RDF Graph Summarization for First-sight Structure Discovery

François Goasdoué, Pawel Guzewicz, Ioana Manolescu

RDF GRAPH SUMMARIZATION FOR FIRST-SIGHT STRUCTURE DISCOVERY

FRANÇOIS GOASDOUÉ, PAWEŁ GUZEWICZ, AND IOANA MANOLESCU

ABSTRACT. To help users get familiar with large RDF graphs, RDF summarization techniques can be used. In this work, we study quotient summaries of RDF graphs, that is: graph summaries derived from a notion of equivalence among RDF graph nodes. We make the following contributions: (*i*) four novel summaries which are often small and easy-to-comprehend, in the style of E-R diagrams; (*ii*) efficient (amortized linear-time) algorithms for computing these summaries either from scratch, or incrementally, reflecting additions to the graph; (*iii*) the first formal study of the interplay between RDF graph saturation in the presence of an RDFS ontology, and summarization; we provide a sufficient condition for a highly efficient shortcut method to build the quotient summary of a graph without saturating it; (*iv*) formal results establishing the shortcut conditions for some of our summaries and others from the literature; (*v*) experimental validations of our claim within a tool available online.

1. INTRODUCTION

The Resource Description Framework (RDF) is the data model recommended by the W3C for data publishing and sharing data. An RDF graph typically consists of *data triples*, stating that a *subject* has a *property* with a certain value, called *object*. An RDF graph may also contain *type triples*, stating that some resource has the W3C standard *rdf:type* property (*type* for short) whose value is a certain type (or, equivalently, class). Finally, an RDF graph may also contain an *ontology*, describing relationships that hold between the properties and classes present in the graph. For instance, u_1 *hasName "Julie"*, u_1 *worksFor "ACME"* describe a resource whose identifier (URI) is u_1, while u_1 *type Person* assigns it a type; we use quote-enclosed strings to denote constants (also called *literals*). An ontology associated to this sample dataset may state e.g., that *worksFor rdfs:domain Employee*: this means "anyone who works for something is of type *Employee*". As a consequence, u_1 *type Employee* also holds in the above graph, although it is not explicitly written there. Such a triple is called *implicit* (or *inferred*, or *entailed*).

RDF graphs enable describing large and heterogeneous data sets, which may be hard to understand by human users, and to analyze by machines. To help address this difficulty, many RDF *summarization* techniques have been proposed in the literature [7], some of which draw upon graph summarization techniques [29] proposed independently of RDF. As stated in [7], RDF summarization techniques fall into four classes: (*i*) structural methods are built considering first and foremost the graph structure, respectively the paths and subgraphs present in the graph; (*ii*) pattern mining methods apply mining techniques to discover patterns in the data and use the patterns as a summary (synthesis) of the graph; (*iii*) statistical methods aim at extracting from the graph a set of quantitative measures or statistics; finally (*iv*) hybrid methods combine elements from more than one of the previous classes. For what concerns summary applications, these range from (RDF) graph indexing, query cardinality estimation, to helping users formulate graph queries, graph visualization and exploration.

A large and useful class of structural graph summaries are based on defining an *equivalence* relation among graph nodes, and creating one summary node for each equivalence class (set of nodes equivalent to each other in the original graph). Then, for every edge labeled p which goes from s to o in the graph, the summary has an edge labeled p from the summary node corresponding to the equivalence class of s, to the node corresponding to the equivalence class of p. Such summaries, also called *quotient summaries*, have many good properties, mainly due to the existence of a graph homomorphism from the original graph into its summary. Quotient summaries proposed for general graphs include [21, 30, 22, 23, 10, 11].

Summarizing RDF graphs raises two new questions w.r.t. to prior (non-RDF) graph summarization setting: (*i*) how to take into account the types that may be attached to the nodes (knowing that a node may have no type, or one type, or several)? On one hand, types bring an opportunity to define node equivalence, since, intuitively, two nodes having the same type(s) are likely similar in sole way. On the other hand, they cannot be solely relied upon, because many RDF graphs lack types for many (or all) of their nodes; prior RDF summary quotients [37, 35, 5] answer this question in different ways; (*ii*) how should a summary reflect the implicit triples that may hold in the graph due

Date: Accepted on April 1, 2020 for publication in VLDB Journal.

to the presence of an ontology? Our prior poster paper [8] is the only work to have addressed this so far.

In this work, we make several theoretical and practical contributions to the area of quotient RDF graph summarization. Specifically:

(1) We formalize an *RDF quotient summarization framework*, taking into account RDF-specific concepts such as ontologies.
(2) We introduce *two novel equivalence relations* between RDF nodes, which rely on the transitive co-occurrence of properties on graph nodes. Based on them we define *two novel summaries called Weak and Strong* respectively, as well as two versions thereof which give priorities to types (for those nodes that have type information); we call these summaries *Typed Weak and Typed Strong*, respectively. The interest of these new equivalence relations is that they lead to summaries that are much more compact (fewer nodes and edges) than quotient summaries previously studied in the literature [21, 30, 22, 23, 10, 11, 37, 35, 5]. This compactness comes at the price of some loss of accuracy. Nevertheless, they do preserve a significant amount of information from the input graph. In particular, for domain-specific graphs, describing applications from a specific area, our summaries are very convenient data discovery tools: a simple summary visualization helps learn a lot about the graph structure. This is why *our work's main target are domain-specific graphs*. For encyclopedic graphs, such as DBPedia or YAGO, our quotients are very likely to be more compact than some studied in prior work, but still too large for human comprehension; non-quotient summaries, e.g., based on pattern mining [7], are more appropriate for such graphs.
(3) We are the first to show that for a large set of RDF equivalence relations, one can build the quotient summary of an RDF graph *including its implicit triples, without materializing them*. Based on our framework, we provide a novel sufficient condition for an RDF equivalence relation, which enables building this through our so-called *shortcut* method; its advantage is to reduce very significantly the summarization time. We prove that *our Weak and Strong summaries satisfy this condition, whereas the Typed Weak and Typed Strong ones do not*; we provide a set of similar results also on previously studied equivalence relations.
(4) Our fourth contribution is a set of *novel algorithms for computing our summaries*, including *incremental* ones which are able to reflect the addition of triples to the graph, without re-traversing the rest of the graph. All our algorithms have *amortized linear complexity in the size of the graph.*
(5) We have implemented these algorithms and summary visualizations in a system called RDFQuotient, available online in open source[1].

Sample summary visualization Below, we show an example where our summarization techniques compress an RDF graph structure by many orders of magnitude, while still supporting an informative visualization.

Figure 1 shows the summary of a WatDiv [2] benchmarkgraph of approximately 11 millions of triples. This visualization reflects the complete structure of the graph, using only 8 nodes and 24 edges, comparable to a simple Entity-Relationship diagram. This summary reads as follows. (*i*) Non-leaf graph nodes belong to one of the eight disjoint entities, each represented by a summary node (box in Figure 1) labeled $N1$ to $N8$. The number of graph nodes in each entity appears in parenthesis after the label Ni of their representative, e.g., 25000 for $N8$. (*ii*) Each entity represents either graph nodes with types or without types. In the first case, the most general types of the graph nodes, according to which they have been grouped in the entity, are given in bold underneath the entity label; these coincide with the original graph node types when an ontology is not present. For instance, all the 25000 graph nodes represented by $N8$ are (implicitly) of type ProductCategory due to the ontology at hand; their distribution according to their original typing is also shown, e.g., 807 are of type ProductCategory0. An entity that does not show types represents untyped graph nodes which have been grouped according to the relationships they have with others, using a novel *transitive* relation of co-occurrence of their properties, which we introduce in this paper. For instance, $N4$ represents both the web pages of the products of $N8$ and these of the persons of $N7$, because these web pages can be home pages for both: there are *homepage* edges from both $N8$ to $N4$ and from $N7$ to $N4$. (*iii*) Graph nodes from an entity may have outgoing properties whose values are *leaf* nodes in the graph; the set of all such properties appears in the corresponding summary node box, one property per line. For each property, e.g., nationality for $N7$, the summary node specifies how many graph nodes represented by this entity have it (19924 in this case), and how many distinct leaf nodes are target of these edges (25 in this case). (*iv*) Graph

[1] `https://rdfquotient.inria.fr`

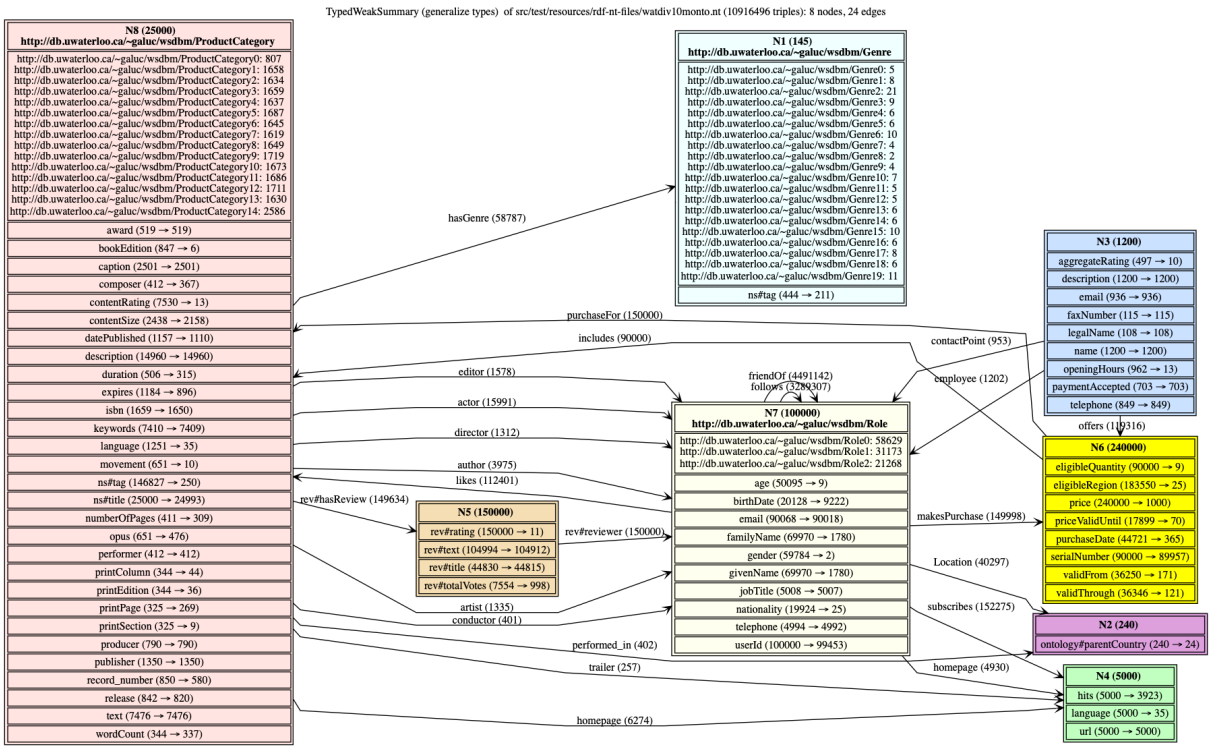


FIGURE 1. ER-style visualization built from one of our summaries.

nodes from an entity may have outgoing properties whose values are *non-leaf* nodes in the graph. For each graph edge $n_1 \xrightarrow{a} n_2$, where n_1, n_2 are non-leaf graph nodes and a is the property (edge label), an a-labeled edge in the summary goes from the representative of n_1 to that of n_2. Next to a, that summary edge is also labeled with the number of graph edges to which it corresponds. (v) Properties from a small, fixed vocabulary are considered *metadata* (as opposed to *data*) and therefore are not used to split graph nodes in entities, e.g., *rdf-schema#comment* and *rdf-schema#label* in Figure 1. More such visualization summaries can be found online[1]; an example leading from an RDF graph to its summary and then such a visualization is worked out in the paper. Most of the material presented here is new. The exceptions are: Theorem 2 appeared without proof in the poster [8]; our algorithms were outlined in the demonstration [15].

Below, Section 2 recalls useful preliminaries, then Section 3 introduces the novel notions of property cliques, based on which we define our summaries. Section 4, respectively, 5 extend this to graphs comprising type triples, respectively, ontologies. In Section 6 we discuss summary visualization. Section 7 describes our summarization algorithms and Section 8 presents our experiments. We then survey related work and conclude. Proofs of the technical results of this paper can be found in the Appendix.

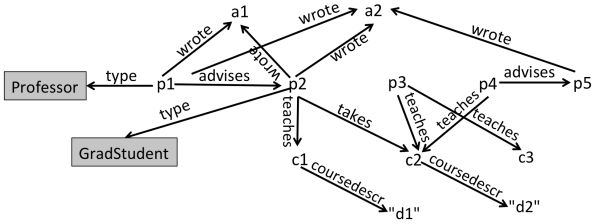


FIGURE 2. Sample RDF graph.

2. Preliminaries

We recall here the starting points of our work: RDF graphs (Section 2.1) and graph quotients (Section 2.2).

2.1. **Data graphs.** An *RDF graph* is 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 [38], using uniform resource identifiers (URIs), typed or untyped literals (constants) and blank nodes (unknown URIs or literals). Blank nodes are essential features of RDF allowing to support *unknown URI/literal tokens*. These are conceptually similar to the labeled nulls or variables used in incomplete relational databases [1], as shown in [16].

As our running example, Figure 2 shows a sample RDF graph G describing a university department, including professors, graduate students, articles they wrote, and courses they teach and/or take. Here and in the sequel, nodes shown in gray are classes (or types). Further, nodes whose labels appear enclosed in quotes, e.g., "d1", are literals, while the others are URIs.

RDFS ontology. One can enhance the resource descriptions comprised in an RDF graph by declaring *ontological constraints* between the classes and the properties they use. For instance, one may add to the graph in Figure 2 the ontology O consisting of the triples

GradStud rdfs:subClassOf Instructor *takes rdfs:range Course*
Professor rdfs:subClassOf Instructor *takes rdfs:domain Student*
advises rdfs:subPropertyOf knows

to state that graduate students, respectively professors, are instructors, that anyone who takes a course is a Student, what can be taken is a course, and that advising someone entails knowing him or her.

From ontological constraints and explicit triples, *implicit* triples may be derived. For instance, from G and O, it follows that p_1 is of type *Instructor*, which is modeled by the implicit triple p_1 *type Instructor*; we say this triple *holds in* G, even though it is not explicitly part of it. Other implicit triples obtained from this ontology based on G are: p_2 *type Instructor*, p_2 *type Student*, and p_1 *knows* p_2. Given a graph G (which may include a set of ontological constraints, denoted O), the *saturation* of G, is obtained by adding to G: (i) all the implicit triples derived from G and O, then (ii) all the implicit triples derived from those of step (i) and O, and so on, until a fixpoint, denoted $\mathtt{G}^\infty$, is reached. These triples are added based on *RDF entailment rules* from the RDF standard. In this work, we consider the widely-used *RDFS entailment rules*. They exploit the simple RDFS ontology language, based on the four standard properties illustrated above, which we denote **subClass**, **subProperty**, **domain** and **range**, to add new ontological constraints or facts. The saturation $\mathtt{G}^\infty$ of any graph G comprising RDFS ontological constraints is finite, unique, and can be computed in polynomial time, e.g., by leveraging a database management system [16]. Crucially, $\mathtt{G}^\infty$ materializes the semantics of G.

Terminology and notations. We call the triples from G whose property is (rdf:)type *type triples*, those whose property is among the standard four RDFS ones *schema triples*, and we call all the other *data triples*. We say a node is *typed* in G if the node is the subject of at least one type triple in G. In the graph shown in Figure 2, *p1 type Professor* is a type triple, hence the node $p1$ is typed, *p1 advises p2* is a data triple, while *Professor rdfs:subClassOf Instructor* may be a schema triple. Further, we say a URI from G is a *class node* if (i) it appears as subject or object of a subClass triple, or an object of range or domain triple; or (ii) it appears as the object of a type triple; or (iii) it appears as a subject of a type triple with object rdfs:Class. We call *property node*, a URI appearing (i) as subject or object in subProperty triple, or as a subject of domain or range triple; or (ii) as a subject of rdf:type triple with object rdf:Property. Together, the class and property nodes are the *schema nodes*; all non-schema nodes are *data nodes*. In Figure 2, Professor and GradStudent are class nodes. If we consider the aforementioned ontology O, takes, advises and knows are property nodes. Finally, the a's, p's, c's and d's nodes are data nodes.

It is important to stress that not all nodes in an RDF graph are typed, e.g., this is only true for p_1 and p_2 in Figure 2. Further, some nodes may have several types, in particular due to saturation (e.g., p_2 is of types GradStudent and Instructor) but not only, e.g., p_1 could also be of type ForeignStudent etc.

2.2. **Quotient RDF summaries.** We recall here quotient RDF summaries as defined in prior work, outline existing work in this area, and discuss their limitations.

Given an RDF graph G and an equivalence relation[2] $\equiv$ over the nodes of G, the quotient of G by $\equiv$, denoted $\mathtt{G}_{/\equiv}$, is the graph having (i) a node for each equivalence class of $\equiv$ (thus, for each set of equivalent G nodes); and (ii) for each edge $n_1 \xrightarrow{a} n_2$ in G, an edge $m_1 \xrightarrow{a} m_2$, where m_1, m_2 are the quotient nodes corresponding to the equivalence classes of n_1, n_2 respectively. Quotients have several desirable properties from a summarization perspective:

Size guarantees:: By definition, $\mathtt{G}_{/\equiv}$ is guaranteed to have at most as many nodes and edges as G. Some non-quotient summaries, e.g., Dataguides [17], cannot guarantee this.

Property completeness:: Every property (edge label) from G is present on some summary edges. This gives first-time users of the dataset a chance to decide, based on their interest and envisioned application, if it is worth further investigation. In some applications, e.g., when data journalists explore open data, or physicians look for a rare diagnosis, it is important not to miss a "weak signal", encoded in RDF as a set of triples using infrequent properties.

Structural homomorphism:: It is easy to see that the function f associating to any

[2]An equivalence relation $\equiv$ is a binary relation that is reflexive, i.e., $x \equiv x$, symetric, i.e., $x \equiv y \Rightarrow y \equiv x$, and transitive, i.e., $x \equiv y$ and $y \equiv z$ implies $x \equiv z$ for any x, y, z.

G node, its representative in $G_{/\equiv}$ is a homomorphism from G into its summary: any subgraph of G is "projected" by f into a subgraph of its quotient.

Quotient graph summaries include e.g., [30, 22, 10, 36, 25, 11, 13]; RDF quotient summaries are described in [37, 35, 5].

Bisimilarity is behind most equivalence relations used in these works. Two nodes n_1, n_2 are *forward bisimilar* [21] (denoted $\equiv_{\mathtt{fw}}$) iff (i) for every G edge $n_1 \xrightarrow{a} m_1$, G also comprises an edge $n_2 \xrightarrow{a} m_2$, such that m_1 and m_2 are also (forward) bisimilar and (ii) a similar statement holds, replacing n_1, m_1 with n_2, m_2 and vice-versa. While forward bisimilarity focuses on *outgoing* edges only, two nodes can also be bisimilar w.r.t. their *incoming* edges, i.e., *backward bisimilar* (denoted $\equiv_{\mathtt{bw}}$). Backward similarity of two nodes n_1, n_2 is recursively defined similarly as above when considering G edges $m_1 \xrightarrow{a} n_1$ and $m_2 \xrightarrow{a} n_2$. Finally, two nodes can be bisimilar based on both their incoming and outgoing edges, i.e., *forward and backward bisimilar* (denoted $\equiv_{\mathtt{fb}}$), when they are both *forward bisimilar* and *backward bisimilar*. We denote the bisimulation based summaries $G_{/\mathtt{fw}}$ (forward), $G_{/\mathtt{bw}}$ (backward) and $G_{/\mathtt{fb}}$ (forward and backward), respectively. They have been studied, in particular for indexing and query processing, in [37, 5, 35].

Leaf (resp. root) collapse An issue encountered when summarizing only according to the properties outgoing a node, e.g., $\equiv_{\mathtt{fw}}$, is that the summary considers equivalent (thus, collapses) all nodes lacking outgoing edges, that is, all the leaves of G. Such leaf nodes may have very little to do with each other. For instance, on the WatDiv graph summarized in Figure 1, all the values of *award*, *numberOfPages*, *keywords*, *email*, *faxNumber* would be summarized together, even though they are very different. We call this situation *leaf collapse through summarization*. Symetrically, a summary whose relation only depends on nodes' incoming properties, e.g., $\equiv_{\mathtt{bw}}$, automatically summarizes together all nodes with no incoming edges, although again they may represent very different things; we call this *root collapse*. We argue that a good summary should not systematically collapse leaves (respectively, roots), but do so only when they really are similar to each other.

