Query-Oriented Summarization of RDF Graphs

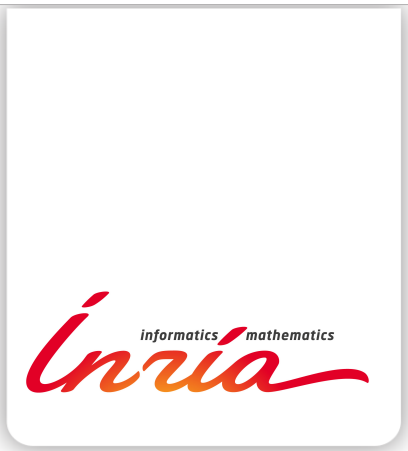
Šejla Čebirić, François Goasdoué, Ioana Manolescu

Query-Oriented Summarization of RDF Graphs

Šejla Čebirić, François Goasdoué, Ioana Manolescu

RESEARCH REPORT
N° 8920
June 2016
Project-Teams CEDAR

ISSN 0249-6399 ISRN INRIA/RR--8920--FR+ENG



Query-Oriented Summarization of RDF Graphs

Šejla Čebirić, François Goasdoué, Ioana Manolescu

Project-Teams CEDAR

Research Report n° 8920 — version 4 — initial version June 2016 — revised version June 2017 — 42 pages

Abstract: RDF is the data model of choice for Semantic Web applications. RDF graphs are often large and heterogeneous, thus users may have a hard time getting familiar with the structure and semantics of a graph, or determining whether a graph is useful for a certain application.
We consider answering such questions by inspecting a *graph summary*, a compact structure conveying as much information as possible about the input graph. A summary is *representative* of a graph if it represents both its explicit and implicit triples, the latter resulting from RDF Schema constraints. To ensure representativeness, we define a novel RDF-specific summarization framework based on *RDF node equivalence* and graph *quotients*; our framework can be instantiated with many different RDF node equivalence relations. We show that our summaries are representative, and establish a *sufficient condition* on the RDF equivalence relation to ensure that a graph can be *efficiently summarized*, without materializing its implicit triples. We demonstrate that the state-of-the art *bisimulation* equivalence relations between graph nodes fit into our framework. Further, we instantiate the framework through *four novel summaries*, based on the new concept of *property cliques*, specifically tailored to cope with highly heterogeneous RDF graphs; we show that they are orders of magnitude more compact than bisimulation summaries. Finally, we show that the bisimulation and two of our clique summaries can be built efficiently so that they represent the explicit and implicit data of the input graph without saturating the graph. The performance benefits of our efficient summarization method is confirmed through a set of experiments.

Key-words: Semantic Web, RDF, data summary, inference, reasoning, data compression

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Résumés orientés requêtes de graphes RDF

Résumé :

RDF est le modèle de données du W3C, fondé sur les graphes, pour les applications du Web Sémantique. Les graphes RDF sont souvent larges et hétérogènes, ce qui complique la tâche des utilisateurs qui tentent de se familiariser avec leurs structure et sémantique, ou de déterminer si un graphe est utile pour une application donnée.

Nous étudions comment répondre à ces besoins en inspectant un *résumé de graphe*, une structure compacte portant autant d'information que possible à propos du graphe d'entrée. Un résumé est *représentatif* d'un graphe s'il représente à la fois ses triplets explicites et implicites, ces derniers découlant de contraintes de RDF Schema. Pour garantir la représentativité, nous définissons un nouvau cadre de résumé de graphe spécifique à RDF, fondé sur une *relation d'équivalence de noeuds RDF* et le *quotient de graphe* ; notre cadre peut être instancié avec une multitude de relations d'équivalence de noeuds RDF. Nous montrons que nos résumés sont représentatifs et nous établissons une *condition suffisante* sur la relation d'équivalence de noeuds RDF pour garantir qu'un graphe peut être *efficacement* résumé, sans matérialiser ses triplets implicites. Nous démontrons que les relations d'équivalence entre noeuds de l'état de l'art, fondées sur la *bissimulation* de graphes, satisfont notre condition. De plus, nous instancions notre cadre avec quatre nouvelles relations d'équivalence de noeuds RDF, donnant lieu à quatre nouveaux types de résumés, fondées sur les *cliques de propriétés*, spécialement adaptées à la grande hétérogénéité des graphes RDF ; nous montrons que ces types de résumé sont plusieurs ordres de magnitude plus compacts que les résumés à base de bissimulation. Enfin, nous montrons que les résumés à base de bissimulation et deux de nos résumés à base de cliques peuvent être construits efficacement, de sorte qu'ils représentent les triplets explicites et implicites du graphe d'entrée sans avoir à le saturer. Les bénéfices de performance de notre méthode de résumé sont confirmés au travers d'un ensemble d'expériences.

Mots-clés : Web Sémantique, RDF, résumé de données, inférence, raisonnement, compression de données

1 Introduction

The Resource Description Framework (RDF) is the W3C standard for Semantic Web applications. RDF graphs are *varied*, coming from scientific applications, social or online media, government data etc. They are often *large*, and *heterogeneous*, i.e., resources described in an RDF graph may have very different sets of properties. An RDF resource may have one or several *types* (which may or may not be related to each other) or lack types. *RDF Schema* (RDFS) statements are sometimes part of an RDF graph, in which case they lead to **implicit data**.

For illustration, consider the RDF graph $\mathtt{G}_1$ *consisting only of the triples shown with solid edges* in Figure 1; *here and throughout the paper, class nodes, property nodes and RDF Schema triples are shown in blue* (we formally revisit them in Section 2). The graph describes resources $p_1, p_2, c_1, o_1, o_2, o_3$ and relationships between them, while the RDF Schema triples state respectively that someone having the property *heldOffice* is a *Politician*, that *havingSharesIn* something is a form of *ownership*, and that something in which one can have shares, is a *Company*. Implicit triples are shown with dotted edges: thus, because p_2 held offices o_2 and o_3, the graph implicitly states that p_2 is of type *Politician*. Similar reasoning leads to the other dotted (implicit) triples in the figure; $\mathtt{G}^\infty$ consists of all the (solid or dotted) triples. Importantly, **the semantics of an RDF graph $\mathtt{G}$, typically denoted $\mathtt{G}^\infty$, comprises both its explicit and implicit data**. In particular, when querying RDF graphs through SPARQL, the W3C's standard query language, query answers must *reflect all the explicit and implicit data*.

