Y. Ren, M. Zameni, J. Chan, and T. Sellis, "Shrink: Distance preserving graph compression," *Inf. Syst.*, vol. 69, 2017.

[16] G. Troullinou, H. Kondylakis, E. Daskalaki, and D. Plexousakis, "Ontology understanding without tears: The summarization approach," *SWJ*, 2017.

[17] S. Campinas, R. Delbru, and G. Tummarello, "Efficiency and precision trade-offs in graph summary algorithms," in *IDEAS*, 2013.

[18] Š. Čebirić, F. Goasdoué, and I. Manolescu, "Query-Oriented Summarization of RDF Graphs," in *BDA (Bases de Données Avancées)*, 2016. [Online]. Available: https://hal.inria.fr/hal-01363625

[19] W3C, "Resource description framework," http://www.w3.org/RDF/.

[20] S. Khatchadourian and M. P. Consens, "Constructing bisimulation summaries on a multi-core graph processing framework," in *GRADES*, 2015. [Online]. Available: http://doi.acm.org/10.1145/2764947.2764955

[21] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 2, pp. 1–24, 2009.

[22] D. Hovland, R. Kontchakov, M. G. Skjæveland, A. Waaler, and M. Zakharyaschev, "Ontology-based data access to slegge," in *ISWC*, 2017.

[23] A. Magkanaraki, S. Alexaki, V. Christophides, and D. Plexousakis, "Benchmarking RDF schemas for the semantic web," in *ISWC*, 2002.