Bisimulation summaries tend to be large, because bisimilarity is rare in heterogeneous graphs. For instance, in Figure 2, none of p_1, $p_2, \ldots p_5$ is bisimilar to the other, due to slight differences in their properties; similarly, the courses c_1, c_2 and c_3 are not bisimilar, because c_3 lacks a description, c_2 is the only one target of a "takes" triple etc. Our experiments in Section 8 confirm this on many graphs.

Bounded bisimilarity To mitigate this problem, k-bisimilarity was introduced [23]. For some integer k, nodes are k-forward (and/or backward) bisimilar iff they are bisimilar *within their k-bound neighborhoods*. One drawback of k-bisimilarity is that it requires users to guess the k value leading to the best compromise between compactness (favored by a low k, e.g., if $k = 0$, all the graph nodes are equivalent, and the summary has a single node) and structural information in the summary (high k). RDF quotients based on k-bisimilarity are studied in [37, 5].

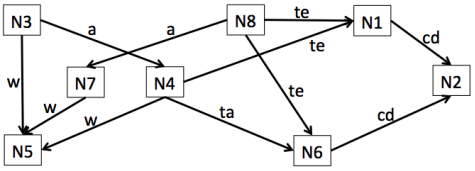


FIGURE 3. 1fb summary of the RDF graph in Figure 2.

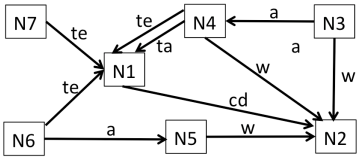


FIGURE 4. 1fw summary of the RDF graph in Figure 2.

It turns out that even 1-bisimilarity is rare in heterogeneous graphs. For instance, Figure 3 shows the 1fb summary of the sample graph in Figure 2[3]. *Here and throughout this paper, summary nodes are shown in rectangles and are labeled N_1, N_2 etc.; we abridge property names to use **w** for wrote, **a** for advises, **te** for teaches, **ta** for takes and **cd** for coursedescr.* Nodes N_3, N_4 and N_8 represent, respectively, p_1, p_2, p_3; the three courses are represented by N_1 and N_6. This summary is almost as complex as the input graph. Figure 4 shows the 1fw summary of our sample G: it is smaller (only 5 nodes and 10 edges, whereas the 1fb one has 8 nodes and 12 edges). However, as our experiments show, 1fw summaries are still too large to be useful for visualization. The 1fw summary, denoted $\sim_{\mathtt{a}}$ in [5], is also very similar to grouping nodes into "characteristic sets" [18, 32]; they all suffer from the leaf collapse issue described above. For instance, they summarize together all articles and course descriptions (N_2 in Figure 4).

To avoid too many characteristic sets (or, equivalently, to reduce the number of summary nodes), [32] proposes a cardinality-based heuristic method which merges them into maximum r sets, for a user-specified threshold r. We do not burden the user with chosing such a threshold; as our experiments show, some graphs are much more complex than

[3]The summaries shown in the Figures 3, 4 and 5 ignore the type triples of G for readability and because they were not used for summarization in the referenced works.

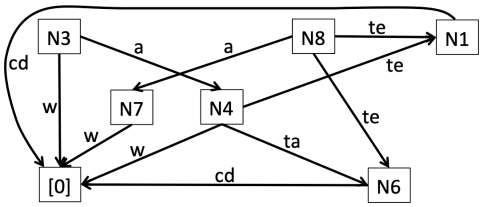


FIGURE 5. $\sim_{\mathtt{ioa}}$ summary [5] of the RDF graph in Figure 2.

others, thus it is not easy to set r, especially for users not yet acquainted with the data, such as the ones we target.

In [5], any RDF equivalence relation (thus, any RDF quotient) considers all the leaf nodes, whether URIs or literals, as equivalent, and represents them by a single summary node denoted [0]. Thus, all their summaries automatically suffer from the leaf collapse issue. For instance, their `1fb` equivalence (denoted $\sim_{\mathtt{ioa}}$) leads to the summary shown in Figure 5. This may significantly reduce the number of summary nodes, since only one of them is a leaf. However, as reported in [5] and verified in our experiments, their bisimilarity-based summaries are still too large for visualization.

Type-driven summarization The remaining equivalence relations used in the literature to summarize RDF graphs are (also) based on *RDF types*.

In [5], two nodes are $\sim_{\mathtt{t}}$-equivalent if they have exactly the same types. This collapses all untyped nodes in a single summary node, e.g., all but p_1 and p_2 in Figure 2, and five out of the eight nodes in Figure 1 (N_2, N_3, N_4, N_5 and N_6), even though they are unrelated.

[5] also introduces $\sim_{\mathtt{ioat}}$, which considers two nodes equivalent iff they are `1fb` equivalent and they have exactly the same types. Thus, a $\sim_{\mathtt{ioat}}$ summary has at least as many nodes and edges as the $\sim_{\mathtt{ioa}}$ one; our experiments confirm it is too large to be used for first-sight visualization.

To conclude, the equivalence relations used in prior RDF quotient summaries are based on:

(1) Bisimilarity (possibly bounded to a maximum distance k), which leads to complex summaries with very high numbers of nodes and edges, unsuited for first-sight discovery.
(2) Uni-directional bisimilarity (i.e., $\equiv_{\mathtt{fw}}$, $\equiv_{\mathtt{bw}}$ and their bounded variants), which suffer from leaf or root collapse;
(3) Types alone: this collapses all untyped nodes, even when their data properties have nothing to do with each other;
(4) Bisimilarity *and* having the same types; this leads to summaries at least as large as those based on bisimilarity alone.

Leaf collapse is also present in all the equivalences of [5].

$\mathtt{G}$	RDF graph
$\mathtt{G}^{\infty}$	the result of saturating $\mathtt{G}$ (Section 2.1)
type	rdf:type (Section 2.1)
sc, *sp*	rdfs:subClassOf, rdfs:subPropertyOf (Section 2.1)
domain, *range*	rdfs:domain, rdfs:range (Section 2.1)
$\equiv_{\mathtt{S}}$	strong equivalence (Definition 2)
$\equiv_{\mathtt{W}}$	weak equivalence (Definition 3)
$\mathtt{G}_{/\mathtt{S}}$	strong summary of $\mathtt{G}$ (Definition 5)
$\mathtt{G}_{/\mathtt{W}}$	weak summary of $\mathtt{G}$ (Definition 4)
$\equiv_{\mathtt{TW}}$	typed weak equivalence (Section 4.2)
$\equiv_{\mathtt{TS}}$	typed strong equivalence (Section 4.2)
$\mathtt{G}_{/\mathtt{TW}}$	typed weak summary of $\mathtt{G}$(Definition 6)
$\mathtt{G}_{/\mathtt{TS}}$	typed strong summary of $\mathtt{G}$ (Section 4.2)
$\widehat{=}$	strong homomorphism (Definition 7)

TABLE 1. Summary of the notations used in the article.

Another limitation of prior work is not considering how summarization interacts with saturation, and instead simply assuming that $\mathtt{G}$ is already saturated. When this is not the case, obtaining the summary of $\mathtt{G}^{\infty}$ requires first computing this saturation, which may be costly in terms of computation time and storage space.

To go beyond these limitations, in the sequel, we introduce our *novel equivalence relations*, leading to *compact and informative* quotient summaries of RDF graphs, whether they are fully typed, have no types at all, or are anywhere in between. Further, we provide *novel, advanced techniques for summarizing a graph's saturation without saturating it*; this can lead to speed-ups of orders of magnitude. The notations we use in this work are compiled in Table 1.

3. Data graph summarization

We first consider graphs made of *data triples* only. We define the novel notion of *property cliques* in Section 3.1; building on them, we devise new graph node equivalence relations and corresponding graph summaries in Section 3.2. Summarization will be generalized to handle also *type triples* in Section 4.

3.1. Data property cliques.

Let us consider the ways in which data properties (edge labels) are organized in a graph. The simplest relation is *co-occurrence*, when a node is the source (or target) of two edges carrying the two labels. However, as illustrated in Figure 2, two properties, such as *wrote* and

	Source clique and nodes having this source clique:
SC_1	{*advises*, *takes*, *teaches*, *wrote*}: p_1, p_2, p_3, p_4, p_5
SC_2	{*coursedescr*}: c_1, c_2, c_3
SC_3	$\emptyset$: a_1, a_2
	Target clique and nodes having this target clique:
TC_1	{*advises*}: p_2, p_5
TC_2	{*teaches*, *takes*}: c_1, c_2, c_3
TC_3	{*coursedescr*}: d_1, d_2
TC_4	{*wrote*}: a_1, a_2
TC_5	$\emptyset$: p_1, p_3, p_4

TABLE 2. Source and target cliques of G nodes (Figure 2).

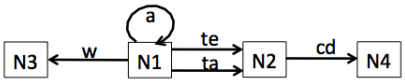


FIGURE 6. Weak summary of the RDF graph in Figure 2 (type triples excluded).

advises, may co-occur on one node, while another one may have *wrote* and *teaches*. The main intuition of our work is to consider *all* these properties (*wrote*, *advises*, *teaches*) related, as they **directly or transitively** co-occur on some nodes. Formally:

Definition 1. ([)*Property relations and cliques]* *Let* $\mathtt{p}_1$, $\mathtt{p}_2$ *be two data properties in* G*:*

(1) $\mathtt{p}_1, \mathtt{p}_2 \in \mathtt{G}$ *are* source-related *iff either: (i) a data node in* G *is the subject of both* $\mathtt{p}_1$ *and* $\mathtt{p}_2$*, or (ii)* G *holds a data node that is the subject of* $\mathtt{p}_1$ *and of a data property* $\mathtt{p}_3$*, with* $\mathtt{p}_3$ *and* $\mathtt{p}_2$ *being source-related.*

(2) $\mathtt{p}_1, \mathtt{p}_2 \in \mathtt{G}$ *are* target-related *iff either: (i) a data node in* G *is the object of both* $\mathtt{p}_1$ *and* $\mathtt{p}_2$*, or (ii)* G *holds a data node that is the object of* $\mathtt{p}_1$ *and of a data property* $\mathtt{p}_3$*, with* $\mathtt{p}_3$ *and* $\mathtt{p}_2$ *being target-related.*

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

In the graph in Figure 2, properties *advises* and *teaches* are source-related due to p_4 (condition (i) in the definition). Similarly, *advises* and *wrote* are source-related due to p_1; consequently, teaches and wrote are source-related (condition (ii)). Further, the graduate student p_2 *teaches* a course and *takes* another, thus *teaches*, *advises*, *wrote* and *takes* are all part of the same source clique. Table 2 shows the target and source cliques of all data nodes from Figure 2.

It is easy to see that the set of non-empty source (or target) property cliques is a *partition over the data properties of* G. Further, if a node $n \in \mathtt{G}$ is a source of some data properties, they are all in the same source clique; similarly, all the properties of which n is a target are in the same target clique.

3.2. **Strong and weak node equivalences.** Building on property cliques, we define two *node equivalence relations* among the data nodes of a graph G:

Definition 2. ([)*Strong equivalence]* *Two data nodes* n_1, n_2 *of* G *are* strongly equivalent*, denoted* $n_1 \equiv_{\mathtt{S}} n_2$*, iff they have the same source* and *target cliques.*

Strongly equivalent nodes have the same structure of *incoming and outgoing* edges. In Figure 2, nodes p_1, p_3 and p_4 are *strongly* equivalent to each other. Among these, note that p_1 and p_3 may seem very dissimilar: p_1 has the properties {*wrote*, *advises*} while p_3 has only {*teaches*}. These nodes are strongly equivalent due to the node p_4 which has a common outgoing property with p_1, and one with p_3. Similarly, p_2, p_5 are strongly equivalent, and so are c_1, c_2 and c_3 etc. The transitivity built in strong equivalence through the use of cliques allows to recognize all these publication nodes as equivalent, and avoid separating them (as $\equiv_{\mathtt{fb}}$ and $\equiv_{\mathtt{fw}}$ do, recall Figures 3, 4 in Section 2.2). Thus, clique-based equivalence avoids the pitfall of leading to too many nodes for a readable visualization.

A second, weaker notion of node equivalence requests only that equivalent nodes share the same *incoming or outgoing* structure, i.e., they share the same source clique or the same target clique. Formally:

Definition 3. ([)*Weak equivalence]* *Two data nodes* n_1, n_2 *are* weakly equivalent*, denoted* $n_1 \equiv_{\mathtt{W}} n_2$*, 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*.*

It is easy to see that $\equiv_{\mathtt{W}}$ and $\equiv_{\mathtt{S}}$ are equivalence relations and that strong equivalence implies weak equivalence, noted $\equiv_{\mathtt{S}} \Rightarrow \equiv_{\mathtt{W}}$.

In Figure 2, $p_1, \ldots, p_5$ are *weakly* equivalent to each other due to their common source clique SC_1; a_1, a_2 are *weakly* equivalent due to their common target clique etc.

From the definitions above and the equivalence notions recalled in Section 2.2, it follows that $\equiv_{\mathtt{1fb}}$ is more restrictive than $\equiv_{\mathtt{S}}$ ($\equiv_{\mathtt{1fb}} \Rightarrow \equiv_{\mathtt{S}} \Rightarrow \equiv_{\mathtt{W}}$). However, $\equiv_{\mathtt{S}}$ and $\equiv_{\mathtt{W}}$, which reflect outgoing *and* incoming properties, are in general incomparable with $\equiv_{\mathtt{1fw}}$ and $\equiv_{\mathtt{1bw}}$, which only outgoing, respectively, incoming properties. The *transitive aspect* of property cliques is a radical departure from previously considered equivalence relations (Section 2). It gives $\equiv_{\mathtt{S}}$ and $\equiv_{\mathtt{W}}$ the flexibility to accept as equivalent structurally heterogeneous nodes, leading to summaries which are both meaningful and compact.

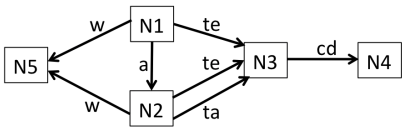


FIGURE 7. Strong summary of the RDF graph in Figure 2 (type triples excluded).

Property nodes equivalence We make an important addition to the clique-based equivalences introduced above. A property node (Section 2.1), that is, a node (subject or object) labeled by an URI which also appears as a property of a data node, is only $\equiv_{\mathtt{W}}$ and $\equiv_{\mathtt{S}}$ to itself. Property nodes are pretty rare, but they do occur in some RDF graphs; an example would be the subject of a triple such as *takes definedBy studentOfficerX*, where takes appears as a data property in Figure 2. A triple comprising a property node can be seen as a form of *metadata*, which helps interpret/understand the graph's data triples. Thus, we consider that a property node is only equivalent to itself.

3.3. **Weak and strong summarization. Weak summarization**. The first summary we define is based on weak equivalence:

Definition 4. ([] *Weak summary] The* weak summary *of a data graph* $\mathtt{G}$*, denoted* $\mathtt{G}_{/\mathtt{W}}$*, is its quotient graph w.r.t. the weak equivalence relation* $\equiv_{\mathtt{W}}$.

The weak summary of the graph in Figure 2 is depicted in Figure 6. N_1 represents all the people (p_1 to p_5), N_2 represents the courses, N_3 the articles and N_4 the course descriptions. Note the self-loop from N_1 to itself; it denotes that *some* nodes represented by N_1 advise *some* nodes represented by N_1. This summary has only 4 nodes and 5 edges; it is smaller (at most half as many edges) and much easier to grasp than the `1fb`, `1fw` and $\sim_{\mathtt{ioa}}$ ones, shown in Section 2. At the same time, it conveys the essential information that some nodes *advise*, *write*, also they *teach* and *take* something that *has course descriptions*.

Generally speaking, node N_i in the weak summary $\mathtt{G}_{/\mathtt{W}}$ of a graph $\mathtt{G}$ represents *all the* $\mathtt{G}$ *nodes whose outgoing (respectively, incoming) properties are a subset of the outgoing (resp., incoming) properties of* N_i.

The weak summary has the following important property:

Proposition 1. (UNIQUE DATA PROPERTIES) *Each* $\mathtt{G}$ *data property appears exactly once in* $\mathtt{G}_{/\mathtt{W}}$.

We exploit this to efficiently build weak graph summaries (Section 7).

We remark that the weak summary $\mathtt{G}_{/\mathtt{W}}$ of a graph $\mathtt{G}$ has *minimal size* (in the number of edges) among *all* the quotient summaries of $\mathtt{G}$: every property labeling a $\mathtt{G}$ edge appears *exactly once* in $\mathtt{G}_{/\mathtt{W}}$, while, by definition, it appears *at least once* in any quotient summary (Section 2.2). Our experiments show that $|\mathtt{G}_{/\mathtt{W}}|$ is typically 3 to 6 orders of magnitude smaller than $|\mathtt{G}|$.

Strong summarization. Next, we introduce:

Definition 5. ([] *Strong summary] The* strong summary *of the graph* $\mathtt{G}$*, denoted* $\mathtt{G}_{/\mathtt{S}}$*, is its quotient graph w.r.t. the strong equivalence relation* $\equiv_{\mathtt{S}}$.

The strong summary of the graph of Figure 2 is shown in Figure 7. Similarly to the weak summary (Figure 6), the strong one groups all courses together, and all articles together. However, it separates the person node in two: those represented by N_1 advise those represented by N_2. This is because the target clique of p_1, p_3 and p_4 is empty, while the target clique of p_2 and p_5 is $\{advises\}$ (Table 2). Due to this finer granularity, in $\mathtt{G}_{/\mathtt{S}}$, several edges may have the same label, e.g., there are two *teaches* and two *wrote* edges in Figure 7, whereas in $\mathtt{G}_{/\mathtt{W}}$, as stated in Proposition 1, this is not possible. Our experiments (Section 8) show that while $\mathtt{G}_{/\mathtt{S}}$ is often somehow larger than $\mathtt{G}_{/\mathtt{W}}$, it still remains many orders of magnitude smaller than the original graph.

By definition of $\equiv_{\mathtt{S}}$, equivalent nodes have the same source clique and the same target clique. This leads directly to the next result, exploited by our algorithms for building strong summaries (Section 7):

Proposition 2. (STRONG SUMMARY NODES AND $\mathtt{G}$ CLIQUES) $\mathtt{G}_{/\mathtt{S}}$ *has exactly one node for each source clique and target clique of a same* $\mathtt{G}$ *data node.*

From the strong to the weak summary. Because strong equivalence implies weak equivalence, it follows that $(\mathtt{G}_{/\mathtt{S}})_{/\mathtt{W}} = \mathtt{G}_{/\mathtt{W}}$. For instance, the nodes N_1 and N_2 in Figure 7 have the same source clique, thus the weak summary of the graph in Figure 7 is exactly the one in Figure 6. Hence, one can get *both* $\mathtt{G}_{/\mathtt{W}}$ *and* $\mathtt{G}_{/\mathtt{S}}$ by building $\mathtt{G}_{/\mathtt{S}}$ and then weakly summarizing it to also get $\mathtt{G}_{/\mathtt{W}}$. This is (much) faster than re-summarizing $\mathtt{G}$, mainly because $\mathtt{G}_{/\mathtt{S}}$ is much smaller than $\mathtt{G}$. Another consequence is that $\mathtt{G}_{/\mathtt{W}}$, intuitively, compresses more (is more imprecise) than $\mathtt{G}_{/\mathtt{S}}$[4]; we demonstrate this also through experiments (Table 6 in Section 8).

Property node representation. Since property nodes represent a form of metadata about the data graph, we decide that in *all* our quotient summaries they are always represented by themselves, i.e., a node labeled with the same URI.

SameAs and generic properties. Some special properties frequently used in RDF deserve a special

[4]One example among many: the `W` summary of a BSBM 1M graph has just one node, whereas the `S` summary has 5 and is quite informative (`https://rdfquotient.inria.fr/files/2019/11/bsbm1m_s_split_and_fold_leaves.png`).

treatment. First, the standard *owl:sameAs* property is used to denote that two URIs should be considered as being "the same"; in particular, the incoming/outgoing edges of one should also be considered as belonging to the other. To reflect this special semantics, we extend our notion of clique to *treat the properties incoming/outgoing two nodes connected by sameAs (directly or indirectly) as if they occurred on the same node*. This ensures that any two nodes connected by sameAs have the same source and target clique, thus they are weakly and strongly equivalent. Second, some generic properties such as *rdfs:label* are sometimes used to annotate RDF nodes with very different meaning. Building cliques based on the co-occurrence of such generic properties may consider too many nodes equivalent. To avoid this, we build our cliques ignoring the triples whose properties are generic, construct $\mathtt{G}_{/\mathtt{W}}$ or $\mathtt{G}_{/\mathtt{S}}$ accordingly, then, for each triple of the form n_1 rdfs:label t (where t is some text), we add an edge N_1 rdfs:label N_2 to the weak or strong summary, where N_1 is the representative of n_1, and N_2 is a new summary node, representing t.