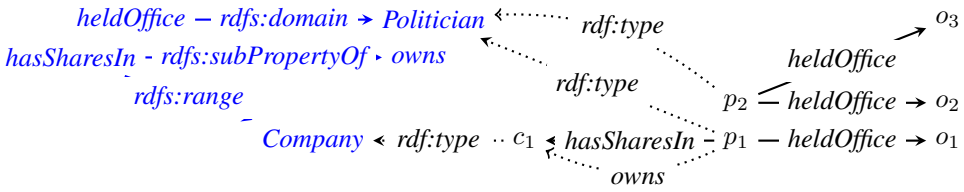


Figure 1: Sample RDF graph $\mathtt{G}_1$ with RDF Schema triples and implicit triples.

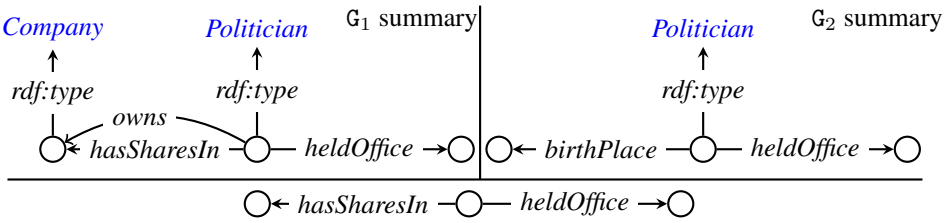


Figure 2: Top left: summary of the implicit and explicit triples of $\mathtt{G}_1$. Top right: summary of the implicit and explicit triples of another graph $\mathtt{G}_2$. Bottom: summary of the explicit triples of $\mathtt{G}_1$ only.

The complexity, heterogeneity, and possible semantics associated to an RDF graph may make them difficult to get acquainted with. For instance, consider a journalist interested in *politicians owning companies*. Let $\mathtt{G}_1$ be the graph in Figure 1, and $\mathtt{G}_2$ be another graph specifying for some politicians the offices they held and their birth places. Clearly, $\mathtt{G}_1$ is the best source to use in this case, as $\mathtt{G}_2$ lacks company information. How can we help the journalist figure this out quickly?

One solution is to use graph summaries [13, 12, 10] and in particular those specific to RDF graphs, e.g., [32, 21, 27]. Without loss of generality, a summary is a smaller RDF graph which "represents" the RDF graphs in some sense. Users may consult the summary to get an idea if the original graph is interesting for their needs. For instance, the summaries of $\mathtt{G}_1$ and $\mathtt{G}_2$ could be those shown in Figure 2,

quickly guiding the journalist toward $\mathtt{G}_1$. *In the figure and throughout this paper, circles denote summary nodes*, each of which is derived from (represents) a set of nodes from the RDF graph.

If $\mathtt{G}$ is a strict subset of $\mathtt{G}^\infty$, the summary of $\mathtt{G}$ may be different from that of $\mathtt{G}^\infty$. However, given that $\mathtt{G}^\infty$ is the semantics of $\mathtt{G}$, *the summary really representing* $\mathtt{G}$ *is in fact the one of* $\mathtt{G}^\infty$! For instance, at the bottom of Figure 2 appears the summary of only the explicit triples of $\mathtt{G}_1$. This summary does not reflect that in $\mathtt{G}_1$, *politician* p_1 *owns part of Company* c_1, as this triple is implicit. In contrast, the summary at top left in Figure 2 fully represents $\mathtt{G}_1$, that is, it summarizes (represents) its full semantics $\mathtt{G}_1^\infty$.

Ideally, the summary of $\mathtt{G}_1^\infty$ could be made available by the site hosting $\mathtt{G}_1$. What if this is not the case? Assuming the owners of $\mathtt{G}_1$ are willing to share it, one could try to saturate it, but RDF/SPARQL endpoints usually do not allow updates, thus one cannot add the implicit triples directly to $\mathtt{G}_1$. Instead, $\mathtt{G}_1$ could be copied elsewhere, saturated, and $\mathtt{G}_1^\infty$ summarized. This is however quite cumbersome, especially if the saturation is large and/or building it takes a long time. It would be more efficient if one could get the summary of $\mathtt{G}_1^\infty$ directly based on $\mathtt{G}_1$, without needing to saturate it.

This work is the first study of fully representative RDF graph summarization, taking into account both its explicit and implicit triples. Toward *representative, compact and efficient summarization of RDF graphs*, this work makes the following contributions:

1. We consider RDF graph summarization based on the classical notion of *quotient graphs*: this corresponds to a very wide family of graph summaries, each defined through one notion of *node equivalence*. While numerous useful graph summaries from the literature fit this framework, we show that *a naïve application of quotient summarization to RDF graphs with type and schema information prevents a summary from being representative*.

2. To solve this problem, we define *RDF node equivalence relations* based on the data triples of an RDF graph, and define an *RDF summaries* as a quotient graph based on such a relation. We show that any such summary is *representative*, that is: for any query q from a rich SPARQL subset $\mathcal{Q}$, if q evaluated on a graph $\mathtt{G}$ has some answers, then q also has answers on the summary. Conversely, we show that the summary is *accurate*: if q has answers on the summary, then it also has answers on *some* graph which it summarizes[1].

3. While the summary of $\mathtt{G}^\infty$ is the one really representing $\mathtt{G}$, building $\mathtt{G}^\infty$ in order to summarize it may be costly, or impossible for $\mathtt{G}$ users that do not have the right to add triples to $\mathtt{G}$, e.g., those querying a graph available through a SPARQL endpoint. To efficiently compute representative summaries, we describe a *shortcut* method to compute the summary of $\mathtt{G}^\infty$ *directly from* $\mathtt{G}$*, without saturating it*, and we provide a *sufficient condition* on an RDF node equivalence relation, for the corresponding RDF summary to be computable through our shortcut. We show that this condition holds for summaries based on the well-known *bisimulation* [16] notion (revisited in our framework).

4. We *define four RDF summaries* (Section 7 and 8) within our framework, based on the new concept of *property cliques* (Section 6), corresponding to a form of "association" between data properties in RDF graphs; a particularly interesting property of cliques is their tolerance to heterogeneity among graph nodes.

5. From an *efficiency* perspective, we demonstrate that two of these four summaries can be built through the shortcut method which the other two summaries do not admit.

6. We implemented a summarization tool based on bisimulation and clique node equivalence relations, and illustrate the performance benefits of building $\mathtt{G}^\infty$ summaries without saturating $\mathtt{G}$, for the summaries which admit the shortcut. Our experiments demonstrate the *compactness* of our four clique summaries, *orders of magnitude smaller than bisimulation summaries*.

As an image is worth a thousand words, the Web site [1] provides graphical representations of many sample summaries.

[1] Many RDF graphs may have the same RDF summary.