4. Typed data graph summarization

We now discuss the summarization of graphs with data and type triples. Types represent domain knowledge that the data producers found meaningful to describe it. However, some or all nodes of a graph may lack types.

Only a few RDF graph summarization works explicitly considered type triples. The equivalence relation $\sim_{\mathtt{t}}$ introduced in [5] follows an approach we call *type-only*: nodes are equivalent if they have exactly the same types. This approach groups together all untyped nodes, which is problematic in graphs such as the one summarized in Figure 1. The same approach is taken in [27] which further assumes that all non-leaf nodes are typed, a supposition not borne out in practice (see again Figure 1). A different approach we term *data-and-type* is taken in [5]: for two nodes to be equivalent, they should both be equivalent according to a relation that only reflects their data properties, and have the same types. For instance, $\sim_{\mathtt{ioat}}$ is based both on having the same input and output properties ($\sim_{\mathtt{ioa}}$), and the same types. As explained in Section 2, this splits graph nodes into many equivalence classes (summary nodes), which is not desirable for first-sight visualization.

A better approach introduced in [5] to reflect types in a quotient summary built from an equivalence relation $\equiv$ is as follows: first, summarize $\mathtt{G}$ ignoring type triples, and second, for each triple n *type* C in $\mathtt{G}$, add to $\mathtt{G}_{/\equiv}$ a triple N *type* C, where N represents n in $\mathtt{G}_{/\equiv}$. We call this quotient summarization approach *data-then-type*. It does not suffer from the disadvantages of *type-only* nor *data-and-type*. Instead, it allows to identify meaninful node groups even in graphs where some or all the nodes lack types.

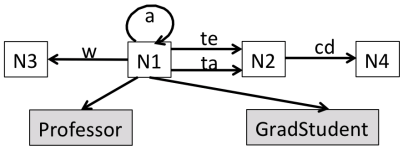


Figure 8. Weak summary of the graph in Figure 2.

Below, we start by formalizing the special treatment we argue should be given to class and property nodes in any RDF quotient summary. Based on this, we extend the *data-then-type* approach to our clique-based equivalence relations (Section 4.1), then present another novel approach which we call *type-then-data*. It gives priority to types when available, while still avoiding the pitfalls of type-only and data-and-type summarization (Section 4.2).

Class node equivalence and representation. To ensure quotient summaries preserve the application knowledge encoded within the classes, properties and ontology of a graph, we decide that in any equivalence relation $\equiv$, any class node is only equivalent to itself, and any class node is represented by itself, and similarly for property nodes. Hence, a typed data graph has the same class nodes, and the same property nodes (Section 3), as its summary.

4.1. Data-then-type summarization.

We extend the $\mathtt{W}$, respectively $\mathtt{S}$ summaries to type triples, by stating that they follow the data-then-type approach.

Figure 8 illustrates this for $\mathtt{G}_{/\mathtt{W}}$; note that N_1 is attached both *Professor* and *GradStudent* types. Generally, a typed node N_i in a typed weak summary represents *all the* $\mathtt{G}$ *nodes whose incoming/outgoing properties are included in those of* N_i*, some of which may also have some of the types of* N_i; the $\mathtt{G}$ nodes represented by an untyped node N_j are the same as in a weak summary.

Figure 9 shows the strong summary of our sample graph when type triples are considered. Note that the *Professor* type is attached to N_1, the representative of p_1, p_3 and p_4 (those who *advise* someone), while *GradStudent* is attached to N_5, representing the *advise*es (p_2 and p_5).

4.2. Type-then-data summarization.

This approach is novel. In contrast with data-then-types, it considers that node types are more important when deciding whether nodes are equivalent, however, it still relies on data properties to summarize untyped nodes. Thus, from an equivalence relation $\equiv$ (based on data properties alone), we derive a *novel typed*

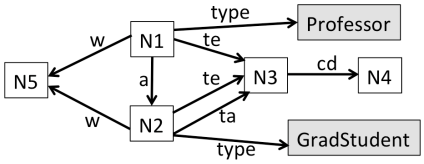


FIGURE 9. Strong summary of the graph in Figure 2.

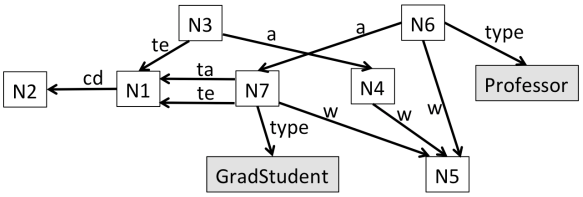


FIGURE 10. Typed weak summary of the graph in Figure 2.

equivalence relation whereas two nodes are equivalent if:

- both are typed, and they have the same set of types;
- or, both are untyped, and they are equivalent according to $\equiv$.

For weak summarization, this approach leads to:

Definition 6. ([]*Typed weak summary] Let* $\equiv_{\mathtt{TW}}$ *(*typed weak equivalence*) be an equivalence relation that holds between two data nodes* n_1, n_2 *iff (i)* n_1, n_2 *have no types in* $\mathtt{G}$ *and* $n_1 \equiv_{\mathtt{W}} n_2$*; or (ii)* n_1, n_2 *have the same non-empty set of types in* $\mathtt{G}$*. The* typed weak summary $\mathtt{G}_{/\mathtt{TW}}$ *of a graph* $\mathtt{G}$ *is denoted* $\mathtt{G}_{/\mathtt{TW}}$*.*

Figure 10 shows the typed weak summary of our sample RDF graph. Unlike $\mathtt{G}_{/\mathtt{W}}$ (Figure 6), $\mathtt{G}_{/\mathtt{TW}}$ represents p_1 by N_1, separately from p_3, because p_3 is of type *Person*, while p_3 is untyped.

In a similar manner, we define **typed strong equivalence**, denoted $\equiv_{\mathtt{TS}}$, as in Definition 6 by replacing $\equiv_{\mathtt{W}}$ with $\equiv_{\mathtt{S}}$, and denoting by $\mathtt{G}_{/\mathtt{TS}}$ the **typed strong summary** of a graph $\mathtt{G}$. In our example, $\mathtt{G}_{/\mathtt{TS}}$ coincides with $\mathtt{G}_{/\mathtt{TW}}$.

From the typed strong to typed weak summary. It is easy to see that if $n_1 \equiv_{\mathtt{TS}} n_2$, then also $n_1 \equiv_{\mathtt{TW}} n_2$, therefore $(\mathtt{G}_{/\mathtt{TS}})_{/\mathtt{TW}} = \mathtt{G}_{/\mathtt{TW}}$. This also allows building $\mathtt{G}_{/\mathtt{TS}}$ and $\mathtt{G}_{/\mathtt{TW}}$ for almost the cost of building $\mathtt{G}_{/\mathtt{TS}}$ alone, as this summary is small and thus summarized quickly.

Summary equality. We now consider when two of our summaries may coincide for a given graph $\mathtt{G}$, i.e., they are the same up to their data node labels N_1, N_2 etc. To formalize this, we define:

Definition 7. ([]*Strong isomorphism* $\widehat{=}$*] A strong isomorphism between two RDF graphs* $\mathtt{G}_1, \mathtt{G}_2$*, noted* $\mathtt{G}_1 \widehat{=} \mathtt{G}_2$*, is an isomorphism which is the identity for the class and property nodes.*

We remark that for visualization purposes, strongly isomorphic summaries can be seen as identical, as they describe exactly the same structure.

5. Summarization of graphs with RDFS ontologies

We now turn to the general case of an RDF graph with an RDFS ontology.

First, observe that any summary of an RDF graph has the same RDFS ontology as this graph. This is because: (*i*) every property node is only represented by itself (Section 3.2), (*ii*) every class node is represented by itself (Section 3.3), and (*iii*) by definition of an RDF quotient, any ontology triple, which only connects two class or property nodes, is "copied" in the summary. We view ontology preservation as a desirable feature, since the ontology has crucial information about the meaning of the data[5].

Second, we identify two ways in which an ontology can impact summarization:

Through type generalization:: Type-then-data summarization (Section 4.2) groups nodes by their sets of types. If the ontology features triples of the form c_1 *sc* c_2, it can be argued that c_2 can be used instead of c_1 to summarize a resource having the type c_1. We study this in Section 5.1.

Through implicit triples:: As we explained in Section 2, the semantics of an RDF graph $\mathtt{G}$ includes its explicit triples, but also its *implicit* triples which are not in $\mathtt{G}$, but hold in $\mathtt{G}^{\infty}$ due to ontological constraints (such as the triples p_2 *type Instructor*, p_2 *type Student*, and p_1 *knows* p_2 in Section 2.1). An interesting question, then, is to determine the interplay between saturation and summarization: how is the summary of $\mathtt{G}^{\infty}$ related to that of $\mathtt{G}$, first, in general (for any quotient summary), and then, for the four summaries we introduced? The rest of the section is devoted to this topic.

5.1. Type-then-data summarization using most general types.

The most commonly used feature of RDFS ontologies is the subClass relationship, stating that any resource of a type c_1 is also of the type c_2; "subtype" and "supertype" are commonly used to denote c_1 and c_2 in such settings. The subgraph consisting of the subClass triples of a graph is typically acyclic (if a loop existed, all the types involved in the loop would be equivalent for

[5]While we consider the ontology very important, our goal is to bring the much more numerous *data and type* (non-ontology) triples to a visually comprehensible size through summarization. When present, the ontology may help visualize the data; the ontology itself may be summarized etc.

all practical purposes and could be replaced with any among them); we assume below that this is the case, thus the types present in G can be organized in a directed acyclic graph (DAG). While class nodes are very often much fewer than data nodes, they can still be too numerous for a small visualization to include or reflect all of them. For instance, there are more than 500 product types labeled ProductType1, ProductType2 etc. in a BSBM benchmark graph of 100M triples, a WatDiv benchmark graph of 10M triple comprises 14 product category types, or the real-life DBLP dataset includes a dozen types of scientific publications. Applying type-then-data summarization to such a graph would lead to a high number of nodes, one for each type; this appears shortsighted, given that in these examples, a natural common supertype can be found, e.g., ProductType, ProductCategory, and Publication, respectively.

To obtain compact type-then-data summaries even in the presence of such ontologies, we adopt the following practical solution:

- For each typed data node $x \in \mathtt{G}$, let $\tau(x)$ be the set of all types associated to x in G, and $\overline{\tau(x)}$ the set comprising the most general supertypes of the types in $\tau(x)$[6], which can be easily computed based on the subClass triples.
- Then, **type-then-data summarization based on most general types** uses $\overline{\tau(x)}$ instead of $\tau(x)$. This is how we obtained the graph in Figure 1: there, N1 represents all the nodes whose most general type set comprises exactly `http://db.uwaterloo.ca/~galuc/wsdbm/Genre`, and similarly for N8 and the type `http://db.uwaterloo.ca/~galuc/wsdbm/ProductCategory`.

This technique can be applied to both typed weak and typed strong summarization.

5.2. **Interactions between summarization and saturation.** First, does saturation *commute with* summarization? In other words, is $(\mathtt{G}^\infty)_{/\equiv}$ strongly isomorphic (Definition 7) to $(\mathtt{G}_{/\equiv})^\infty$? Figure 11 shows that this is not always the case; *sp* denotes the standard property RDFS *rdfs:subPropertyOf* (Section 2.1). For a given graph G, the figure shows its weak summary $\mathtt{G}_{/\mathtt{W}}$ and its saturation $(\mathtt{G}_{/\mathtt{W}})^\infty$, as well as $\mathtt{G}^\infty$ and its summary $(\mathtt{G}^\infty)_{/\mathtt{W}}$. Here, saturation leads to b edges outgoing both r_1 and r_2 which makes them equivalent in $\mathtt{G}^\infty$. In contrast, summarization *before* saturation represents them separately; saturating the summary cannot unify them

[6]We exclude a few "standard" root types, such as rdfs:Resource in RDF Schema or OWL:Thing, from the supertype hierarchy, as these would not bring useful information to summary users.

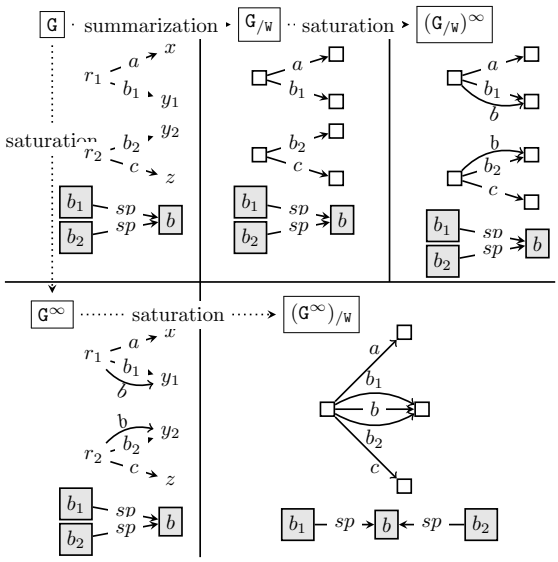


FIGURE 11. Saturation and summarization example.

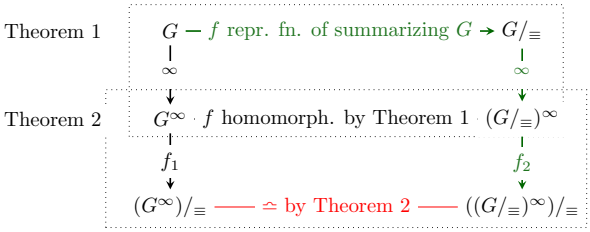


FIGURE 12. Illustration for Theorem 1 and Theorem 2.

as in $(\mathtt{G}^\infty)_{/\mathtt{W}}$ (recall from Section 2.1 that saturation can only *add edges* in a graph).

While $(\mathtt{G}^\infty)_{/\equiv}$ and $(\mathtt{G}_{/\equiv})^\infty$ are *not* strongly isomorphic in general, we establish that they *always* relate as follows (see the diagram in Figure 12):

Theorem 1. ([)*Summarization homomorphism]* *Let* G *be an RDF graph,* $\mathtt{G}_{/\equiv}$ *its summary and* f *the corresponding representation function from* G *nodes to* $\mathtt{G}_{/\equiv}$ *nodes. Then* f *defines a homomorphism from* $\mathtt{G}^\infty$ *to* $(\mathtt{G}_{/\equiv})^\infty$.

Since $(\mathtt{G}_{/\equiv})^\infty$ is homomorphic to $\mathtt{G}^\infty$, would *their* summaries coincide, i.e., be strongly isomorphic? It turns out that this may hold or not depending on the RDF equivalence relation under consideration. When it holds, we call *shortcut* the following three-step transformation aiming at obtaining a summary strongly isomorphic to $(\mathtt{G}^\infty)_{/\equiv}$, instead of $(\mathtt{G}^\infty)_{/\equiv}$ itself: first summarize G; then saturate its summary; finally, summarize it again in order to build $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$:

Definition 8. ([)*Shortcut]* *We say* the shortcut holds *for a given RDF node equivalence relation* $\equiv$

iff for any $\mathtt{G}$, $(\mathtt{G}^{\infty})_{/\equiv}$ *and* $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ *are strongly isomorphic.*

Note that from a practical viewpoint, hence for visualization, $(\mathtt{G}^{\infty})_{/\equiv}$ and $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ are equivalent as they differ just in their data node IDs (e.g., N1, N2 etc. in Figure 1), which carry no particular meaning.

Next, we establish one of our main contributions: a *sufficient condition* under which for *any* quotient summary based on an equivalence relation $\equiv$ as discussed above (where class and property nodes are preserved by summarization), the shortcut holds. In particular, as we will demonstrate (Section 8), the existence of the shortcut can lead to computing $(\mathtt{G}^{\infty})_{/\equiv}$ *substantially faster* by actually computing $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$.

Theorem 2. ([])*Sufficient shortcut condition] Let* $\mathtt{G}_{/\equiv}$ *be a summary of* $\mathtt{G}$ *through* $\equiv$ *and* f *the corresponding representation function from* $\mathtt{G}$ *nodes to* $\mathtt{G}_{/\equiv}$ *nodes (see Figure 12).*

If $\equiv$ *satisfies: for any RDF graph* $\mathtt{G}$ *and any pair* (n_1, n_2) *of* $\mathtt{G}$ *nodes,* $n_1 \equiv n_2$ *in* $\mathtt{G}^{\infty}$ *iff* $f(n_1) \equiv f(n_2)$ *in* $(\mathtt{G}_{/\equiv})^{\infty}$*, then the shortcut holds for* $\equiv$*.*

Figure 12 depicts the relationships between an RDF graph $\mathtt{G}$, its saturation $\mathtt{G}^{\infty}$ and summarization $(\mathtt{G}^{\infty})_{/\equiv}$ thereof, and the RDF graphs that appear at each step of the shortcut computation. The intuition for the sufficient condition is the following. On any path in Figure 12, saturation adds edges to its input graph, while summarization "fuses" nodes into common representatives. On the regular path from $\mathtt{G}$ to $(\mathtt{G}^{\infty})_{/\equiv}$, edges are added in the first step, and nodes are fused in the second. On the shortcut (green) path, edges are added in the second step, while nodes are fused in the first and third steps. The two paths starting from $\mathtt{G}$ can reach $\eqsim$ results only if $\mathtt{G}$ nodes fused on the shortcut path are also fused (when summarizing $\mathtt{G}^{\infty}$) on the standard path. In particular, *the first summarization along the shortcut path should not make wrong node fusions*, that is, fusions not made when considering the full $\mathtt{G}^{\infty}$: such a "hasty" fusion *can never be corrected later on along the shortcut path*, as neither summarization nor saturation split nodes. Thus, an erroneous fusion made in the first summarization step irreversibly prevents the end of the shortcut path from being $\eqsim$ to $(\mathtt{G}_{/\equiv})^{\infty}$.

When the condition is met, summarizing $\mathtt{G}$, then saturating its summary, then summarizing the graph thus obtained leads to $(\mathtt{G}^{\infty})_{/\equiv}$ (up to data node labels) *without the need to saturate* $\mathtt{G}$. The shortcut can be faster than saturating $\mathtt{G}$ then summarizing the result, because the shortcut avoids the cost to *find, store, and summarize* the implicit triples derived from $\mathtt{G}$; it only deals with the implicit triples derived from $\mathtt{G}_{/\equiv}$, which (depending on $\equiv$) may be much smaller than $\mathtt{G}$.

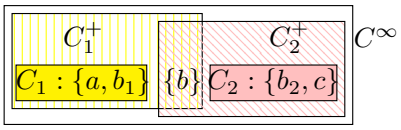


FIGURE 13. Two source cliques from the graph in Figure 11, their saturations, and their enclosing clique C^{∞} in $\mathtt{G}^{\infty}$.

5.3. **Shortcut results.** In order to establish shortcut results for our summaries, we start by investigating how property cliques are impacted by saturation.

In $\mathtt{G}^{\infty}$, every $\mathtt{G}$ node has all the data properties it had in $\mathtt{G}$, therefore two data properties belonging to a $\mathtt{G}$ clique are also in the same clique of $\mathtt{G}^{\infty}$. Further, if the schema of $\mathtt{G}$ comprises subProperty constraints, a node may have in $\mathtt{G}^{\infty}$ a data property that it did not have in $\mathtt{G}$. As a consequence, each $\mathtt{G}^{\infty}$ clique includes one or several cliques from $\mathtt{G}$, which may "fuse" by acquiring more properties due to saturation with subProperty constraints. An example is given in Figure 13, where C_1^+ and C_2^+ are the saturations of the source cliques C_1, C_2, while $C^{\infty} = \{a, b_1, b, b_2, c\}$ is a source clique of the graph $\mathtt{G}^{\infty}$ (also in Figure 11).

Based on Theorem 2 and the above observations, we show:

Theorem 3. ([])$\mathtt{W}$ *shortcut] The shortcut holds for* $\equiv_{\mathtt{W}}$.

For instance, on the graph in Figure 11, it is easy to check that applying summarization on $(\mathtt{G}_{/\mathtt{W}})^{\infty}$ (as prescribed by the shortcut) leads exactly to a graph strongly isomorphic to $(\mathtt{G}^{\infty})_{/\mathtt{W}}$.

Showing Theorem 3 is rather involved; we do it in several steps. First, based on a technical Lemma (see Appendix E) , we show:

Lemma 1. ([])*Property relatedness in* $\mathtt{W}$ *summaries] Data properties are target-related (resp. source-related) in* $(\mathtt{G}_{/\mathtt{W}})^{\infty}$ *iff they are target-related (resp. source-related) in* $\mathtt{G}^{\infty}$.