Assertion	Triple
Class	s rdf:type o
Property	s p o

Constraint	Triple
Subclass	s $\prec_{sc}$ o
Subproperty	s $\prec_{sp}$ o
Domain typing	p $\hookleftarrow_{d}$ o
Range typing	p $\hookrightarrow_{r}$ o

Figure 3: RDF (left) & RDFS (right) statements.

2 Preliminaries: RDF Graphs and Queries

An *RDF graph* (or *graph*, in short) 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 [33], using uniform resource identifiers (URIs), typed or untyped literals (constants) and blank nodes (unknown URIs or literals).

Notations. We use s, p, and o as placeholders for subjects, properties and objects, respectively. Literals are shown as strings between quotes, e.g., "$string$". Figure 3 (left) shows how to use triples to describe resources, that is, to express class and property assertions. The RDF standard [33] has a set of *built-in classes and properties*, as part of the rdf: and rdfs: pre-defined namespaces. We use these namespaces exactly for these classes and properties, e.g., rdf:type specifies the class(es) to which a resource belongs. *For brevity, we will sometimes use τ to denote* rdf:type.

For example, the RDF graph in Figure 1 describes two politicians identified by the respective URIs p_1 and p_2, three offices o_1 to o_3, and a company c_1.

RDF Schema (RDFS) allows specifying *semantic constraints* between the classes and the properties of an RDF graph. Figure 3 (right) shows the four kinds of RDFS constraints, and how to express them through triples. For concision, we denote the properties expressing *subclass*, *subproperty*, *domain* and *range* constraints by the symbols $\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_{d}$ and $\hookrightarrow_{r}$, respectively.

Implicit triples are an important RDF feature, considered part of the RDF graph even though they are not explicitly present in it. W3C names *RDF entailment* the mechanism through which, based on a set of explicit triples and some *entailment rules*, implicit RDF triples are derived. We denote by $\vdash^{i}_{\text{RDF}}$ *immediate entailment*, i.e., the process of deriving new triples through a *single* application of an entailment rule. More generally, a triple s p o is entailed by a graph G, denoted G $\vdash_{\text{RDF}}$ s p o, if and only if there is a sequence of applications of immediate entailment rules that leads from G to s p o (where at each step, triples previously entailed are also taken into account).

For instance, the graph in Figure 1 has the constraints: *heldOffice* $\hookleftarrow_{d}$ *Politician*, *hasSharesIn* $\prec_{sp}$ *owns* and *hasSharesIn* $\hookrightarrow_{r}$ *Company*. These constraints lead to the following implicit triples being part of the RDF graph: p_1 τ *Politician*, p_2 τ *Politician*, c_1 τ *Company* and p_1 *owns* c_1. These triples are shown with dotted lines in the figure.

RDF graph saturation. The immediate entailment rules allow defining the finite *saturation* (a.k.a. closure) of an RDF graph G, which is the RDF graph G^{∞} defined as the fixpoint obtained by repeatedly applying $\vdash^{i}_{\text{RDF}}$ rules on G. The saturation of an RDF graph is unique (up to blank node renaming), and does not contain implicit triples (they have all been made explicit by saturation). An obvious connection holds between the triples entailed by a graph G and its saturation: G $\vdash_{\text{RDF}}$ s p o if and only if s p o $\in \text{G}^{\infty}$.

The semantics of an RDF graph is its saturation. RDF entailment is part of the RDF standard itself; in particular, *the answers to a query posed on* G *must take into account all triples in* G^{∞} [33].

For presentation purposes, we may use a *triple-based* or a *graph-based* representation of an RDF graph:

1. The triple-based representation of an RDF graph. We see an RDF graph G as a *partition* of its

triples into three components $\mathtt{G} = \langle \mathtt{D_G}, \mathtt{S_G}, \mathtt{T_G} \rangle$, where: (i) $\mathtt{S_G}$, the **schema** component, is the set of all $\mathtt{G}$ triples whose properties are $\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_d$ or $\hookrightarrow_r$; (ii) $\mathtt{T_G}$, the **type** component, is the set of τ triples from $\mathtt{G}$; (iii) $\mathtt{D_G}$, the **data** component, holds all the remaining triples of $\mathtt{G}$. Note that each of $\mathtt{D_G}$, $\mathtt{S_G}$, and $\mathtt{T_G}$ is an RDF graph by itself.

Further, we call *data property* any property $\mathtt{p}$ occurring in $\mathtt{D_G}$, and *data triple* any triple in $\mathtt{D_G}$. For instance, in the RDF graph of Figure 1, data properties are *heldOffice* and *hasSharesIn*, used in the graph's four data triples.

2. The graph-based representation of an RDF graph. As per the RDF specification [33], *the set of nodes* of an RDF graph is the set of subjects and objects of triples in the graph, while its *edges* correspond to its triples. We define three categories of RDF graph nodes: (i) a *class node* is any node whose URI appears as subject or object of a $\prec_{sc}$ triple, or object of a $\hookleftarrow_d$ or $\hookrightarrow_r$ or τ triple; (ii) a *property node* is any node whose URI appears as subject/object of a $\prec_{sp}$ triple, or subject of a $\hookleftarrow_d$ or $\hookrightarrow_r$ triple, or property of a triple[2] (iii) a *data node* as any node that is neither a class nor a property node. Note that the sets of class nodes and of property nodes may intersect (indeed, nothing in the RDF specification forbids it), while class nodes are disjoint from data nodes, and property nodes are disjoint from data nodes, too.

Size and cardinality notations. We denote by $|\mathtt{G}|_n$ the number of nodes in a graph $\mathtt{G}$, and by $|\mathtt{G}|$ its number of edges. Further, for a given attribute $x \in \{\mathtt{s}, \mathtt{p}, \mathtt{o}\}$ and graph $\mathtt{G}$, we note $|\mathtt{G}|_x^0$ the number of distinct values of the attribute x within $\mathtt{G}$. For instance, $|\mathtt{D_G}|_{\mathtt{p}}^0$ is the number of distinct properties in the data component of $\mathtt{G}$.

Queries. We consider the SPARQL dialect consisting of *basic graph pattern* (BGP) queries, a.k.a. conjunctive queries, widely considered in research but also in real-world applications. A BGP is a set of *triple patterns*, or triples in short; each triple has a subject, property and object, some of which can be variables.

Notations. We use the usual conjunctive query notation $q(\bar{x})$:- $t_1, \ldots, t_\alpha$, where $\{t_1, \ldots, t_\alpha\}$ is a BGP; the query head variables $\bar{x}$ are called *distinguished variables*, and are a subset of those in $t_1, \ldots, t_\alpha$. For boolean queries, $\bar{x}$ is empty. The head of q is $q(\bar{x})$, its body is $t_1, \ldots, t_\alpha$; x, y, z, etc. denote variables.

Query answering. The *evaluation* of a query q against $\mathtt{G}$ has access only to the explicit triples of $\mathtt{G}$, thus it may fail to return the complete answer; the latter is obtained by evaluating q against $\mathtt{G}^\infty$. For instance, the query $q(x_2)$:- x_1 τ *Politician*, x_1 *hasName* x_2, x_1 *owns* c_1 asks for name of the politician(s) owning company c_1. Assuming the graph in Figure 1 has an extra triple stating that the name of p_1 is "Cahuzac", the answer to this query against the explicit and implicit triples of the graph is: $q(\mathtt{G}^\infty) = \{\langle \text{"}\textit{Cahuzac}\text{"} \rangle\}$. Note that evaluating q only against $\mathtt{G}$ leads to the empty answer.

3 Summarization principles

Without loss of generality, we consider that *the summary of an RDF graph* $\mathtt{G}$ *is an RDF graph itself.* This allows in particular to use existing RDF tools to query or visualize the summary. Further, summarization should satisfy the following conditions:

- *Schema independence:* It must be possible to summarize $\mathtt{G}$ whether or not it has RDF Schema information.

[2] A property node must be a *node*, i.e., merely appearing in a property position does not make an URI a property node; it also needs to appear a subject or object in the same or another triple.

- *Fully automatic:* Summarization should not require user input: the summary should help users learn about the data, not expect them to already know it[3].
- *Equal treatment of implicit and explicit data:* From the summary of $\mathtt{G}$, it must be possible to obtain as much information about the explicit triples of $\mathtt{G}$ as about the implicit ones, given that the semantic of $\mathtt{G}$ considers them equally.

Another desirable property is *accuracy*, i.e., the summary should avoid, to the extent possible, reflecting data that does not exist in $\mathtt{G}^\infty$.

Criteria for representativeness and accuracy. We formalize these criteria through the prism of query answering, since RDF graphs are typically accessed and exploited by means of queries. For instance, in our motivating example (Section 1), to find interesting graphs in a large graph collection, one can answer the query "politicians owning companies" over all the graph summaries, and focus on those leading to non-empty answers, e.g., $\mathtt{G}_1$ in Section 1. Given an *RDF query language (dialect)* $\mathcal{Q}$, we define:

Definition 1. (REPRESENTATIVENESS) *Let* $\mathtt{G}$ *be any RDF graph. Its summary* $\mathtt{Sum}$ *is* $\mathcal{Q}$*-representative of* $\mathtt{G}$ *if and only if for any query* $q \in \mathcal{Q}$ *such that* $q(\mathtt{G}^\infty) \neq \emptyset$*, we have* $q(\mathtt{Sum}^\infty) \neq \emptyset$*.*

Informally, queries having answers on $\mathtt{G}$ should also have answers on the summary. This is desirable in order for the summary to help users formulate queries: the summary should reflect all graph patterns that occur in the data.

Formalizing accuracy is a bit more involved. Summarization inevitably leads to some loss of information, which from our query-oriented perspective can be thought of as "false positives" (queries having answers on the summary of $\mathtt{G}$, but not on $\mathtt{G}$) and/or "false negatives" (queries having answers on $\mathtt{G}$, but not on its summary). As our discussion has highlighted, representativeness is an important asset, thus we rule out false negatives. Therefore, we allow summarization to introduce some false positives, and take this into account when defining accuracy. Note that several RDF graphs may have the same summary. We term *inverse set* of a summary $\mathtt{Sum}$, the set of all RDF graphs whose summary is $\mathtt{Sum}$; the inverse set is purely conceptual, i.e., we never actually compute it. Based on this, we define accuracy as:

Definition 2. (ACCURACY) *Let* $\mathtt{G}$ *be any RDF graph,* $\mathtt{Sum}$ *its summary, and* $\mathcal{G}$ *the inverse set of* $\mathtt{Sum}$*. The summary* $\mathtt{Sum}$ *is* $\mathcal{Q}$-accurate *if for any query* $q \in \mathcal{Q}$ *such that* $q(\mathtt{Sum}^\infty) \neq \emptyset$*, there exists* $\mathtt{G}' \in \mathcal{G}$ *such that* $q(\mathtt{G}'^\infty) \neq \emptyset$*.*

The above characterizes the accuracy of a summary with respect to *any graph it may correspond to*.

Which query dialect should we define representativeness and accuracy for? In order for summaries to be compact, they should not include subject and object values (whether they are URIs, literals or blank nodes); this follows the observation that there are many orders of magnitude more distinct subject and objects, than there are distinct properties in an RDF graph [35]. However, we chose to preserve types (classes) and data properties, as these are essential features of an RDF graph. These choices *rule out representativeness for queries with URIs or literals in subjects or objects positions*. Instead, we identify two large and useful classes of BGP queries:

Definition 3. (RELATIONAL AND RBGP* QUERIES) *A* relational BGP *(RBGP, in short) query is a BGP query whose body has: (i) URIs in all the property positions, (ii) a URI in the object position of every* τ *triple, and (iii) variables in any other positions.*

An extended relational *(RBGP*, in short) query is a BGP query whose body has (i) URIs* or variables *in all the property positions, (ii) a URI in the object position of every* τ *triple, and (iii) variables in any other positions.*

[3]For a different summary usage scenario, it is easy to extend our approach to summarize only triples referring e.g., to some classes and/or properties; we do not pursue this further here.

Both languages forbid URIs or literals in subject and object positions, and require that if type triples appear in the query, then the type is known. They differ in that RBGPs require URIs in the property positions, whereas RBGP*s also allow variables there. Clearly, RBGPs are a restriction of RBGP*s.

A sample RBGP is $q(x_2)$:- x_1 τ *Politician*, x_1 *hasName* x_2, x_1 *owns* c_1 while a similar RBGP* is $q^*(x_2)$:- x_1 τ *Politician*, x_1 *hasName* x_2, x_1 y c_1.

We define *RBGP* representativeness* and *RBGP* accuracy* by instantiating $\mathcal{Q}$ in Definition 1 and Definition 2, respectively, to RBGP* queries (Definition 3).