Based on the above Lemma and Theorem 1, we establish the next result from which Theorem 3 directly follows:

Proposition 3. (SAME CLIQUES-$\mathtt{W}$) $\mathtt{G}^{\infty}$ *and* $(\mathtt{G}_{/\mathtt{W}})^{\infty}$ *have identical source clique sets, and identical target cliques sets. Further, a node* $n \in \mathtt{G}^{\infty}$ *has exactly the same source and target clique as* $f_{\mathtt{W}}(n)$ *in* $(\mathtt{G}_{/\mathtt{W}})^{\infty}$.

Theorem 4. ([])$\mathtt{S}$ *shortcut] The shortcut holds for* $\equiv_{\mathtt{S}}$.

We prove this based on counterparts of statements established for $\mathtt{G}_{/\mathtt{W}}$. First we show:

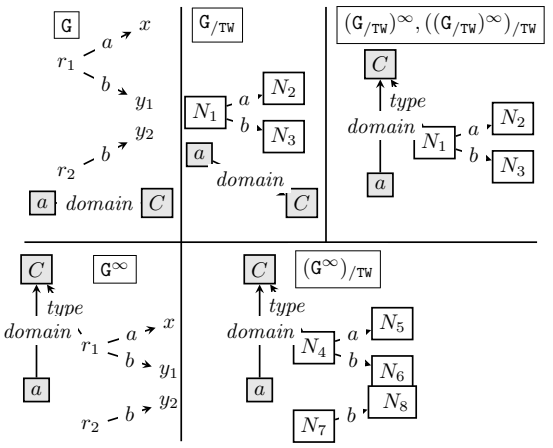


FIGURE 14. Shortcut counter-example.

Lemma 2. ([)*Property relatedness in* S *summaries]* *Data properties are target-related (resp. source-related) in* $(\mathtt{G}_{/\mathtt{S}})^{\infty}$ *iff they are target-related (resp. source-related) in* $\mathtt{G}^{\infty}$.

Then, from Theorem 1 and the above Lemma, we obtain the next proposition from which Theorem 4 directly follows:

Proposition 4. (SAME CLIQUES-S) $\mathtt{G}^{\infty}$ *and* $(\mathtt{G}_{/\mathtt{S}})^{\infty}$ *have identical source clique sets, and identical target clique sets. Further, a node* $n \in (\mathtt{G}_{/\mathtt{S}})^{\infty}$ *has exactly the same source and target clique as* $f_{\mathtt{S}}(n)$ *in* $(\mathtt{G}_{/\mathtt{S}})^{\infty}$.

Finally, we have:

Theorem 5. ([)*No shortcut for* $\equiv_{\mathtt{TW}}$*]* *The shortcut does not hold for* $\equiv_{\mathtt{TW}}$.

We prove this by exhibiting in Figure 14 a counter-example. In G and $\mathtt{G}_{/\mathtt{TW}}$, all data nodes are untyped; only after saturation a node gains the type C. Thus, in $\mathtt{G}_{/\mathtt{TW}}$, one (untyped) node represents all data property subjects; this is exactly a "hasty fusion" as discussed below Theorem 2. In $(\mathtt{G}_{/\mathtt{TW}})^{\infty}$, this node gains a type, and in $((\mathtt{G}_{/\mathtt{TW}})^{\infty})_{/\mathtt{TW}}$, it is represented by a single node. In contrast, in $\mathtt{G}^{\infty}$, r_1 is typed and r_2 isn't, leading to two distinct nodes in $(\mathtt{G}^{\infty})_{/\mathtt{TW}}$. This is not strongly isomorphic with $(\mathtt{G}_{/\mathtt{TW}})^{\infty}$ which, in this example, is strongly isomorphic to $((\mathtt{G}_{/\mathtt{TW}})^{\infty})_{/\mathtt{TW}}$. Thus, the shortcut does not hold for $\equiv_{\mathtt{TW}}$.

Theorem 6. ([)*No shortcut for* $\equiv_{\mathtt{TS}}$*]* *The shortcut does not hold for* $\equiv_{\mathtt{TS}}$.

The graph in Figure 14 is also a shortcut counter-example for TS.

Based on Theorem 2, we have also established:

Theorem 7. ([)*Bisimilarity shortcut]* *The shortcut holds for the forward* ($\equiv_{\mathtt{fw}}$)*, backward* ($\equiv_{\mathtt{bw}}$)*, and forward-and-backward* ($\equiv_{\mathtt{fb}}$) *bisimilarity equivalence relations (recalled in Section 2.2).*

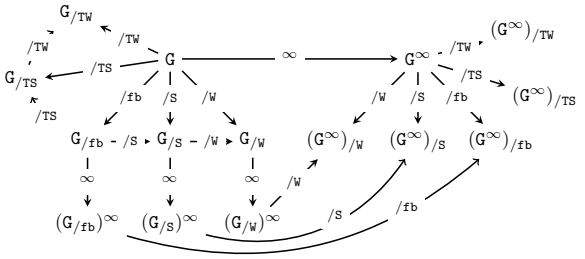


FIGURE 15. Relations between quotient summaries.

5.4. **Relationships between summaries.** From the definition of weak and strong equivalence, it is easy to show that $(\mathtt{G}_{/\mathtt{S}})_{/\mathtt{W}} = \mathtt{G}_{/\mathtt{W}}$, i.e., one could compute $\mathtt{G}_{/\mathtt{W}}$ by first summarizing G into $\mathtt{G}_{/\mathtt{S}}$, and then applying weak summarization on this (typically much smaller) graph; similarly, $(\mathtt{G}_{/\mathtt{TS}})_{/\mathtt{TW}} = \mathtt{G}_{/\mathtt{TW}}$. It is also the case that $(\mathtt{G}_{/\mathtt{W}})_{/\mathtt{S}} = \mathtt{G}_{/\mathtt{W}}$, i.e., strong summarization cannot compress a weak summary further, and similarly $(\mathtt{G}_{/\mathtt{TW}})_{/\mathtt{TS}} = \mathtt{G}_{/\mathtt{TW}}$. Figure 15 summarizes the main relationships between G, $\mathtt{G}^{\infty}$, our summaries and bisimilarity-based ones.

6. FROM SUMMARIES TO VISUALIZATIONS

We now describe how to go from a quotient summary to a graphical visualization such as the one illustrated in the Introduction.

6.1. **Leaf and type inlining.** For structurally simple graphs like our sample G shown in Figure 2, quotient summaries have very few nodes and edges, and any node-link visualization method can be used. We explain here how we obtained our visualizations, illustrated in our *online gallery*[1].

To further simplify summaries, we apply *leaf and type inlining*, as follows. We remove type edges; instead, each type attached to a node in the summary is shown in the box corresponding to the node, after the node ID. Similarly, for each edge $n \xrightarrow{a} m$ where m is a *leaf*, we include a as an "attribute" of n, and do not render m (we say it has been "inlined" within n). A sizable part of an RDF graph's nodes are leaves; as we will show, inlining them into their parent nodes greatly simplifies the visualization.

Figure 16 illustrates inlining for the S summary (Figure 9) of our sample graph. This summary is extremely compact, yet rich with information; professors, students, and courses are visible at a glance. Articles have been inlined within their authors as they were leaves in $\mathtt{G}_{/\mathtt{S}}$. This simplification can also be seen as a small loss of information: Figure 16 does not immediately suggest that Professors may have written articles together with GradStudents. However, (*i*) only leaf nodes are folded and (*ii*) after a first glance, users may pursue exploration by other means (e.g., queries to check for such joint

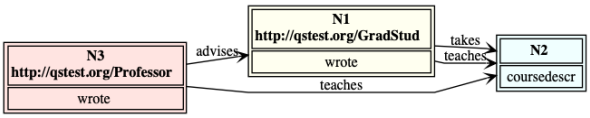


FIGURE 16. Visualization resulting from leaf and type inlining on the sample Strong summary from Figure 9.

articles). Thus, we consider that inlining is overall beneficial, and *systematically apply it on summaries before visualization.*

If type-then-data summarization is used based on the most general types (Section 5.1), the most general types are shown at the top of each typed summary node (immediately under the node ID), then the actual types of the graph nodes represented by the summary node are shown one per line, under the most general types. N1, N7 and N8 in Figure 1 illustrate this.

6.2. **Summary statistics.** If users are interested (also) in a quantitative view of an RDF graph, our summaries can also plot a set of statistics. We describe them below and illustrate them based on Figure 1. For each summary node N_i, we display:

- The number of G nodes represented by N_i, in parenthesis after "N_i" in the corresponding box, e.g., "N8 (25000)";
- For each type c such that $x\ \tau\ c$ for some x represented by N_i, the number of G nodes represented by N_i which are of type c, e.g., `http://db.uwaterloo.ca/~galuc/wsdbm/ProductCategory0:807`
- For each data property p such that (i) $x\ p\ y$ for some x represented by N_i and (ii) all the objects of p triples whose subjects are represented by N_i are leaves in G, the number of such p triples, and the number of distinct targets of such triples. For instance, within N8, "bookedition (847 → 6)" denotes that there are 847 bookedition triples whose subjects are represented by N8, and they reach a total of 6 distinct objects (which are leaf nodes).

For each summary edge $N_i \xrightarrow{a} N_j$ where a is a data property, the number of $x\ a\ y$ triples in G such that x is represented by N_i and y is represented by N_j. The label "hasGenre (58787)" on the edge from N8 to N1 is an example of such an edge statistic.

All these statistics can be gathered by our summary construction algorithms (Section 7), at no extra computational cost.

6.3. **Visualizing very large summaries.** For a very complex (e.g., encyclopedic) dataset, even the graph obtained from inlining may have too many nodes and edges for an effective visualization. If it has several connected components (one per domain), each of them can be viewed separately. Otherwise, it can be split into several, possibly overlapping subgraphs, using any graph decomposition strategy (for instance, minimize the number of times the representatives of two nodes connected in G appear in different summary subgraphs etc.) Information discovery in such graphs requires more computational and cognitive effort.

7. SUMMARIZATION ALGORITHMS

We now present summarization algorithms which, given as input a graph G, construct $G_{/W}$, $G_{/S}$, $G_{/TW}$ and $G_{/TS}$. All our algorithms have an *amortized linear complexity in the size of* G: they can be built in just one or two passes over the data. Our incremental algorithms, capable of reflecting additions to G, into its previously computed summaries, are the most involved.

7.1. **Global data graph summarization.** The first algorithms we present summarize *only the data triples* through *two graph traversals*: one to learn the equivalence relation[7] and create the summary nodes, the second to determine the representative of each G node and, as a consequence, add triples to the summary.

We start with our *global* W *summarization algorithm* (Algorithm 1). It exploits Proposition 1, which guarantees that any data property occurs only once in the summary. To each data property p encountered, it associates a summary node (integer) s_p which will be the (unique) source of p in the summary, and similarly a node t_p target of p; these are initially unknown, and evolve as G is traversed. Further, it uses two maps *op* and *ip* that associate to each data node n, the set of its outgoing, resp. incoming data properties. These are filled during the first traversal of G (step 1.) Steps 2. to 2.5 ensure that for each node n having outgoing properties and possibly incoming ones, s_p for all the outgoing ones are equal, and equal also to t_p for all the incoming ones. This is performed using a function fuse which, given a set of summary nodes, picks one that will replace all of them. In our implementation, summary nodes are assigned integer IDs, and fuse is simply *min*; we just need fuse to be distributive over ∪, i.e., $\text{fuse}(A, (B \cup C)) = \text{fuse}(\text{fuse}(A, B), \text{fuse}(A, C))$. Symmetrically, step 3. ensures that the incoming properties of nodes lacking outgoing properties (thus, absent from *op*) also have the same target. In Step 4., we represent s and o based on the source/target of the property p connecting them.

[7]Our equivalence relations are defined *based on the triples of a given graph* G, thus when summarization starts, we do not know whether any two nodes are equivalent; the full equivalence relation is known only after inspecting all G triples.

global-W(G)

1. For each s p o ∈ G, add p to op(s) and to ip(o).
2. For each node $n \in op$:
2.1. Let $X \leftarrow \text{fuse}\{s_\text{p} \mid \text{p} \in op(n)\}$.
If X is undefined, let $X \leftarrow$nextNode();
2.2. Let $Y \leftarrow \text{fuse}\{t_\text{p} \mid \text{p} \in ip(n)\}$.
If Y is undefined, let $Y \leftarrow$nextNode();
2.3. Let $Z \leftarrow \text{fuse}(X, Y)$;
2.4. For each $\text{p} \in ip(n)$, let $s_\text{p} \leftarrow Z$;
2.5. For each $\text{p} \in op(n)$, let $t_\text{p} \leftarrow Z$;
3. Repeat **2** to **2.5** swapping ip with op and t_p with s_p;
4. For each s p o ∈ G: let f_W(s) ← s_p, f_W(o) ← t_p;
Add f_W(s) p f_W(o) to $\text{G}_{/\text{W}}$.

Algorithm 1: Global W summarization of a graph

global-S(G)

1. For each s p o ∈ G:
1.1. Check if src_p, trg_p, sc(s) and tc(o) are known; those not known are initialized with {p};
1.2. If $sc(\text{s}) \neq src_\text{p}$, fuse them into new clique $src'_\text{p} = sc(\text{s}) \cup src_\text{p}$; similarly, if $tc(\text{o}) \neq trg_\text{p}$, fuse them into $trg'_\text{p} = tc(\text{o}) \cup trg_\text{p}$. **2.** For each s p o ∈ G:
2.1. f_S(s) ← the (unique) summary node corresponding to the cliques $(sc(\text{s}), tc(\text{s}))$; similarly, f_S(o) ← the node corresponding to $(sc(\text{o}), tc(\text{o}))$ (create the nodes if needed).
2.2 Add f_S(s) p f_S(o) to $\text{G}_{/\text{S}}$.

Algorithm 2: Global S summarization of a graph

The fuse operations in 2. and 3. have ensured that, while traversing G triples in 4., any data node n is always represented by the same summary node $f_\text{W}(n)$.
Our *global* S *summarization algorithm* (Algorithm 2) uses two maps sc and tc which store for each data node n, its source clique $sc(n)$, and its target clique $tc(n)$, and for each data property p, its source clique src_p and target clique trg_p. Further, for each (source clique, target clique) pair encountered during summarization, we store the (unique) corresponding summary node. Steps 1.-1.2. build the source and property cliques present in G and associate them to every subject and object node (in sc and tc), as well as to any data property (in src_p and trg_p). For instance, on the sample graph in Figure 2, these steps build the cliques in Table 2. Steps 2-2.2. represent the nodes and edges of G.

The correctness of algorithms **global-W** and **global-S** follows quite easily from their descriptions and the summary definitions.

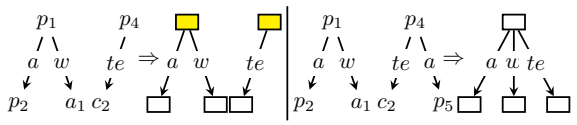
7.2. **Incremental data graph summarization.** These algorithms are particularly suited for incremental *summary maintenance*: if new triples Δ^+_G are added to G, it suffices to summarize only Δ^+_G, based on $\text{G}_{/\equiv}$ and its representation function $f_\equiv$, in order to obtain $(\text{G} \cup \Delta^+_\text{G})_{/\equiv}$. Incremental algorithms are considerably *more complex*, since various decisions (assigning sources/targets to properties in W, source/target cliques in S, node representatives in both) must be *repeatedly revisited* to reflect newly acquired information about G triples, as we shall see.
Each *incremental summarization* algorithm consists of an *incremental update method, called for every data triple*, which adjusts the summary's data structures, so that at any point, the summary reflects exactly the graph triples visited until then.

increm-W(s p o)

1. Check if s_p and o_p are known: either both are known (if a triple with property p has already been traversed), or none;
2. Check if f_W(s) and f_W(o) are known; none, one, or both may be, depending on whether s, respectively o have been previously encountered;
3. Fuse s_p with f_W(s) (if one is unknown, assign it the value of the other), and o_p with f_W(o);
4. Update f_W(s) and f_W(o), if needed;
5. Add the edge f_W(s) p f_W(o) to $\text{G}_{/\text{W}}$.

Algorithm 3: Incremental W summarization of one triple

Algorithm 3 outlines incremental W summarization. For example (see the figure below), let's assume the algorithm traverses the graph G in Figure 2 starting with: p_1 *advises* p_2, then p_1 *wrote* a_1, then p_4 *teaches* c_2. When we summarize this third triple, we do not know yet that p_1 is equivalent to p_4, because no common source of *teaches* and *advises* (e.g., p_3 or p_4) has been seen so far. Thus, p_4 is found not equivalent to any node visited so far, and represented separately from p_1. Now assume the fourth triple traversed is p_4 *advises* p_5: at this point, we know that *advises*, *wrote* and *teaches* are in the same source clique, thus $p_1 \equiv_\text{W} p_4$, and their representatives (highlighted in yellow) must be *fused* in the summary (Step 3.) More generally, it can be shown that $\equiv_\text{W}$ *only grows* as more triples are visited, in other words: if in a subset G' of G's triples, two nodes n_1, n_2 are weakly equivalent, then this holds in any G'' with $\text{G}' \subseteq \text{G}'' \subseteq \text{G}$.



Summary node *fusion* dominates the algorithm's *complexity*. Let N_1, N_2 be two sets of G nodes, represented at a certain point by the distinct summary nodes m_1, m_2. When fusing them in a single m, we must also record that all the nodes in $N_1 \cup N_2$ are now represented by m. A naïve implementation leads to $O(N^2)$ complexity, where N is the number of data nodes, since each new node may lead to a fusion whose cost is $O(N)$; in the worst case N could be proportional to $|\mathtt{G}|$, the number of triples in G, leading to an overall complexity of $O(|\mathtt{G}|^2)$ for the incremental weak summarization.

Instead, we rely on a *Union-Find* [14] (aka Disjoint Sets) data structure, with the *path compression* and *union by size* optimizations, which guarantee an overall *quasi-linear worst-case complexity* to our incremental weak summarization algorithm. The exact complexity is $O(N\alpha(N))$ where $\alpha(N)$, the inverse Ackermann's function, is smaller than 5 for any machine-representable input N. Assimilating this to *linear-time*, the algorithm's complexity class is in $O(|\mathtt{G}|)$, which is also *optimal*, as summarization must fully traverse G.

Algorithm 4 outlines the incremental update of the S summary due to the traversal of the triple s p o. Conceptually, the algorithm is symetric for the source (s) and target (o) of the edge, we only discuss the source side below. Steps 1. and 2. start by determining the source clique of s, based on its previously known source clique (if any) and the previously known target clique of p (if any); after step 2., s's source (and target) clique reflecting also the newly seen triple s p o are completely known. Determining them may have involved fusing some previously separate cliques. For instance, on the graph in Figure 2, assume we first traverse the triple $p1$ *advises* p_2, then p_4 *teaches* c_2; so far we have the source cliques $\{advises\}$, $\{teaches\}$ and $\emptyset$. If the next traversed triple is p_4 *advises* p_5, we fuse the source cliques (step 3.1) $\{advises\}$ and $\{teaches\}$ into $\{advises, teaches\}$. This requires fusing the summary node whose (source, target) cliques were ($\{advises\}$, $\emptyset$) with the one which had ($\{teaches\}$, $\emptyset$) (Step 3.2).

The last intricacy of incremental strong summarization is due to the fact that unlike $\equiv_{\mathtt{W}}$, $\equiv_{\mathtt{S}}$ *may grow and shrink* during summarization. For instance, assume incremental strong summarization of the graph in Figure 2 starts with p_1 *wrote* a_1, p_2 *wrote* a_2, p_2 *takes* c_2 (see the figure below). After these, we know $p_1 \equiv_{\mathtt{S}} p_2$; their source clique

increm-S(s p o)

1. Check if we already know a source clique $src_{\mathtt{p}}$ (resp. target clique $trg_{\mathtt{p}}$). Either both are known (if a p triple has already been traversed), or none. Those not known are initialized with $\{\mathtt{p}\}$;
2. Check if $sc(\mathtt{s})$ (resp. $tc(\mathtt{o})$) are known; those unknown are initialized with $\{\mathtt{p}\}$;
3. If $sc(\mathtt{s}) \neq src_{\mathtt{p}}$, fuse them into new clique $src'_{\mathtt{p}} = sc(\mathtt{s}) \cup src_{\mathtt{p}}$, using Union-Find; similarly, if $tc(\mathtt{o}) \neq trg_{\mathtt{p}}$, fuse them into $trg'_{\mathtt{p}} = tc(\mathtt{o}) \cup trg_{\mathtt{p}}$, and:
3.1 Replace $sc(\mathtt{s})$ and $src_{\mathtt{p}}$ with $src'_{\mathtt{p}}$ throughout the summary (respectively, replace $tc(\mathtt{o})$ and $trg_{\mathtt{p}}$ with $trg'_{\mathtt{p}}$);
3.2 The above may entail summary node fusions; in this case, update $f_{\mathtt{S}}$ (use Union-Find) and the summary edges to reflect it;
4. If before seeing s p o s had been already represented and it had an empty source clique, then s needs to split, i.e., be represented separately from the nodes to which it was $\equiv_{\mathtt{S}}$ previously; call **split-source(s)**. (Symmetric discussion for o, call **split-target(o)**).
5. Update $f_{\mathtt{S}}(\mathtt{s})$ and $f_{\mathtt{S}}(\mathtt{o})$, if needed;
6. Add the edge $f_{\mathtt{S}}(\mathtt{s})$ p $f_{\mathtt{S}}(\mathtt{o})$ to $\mathtt{G}_{/\mathtt{S}}$.

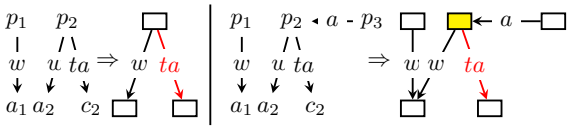
Algorithm 4: Incremental S summarization of one triple

split-source(s)

1. Collect all G edges adjacent to s into $transfer$ set.
2. For each s p o $\in transfer$, decrement by 1 the counter for $f_{\mathtt{S}}(\mathtt{s})$ p $f_{\mathtt{S}}(\mathtt{o})$ in the summary.
3. Update $f_{\mathtt{S}}(\mathtt{s})$.
4. For each s p o $\in transfer$, if such edge already exists in the summary, then increment its counter by 1, otherwise add $f_{\mathtt{S}}(\mathtt{s})$ p $f_{\mathtt{S}}(\mathtt{o})$ to the summary with counter equal to 1.

Procedure 5: Splitting summary node s on source

is $\{wrote, takes\}$ and their target clique is $\emptyset$. Assume the next triple traversed is p_3 *advises* p_2: at this point, p_1 is *not* $\equiv_{\mathtt{S}}$ *to* p_2 *any more*, because p_2's target clique is now $\{advises\}$ instead of the empty $\emptyset$. Thus, p_2 *splits* from p_1, that is, it needs to be represented by a new summary node (shown in yellow below), distinct from the representative of p_1.



Further, note that the representative of p_1 and p_2 (at left above) had one *takes* edge (highlighted in red) which was *solely due to p_2's outgoing takes edge*. By definition of a quotient summary (Section 2.2), that edge *moves* from the old to the new representative of p_2 (the yellow node). If, above at left, p_1 had also had an outgoing edge labeled *takes*, at right, both nodes in the top row would have had an outgoing *takes* edge. It can be shown that *splits only occur in such cases*, i.e., `o` whose target clique becomes non-empty (respectively, `s` whose source clique becomes non-empty, and the node was previously represented together with other nodes; if it was represented alone, we just update the respective clique of its representative).

The procedure **split-source(s)** (Procedure 5) represents `s` separately, to reflect it no longer has an empty target clique, and, for each outgoing edge of `s`: adds a corresponding edge to the new representative of `s`; and checks if, as a consequence, an edge needs to be removed from its previous representative. We establish:

Proposition 5. (Algorithm correctness) *Applying algorithm* **increm-W** *(resp.,* **increm-S***) successively on each triple of* G*, in any order, builds* $\mathtt{G}_{/\mathtt{W}}$ *(resp.,* $\mathtt{G}_{/\mathtt{S}}$*).*

Splitting requires inspecting the data edges attached to the node, in order to add edges to its new representative (such as p_2 *ta* c_2 above). We make the hypothesis, denoted ($\star$), that the average number of edges incoming/outgoing a data node is small (and basically constant) compared to the size of `G`; this was the case in the graphs we have experimented with. Under the ($\star$) hypothesis, using the above data structures (including Union-Find), *the complexity of incremental strong summarization is amortized constant per added triple.*

All our algorithms require $O(|\mathtt{G}|)$ space to store the summary, the representation function, and their other data structures. Encoding all graph and summary nodes as integers, however, reduces the actual memory needs significantly.

7.3. **Incremental summarization of typed graphs.** We now explain how to extend our incremental data triple summarization algorithms to type triples.

To extend `W`, respectively, `S` summarization to type triples in "data-then-type" fashion (Section 4.1), we run `W`, resp. `S` summarization *first, over the data triples only*, as described in the preceding two sections. This assigns their (final) representatives to all data nodes. Then, for each `s` *type* C triple, we simply add to the summary the edge $f_{\mathtt{W}}(\mathtt{s})$ *type* C (resp. $f_{\mathtt{S}}(\mathtt{s})$ *type* C); recall from Section 4 that any class node C is represented by itself.

For "type-then-data" summarization (Section 4.2), *we first traverse the type triples only*, compute all the class sets, and assign to each typed data node a representative based on its class set. Then, we run a *type-aware variant* of a `W` (resp. `S`) algorithm, either global or incremental. The differences are: (*i*) In `TW` summarization, a data property `p` may lack an untyped source (and/or target), if `p` has only typed subjects (resp. objects), e.g., property e in the two-triples graph n_1 *type* C, n_1 e a_1. Similarly, in `TS` summarization, a property like e will lack a source clique, if it does not have an untyped source. Whenever a clique-based representative for a property's source and target is missing, the algorithms will instead use the type-based representative, e.g., the representative of n_1 in n_1 e a_1 will be the one for the type set $\{C\}$. (*ii*) Summarizing the data triple `s p o` does not fuse nor split the representative of `s` (resp. `o`) if `s` (resp. `o`) is typed; the representatives of typed nodes never change. We establish:

Proposition 6. (Algoritm correctness) *Applying global-*W *(respectively global-*S*) on* G*, or applying increm-*W *(resp., increm-*S*) on each triple of* G*, extended as described above for data-then-type or type-then-data summarization leads, resp., to* $\mathtt{G}_{/\mathtt{W}}$*,* $\mathtt{G}_{/\mathtt{S}}$*,* $\mathtt{G}_{/\mathtt{TW}}$ *and* $\mathtt{G}_{/\mathtt{TS}}$*.*

These algorithms need to work with the data and type triples *separately*. Fortunately, most popular RDF store allow such direct access. The space needed to also represent type triples remains linear in $|\mathtt{G}|$.

8. Experimental study

Algorithms compared. We have implemented in a Java 1.8 tool available online[1]:

- Our *global* and *incremental* algorithms for building $\mathtt{G}_{/\mathtt{W}}$, $\mathtt{G}_{/\mathtt{S}}$, $\mathtt{G}_{/\mathtt{TW}}$, $\mathtt{G}_{/\mathtt{TS}}$ (a total of eight);
- An algorithm which computes the `fb` (full bisimilarity) summary of an RDF graph, used in many prior works, e.g., [22]. The summary called $\sim_b$ in [5] is obtained from `fb` by collapsing all leaves together; as explained in Section 2, this leaf collapse introduces a loss of information, thus we do not adopt it.
- Algorithms to build the k-bounded bisimilarity summaries for $k = 1$, denoted `1fw`, `1bw` and `1fb` [23] (`1fw` also corresponds to the characteristic sets of [32]). We do not use higher k values because, as we will show, even at this smallest k, bisimilarity summaries are too large for visualization; higher k would only increase the number of nodes and edges. The $\sim_{ioa}$ summary of [5] is obtained from `1fb` by collapsing all leaves.

Real datasets	|G|	|G$^\infty$|	u%	#p	#C
DBLP	88,153,334	88,153,334	87	26	14
DBpedia Person	7,889,268	7,889,268	63	9	1
Foodista	1,019,799	1,019,799	75	13	5
Nobel Prizes	87,549	119,457	71	45	26
Springer LOD	145,136	213,017	77	26	11
Synth. datasets	|G|	|G$^\infty$|	u%	#p	#C
BSBM [3] 1M	1,000,708	1,009,138	68	38	159
BSBM 10M	10,538,484	10,628,484	66	38	593
BSBM 100M	104,115,556	105,315,556	61	38	2019
BSBM 138M	138,742,476	140,342,476	61	38	2,421
LUBM [19] 1M	1,001,658	1,227,984	34	18	45
LUBM 10M	10,728,460	13,147,069	34	18	45
LUBM 100M	106,778,632	130,843,944	34	18	45

TABLE 3. Datasets used in experiments.

- The best algorithms from [5] building the $\sim_t$ and $\sim_{ioat}$ summaries, but without their leaf collapse.

Settings. All experiments ran on a Linux server with an Intel Xeon CPU E5-2640 v4 @2.40GHz and 124 GB RAM. We used PostgreSQL v9.6 to store RDF triples in an integer-encoded triple table, indexed by s, p and o; the server had 30 GB of shared buffers and 640 MB working memory. The JVM had 90 GB of RAM.

Datasets. We have experimented with real and synthetic graphs of up to 36.5 GB. Table 3 shows for each graph its number of triples |G|, the number of triples in its saturation |G$^\infty$|, the percentage of untyped nodes in the graph u%, and the number of distinct data properties #p and classes #C. Each graph has at least 30% untyped nodes; these form a strong majority in all but the LUBM graphs. Note that in BSBM graphs, the number of classes grows with the data size.

Summary size. Table 4 reports the node and edge counts, denoted $(n|e)$, for the compared summaries, both directly, and after applying inlining (Section 6, denoted with the ^{in} superscript). We report only the data edges in the summaries, and omit (i) schema triples, or (ii) metadata triples, e.g., C *dc:publisher* a_1 states that a_1 published class C etc. These omitted triples are the same for all summaries[8]. For space reasons, we delegate the 1bw and 1fb numbers to [6]; they are slightly worse (i.e., more nodes and edges) than those of 1fw. We also delegate to [6] the TW results, which are often identical to those for TS. Numbers are missing (-|-) when algorithms ran out of memory or longer than 3 hours.

[8]In details, for the graphs in Table 4, we omitted: 1008, 4028, 13352, 16020, 19, 1, 5, 246, 246, 246, 171, resp. 108 triples.

fb summarization failed to complete within 3 hours, on all but the smallest graphs (Springer, Nobel Prizes and BSBM 1M). This is partially due to our simple, single-computer implementation, but computing fb is also intrinsically hard, as it requires many iterations. More efficient methods to build the fb summary are parallel [26]; existing algorithms to build the $\sim_t$ and $\sim_{ioat}$ summaries are based on MapReduce [5]. Here, we study the size, precision, as well as qualitative properties (see Section 2.2) of prior-work summaries, based on simple (if not the most efficient) centralized implementations. The fb results we obtained confirm the observation in Section 2.2 that such summaries are much too complex to be used for a first visualization.

Among the other summaries, *before inlining*, as theoretically expected (Proposition 1), $G_{/W}$ always has the fewest edges; $G_{/S}$ is close. In contrast, summaries that group nodes (also) by the types, such as $G_{/TS}$, $G_{/t}$ and $G_{/ioat}$ are much larger. This is particularly visible for larger BSBM datasets: as the schema complexity grows, these summaries have tens of thousands of edges. $G_{/ioat}$ remains too complex even for the simpler-schema graphs (excluding BSBM), with hundreds or thousands of edges, whereas $G_{/W}$ and $G_{/S}$ only have a few dozen edges. $G_{/1fw}$ is always smaller than $G_{/ioat}$ (this can be shown theoretically based on their definitions), but it remains much larger (by a factor of 2 in the case of Springer, up to 122 in the case of Foodista) than $G_{/S}$. Among the graphs other than BSBM, $G_{/t}$ has less edges than $G_{/S}$ on the DBpedia Person graph, and less nodes on the Springer graph; on all the others, $G_{/t}$ is several times larger than $G_{/S}$. Further, $G_{/t}$ has the qualitative drawback of considering all untyped nodes equivalent; none of our summaries has this problem.

After inlining, $G^{in}_{/W}$ and $G^{in}_{/S}$ are reduced to very few (1 to 21) nodes and 2 to 36 edges. In contrast, $G^{in}_{/t}$ has significantly more edges than $G^{in}_{/S}$ (up to 11× for BSBM10 or DBLP); it is smaller than $G^{in}_{/S}$ (by a small margin) only on DBPedia Person and Nobel. The summaries $G^{in}_{/1fw}$, $G^{in}_{/ioat}$ still remain very large, thus, not useful for first-sight visualization.

Summarization time. The times to build our summaries using the global and the incremental algorithms are plotted as a function of |G| in Figure 17; both axes are in log scale. For each summary type, the summarization time is roughly linear in the size of G, confirming the expectations stated in Section 7. Increm-W is the fastest overall; it traverses G only once, thus it is faster than global-W which performs several passes. S, TW and TS, in this order, are more expensive, and finally incremental S which, as we explained, is quite complex. Since increm-S is rather expensive per-triple, it is more efficient to first summarize a graph using global-S, and call increm-S only to maintain

G	$G_{/W}$	$G^{in}_{/W}$	$G_{/S}$	$G^{in}_{/S}$	$G_{/TS}$	$G^{in}_{/TS}$	$G_{/1fw}$	$G^{in}_{/1fw}$	$G_{/\sim t}$	$G^{in}_{/\sim t}$	$G_{/\sim ioat}$	$G^{in}_{/\sim ioat}$	$G_{/fb}$	$G^{in}_{/fb}$
BSBM1	**179|38**	**1|7**	184|57	5|9	281|1951	7|10	208|1017	48|573	264|2046	102|485	312|4477	61|617	279999|999436	-|-
BSBM10	**613|38**	**1|7**	618|68	5|11	1131|10274	7|10	642|1023	48|579	1114|10785	518|749	719|10320	86|740	-|-	-|-
BSBM100	**2039|38**	**1|7**	2044|70	5|11	3325|25639	7|10	2068|1023	48|579	3308|26918	-|-	2149|34503	105|895	-|-	-|-
BSBM138	**2441|38**	**1|7**	2446|71	5|11	3983|30759	-|-	2470|1023	-|-	3966|32294	-|-	2556|41963	110|892	-|-	-|-
DBLP	**23|26**	**2|8**	30|66	7|17	38|196	14|66	267|4344	62|938	55|237	13|188	439|8390	-|-	-|-	-|-
DB. Person	**4|9**	**1|2**	6|21	2|4	5|11	**1|2**	67|377	65|83	3|11	**1|2**	135|1651	112|91	-|-	-|-
Foodista	**11|13**	**3|6**	12|16	**3|6**	17|33	6|13	72|1959	25|975	12|36	6|33	191|4518	169|934	-|-	-|-
LUBM1	**82|18**	**3|12**	86|43	7|27	95|94	16|58	89|81	14|75	92|94	16|58	103|173	24|111	-|-	-|-
LUBM10	**82|18**	**3|12**	86|44	7|28	95|94	16|58	89|83	14|77	92|94	16|58	101|163	22|100	-|-	-|-
LUBM100	**82|18**	**3|12**	86|44	7|27	95|94	16|58	89|83	14|77	92|94	16|58	101|167	22|104	-|-	-|-
Nobel	**97|59**	**9|18**	110|110	21|36	105|85	16|33	117|555	48|434	83|82	14|65	164|697	70|316	22640|83206	10765|32671
Springer	**49|24**	**4|2**	49|24	**4|2**	49|24	**4|2**	40|51	8|9	37|24	**4|2**	73|90	8|9	65297|139566	15530|30028

TABLE 4. Summary sizes: direct and after inlining.

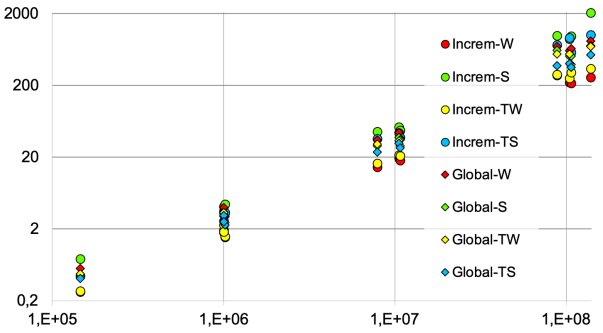


FIGURE 17. Summarization time (s) vs. graph size $|G|$.

Dataset	$G_{/W}$	$G_{/S}$	$G_{/TS}$	$G_{/1fw}$	$G_{/\sim t}$
BSBM1	0.56	0.26	**0.01**	0.46	0.46
BSBM10	0.56	0.45	**0.01**	0.46	0.46
BSBM100	0.56	0.48	**0.01**	0.46	0.46
BSBM138	0.56	0.49	**0.01**	0.46	0.46
DBLP	0.23	0.09	**0.03**	0.32	0.37
DB. Person	0.09	**0.08**	0.09	0.28	0.31
Foodista	0.15	0.15	**0.00**	0.25	0.32
LUBM1	0.50	0.25	**0.00**	0.15	0.03
LUBM10	0.50	0.25	**0.00**	0.15	0.03
LUBM100	0.50	0.25	**0.00**	0.15	0.03
Nobel	0.27	0.04	**0.01**	0.59	0.49
Springer	**0.01**	**0.01**	**0.01**	0.59	0.59

TABLE 6. Precision loss experiments.

Dataset	dt_W (s)	st_W (s)	x_W (%)	dt_S (s)	st_S (s)	x_S (%)
BSBM1M	6.56	2.12	0.68	9.52	4.37	0.54
BSBM10M	73.96	19.34	0.74	123.73	52.84	0.67
BSBM100M	797.50	218.83	0.73	1451.40	884.35	0.39
BSBM138M	1393.69	257.19	0.82	3627.19	2049.22	0.44
LUBM1M	24.53	1.99	0.92	27.47	3.76	0.86
LUBM10M	302.24	18.13	**0.94**	354.81	47.09	**0.87**
LUBM100M	3472.07	214.28	**0.94**	4247.83	974.20	0.77
Nobel	3.32	0.46	0.86	3.55	0.70	0.80
Springer	4.07	0.42	0.90	4.52	0.96	0.79

TABLE 5. Shortcut experiments.

it later. This is significantly faster: for instance, global-S on BSBM138M takes only 11.85 minutes, while increm-S takes 34.5. Increm-TS is often faster than increm-S because typed nodes do not lead to splits during TS summarization.

Shortcut speed-up. Table 5 shows the time to build $(G^\infty)_{/\equiv}$ in two ways: (*i*) *direct*, i.e., saturate G then summarize, denoted $dt_\equiv$, and (*ii*) *shortcut* (Section 5.3), summarize G, then saturate the summary and summarize again, denoted $st_\equiv$. It also shows the **shortcut speed-up** $x_\equiv$ for $\equiv \in \{W, S\}$ defined as $(dt_\equiv - st_\equiv)/dt_\equiv$. The speed-up ranges between 39% and 94% in all cases, a direct consequence of $\equiv_W$ and $\equiv_S$ compression. Indeed, $dt_\equiv$ includes the time to summarize G^∞, while $st_\equiv$ includes the time to summarize $(G_{/\equiv})^\infty$; the smaller this is, the higher $x_\equiv$.

Summary precision. We now attempt to quantify the loss of precision of our structural summaries. A simple measure is the fraction of summary subgraphs having no isomorphic counterpart in the data; intuitively, summary users may believe G exhibits such structures, while this is not true. For instance, node N1 in Figure 6 has, among others, two outgoing edges labeled a and ta, whereas such a node does not exist in the original graph (Figure 2). We define the **precision loss at** l, or PL_l as the fraction of connected l-edges subgraphs of the summary $G_{/\equiv}$ without a counterpart in G. Table 6 shows PL_2 scores for our datasets. By definition, fb, 1fb and ioat have $PL_2 = 0$ for any graph, since they reflect completely node neighborhoods at distance 1. In contrast, $G_{/W}$ has the highest precision loss; $G_{/TS}$ is the most precise, much better than $\sim_t$ which blindly collapses untyped nodes. 1fw is quite imprecise in some cases, since it ignores incoming edges.

Experiment conclusion. Our four summaries can be built efficiently in linear time. They strongly reduce graph sizes and, through inlining, they lead to compact, understandable graphs which fit human comprehension capacity at first sight. If all non-leaf G nodes are typed, $G_{/TS}$ (or, equivalently, $G_{/TW}$), with type generalization, are the most informative and most precise, given that type information specified by humans carries precious information about the graphs. Otherwise, $G_{/S}$ strikes

the best balance between concision and informative content, while $\mathtt{G}_{/\mathtt{W}}$ loses more precision. The shortcut speeds up W and S summarization of $\mathtt{G}^{\infty}$ by 39% to 94%. Thus, *we find the strong summary the safest and most interesting choice: it can be built efficiently and is most likely to lead to informative yet understandable RDF graph summaries.* More generally, the data publisher can build them all (first build $\mathtt{G}_{/\mathtt{S}}$ and $\mathtt{G}_{/\mathtt{TS}}$, then $\mathtt{G}_{/\mathtt{W}}$ from $\mathtt{G}_{/\mathtt{S}}$ and $\mathtt{G}_{/\mathtt{TW}}$ from $\mathtt{G}_{/\mathtt{TS}}$) and select the one(s) to share with potential data users.