Discussion: preserving joins on literals While an RGBP (or RBGP*) query cannot enforce that the subject, property or object in a triple is a certain literal or URI (in other words, it cannot express selections), it does express joins. In particular, if the same literal appears in several places in the graph, an RBGP*-representative summary must preserve the joins enabled by these multiple occurrences. For instance, consider the graph:

$\mathtt{s}_1$ rdf:type `city`	$\mathtt{s}_1$ `zipCode` "91120"
$\mathtt{s}_2$ rdf:type `company`	$\mathtt{s}_2$ `employeeNo` "91120"
$\mathtt{s}_3$ rdf:type `person`	$\mathtt{s}_3$ `livesIn` "91120"

An RBGP* representative summary *must* have some results to the hypothetical query asking "cities and companies such that the zipcode of the first is the number of employees of the second", because on the above graph, the query does have results (due to $\mathtt{s}_1$ and $\mathtt{s}_2$). One may find this query meaningless, and argue that its non-emptiness on `G` is an accident. However, the very similar RGBP query "people living in the zipcode of a city" (which has results based on $\mathtt{s}_1$ and $\mathtt{s}_3$) is meaningful. In general, without human user input, it is hard to detect that the summary should preserve the results of one, and not of the other. Thus, in the remainder of the work, *we rely on RBGP* representativeness, knowing it preserves literal-based joins which may sometimes be seen as accidental*. To disable this, it suffices to redefine BGP (thus, RBGP and RBGP*) query semantics to consider that for query embedding purposes, two occurrences of the same literal in different places of the graph are not equal, and all our framework would adapt to this; we do not pursue this option further. (Note that the above discussion only concerns *joins on literals*; in contrast, joins on URI are a fundamental aspect of RDF (*linked*) data, and we consider they must be preserved by a meaningful summary.)

4 RDF summarization

As has often been attempted for non-RDF and RDF graphs, we define summaries through the classical notion of *quotient graph* from graph theory, parameterized by a *node equivalence relation*. Section 4.1 recalls these notions and shows that applied as such, they prevent RDF summary representativeness. Section 4.2 presents our approach toward defining RDF summaries that are rich with information and satisfy requirements identified in Section 3, in particular representativeness.

4.1 Quotient summaries and their limitations for RDF graphs

Recall that an *equivalence relation* $\sim$ over a set S is a binary relation (subset of $S \times S$) which is reflexive, transitive, and symmetric. Further, we recall:

Definition 4. (QUOTIENT GRAPH) *Let A be a label set, $G = (V, E)$ be a labeled directed graph whose vertices are V, whose edges are $E \subseteq V \times V \times A$, with labels from A. Let $\sim$ be an equivalence relation over the nodes of V.*

The quotient graph of G using $\sim$*, denoted $G_{/\sim}$, is a labeled directed graph having (i) a node n_S for each set S of equivalent V nodes, and (ii) an edge $n_{S_1} \xrightarrow{a} n_{S_2}$ for some label $a \in A$ iff there exist two V nodes $n_1 \in S_1$ and $n_2 \in S_2$ such that the edge $n_1 \xrightarrow{a} n_2 \in E$.*

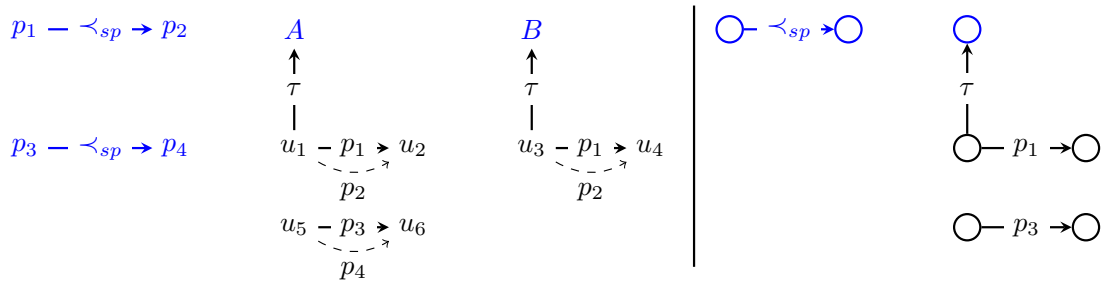


Figure 4: Sample RDF graph G (left) and its quotient graph based on $\sim_{\text{fb}}$ (right).

Many prior graph summarization and indexing works have relied on graph quotients, e.g., [25, 10, 28] (see also Section 11). In particulary, these works used the classical notion of *bisimulation* as a node equivalence relation; we recall it below:

Definition 5. (FW BISIMILARITY) *Forward (FW) bisimilarity is a node equivalence relation such that two nodes $n_1, n_2 \in G$ are FW-bisimilar, denoted $n_1 \sim_{\texttt{fw}} n_2$, iff (i) for every G edge of the form $n_1 \xrightarrow{a} m_1$ there exists a G edge of the form $n_2 \xrightarrow{a} m_2$ such that $m_1 \sim_{\texttt{fw}} m_2$, and conversely (ii) for every G edge of the form $n_2 \xrightarrow{a} m_2$ there exists a G edge of the form $n_1 \xrightarrow{a} m_1$ such that $m_1 \sim_{\texttt{fw}} m_2$.*

Intuitively, FW bisimilarity requires nodes to have the same sets of outgoing properties, and that these properties lead to respectively bisimilar nodes. The dual relation of forward bisimilarity is *backward (BW) bisimilarity*, noted $\sim_{\texttt{bw}}$, which compares nodes based on the incoming edges *only*. Nodes can also be compared based on their incoming *and* outgoing edges using *forward-backward (FB) bisimilarity*, noted $\sim_{\texttt{fb}}$. The definitions of these node equivalence relations easily derive from Definition 5; clearly, $\sim_{\texttt{fb}}$ implies $\sim_{\texttt{fw}}$ and $\sim_{\texttt{bw}}$ but the opposites do not hold. For instance, in the graph G shown Figure 4, it is easy to check that $u_1 \sim_{\texttt{fb}} u_3$, $u_2 \sim_{\texttt{fb}} u_4$, $p_1 \sim_{\texttt{fb}} p_3$ and $p_2 \sim_{\texttt{fb}} p_4$.

Bisimilarity essentially requires that the graph structure leading to/starting from two bisimilar nodes n_1, n_2 be the same. A more relaxed condition of *bounded- or k-bisimilarity* requires that these be equal only up to length k. For instance, if $\sim^k_{\texttt{fb}}$ is the bounded version of $\sim_{\texttt{fb}}$, and $n_1 \sim^k_{\texttt{fb}} n_2$, the graph structures at a distance greater than k from n_1 and n_2 may differ. Clearly, as bounded bisimilarity is easier to satisfy, a summary based on it will have fewer nodes than one based on pure bisimilarity. Our analysis in this paper applies to bounded and unbounded bisimilarity alike; for space reasons, we will refer only to the latter in the sequel.

Importantly, *straightforward application of quotient summarization on RDF graphs prevents the summary from being representative*. Figure 4 illustrates this on a FB bisimilarity quotient; similar examples can be exhibited for any quotient based on an equivalence relation which ignores the special role types and schema play in RDF graphs. The quotient shown at right in Figure 4 is not a representative summary: for instance, although the query $q(x)$:- $x \; \tau \; A$ has a non-empty answer on the graph G, it is empty on the quotient, and the same holds for the query $q(x)$:- $x \; p_2 \; y$. This loss of representativity stems from three issues: (•) *loss of class and (some) property names:* the nodes corresponding to classes A and B are equivalent and thus fused, since they only have incoming type edges and no outgoing edges; similarly, the property nodes p_2 and p_4 are fused thus they are absent from the summary; (⋄) *loss of RDF Schema triples:* the subproperty ($\prec_{sp}$) triples are summarized in a single one which states that "some properties are subproperties of others"; (○) *loss of implicit triple information:* the G triples shown in dashed edges are not reflected in the quotient, nor can they be inferred from it, given the loss of the schema triples.

4.2 Our RDF summarization framework

The first step we take toward RBGP* representative summaries is to define a generic notion of node equivalence specifically designed for RDF graphs:

Definition 6. (RDF NODE EQUIVALENCE) *Let $\equiv$ be a binary relation between the nodes of an RDF graph. We say $\equiv$ is an* RDF node equivalence relation *(or* RDF equivalence*, in short) iff (i) $\equiv$ is an equivalence relation (ii) any class node is $\equiv$ only to itself, and (iii) any property node is $\equiv$ only to itself.*

With this generic equivalence relation in place, we define our summary of RDF graphs as follows:

Definition 7. (RDF SUMMARY) *The* summary *of an RDF graph $\mathtt{G}$ by an RDF node equivalence relation $\equiv$, denoted $\mathtt{G}_{/\equiv}$, is the RDF graph defined as the quotient graph of $\mathtt{G}$ by $\equiv$, in which every class (resp. property) node is named after its $\mathtt{G}$ URI. Further, its data nodes are fresh URIs.*

Importantly, the RDF summaries enjoy the following property:

Property 1 (Class, property and schema preservation)**.** *An RDF graph $\mathtt{G}$ and an RDF summary $\mathtt{G}_{/\equiv}$ of it have the same sets of classes names, of property names, and of schema triples.*

Indeed, Definitions 6 and 7 ensure that class and property nodes are preserved through summarization, as well as their URI labels, since they cannot be equivalent (hence fused) with other nodes. Further, because our summarization approach relies on graph quotients, all property names labelling edges in a given graph also label edges in its summary. This obviously implies that schema triples are preserved, as they only involve class or property nodes.

It is worth noting that the above property de facto prevents in RDF summaries the issues ($\bullet$) and ($\diamond$) pointed out in the preceding section. However, a solution to the third issue ($\circ$) requires the following result:

Proposition 1. (SUMMARY REPRESENTATIVENESS) *An RDF summary is RBGP*-representative.*

Proof. We prove the statement for **RBGP** queries; Proposition 2 (below) carries the statement over to RBGP*.

Let q be a query such that $q(\mathtt{G}^\infty) \neq \emptyset$; we need to show that $q((\mathtt{G}_{/\equiv})^\infty) \neq \emptyset$.

Let $\phi : q \to \mathtt{G}^\infty$ be an embedding, assigning to each query variable v, a node from $\mathtt{G}^\infty$; we extend ϕ to say it maps triple patterns from q into triples from $\mathtt{G}^\infty$. We need to produce an embedding from q into $(\mathtt{G}_{/\equiv})^\infty$.

First, consider a triple pattern t of q whose embedding $\phi(t) \in \mathtt{G}^\infty$ also belongs to $\mathtt{G}$.

- If $\phi(t)$ is a schema triple, then $\phi(t)$ is also in $\mathtt{G}_{/\equiv}$, since $\mathtt{G}$ and $\mathtt{G}_{/\equiv}$ have the same schema triples.
- Else if $\phi(t)$ is a type triple of the form $\mathtt{s}\ \tau\ \mathtt{c}$, the triple $f(\mathtt{s})\ \tau\ \mathtt{c}$ belongs to $\mathtt{G}_{/\equiv}$, thus also to $(\mathtt{G}_{/\equiv})^\infty$.
- Otherwise, $\phi(t) = \mathtt{s\ p\ o} \in \mathtt{G}$ is a data triple, $\mathtt{G}_{/\equiv}$ holds the triple $f(\mathtt{s})\ \mathtt{p}\ f(\mathtt{o})$, thus $(\mathtt{G}_{/\equiv})^\infty$ also comprises it.

Now consider a triple pattern t' of q whose embedding $\phi(t') \in \mathtt{G}^\infty$ does not belong to $\mathtt{G}$.

- If $\phi(t')$ is a *schema* triple, then by definition of RDF entailment $\phi(t') \in (\mathtt{S_G})^\infty$, thus it is also in $(\mathtt{G}_{/\equiv})^\infty$ since $\mathtt{S_G} = \mathtt{S}_{\mathtt{G}_{/\equiv}} \subseteq \mathtt{G}_{/\equiv}$ by definition of a summary.
- Else if $\phi(t')$ is a *data* triple in $\mathtt{G}^\infty$, then by definition of RDF entailment, this triple $\phi(t')$ must be entailed by a *data* triple $t_d = \mathtt{s}_d\ \mathtt{p}_d\ \mathtt{o}_d$ in $\mathtt{G}$ and a subproperty constraints t_s, i.e., a schema triple in $\mathtt{S}^\infty$. As explained above, $f(\mathtt{s}_d)\ \mathtt{p}_d\ f(\mathtt{o}_d) \in \mathtt{G}_{/\equiv}$; at the same time, t_s also belongs $(\mathtt{G}_{/\equiv})^\infty$, since $\mathtt{S_G} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ hence $(\mathtt{S_G})^\infty = (\mathtt{S}_{\mathtt{G}_{\equiv}})^\infty$. It follows that the inference step which entailed $\phi(t')$ from t_d and t_s in $\mathtt{G}^\infty$ also applies on $f(\mathtt{s})\ \mathtt{p}_d\ f(\mathtt{o}_d)$ and t_s in $(\mathtt{G}_{/\equiv})^\infty$.

- Otherwise, $\phi(t')$ is a τ triple in $\mathtt{G}^\infty$. This may result either:

  - from a $\mathtt{D_G}$ triple $t_d = \mathtt{s}_d\ \mathtt{p}_d\ \mathtt{o}_d$ and a triple $t_s \in (\mathtt{S_G})^\infty$, if t_s is a $\hookleftarrow_d$ or $\hookrightarrow_r$ triple. This case is very similar to the one above.
  - from a $\mathtt{T_G}$ triple of the form $\mathtt{s}\ \tau\ \mathtt{c}_1$ and a schema triple $t_s = \mathtt{c}_1\ \prec_{sc}\ \mathtt{c}_2 \in (\mathtt{S_G})^\infty$, such that $\phi(t') = \mathtt{s}\ \tau\ \mathtt{c}_2$. In this case, $\mathtt{G}_{/\equiv}$ holds the triple $f(\mathtt{s})\ \tau\ \mathtt{c}_1$ which is also present in $(\mathtt{G}_{/\equiv})^\infty$, thus the same inference step applies in $(\mathtt{G}_{/\equiv})^\infty$ to produce $f(\mathtt{s})\ \tau\ \mathtt{c}_2$, since $\mathtt{S_G} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ hence $(\mathtt{S_G})^\infty = (\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$.