9. Related Work and Conclusion

As discussed in the Introduction and in Section 2.2, summarization is a well-studied notion for general graphs [29] and RDF ones in particular [7]. Our work pertains to the family of quotient summaries of RDF graphs, and most directly compares to bisimilarity-based quotients as well as the more RDF-specific ones. In this comparison, our summaries, and among them the strong summary which we find generally the best, has the advantages of being (*i*) compact and easy to understand for domain-specific graphs, (*ii*) efficiently computed in linear-time, and (*iii*) benefitting from the original shortcut procedure we introduced. None of our summaries has the drawback of collapsing all leaves, all roots, or all untyped nodes. In contrast, more complex summaries such as 1fb and $\sim_{ioat}$ are better suited for indexing, but not as a first interface to show users.

Many non-quotient RDF graph summaries exist, see, e.g., [7]. [13] builds answer-preserving summaries for reachability and graph pattern queries. However, these summaries do not preserve query structure (i.e., joins), which quotient summaries do preserve.

Other graph summaries compress graphs with bounded "error" (number of edges to be added as "corrections" after decompression, to retrieve the original graph) [31, 24]. Nodes and edges are summarized according to their frequencies and/or based on ontology patterns in [28, 33]. Summaries where nodes are grouped by graph clustering [20], user-defined aggregation rules [34], mining [9], and identification of frequent subtrees [39] do not reflect the complete structure, and/or require user input. With different objectives, these summaries may omit part of the graph structure, or be much too large for visualization.

Work on fitting XML data into relational stores [12, 4] also aimed at finding "homogeneous" node groups; "inlining" there meant storing in the relation of node n, its properties which occur at most once. Our inlining (Section 6) pushes leaf nodes within their parents, regardless of their number of occurrences.

Quality metrics for RDF graph summaries have been proposed in [40]. Along these metrics, our summaries score high for preserving all classes and properties from G; the price they pay for compactness is to sometimes show instances that do not exist in G (as our precision loss experiments show).

We currently work on using summaries to automatically identify interesting analytical queries in large RDF graphs and to speed up RDF keyword search.

Acknowledgements Šejla Čebirić has contributed to discussions on early versions of this work.

References

[1] Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
[2] Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of RDF data management systems. In: ISWC, pp. 197–212 (2014)
[3] Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. Int. J. Semantic Web Inf. Syst. **5**(2) (2009)
[4] Bohannon, P., Freire, J., Roy, P., Siméon, J.: From XML schema to relations: A cost-based approach to XML storage. In: ICDE (2002)
[5] Campinas, S., Delbru, R., Tummarello, G.: Efficiency and precision trade-offs in graph summary algorithms. In: IDEAS (2013)
[6] Čebirić, Š., Goasdoué, F., Guzewicz, P., Manolescu, I.: Compact Summaries of Rich Heterogeneous Graphs. Research Report RR-8920, INRIA and U. Rennes 1 (2018). URL https://hal.inria.fr/hal-01325900v6. See also previous version (v5)
[7] Cebiric, S., Goasdoué, F., Kondylakis, H., Kotzinos, D., Manolescu, I., Troullinou, G., Zneika, M.: Summarizing Semantic Graphs: A Survey. The VLDB Journal (2018). URL https://hal.inria.fr/hal-01925496
[8] Čebirić, Š., Goasdoué, F., Manolescu, I.: A framework for efficient representative summarization of RDF graphs. In: ISWC (poster) (2017)
[9] Chen, C., Lin, C.X., Fredrikson, M., Christodorescu, M., Yan, X., Han, J.: Mining graph patterns efficiently via randomized summaries. PVLDB **2**(1) (2009)
[10] Chen, Q., Lim, A., Ong, K.W.: $D(K)$-index: An adaptive structural summary for graph-structured data. In: SIGMOD (2003)
[11] Consens, M.P., Miller, R.J., Rizzolo, F., Vaisman, A.A.: Exploring XML web collections with DescribeX. TWEB **4**(3) (2010)
[12] Deutsch, A., Fernández, M.F., Suciu, D.: Storing semistructured data with STORED. In: SIGMOD (1999)
[13] Fan, W., Li, J., Wang, X., Wu, Y.: Query preserving graph compression. In: SIGMOD (2012)
[14] Galil, Z., Italiano, G.F.: Data structures and algorithms for disjoint set union problems. ACM Comput. Surv. **23**(3), 319–344 (1991)
[15] Goasdoué, F., Guzewicz, P., Manolescu, I.: Incremental structural summarization of RDF graphs. In: EDBT. Lisbon, Portugal (2019). URL https://hal.inria.fr/hal-01978784
[16] Goasdoué, F., Manolescu, I., Roatiş, A.: Efficient query answering against dynamic RDF databases. In: EDBT (2013)
[17] Goldman, R., Widom, J.: Dataguides: Enabling query formulation and optimization in semistructured databases. In: VLDB (1997)

[18] Gubichev, A., Neumann, T.: Exploiting the query structure for efficient join ordering in SPARQL queries. In: EDBT, pp. 439–450 (2014)
[19] Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Sem. **3**(2-3) (2005)
[20] Gurajada, S., Seufert, S., Miliaraki, I., Theobald, M.: Using graph summarization for join-ahead pruning in a distributed RDF engine. In: SWIM Workshop (2014)
[21] Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: FOCS (1995)
[22] Kaushik, R., Bohannon, P., Naughton, J.F., Korth, H.F.: Covering indexes for branching path queries. In: SIGMOD (2002)
[23] Kaushik, R., Shenoy, P., Bohannon, P., Gudes, E.: Exploiting local similarity for indexing paths in graph-structured data. In: ICDE (2002)
[24] Khan, K., Nawaz, W., Lee, Y.: Set-based approximate approach for lossless graph summarization. Computing **97**(12), 1185–1207 (2015)
[25] Khatchadourian, S., Consens, M.P.: ExpLOD: Summary-based exploration of interlinking and RDF usage in the linked open data cloud. In: ESWC (2010)
[26] Khatchadourian, S., Consens, M.P.: Constructing bisimulation summaries on a multi-core graph processing framework. In: GRADES Workshop (2015)
[27] Le, W., Li, F., Kementsietsidis, A., Duan, S.: Scalable keyword search on large RDF data. IEEE TKDE **26**(11) (2014)
[28] LeFevre, K., Terzi, E.: GraSS: Graph structure summarization. In: SDM (2010)
[29] Liu, Y., Safavi, T., Dighe, A., Koutra, D.: Graph summarization methods and applications: A survey. ACM Comput. Surv. **51**(3) (2018)
[30] Milo, T., Suciu, D.: Index structures for path expressions. In: ICDT (1999)
[31] Navlakha, S., Rastogi, R., Shrivastava, N.: Graph summarization with bounded error. In: SIGMOD (2008)
[32] Neumann, T., Moerkotte, G.: Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In: ICDE (2011)
[33] Principe, R.A.A., Spahiu, B., Palmonari, M., Rula, A., Paoli, F.D., Maurino, A.: ABSTAT 1.0: Compute, manage and share semantic profiles of RDF knowledge graphs. In: ESWC (2018)
[34] Rudolf, M., Paradies, M., Bornhövd, C., Lehner, W.: SynopSys: large graph analytics in the SAP HANA database through summarization. In: GRADES (2013)
[35] Schätzle, A., Neu, A., Lausen, G., Przyjaciel-Zablocki, M.: Large-scale bisimulation of RDF graphs. In: SWIM Workshop (2013)
[36] Tian, Y., Hankins, R.A., Patel, J.M.: Efficient aggregation for graph summarization. In: SIGMOD. ACM (2008)
[37] Tran, T., Ladwig, G., Rudolph, S.: Managing structured and semistructured RDF data using structure indexes. IEEE TKDE **25**(9) (2013)
[38] W3C: Resource description framework. http://www.w3.org/RDF/
[39] Zhao, P., Yu, J.X., Yu, P.S.: Graph indexing: Tree + delta >= graph. In: VLDB (2007)
[40] Zneika, M., Vodislav, D., Kotzinos, D.: Quality metrics for RDF graph summarization. Semantic Web (2018)

Appendix

We include here the proofs of all the statements made in the paper. The main ones are for the general shortcut Theorems 1 and 2, the W and S shortcuts (Theorems 3 and 4), and the correctness of our incremental algorithms (Propositions 5 and 6). The other serve as ingredientes for these main proofs.

Appendix A. Proof of Proposition 1

Proof. First, note that any two weak summary nodes n_1, n_2 cannot be targets of the same data property. Indeed, if such a data property p existed, let TC be the target clique it belongs to. By the definition of the weak summary, n_1 corresponds to a set of (disjoint) target cliques STC_1, which includes TC, and a set of disjoint source cliques SSC_1. Similarly, n_2 corresponds to a set of (disjoint) target cliques STC_2, which includes TC, and a set of disjoint source cliques SSC_2. The presence of TC in STC_1 and STC_2 contradicts the fact that different equivalence classes of G nodes correspond to disjoint sets of target cliques. The same holds for the sets of properties of which weak summary nodes are sources. Thus, any data property has at most one source and at most one target in $\mathtt{G}_{/\mathtt{W}}$. Further, by the definition of the summary as a quotient, every data property present in G also appears in the summary. Thus, there is exactly one p-labeled edge in $\mathtt{G}_{/\mathtt{W}}$ for every data property in G. □

Appendix B. Proof of Proposition 2

Proof. If two $\mathtt{G}_{/\mathtt{S}}$ distinct nodes had the same source and the same target clique, they would be strongly equivalent. This cannot be the case in a quotient summary obtained through $\equiv_{\mathtt{S}}$, since by definition, such a summary has one node for each $\equiv_{\mathtt{S}}$ equivalence class. Thus, any two distinct $\mathtt{G}_{/\mathtt{S}}$ have distinct source cliques and/or distinct target cliques.

Now, let m be a $\mathtt{G}_{/\mathtt{S}}$ node, and $S_m = f_{\mathtt{S}}^{-1}(m)$ be the set of all G nodes represented by m. By the definition of a quotient summary, m must be the target (resp. the source) of an edge carrying each of the labels on the edges entering (resp. going out of) any node $n \in S_m$. Thus, m is source of all the properties in the source clique shared by the nodes in S_m, and is target of all the properties in the target clique shared by the nodes in S_m. Thus, m has the source and target clique of any node from S_m; this concludes our proof. □

Appendix C. Proof of Theorem 1

Proof. We first show that an homomorphism can be established from the node sets of $\mathtt{G}^{\infty}$ to that of $(\mathtt{G}_{/\equiv})^{\infty}$.

Observe that RDF saturation with RDFS constraints only adds edges between graph nodes, but does not add nodes. Thus, a node n is in $\mathtt{G}^{\infty}$ iff n

is in $\mathtt{G}$. Further, by the definition of our quotient-based summaries, n is in $\mathtt{G}$ iff $f_{\equiv}(n)$ is in $\mathtt{G}_{/\equiv}$. Finally, again by the definition of saturation, $f_{\equiv}(n)$ is in $\mathtt{G}_{/\equiv}$ iff $f_{\equiv}(n)$ is in $(\mathtt{G}_{/\equiv})^{\infty}$.

Therefore, every $\mathtt{G}^{\infty}$ node n maps the $f_{\equiv}(n)$ $(\mathtt{G}_{/\equiv})^{\infty}$ node (*).

Next, we show that there is a one-to-one mapping between $\mathtt{G}^{\infty}$ edges and those of $(\mathtt{G}_{/\equiv})^{\infty}$.

If $n_1\ p\ n_2$ is an edge in $\mathtt{G}^{\infty}$, at least one of the following two situations holds:

- $n_1\ p\ n_2$ is an edge in $\mathtt{G}$. This holds iff $f_{\equiv}(n_1)\ p\ f_{\equiv}(n_2)$ is an edge in $\mathtt{G}_{/\equiv}$, by definition of an RDF summary. Finally, if $f_{\equiv}(n_1)\ p\ f_{\equiv}(n_2)$ is an edge in $\mathtt{G}_{/\equiv}$, then $f_{\equiv}(n_1)\ p\ f_{\equiv}(n_2)$ is also an edge in $(\mathtt{G}_{/\equiv})^{\infty}$.
- $n_1\ p'\ n_2$ is an edge in $\mathtt{G}$, and $p'\ sc\ p$ belongs to schema triples of the saturated graph, thus $n_1\ p\ n_2$ is produced by saturation in $\mathtt{G}^{\infty}$. In this case, we show similarly to the preceding item that $f_{\equiv}(n_1)\ p'\ f_{\equiv}(n_2)$ is an edge in $(\mathtt{G}_{/\equiv})^{\infty}$, hence $f_{\equiv}(n_1)\ p\ f_{\equiv}(n_2)$ is also an edge added to $(\mathtt{G}_{/\equiv})^{\infty}$ by saturation, since $(\mathtt{G}_{/\equiv})^{\infty}$ and $\mathtt{G}^{\infty}$ have the same (saturated) schema triples (Section 5).

If $n_1\ type\ c$ is an edge in $\mathtt{G}^{\infty}$, at least one of the following two situations holds:

- $n_1\ type\ c$ is an edge in $\mathtt{G}$. This holds iff $f_{\equiv}(n_1)\ type\ c$ is an edge in $\mathtt{G}_{/\equiv}$, by definition of an RDF summary (recall that $f_{\equiv}(c) = c$ for classes). Finally, if $f_{\equiv}(n_1)\ type\ c$ is an edge in $\mathtt{G}_{/\equiv}$, then $f_{\equiv}(n_1)\ type\ c$ is also an edge in $(\mathtt{G}_{/\equiv})^{\infty}$.
- $n_1\ p\ n_2$ is an edge in $\mathtt{G}$ and $p\ domain\ c$ (or $p\ range\ c$) belongs to schema triples of the saturated graph, thus $n_1\ type\ c$ is produced by saturation in $\mathtt{G}^{\infty}$. In this case, we show similarly as above that $f_{\equiv}(n_1)\ p\ f_{\equiv}(n_2)$ is an edge in $(\mathtt{G}_{/\equiv})^{\infty}$, hence $f_{\equiv}(n_1)\ type\ c$ is also an edge added to $(\mathtt{G}_{/\equiv})^{\infty}$ by saturation, since $(\mathtt{G}_{/\equiv})^{\infty}$ and $\mathtt{G}^{\infty}$ have the same (saturated) schema triples (Section 5).

Therefore, every $\mathtt{G}^{\infty}$ edge $n_1\ p\ n_2$ (resp. $n_1\ type\ c$) maps into the $(\mathtt{G}_{/\equiv})^{\infty}$ edge $f_{\equiv}(n_1)\ p\ f_{\equiv}(n_2)$ (resp. $f_{\equiv}(n_1)\ type\ c$) (**).

From (*) and (**), it follows that f is an homomorphism from $\mathtt{G}^{\infty}$ to $(\mathtt{G}_{/\equiv})^{\infty}$. □

Appendix D. Proof of Theorem 2

Proof. We start by introducing some **notations** (see Figure 18). Let f_1 be the representation function from $\mathtt{G}^{\infty}$ into $(\mathtt{G}^{\infty})_{/\equiv}$, and f_2 be the representation function from $(\mathtt{G}_{/\equiv})^{\infty}$ into $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$.

Let the function φ be a function from the $(\mathtt{G}^{\infty})_{/\equiv}$ nodes to the $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ nodes defined as: $\varphi(f_1(n)) = f_2(f(n))$ for n any $\mathtt{G}^{\infty}$ node.

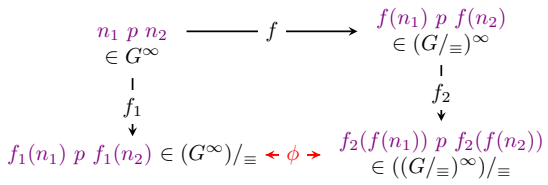


Figure 18. Diagram illustrating Theorem 2.

Suppose that for every pair (n_1, n_2) of $\mathtt{G}$ nodes, $n_1 \equiv n_2$ in $\mathtt{G}^{\infty}$ iff $f(n_1) \equiv f(n_2)$ in $(\mathtt{G}_{/\equiv})^{\infty}$ holds. Let us show that this condition suffices to ensure $(\mathtt{G}^{\infty})_{/\equiv} \equiv ((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ holds, i.e., the φ function defines an isomorphism from $(\mathtt{G}^{\infty})_{/\equiv}$ to $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$.

First, let us show that φ is a bijection from all the $(\mathtt{G}^{\infty})_{/\equiv}$ nodes to all the $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ nodes. Since for every pair n_1, n_2 of $\mathtt{G}^{\infty}$ nodes, $n_1 \equiv n_2$ iff $f(n_1) \equiv f(n_2)$ in $(\mathtt{G}_{/\equiv})^{\infty}$, it follows that $(\mathtt{G}^{\infty})_{/\equiv}$ and $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ have the same number of nodes (*).

Further, a given node n in $(\mathtt{G}^{\infty})_{/\equiv}$ represents a set of equivalent nodes $n_1, \ldots, n_k$ from $\mathtt{G}^{\infty}$. By hypothesis, $n_1 \equiv \cdots \equiv n_k$ in $\mathtt{G}^{\infty}$ iff $f(n_1) \equiv \cdots \equiv f(n_k)$ in $\mathtt{G}^{\infty}_{/\equiv}$ holds. Hence, every node $n = f_1(n_1) = \cdots = f_1(n_k)$ of $(\mathtt{G}^{\infty})_{/\equiv}$ maps to a distinct node $n' = f_2(f(n_1)) = \cdots = f_2(f(n_k))$ in $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ (**).

Similarly, a given node n' in $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ represents a set of equivalent nodes $n'_1 = f(n_1), \ldots, n'_k = f(n_k)$ in $(\mathtt{G}_{/\equiv})^{\infty}$. By hypothesis, $f(n_1) \equiv \cdots \equiv f(n_k)$ in $\mathtt{G}^{\infty}_{/\equiv}$ iff $n_1 \equiv \cdots \equiv n_k$ in $\mathtt{G}^{\infty}$ holds. Hence, every node $n' = f_2(f(n_1)) = \cdots = f_2(f(n_k))$ in $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ maps to a distinct node $n = f_1(n_1) = \cdots = f_1(n_k)$ of $(\mathtt{G}^{\infty})_{/\equiv}$ (***).

From (*), (**) and (***), it follows that φ is a bijective function from all the $(\mathtt{G}^{\infty})_{/\equiv}$ nodes to all the $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ nodes.

Now, let us show that φ defines an isomorphism from $(\mathtt{G}^{\infty})_{/\equiv}$ to $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$.

For every edge $n'_1\ p\ n'_2$ in $(\mathtt{G}^{\infty})_{/\equiv}$, by definition of an RDF summary, there exists an edge $n_1\ p\ n_2$ in $\mathtt{G}^{\infty}$ such that $n'_1\ p\ n'_2 = f_1(n_1)\ p\ f_1(n_2)$. Figure 18 illustrates the discussion. Further, if $n_1\ p\ n_2$ is in $\mathtt{G}^{\infty}$, then $f(n_1)\ p\ f(n_2)$ is in $(\mathtt{G}_{/\equiv})^{\infty}$ (Theorem 1), hence $f_2(f(n_1))\ p\ f_2(f(n_2))$ is in $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$. Therefore,

- since for every $f_1(n_1)\ p\ f_1(n_2)$ edge in $(\mathtt{G}^{\infty})_{/\equiv}$, there is an edge $f_2(f(n_1))\ p\ f_2(f(n_2))$ in $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$, and
- since $\varphi(f_1(n)) = f_2(f(n))$, for n any $\mathtt{G}^{\infty}$ node, is a bijective function from all $(\mathtt{G}^{\infty})_{/\equiv}$ nodes to all $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ nodes,
- it follows that $((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ contains the image of all $(\mathtt{G}^{\infty})_{/\equiv}$ $f_1(n_1)\ p\ f_1(n_2)$ triples through φ (*).