Thus, any q triple mapped by ϕ into a data $\mathtt{G}^\infty$ triple (which may or may not explicitly belong to $\mathtt{G}$) is also mapped into a corresponding triple in $(\mathtt{G}_{/\equiv})^\infty$.

To conclude this proof, we now need to show that, in addition to the fact that each q triple that has an embedding in $\mathtt{G}^\infty$ has also necessarily an embedding in $\mathtt{G}^\infty_{/\equiv}$, if q has an embedding in $\mathtt{G}^\infty$, then q has also an embedding in $(\mathtt{G}_{/\equiv})^\infty$. This amounts to show that any two q triples t_1 and t_2 that join and that have an embedding in $\mathtt{G}^\infty$ also embed in $(\mathtt{G}_{/\equiv})^\infty$ (i.e., q joins are preserved).

- If both $\phi(t_1)$ and $\phi(t_2)$ are schema triples in $\mathtt{G}^\infty$, then these two triples are also in $(\mathtt{G}_{/\equiv})^\infty$, since $\mathtt{S_G} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ hence $(\mathtt{S_G})^\infty = \mathtt{S}^\infty_{\mathtt{G}_{/\equiv}}$.

- Else if both $\phi(t_1)$ and $\phi(t_2)$ are non-schema triples in $\mathtt{G}^\infty$:

  - If both $\phi(t_1)$ and $\phi(t_2)$ are data triples in $\mathtt{G}^\infty$, there exists a triple t'_1) (respt'_2) in $\mathtt{G}$, with same subject/object, from which $\phi(t_1)$ (resp. $\phi(t_2)$) is entailed using a sub-property constraint t^1_s (resp. t^2_s) from $\mathtt{S}^\infty$. Since $\phi(t_1)$ and t'_1 (resp. $\phi(t_2)$ and t'_2) have the same subject and object values, then t'_1 and t'_2 have same values on the places where t_1 and t_2 join. Therefore, if we assume that $t'_1 = \mathtt{s}_1\ \mathtt{p}_1\ \mathtt{o}_1$ and $t'_2 = \mathtt{s}_2\ \mathtt{p}_2\ \mathtt{o}_2$, the $\mathtt{G}_{/\equiv}$ triples $f(\mathtt{s}_1)\ \mathtt{p}_1\ f(\mathtt{o}_1)$ and $f(\mathtt{s}_2)\ \mathtt{p}_2\ f(\mathtt{o}_2)$ necessarily have same values on the places where t_1 and t_2 join. Moreover, since $\mathtt{S_G} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ hence $(\mathtt{S_G})^\infty = (\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$, these two $\mathtt{G}_{/\equiv}$ triples and the above-mentioned t^1_s and t^2_s schema triples, produce the counterpart triples of $\phi(t_1)$ and $\phi(t_2)$ in $(\mathtt{G}_{/\equiv})^\infty$, which have same subject and object values. Thus, the q triples t_1 and t_2 embed in these two $(\mathtt{G}_{/\equiv})^\infty$ triples, if they embed in $\phi(t_1)$ and $\phi(t_2)$ in $\mathtt{G}^\infty$.
  - Else if both $\phi(t_1)$ and $\phi(t_2)$ are type triples in $\mathtt{G}^\infty$, say $\phi(t_1) = u\ \tau\ c_1$ and $\phi(t_2) = u\ \tau\ c_2$, then $t_1 = x\ \tau\ c_1$ and $t_2 = x\ \tau\ c_2$ by definition of an RBGP query. As in the cases of single q triple embeddings, $\phi(t_1)$ (resp. $\phi(t_2)$) results either from a $\mathtt{G}$ triple $u\ \tau\ c$ and a $\mathtt{S}^\infty_\mathtt{G}$ triple $c\ \prec_{sc}\ c_1$, or a $\mathtt{G}$ triple $u\ p\ u_1$ and a $\mathtt{S}^\infty_\mathtt{G}$ triple $p\ \hookleftarrow_d\ c_1$, or a $\mathtt{G}$ triple $u_1\ p\ u$ and a $(\mathtt{S_G})^\infty$ triple $p\ \hookrightarrow_r\ c_1$. Therefore, since $\mathtt{S_G} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ hence $\mathtt{S}^\infty_\mathtt{G} = (\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$, for $\phi(t_1)$ (resp. $\phi(t_2)$), there are either a $\mathtt{G}_{/\equiv}$ triple $f(u)\ \tau\ c$ and a $(\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$ triple $c\ \prec_{sc}\ c_1$, or a $\mathtt{G}_{/\equiv}$ triple $f(u)\ p\ f(u_1)$ and a $(\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$ triple $p\ \hookleftarrow_d\ c_1$, or a $\mathtt{G}_{/\equiv}$ triple $f(u_1)\ p\ f(u)$ and a $(\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$ triple $p\ \hookrightarrow_r\ c_1$, which entail $f(u)\ \tau\ c_1$ and $f(u)\ \tau\ c_2$ in $\mathtt{G}^\infty_{/\equiv}$. Thus, the q triples t_1 and t_2 embed in these two $\mathtt{G}^\infty_{/\equiv}$ triples, if they embed in $\phi(t_1)$ and $\phi(t_2)$ in $\mathtt{G}^\infty$.
  - Otherwise, $\phi(t_1)$ is a data triple in $\mathtt{G}^\infty$ and $\phi(t_2)$ is a type triple in $\mathtt{G}^\infty$. This case is very similar to the two above case, hence we do not detail it.

- Otherwise, $\phi(t_1)$ is a schema triple in $\mathtt{G}^\infty$ and $\phi(t_2)$ is not a schema triples in $\mathtt{G}^\infty$. In this case, since q is an RBGP query, q triples must be such that t_1 is $\mathtt{s}_1\ \mathtt{p}_1\ \mathtt{o}_1$ with $\mathtt{p}_1 \in \{\prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r\}$ and t_2 is either $\mathtt{s}_2\ p\ \mathtt{o}_2$ or $\mathtt{s}_2\ \tau\ c$.