Now, for every edge $n_1''\ p\ n_2''$ in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, by definition of an RDF summary, there exists an edge $n_1'\ p\ n_2'$ in $(\mathtt{G}_{/\equiv})^\infty$ such that $n_1''\ p\ n_2'' = f_2(n_1')\ p\ f_2(n_2')$. Hence, by Theorem 1, there exists an edge $n_1\ p\ n_2$ in $\mathtt{G}^\infty$ such that $n_1'\ p\ n_2' = f(n_1)\ p\ f(n_2)$. Moreover, since $n_1\ p\ n_2$ is in $\mathtt{G}^\infty$, $f_1(n_1)\ p\ f_1(n_2)$ is in $(\mathtt{G}^\infty)_{/\equiv}$. Therefore, since for every $f_2(f(n_1))\ p\ f_2(f(n_2))$ edge in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, there is an edge $f_1(n_1)\ p\ f_1(n_2)$ in $(\mathtt{G}^\infty)_{/\equiv}$, and since $\varphi(f_1(n)) = f_2(f(n))$, for n any $\mathtt{G}^\infty$ node, is a bijective function from all $(\mathtt{G}^\infty)_{/\equiv}$ nodes to all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ nodes, $(\mathtt{G}^\infty)_{/\equiv}$ contains the image of all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ $n_1''\ p\ n_2''$ triples through φ^{-1} (**).

Similarly, for every edge n_1' *type* c in $(\mathtt{G}^\infty)_{/\equiv}$, by definition of an RDF summary, there exists an edge n_1 *type* c in $\mathtt{G}^\infty$ such that n_1' *type* $c = f_1(n_1)$ *type* c. Further, if n_1 *type* c is in $\mathtt{G}^\infty$, then $f(n_1)$ *type* c is in $(\mathtt{G}_{/\equiv})^\infty$ (Theorem 1), hence $f_2(f(n_1))$ *type* c is in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$. Therefore,

- since for every $f_1(n_1)$ *type* c edge in $(\mathtt{G}^\infty)_{/\equiv}$, there is an edge $f_2(f(n_1))$ *type* c in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, and
- since $\varphi(f_1(n)) = f_2(f(n))$, for n any $\mathtt{G}^\infty$ node, is a bijective function from all $(\mathtt{G}^\infty)_{/\equiv}$ nodes to all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ nodes,
- it follows that $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ contains the image of all $(\mathtt{G}^\infty)_{/\equiv}$ $f_1(n_1)$ *type* c triples through φ (*').

Now, for every edge n_1'' *type* c in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, by definition of an RDF summary, there exists an edge n_1' *type* c in $(\mathtt{G}_{/\equiv})^\infty$ such that n_1'' *type* $c = f_2(n_1')$ *type* c. Hence, by Theorem 1, there exists an edge n_1 *type* c in $\mathtt{G}^\infty$ such that n_1' *type* $c = f(n_1)$ *type* c. Moreover, since n_1 *type* c is in $\mathtt{G}^\infty$, $f_1(n_1)$ *type* c is in $(\mathtt{G}^\infty)_{/\equiv}$. Therefore, since for every $f_2(f(n_1))$ *type* c edge in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, there is an edge $f_1(n_1)$ *type* c in $(\mathtt{G}^\infty)_{/\equiv}$, and since $\varphi(f_1(n)) = f_2(f(n))$, for n any $\mathtt{G}^\infty$ node, is a bijective function from all $(\mathtt{G}^\infty)_{/\equiv}$ nodes to all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ nodes, $(\mathtt{G}^\infty)_{/\equiv}$ contains the image of all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ n_1'' *type* c triples through φ^{-1} (**').

From (*) and (**), and, (*') and (**'), it follows that φ defines an isomorphism from $(\mathtt{G}^\infty)_{/\equiv}$ to $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$. □

Appendix E. Saturation and property cliques

The next Lemma describes the relationships between a clique C of $\mathtt{G}$, its saturated version C^+, and the cliques of $\mathtt{G}^\infty$:

Lemma 3. ([)*Saturation vs. property cliques] Let C, C_1, C_2 be non-empty source (or target) cliques of $\mathtt{G}$.*

(1) *There exists exactly one source (resp. target) clique C^∞ of $\mathtt{G}^\infty$ such that $C \subseteq C^\infty$.*
(2) *If $C_1^+ \cap C_2^+ \neq \emptyset$, then all the properties in C_1 and C_2 are in the same $\mathtt{G}^\infty$ clique C^∞.*
(3) *Any non-empty source (or target) clique C^∞ is a union of the form $C_1^+ \cup \cdots \cup C_k^+$ for some $k \geq 1$, where each C_i is a non-empty source (resp. target) clique of $\mathtt{G}$, and for any C_i, C_j where $1 \leq i, j \leq k$ with $i \neq j$, there exist some cliques $D_1 = C_i, \ldots, D_n = C_j$ in the set $\{C_1, \ldots, C_k\}$ such that:*
$$D_1^+ \cap D_2^+ \neq \emptyset,\ \ldots,\ D_{n-1}^+ \cap D_n^+ \neq \emptyset$$
(4) *Let $\mathtt{p}_1, \mathtt{p}_2$ be two data properties in $\mathtt{G}$, whose source (or target) cliques are C_1 and C_2. Properties $\mathtt{p}_1, \mathtt{p}_2$ are in the same source (resp. target) clique C^∞ of $\mathtt{G}^\infty$ if and only if there exist k non-empty source (resp. target) cliques of $\mathtt{G}$, $k \geq 0$, denoted $D_1, \ldots, D_k$ such that:*
$$C_1^+ \cap D_1^+ \neq \emptyset,\ D_1^+ \cap D_2^+ \neq \emptyset, \ldots, D_{k-1}^+ \cap D_k^+ \neq \emptyset, D_k^+ \cap C_2^+ \neq \emptyset.$$

Proof. We prove the lemma only for source cliques; the proof for the target cliques is very similar.

(1) Any resource $r \in \mathtt{G}$ having two data properties also has them in $\mathtt{G}^\infty$; thus, any data properties in the same source clique in $\mathtt{G}$ are also in the same source clique in $\mathtt{G}^\infty$. The unicity of C^∞ is ensured by the fact that the source cliques of $\mathtt{G}^\infty$ are by definition disjoint.
(2) C_1^+ and C_2^+ intersect on property $\mathtt{p}$ iff there exist some $\mathtt{p}_1 \in C_1$ and $\mathtt{p}_2 \in C_2$ which are specializations of the same $\mathtt{p}$ (one, but not both, may also be $\mathtt{p}$ itself). Independently, we know that there exist $r_1, r_2 \in \mathtt{G}$ such that r_1 has $\mathtt{p}_1$ and r_2 has $\mathtt{p}_2$; in $\mathtt{G}^\infty$, r_1 has $\mathtt{p}_1$ and $\mathtt{p}$, thus these two properties are in the same $\mathtt{G}^\infty$ clique. Similarly, r_2 has $\mathtt{p}_2$ and $\mathtt{p}$, which ensures that $\mathtt{p}$ is also in the same $\mathtt{G}^\infty$ source clique.
(3) Let $\{\mathtt{p}_1, \ldots, \mathtt{p}_k\}$ be the data properties that appear both in $\mathtt{G}$ and in C^∞; it follows from the saturation rules and the definition of cliques, that $k > 0$. For $1 \leq i \leq k$, let C_i be the $\mathtt{G}$ source clique comprising $\mathtt{p}_i$. Applying lemma point 1., $C_i \subseteq C^\infty$ for each $1 \leq i \leq k$. Further, it is easy to see that $C_i^+ \subseteq C^\infty$, since any property that saturation adds to C_i^+ is also added by saturation to C^∞. Thus, $\bigcup_{1 \leq i \leq k} C_i^+ \subseteq C^\infty$.

Let us now show that $C^\infty \subseteq \bigcup_{1 \leq i \leq k} C_i^+$. Let $\mathtt{p} \in C^\infty$ be a data property, then there exists a resource r having $\mathtt{p}$ in $\mathtt{G}^\infty$. Then, in $\mathtt{G}$, r has a property $\mathtt{p}'$ which is either $\mathtt{p}$, or is such that $\mathtt{p}'$ *sp* $\mathtt{p}$ in $\mathtt{G}^\infty$. Then, in $\mathtt{G}^\infty$, r has both $\mathtt{p}$ and $\mathtt{p}'$, which entails that $\mathtt{p}' \in C^\infty$. Therefore, $\mathtt{p}'$ is a data property occurring both in C^∞ and in $\mathtt{G}$, therefore $\mathtt{p}'$ is one of

the properties $\mathtt{p}_i$, for some $1 \leq i \leq k$, that is, $\mathtt{p}' \in C_i$, and accordingly, $\mathtt{p} \in C_i^+$ due to $\mathtt{p}'$ *sp* $\mathtt{p}$.

Thus, any data property $\mathtt{p} \in C^\infty$ is part of some C_i^+.

We must still show that the saturated cliques intersect. If $k = 1$ the statement is trivially true. Suppose $k \geq 2$ and the statement is false. Let $\mathcal{C}$ denote the set $\{C_1, \ldots, C_m\}$; the cliques in $\mathcal{C}$ are pairwise disjoint by definition. Let $\mathcal{I} \subseteq \mathcal{C}$ be a *maximal* subset of $\mathcal{C}$ cliques such that the saturations of $\mathcal{I}$ cliques all intersect (directly or indirectly). Let $\mathcal{J} = \mathcal{C} \setminus \mathcal{I}$ be the complement of $\mathcal{I}$; if the last part of 4. is false, $\mathcal{J}$ is not empty. We denote $\mathcal{I}^+$, respectively $\mathcal{J}^+$, the set of the saturated cliques from $\mathcal{I}$, resp. $\mathcal{J}$.

No data property $\mathtt{p}_i$ from $\mathcal{I}^+$ can be source-related in $\mathtt{G}^\infty$ to any data property $\mathtt{p}_j$ from $\mathcal{J}^+$. This is because source-relatedness requires a resource r having in $\mathtt{G}^\infty$ both $\mathtt{p}_i$ and a property $\mathtt{p}$ source-related to $\mathtt{p}_j$. If such a property $\mathtt{p}$ existed, it would belong both to $\mathcal{I}^+$ (since $\mathtt{p}$ has a common source with $\mathtt{p}_i$) and to $\mathcal{J}^+$ (since $\mathtt{p}$ is source-related to $\mathtt{p}_j$); or, $\mathcal{I}^+$ and $\mathcal{J}^+$ have no property in common.

The lack of source-relatedness in $\mathtt{G}^\infty$ between $\mathtt{p}_i$ and $\mathtt{p}_j$ chosen as above contradicts the hypothesis that they are part of the same source clique of $\mathtt{G}^\infty$, namely C^∞.

(4) The statement follows quite directly as a consequence of the previous one, concluding our proof.

□

Appendix F. Proof of Lemma 1

Proof. We prove the lemma for target-related properties.

"Only if": If data properties are target-related in $(\mathtt{G}_{/\mathtt{W}})^\infty$, then they belong to the same target clique $TC_{\mathtt{W}}^\infty$ in $(\mathtt{G}_{/\mathtt{W}})^\infty$.

By Lemma 3, point 3, it follows that $TC_{\mathtt{W}}^\infty$ is the union of the saturations of a set of $\mathtt{G}_{/\mathtt{W}}$ cliques $(TC_{\mathtt{W}}^1)^+$, $(TC_{\mathtt{W}}^2)^+$, …, $(TC_{\mathtt{W}}^m)^+$. Then:

- For every $1 \leq j \leq m$:
  - $TC_{\mathtt{W}}^j$ is the target clique of a $\mathtt{G}_{/\mathtt{W}}$ node n^j;
  - n^j represents a set of weakly-equivalent $\mathtt{G}$ resources, which are targets only of properties in $TC_{\mathtt{W}}^j$. Thus, the properties in $TC_{\mathtt{W}}^j$ are target-related in $\mathtt{G}$.
  - Thus, in $\mathtt{G}^\infty$, also, the properties in $TC_{\mathtt{W}}^j$ are target-related.
  - From this and the definition of a saturated graph and of a saturated target clique, it follows that the properties from $(TC_{\mathtt{W}}^j)^+$ are target-related in $\mathtt{G}^\infty$.
- Further, still by Lemma 3, point 4, each $(TC_{\mathtt{W}}^j)^+$ intersects at least another $(TC_{\mathtt{W}}^l)^+$ for $1 \leq l \neq j < m$, thus the target properties in all the $(TC_{\mathtt{W}}^j)^+$ for $1 \leq j \leq m$, and in particular $\mathtt{p}$, are target-related to each other in $\mathtt{G}^\infty$. Thus, $\mathtt{p}$ is target-related in $\mathtt{G}^\infty$ to all properties from $TC_{\mathtt{W}}^\infty$.

"If": if data properties are target-related in $\mathtt{G}^\infty$, then they belong to the same target clique TC^∞ in $\mathtt{G}^\infty$. Let $n_1, \ldots, n_k$ be the set of all $\mathtt{G}$ resources which are values of some properties in TC^∞. By definition of an RDF summary and Theorem 1, each summary representative $f(n_i)$ of n_i, for $1 \leq i \leq k$, is at least the object of the same properties as n_i, hence all the properties of TC^∞ in $\mathtt{G}^\infty$ are target-related in $(\mathtt{G}_{/\mathtt{W}})^\infty$. □

Appendix G. Proof of Proposition 3

Proof. Recall from Lemma 3 that:

$$SC_{\mathtt{W}}^\infty = (SC_{\mathtt{W}}^1)^+ \cup (SC_{\mathtt{W}}^2)^+ \cup \ldots \cup (SC_{\mathtt{W}}^m)^+ \text{ and}$$
$$TC_{\mathtt{W}}^\infty = (TC_{\mathtt{W}}^1)^+ \cup (TC_{\mathtt{W}}^2)^+ \cup \ldots \cup (TC_{\mathtt{W}}^n)^+$$

for some $\mathtt{G}_{/\mathtt{W}}$ source cliques $SC_{\mathtt{W}}^1, \ldots, SC_{\mathtt{W}}^m$ and target cliques $TC_{\mathtt{W}}^1, \ldots, TC_{\mathtt{W}}^n$.

Lemma 1 ensures that the data properties in $(SC_{\mathtt{W}}^1)^+ \cup \ldots \cup (SC_{\mathtt{W}}^m)^+$ are related in $\mathtt{G}^\infty$, and those of $(TC_{\mathtt{W}}^1)^+ \cup \ldots \cup (TC_{\mathtt{W}}^n)^+$ are related in $\mathtt{G}^\infty$.

Moreover, $n_{\mathtt{W}}$ was created in $\mathtt{G}_{/\mathtt{W}}$ from a set of weakly-equivalent $\mathtt{G}$ nodes having as source clique one among $SC_{\mathtt{W}}^1, \ldots, SC_{\mathtt{W}}^m$ and as target clique one among $TC_{\mathtt{W}}^1, \ldots, TC_{\mathtt{W}}^n$. In $\mathtt{G}^\infty$, these nodes connect the data properties of $SC_{\mathtt{W}}^\infty$ with those of $TC_{\mathtt{W}}^\infty$. □

Appendix H. Proof of Theorem 3

Proof. We show that $\mathtt{W}$ summaries enjoy the sufficient condition for completenes stated in Theorem 2, i.e., given two nodes n_1, n_2 in $\mathtt{G}^\infty$, f the representation homomorphism corresponding to the weak-equivalence relation $\equiv_{\mathtt{W}}$, and $f(n_1)$, $f(n_2)$ the images of n_1, n_2 in $(\mathtt{G}_{/\mathtt{W}})^\infty$ through f (recall Theorem 1), it holds that: $n_1 \equiv_{\mathtt{W}} n_2$ in $\mathtt{G}^\infty$ iff $f(n_1) \equiv_{\mathtt{W}} f(n_2)$ in $(\mathtt{G}_{/\mathtt{W}})^\infty$.

"Only if": $n_1 \equiv_{\mathtt{W}} n_2$ in $\mathtt{G}^\infty$ iff they are connected by an alternating chain of source and target cliques of $\mathtt{G}^\infty$ (as shown in Figure 19); to reuse that figure for the current proof, let us use n_{2k} to denote the n_2 of the current lemma statement. Note that $\mathtt{G}^\infty$ only adds triples not nodes, thus all the nodes shown in the figure also exist in $\mathtt{G}$. Now, let us consider the $\mathtt{G}_{/\mathtt{W}}$ nodes $f(n_1), f(n_2), \ldots, f(n_{2k})$ obtained by applying the representation function f on $n_1, \ldots, n_{2k}$.

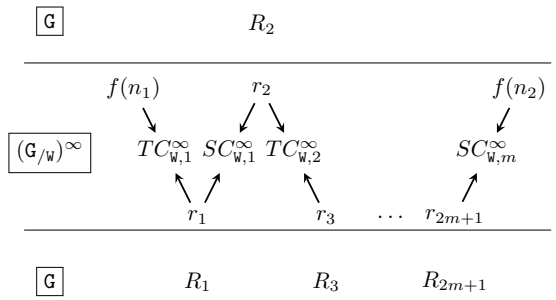


FIGURE 19. Sketch for the sufficient condition for the weak summary to enjoy the shortcut property.

By Theorem 1, f is also a homomorphism from $\mathtt{G}^\infty$ to $(\mathtt{G}_{/\mathtt{W}})^\infty$, therefore any incoming (outgoing) edge into (from) a node n_j of $\mathtt{G}^\infty$ is also incoming (resp. outgoing) into (from) the respective node $f(n_j)$ of $(\mathtt{G}_{/\mathtt{W}})^\infty$. As a consequence, we can reproduce the alternating clique structure into $(\mathtt{G}_{/\mathtt{W}})^\infty$, which suffices to make $f(n_1)$ and $f(n_{2k})$ weakly equivalent in $(\mathtt{G}_{/\mathtt{W}})^\infty$.

"If": $f(n_1) \equiv_{\mathtt{W}} f(n_2)$ in $(\mathtt{G}_{/\mathtt{W}})^\infty$ iff they are connected by an alternating chain of source and target cliques in $(\mathtt{G}_{/\mathtt{W}})^\infty$. Assume w.l.o.g. that the chain is as shown in Figure 19, that is, of the form:

- $f(n_1)$ shares a target clique $TC^\infty_{\mathtt{W},1}$ with r_1
- $r_1(TC^\infty_{\mathtt{W},1}, SC^\infty_{\mathtt{W},1})$
- $r_2(TC^\infty_{\mathtt{W},2}, SC^\infty_{\mathtt{W},1})$, ...,
- $r_{2m+1}(TC^\infty_{\mathtt{W},m-1}, SC^\infty_{\mathtt{W},m})$, and $f(n_2)$ has the source clique $SC^\infty_{\mathtt{W},m}$

The alternating chain starts with a target clique and ends with a source clique (of course three other combinations are possible). In the chain, each resource is either:

- $r_{2i+1}(TC^\infty_{\mathtt{W},i+1}, SC^\infty_{\mathtt{W},i+1})$ or
- $r_{2i+2}(TC^\infty_{\mathtt{W},i+2}, SC^\infty_{\mathtt{W},i+1})$

for some $0 \leq i < m$. Every r_{2i+1} and r_{2i+2} resource is a node from $(\mathtt{G}_{/\mathtt{w}})^\infty$, thus a node from $\mathtt{G}_{/\mathtt{W}}$ (because saturating $\mathtt{G}_{/\mathtt{W}}$ does not create nodes). For a given r_j, let R_j be the set of weakly-equivalent $\mathtt{G}$ resources from which r_j was created; all resources in R_j are by definition weakly equivalent in $\mathtt{G}$, and this also holds in $\mathtt{G}^\infty$.

By Proposition 3, $TC^\infty_{\mathtt{W},1}$ is also a target clique of $\mathtt{G}^\infty$, and it must be the target clique of n_1 in $\mathtt{G}^\infty$ (because of the f homomorphism from $\mathtt{G}^\infty$ into $(\mathtt{G}_{/\mathtt{W}})^\infty$ ensured by Theorem 1). Similarly, $SC^\infty_{\mathtt{W},m}$ must be a source clique in $\mathtt{G}^\infty$ and in particular the source clique of n_2.

In $\mathtt{G}^\infty$, n_1 shares its target clique $TC^\infty_{\mathtt{W},1}$ with the nodes in R_1, thus n_1 is weakly-equivalent to any node from R_1.

Further, by Proposition 3, if the node r_1 has the target clique $TC^\infty_{\mathtt{W},1}$ and the source clique $SC^\infty_{\mathtt{W},1}$ in $(\mathtt{G}_{/\mathtt{W}})^\infty$, then the $\mathtt{G}^\infty$ node whose target clique is $TC^\infty_{\mathtt{W},1}$ must also have the source clique $SC^\infty_{\mathtt{W},1}$ in $\mathtt{G}^\infty$. (Proposition 3 also ensures that a node in $\mathtt{G}^\infty$ has $SC^\infty_{\mathtt{W},1}$ as its source clique.)

If the alternating chain is long enough to comprise r_2 (that is: if the chain does not degenerate in a single node), that corresponds to the set R_2 of $\mathtt{G}$ nodes which, in $\mathtt{G}^\infty$, have the source clique $SC^\infty_{\mathtt{W},1}$, therefore they are weakly equivalent to all nodes from R_1 which have the same source clique. Thus, n_1 is weakly equivalent in $\mathtt{G}^\infty$ to the nodes from R_1 and R_2.

The above reasoning can be applied on each edge in the alternating chain, extending weak equivalence from n_1 through all the R_j sets until n_2. □

APPENDIX I. PROOF OF LEMMA 2

Proof. "If": if data properties are target-related in $\mathtt{G}^\infty$, then they belong to the same target clique TC^∞ in $\mathtt{G}^\infty$. Let $n_1, \ldots, n_k$ be the set of all $\mathtt{G}$ resources which are values of some properties in TC^∞. By definition of an RDF summary and Theorem 1, each image $f_{\mathtt{S}}(n_i)$ of n_i, for $1 \leq i \leq k$ is at least the object of the same properties as n_i, hence all the properties of TC^∞ in $\mathtt{G}^\infty$ are target-related in $(\mathtt{G}_{/\mathtt{S}})^\infty$.

"Only If": if two data properties p_1 and p_2 are target-related in $(\mathtt{G}_{/\mathtt{S}})^\infty$, then they belong to the same target clique $TC^{\mathtt{S},\infty}$, in which they are at distance $n \geq 0$, i.e., they are target-related because of a set $\bigcup_{i=0}^{n}\{r_{i+1}\}$ of nodes which all have the target clique $TC^{\mathtt{S},\infty}$. In $\mathtt{G}_{/\mathtt{S}}$, each such r_{i+1} has a target clique $TC^{\mathtt{S}}_i \subseteq TC^{\mathtt{S},\infty}$, moreover each r_{i+1} results from a set of $\mathtt{G}$ nodes $n^j_{i+1}, j \geq 1$, which by definition of a strong RDF summary, have all the source clique $TC^{\mathtt{S}}_i$. Hence, every such n^j_{i+1} node has target clique $TC^{\mathtt{S},\infty}$ in $\mathtt{G}^\infty$ (since $\mathtt{G}$ and $\mathtt{G}_{/\mathtt{S}}$ have the same schema), in which p_1 and p_2 are target related. □

APPENDIX J. PROOF OF PROPOSITION 4

Proof. Recall from Lemma 3 that:

$$SC^\infty_{\mathtt{S}} = (SC^1_{\mathtt{S}})^+ \cup (SC^2_{\mathtt{S}})^+ \cup \ldots \cup (SC^m_{\mathtt{S}})^+ \text{ and}$$
$$TC^\infty_{\mathtt{S}} = (TC^1_{\mathtt{S}})^+ \cup (TC^2_{\mathtt{S}})^+ \cup \ldots \cup (TC^n_{\mathtt{S}})^+$$

for some $\mathtt{S}_{\mathtt{G}}$ source cliques $SC^1_{\mathtt{S}}, \ldots, SC^m_{\mathtt{S}}$ and target cliques $TC^1_{\mathtt{S}}, \ldots, TC^n_{\mathtt{S}}$.

Lemma 2 ensures that the data properties in $(SC^1_{\mathtt{S}})^+ \cup \ldots \cup (SC^m_{\mathtt{S}})^+$ are related in $\mathtt{G}^\infty$, and those of $(TC^1_{\mathtt{S}})^+ \cup \ldots \cup (TC^n_{\mathtt{S}})^+$ are related in $\mathtt{G}^\infty$.

Moreover, $n_{\mathtt{S}}$ was created in $\mathtt{S}_{\mathtt{G}}$ from a set of strongly-equivalent $\mathtt{G}$ nodes all sharing a source clique $SC^i_{\mathtt{S}}$, for $1 \leq i \leq m$, and all sharing a target clique $TC^j_{\mathtt{S}}$, for some $1 \leq j \leq n$. Thus, in $\mathtt{G}^\infty$, these nodes connect the data properties of $SC^\infty_{\mathtt{S}}$ with those of $TC^\infty_{\mathtt{S}}$. □

Appendix K. Proof of Theorem 4

Proof. "Only if" follows directly from Theorem 1.

To prove "if", note that $f(n_1) \equiv_{\mathtt{S}} f(n_2)$ in $(\mathtt{S}_\mathtt{G})^\infty$ iff they have the same source clique $SC_\mathtt{S}^\infty$ and the same target clique $TC_\mathtt{S}^\infty$ in $(\mathtt{S}_\mathtt{G})^\infty$. By Proposition 4, $TC_\mathtt{S}^\infty$ is also a target clique of $\mathtt{G}^\infty$, and it must be the target clique of n_1 and n_2 in $\mathtt{G}^\infty$ (because of the f homomorphism from $\mathtt{G}^\infty$ into $(\mathtt{S}_\mathtt{G})^\infty$ ensured by Theorem 1). Similarly, $SC_\mathtt{S}^\infty$ is a source clique in $\mathtt{G}^\infty$, and in particular the source clique of n_1 and n_2.

Thus, $n_1 \equiv_{\mathtt{S}} n_2$ in $\mathtt{G}^\infty$. □

Appendix L. Proof of Theorem 7

Proof. We first prove the claim for $\equiv_{\mathtt{fw}}$.

We show this result using the sufficient condition stated in Theorem 2. That is, $n_1 \equiv_{\mathtt{fw}} n_2$ in $\mathtt{G}^\infty$ holds iff $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ in $(\mathtt{G}_{/\mathtt{fw}})^\infty$ holds.

This holds for class nodes and for property nodes since, by definition, they are only equivalent to themselves through some RDF node equivalence relation.

Now, consider two data nodes n_1, n_2 in $\mathtt{G}^\infty$ such that $n_1 \equiv_{\mathtt{fw}} n_2$ in $\mathtt{G}^\infty$, and let us show that $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ in $(\mathtt{G}_{/\mathtt{fw}})^\infty$.

If $n_1 \equiv_{\mathtt{fw}} n_2$ holds in $\mathtt{G}^\infty$, then for every triple $n_1\ \mathtt{p}\ m_1$ there exists a triple $n_2\ \mathtt{p}\ m_2$ such that $m_1 \equiv_{\mathtt{fw}} m_2$ holds, and conversely for every triple $n_2\ \mathtt{p}\ m_2$ there exists a triple $n_1\ \mathtt{p}\ m_1$ such that $m_1 \equiv_{\mathtt{fw}} m_2$ holds.

Let $\mathcal{P}^\infty_{n_1,n_2 \to m_1,m_2}$ be the set of outgoing properties from n_1 to m_1 and from n_2 to m_2 in $\mathtt{G}^\infty$.

In $\mathtt{G}$, the set of outgoing properties from n_1 to m_1, denoted $\mathcal{P}_{n_1 \to m_1}$ is a subset of $\mathcal{P}^\infty_{n_1,n_2 \to m_1,m_2}$, since by definition the saturation of a graph only adds edges; similarly, in $\mathtt{G}$, the set of outgoing properties from n_2 to m_2, denoted $\mathcal{P}_{n_2 \to m_2}$ is a subset of $\mathcal{P}^\infty_{n_1,n_2 \to m_1,m_2}$, which may be different from $\mathcal{P}_{n_1 \to m_1}$.

By definition of a $\equiv_{\mathtt{fw}}$-summary, the set of outgoing properties from $f(n_1)$ to $f(m_1)$ in $\mathtt{G}_{/\mathtt{fw}}$ is exactly $\mathcal{P}_{n_1 \to m_1}$ and similarly the set of outgoing properties from $f(n_2)$ to $f(m_2)$ in $\mathtt{G}_{/\mathtt{fw}}$ is exactly $\mathcal{P}_{n_2 \to m_2}$.

Since $\mathtt{G}$ and $\mathtt{G}_{/\mathtt{fw}}$ have the same schema (Section 5), it follows that in $(\mathtt{G}_{/\mathtt{fw}})^\infty$, the set of outgoing properties from $f(n_1)$ to $f(m_1)$, and from $f(n_2)$ to $f(m_2)$, is exactly $\mathcal{P}^\infty_{n_1,n_2 \to m_1,m_2}$ (data edges can only be added through subProperty constraints).

Since the above holds for any pair of data nodes n_1, n_2 such that $n_1 \equiv_{\mathtt{fw}} n_2$ in $\mathtt{G}^\infty$, and for any of their $\mathtt{G}^\infty$ outgoing edges $n_1\ \mathtt{p}\ m_1$ and $n_2\ \mathtt{p}\ m_2$, hence $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ in $(\mathtt{G}_{/\mathtt{fw}})^\infty$ holds.

Now, consider two data nodes $f(n_1), f(n_2)$ in $(\mathtt{G}_{/\mathtt{fw}})^\infty$ such that $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ in $(\mathtt{G}_{/\mathtt{fw}})^\infty$ and let us show that $n_1 \equiv_{\mathtt{fw}} n_2$ holds in $\mathtt{G}^\infty$.

If $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ holds in $(\mathtt{G}_{/\mathtt{fw}})^\infty$, then for every triple $f(n_1)\ \mathtt{p}\ f(m_1)$ there exists a triple $f(n_2)\ \mathtt{p}\ f(m_2)$ such that $f(m_1) \equiv_{\mathtt{fw}} f(m_2)$ holds, and conversely for every triple $f(n_2)\ \mathtt{p}\ f(m_2)$ there exists a triple $f(n_1)\ \mathtt{p}\ f(m_1)$ such that $f(m_1) \equiv_{\mathtt{fw}} f(m_2)$ holds.

Let $\mathcal{P}^\infty_{f(n_1),f(n_2) \to f(m_1),f(m_2)}$ be the set of outgoing properties from $f(n_1)$ to $f(m_1)$ and from $f(n_2)$ to $f(m_2)$ in $(\mathtt{G}_{/\mathtt{fw}})^\infty$.

In $\mathtt{G}_{/\mathtt{fw}}$, the set of outgoing properties from $f(n_1)$ to $f(m_1)$, denoted $\mathcal{P}_{f(n_1) \to f(m_1)}$ is a subset of $\mathcal{P}^\infty_{fn_1),f(n_2) \to f(m_1),f(m_2)}$, since by definition the saturation of a graph only adds edges; similarly, in $\mathtt{G}_{/\mathtt{fw}}$, the set of outgoing properties from $f(n_2)$ to $f(m_2)$, denoted $\mathcal{P}_{f(n_2) \to f(m_2)}$ is a subset of $\mathcal{P}^\infty_{f(n_1),f(n_2) \to f(m_1),f(m_2)}$, which may be different from $\mathcal{P}_{f(n_1) \to f(m_1)}$.

By definition of a $\equiv_{\mathtt{fw}}$-summary, the set of outgoing properties from n_1 to m_1 in $\mathtt{G}$ is exactly $\mathcal{P}_{f(n_1) \to f(m_1)}$ and similarly the set of outgoing properties from n_2 to m_2 in $\mathtt{G}$ is exactly $\mathcal{P}_{f(n_2) \to f(m_2)}$.

Since $\mathtt{G}$ and $\mathtt{G}_{/\mathtt{fw}}$ have the same schema (Section 5), it follows that in $\mathtt{G}^\infty$, the set of outgoing properties from n_1 to m_1, and from n_2 to m_2, is exactly $\mathcal{P}^\infty_{f(n_1),f(n_2) \to f(m_1),f(m_2)}$ (data edges can only be added through subProperty constraints).

Since the above holds for any pair of data nodes $f(n_1), f(n_2)$ such that $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ in $(\mathtt{G}_{/\mathtt{fw}})^\infty$, and for any of their $(\mathtt{G}_{/\mathtt{fw}})^\infty$ outgoing edges $f(n_1)\ \mathtt{p}\ f(m_1)$ and $f(n_2)\ \mathtt{p}\ f(m_2)$, hence $n_1 \equiv_{\mathtt{fw}} n_2$ in $\mathtt{G}^\infty$ holds.

The proof for $\equiv_{\mathtt{bw}}$ directly derives from the above one by considering incoming edges instead of outgoing ones; the proof for $\equiv_{\mathtt{fb}}$ then derives from those of $\equiv_{\mathtt{fw}}$ and $\equiv_{\mathtt{bw}}$ by considering both incoming and outgoing edges. □

Appendix M. Proof of Proposition 5

Proof. All our algorithms (global or incremental) start by identifying the class and property nodes: this is done retrieving all the subjects and objects from schema triples, and also all the objects of type triples. As previously stated, triple stores routinely support such retrieval efficiently. Our algorithms start by representing these special schema nodes exactly by themselves, and copying in the summary all the schema triples. This exploits the observation made in Section 5 ($\mathtt{G}$ and $\mathtt{G}_{/\equiv}$ have the same schema triples).

Below, we show the correctness of **incremental W and S summarization** on data triples. The proof of Proposition 6 (below) extends this also to type triples.

The correctness of **incremental W summarization** on data triples follows from the fact that Algorithm **increm-W** preserves a set of invariants. Let $\mathtt{G}_k$ be the first k triples of $\mathtt{G}$, in the order in which they are traversed by the algorithm. For any $1 \leq k \leq |\mathtt{G}|$,

after applying **increm-W** on k data triples, the following invariants are preserved:

(1) The source and target $src_{\mathtt{p}}$ and $trg_{\mathtt{p}}$ of any property p present in these k triples are known.
(2) For any summarized triple s p o, we have $f_{\mathtt{W}}(\mathtt{s}) = src_{\mathtt{p}}$ and $f_{\mathtt{W}}(\mathtt{o}) = trg_{\mathtt{p}}$; further, the summary contains the edge $f_{\mathtt{W}}(\mathtt{s})$ p $f_{\mathtt{W}}(\mathtt{o})$.

The preservation of these invariants is shown by considering all the cases which may occur for a given summarized triple s p o: the subject s may have *already been seen* (in which case this triple may lead to a fusion), *or not* (in which case we create the new representative of s), and similarly for o. For p there are also two cases (depending on whether we had already encountered it or not, we may create $src_{\mathtt{p}}$ and $trg_{\mathtt{p}}$, or just fuse them with pre-existing representatives of s and o). There are 8 cases overall. The replacements and fusions detailed in Algorithm 3 guarantee these invariants.

While, for simplicity of presentation, Algorithm **increm-W** considers the possible fusions due to s and o separately, in reality, given that they may impact the same node(s) (e.g., if $f_{\mathtt{W}}(\mathtt{s}) = f_{\mathtt{W}}(\mathtt{o})$), all the replacements are first computed, then reconciled into a *list of summary node substitutions*, applied in all the data structures. For instance, suppose we need to replace summary node 3 with 1 because of a fusion on the subject side, and also summary node 5 with 3 because of a fusion on the object side. In this case, the algorithm will replace 5 and 3 directly with 1. If the replacements were applied sequentially, e.g., first 3 with 1, the second replacement would leave 3 (not 1) instead of 5, which would be an error.

Similarly, the correctness of **incremental S summarization** on data triples follows from the fact that Algorithm **increm-S** preserves the following invariants after having been called on k successive data triples, with $1 \leq k \leq |\mathtt{G}|$:

(1) The source and target clique $sc(\mathtt{p})$ and $tc(\mathtt{p})$ of any property p present in these k triples are known, and they contain p.
(2) For any summarized triple s p o, we have $f_{\mathtt{S}}(\mathtt{s}) = sc(\mathtt{p})$ and $f_{\mathtt{S}}(\mathtt{o}) = tc(\mathtt{p})$; further, the summary contains the edge $f_{\mathtt{S}}(\mathtt{s})$ p $f_{\mathtt{S}}(\mathtt{o})$.
(3) For any source clique sc and target clique tc of a node n appearing in the summarized triple, the summary contains exactly one node.
(4) For any summary node m, the count $m_\sharp$ is exactly the cardinality of the set $\{n \in \mathtt{G} \mid f_{\mathtt{S}}(n) = m\}$.
(5) For any summary edge $m \xrightarrow{\mathtt{p}} m'$, the count $e_\sharp$ is exactly the cardinality of the set $\{n \xrightarrow{\mathtt{p}} n'$ edge of $\mathtt{G} \parallel f_{\mathtt{S}}(n) = m$ and $f_{\mathtt{S}}(n') = m'\}$.
(6) For any (subject, property) combination occurring in the summarized triples, the count $(\mathtt{sp})_\sharp$ is exactly the number of times this occurred in the triples. Similarly, fo r any (property, object) combination appearing in the summarized triples, the count $(\mathtt{po})_\sharp$ is exactly the number of times it appeared.

Like for **increm-W**, there are eight cases depending on whether s, p and o have been previously seen. Further, in the four cases where s has been seen, we may need to split s's representative, or not, and similarly for o; thus, the six cases original cases where at least one of them had been seen lead to 12 cases (to which we add the remaining two, where neither s nor o had been seen), for a total of 14 cases.

Items 4, 5 and 6 are ensured during: the addition of an edge to the summary (this sets $e_\sharp$ to 1 or increases it); the assignment of representatives to nodes (this sets $m_\sharp$ to 1 or increments it); the edge repartition during split (this substracts from one edges $e_\sharp$ exactly the count that it adds to another new edge); and node replacements (which, when replacing u with v, either carry $u_\sharp$ into $v_\sharp$, if v did not exist in the summary previously, or add $u_\sharp$ to $v_\sharp$ if it did). Together, 4, 5 and 6 ensure the correctness of the **split** algorithm (explained in Section 7.1).

The previous items are ensured by the creation of summary nodes (at most one exists at any time for a given source and target clique), fusing cliques (this guarantees each property is in the right clique, and remove cliques input to the fusion), and replacing / fusing summary nodes, as well as from the correctness of the split procedure. □

Appendix N. Proof of Proposition 6

Proof. First, recall that TW and TS summarization start with the type triples, which means all type nodes are detected and represented according to their class sets, before the data triples are summarized. This entails that among the cases which occur for W and S summarization (8, respectively, 14, see discussion in the proof of Proposition 5), those in which the subject, respectively, the object was already represented are further divided in two, depending on whether the subject, respectively, object was a typed node.

This shows that *incremental* TW *summarization handles a superset of the cases handled by the* W *one*, and similarly for TS and TS. Thus, **increm-TW**, respectively, **increm-TS** preserve all the invariants of **increm-W**, respectively, **increm-S**[9], with some additions, which we highlight in italics below.

Additions of TW summarization w.r.t. W:

[9]Note that in the particular case of triples connecting untyped nodes, the algorithms coincide.

(1) The source and target $src_{\mathtt{p}}$ and $trg_{\mathtt{p}}$ of any property p present *with an untyped source, respectively, an untyped target* in the summarized triples are known.
(2) For any summarized triple s p o, we have $f_{\mathtt{W}}(\mathtt{s}) = src_{\mathtt{p}}$ *if* s *is untyped* and $f_{\mathtt{W}}(\mathtt{o}) = trg_{\mathtt{p}}$ *if* o *is untyped*; further, the summary contains the edge $f_{\mathtt{W}}(\mathtt{s})$ p $f_{\mathtt{W}}(\mathtt{o})$.

Additions of TS summarization w.r.t. S:

(1) The source and target clique $sc(\mathtt{p})$ and $tc(\mathtt{p})$ of any property p present in these k triples *with an untyped source, respectively, with an untyped target* are known, and they contain p.
(2) For any summarized triple s p o, we have $f_{\mathtt{S}}(\mathtt{s}) = sc(\mathtt{p})$ *if* s *is untyped*, and $f_{\mathtt{S}}(\mathtt{o}) = tc(\mathtt{p})$ *if* o *is untyped*; further, the summary contains the edge $f_{\mathtt{S}}(\mathtt{s})$ p $f_{\mathtt{S}}(\mathtt{o})$.

Further, they also preserve:

(7) The summary contains one node for each set of classes belonging to some resource in the input.
(8) For any node n with a non empty class set, $f_{\mathtt{TW}}(n)$ (respectively, $f_{\mathtt{TS}}(n)$) is the node corresponding to the class set of n.

These invariants are ensured by the way in which we collect all class sets during the initial traversal of type triples (common to the TW and TS algorithms). Further, during the TW and TS summarization, as said in Section 7.3, the representatives of typed nodes never fuse, and never split.

The 6 invariants from the proof of Prop. 5 ensure the correct summarization of data triples when s and o are untyped. Together with the two above, they also ensure the correct summarization of triples having a typed s and/or o. □

(F. Goasdoué) Univ. Rennes, Inria, CNRS, IRISA, France
Email address, F. Goasdoué: fg@irisa.fr

(P. Guzewicz) Institut Polytechnique de Paris, Inria, France
Email address, P. Guzewicz: pawel.guzewicz@inria.fr

(I. Manolescu) Inria and Institut Polytechnique de Paris, France
Email address, I. Manolescu: ioana.manolescu@inria.fr