  Since $\mathtt{G}^\infty$ and $\mathtt{G}^\infty_{/\equiv}$ have the same schema, $\phi(t_1)$ also belongs to $\mathtt{G}^\infty_{/\equiv}$. Now:

– If t_2 is $\mathtt{s}_2\ p\ \mathtt{o}_2$ then $\phi(t_2) = s, p, o$ must be either in $\mathtt{G}$ or entailed from a $\mathtt{G}$ data triple s, p', o and a $\mathtt{S}^\infty_\mathtt{G}$ sub-property triple $p', \prec_{sp}, p$ (see above, for single q triple embedding). If $\phi(t_2)$ is in $\mathtt{G}$, then $f(s)\ p\ f(o)$ is in $\mathtt{G}_{/\equiv}$, hence in $\mathtt{G}^\infty_{/\equiv}$. Otherwise, $f(s)\ p'\ f(o)$ is in $\mathtt{G}_{/\equiv}$, $p'\ \prec_{sp}\ p$ is in $\mathtt{S}_{\mathtt{G}_{/\equiv}}$ (since $\mathtt{G}$ and $\mathtt{G}_{/\equiv}$ have the same schema), thus $f(s)\ p\ f(o)$ is in $\mathtt{G}^\infty_{/\equiv}$. Since t_1 and t_2 joins, s and/or p are class/property nodes. If s (resp. o) is a class/property node, then $f(s) = s$ (resp. $f(o) = o$). Hence, $\phi(t_1) \in \mathtt{G}^\infty_{/\equiv}$ joins with $f(s)\ p\ f(o) \in \mathtt{G}^\infty_{/\equiv}$.

– If t_2 is $\mathtt{s}_2\ \tau\ c$ then $\phi(t_2) = s\ \tau\ c$ must be either in $\mathtt{G}$ or entailed from (i) a $\mathtt{G}$ data triple $s\ p\ o$ and a $\mathtt{S}^\infty_\mathtt{G}$ triple $p\ \hookleftarrow_d\ c$, or a $\mathtt{G}$ data triple $s_1\ p\ s$ and a $\mathtt{S}^\infty_\mathtt{G}$ triple $p\ \hookrightarrow_r\ c$, or (iii) a $\mathtt{G}$ type triple $s\ \tau\ c'$ and a $\mathtt{S}^\infty_\mathtt{G}$ triple $c'\ \prec_{sc}\ c$ (see above, for single q triple embedding). If $\phi(t_2)$ is in $\mathtt{G}$, then $f(s)\ \tau\ c$ is in $\mathtt{G}_{/\equiv}$, hence in $\mathtt{G}^\infty_{/\equiv}$. Otherwise, $f(s)\ p\ f(o)$ or $f(s_1)\ p\ f(s)$ or $f(s)\ \tau\ c'$ is in $\mathtt{G}_{/\equiv}$, and (since $\mathtt{G}$ and $\mathtt{G}_{/\equiv}$ have the same schema) thus $f(s)\ \tau\ c$ is in $\mathtt{G}^\infty_{/\equiv}$. Since t_1 and t_2 can only join on $\mathtt{s}_2$, s is a class or property nodes, hence $f(s) = s$. Therefore, $\phi(t_1) \in \mathtt{G}^\infty_{/\equiv}$ joins with $f(s)\ \tau\ c \in \mathtt{G}^\infty_{/\equiv}$.

□

Property 2. *RBGP representativeness entails RBGP* representativeness.*

Proof. Let q^* be an RBGP* query which is non-empty on $\mathtt{G}^\infty$ and $\mathtt{G}_{/\equiv}$ be an RBGP-representative summary of $\mathtt{G}$. We show that non-emptiness of q^* on $\mathtt{G}^\infty$ entails its non-emptiness on $(\mathtt{G}_{/\equiv})^\infty$.
Given that $q^*(\mathtt{G}^\infty)$ is non-empty, there exists at least an embedding of q^* into $\mathtt{G}^\infty$; let q be the query obtained by replacing in q^*, each variable occurring in the property position by the concrete property matching it in $\mathtt{G}^\infty$. Clearly, q has results on $\mathtt{G}^\infty$, and since $\mathtt{G}_{/\equiv}$ is RBGP representative, q also has results on $(\mathtt{G}_{/\equiv})^\infty$. Therefore, $q(\mathtt{G}^\infty_{/\equiv}) \neq \emptyset$, and given that $q \subseteq q^*$ (query containment), it follows that $q^*((\mathtt{G}_{/\equiv})^\infty) \neq \emptyset$. □

One can prove in a very similar fashion that summaries are also representative with respect to *regular path expression queries* such as SPARQL 1.1 property paths [34]: any regular path expression query whose results are non empty on an RDF graph $\mathtt{G}$ also has non-empty results on a summary of $\mathtt{G}$. As a consequence, our summaries also preserve reachability in $\mathtt{G}$: if a path leads from n_1 to n_2 in $\mathtt{G}$, the same path also leads from the node representing n_1 in the summary, to the node representing n_2.

We identify the following strong connection between two RDF graphs:

Definition 8. (STRONG ISOMORPHISM $\widehat{=}$) *A strong isomorphism between two RDF graphs $\mathtt{G}_1, \mathtt{G}_2$ is an isomorphism which is the identity for the class and property nodes. We write this: $\mathtt{G}_1 \widehat{=} \mathtt{G}_2$.*

From the definition of an RDF summary, it follows that any summary is (also) a summary of itself, more precisely $(\mathtt{G}_{/\equiv})_{/\equiv} \widehat{=} \mathtt{G}_{/\equiv}$. We call this the **fixpoint property**; it captures the intuition that a summary cannot be summarized any further. The reason why $(\mathtt{G}_{/\equiv})_{/\equiv}$ and $\mathtt{G}_{/\equiv}$ are $\widehat{=}$ and not *identical* is that summarization creates "new" summary nodes (with fresh URIs). The summary preserves (reflects) the *structure* of the original graph, but it does not preserve the URIs of data nodes.

Accuracy. From the fixpoint property, it follows easily that any RDF summary is accurate (Definition 2). Indeed, for any graph $\mathtt{G}$ and RDF summary $\mathtt{G}_{/\equiv}$ of $\mathtt{G}$, since $(\mathtt{G}_{/\equiv})_{/\equiv}$ and $\mathtt{G}_{/\equiv}$ are isomorphic, any query non-empty on $(\mathtt{G}_{/\equiv})_{/\equiv}$ is also non-empty on $\mathtt{G}_{/\equiv}$.

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

Recall from Section 2 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 (Definition 7), n is in $\mathtt{G}$ iff $f(n)$ is in $\mathtt{G}_{/\equiv}$. Finally, again by the definition of saturation, $f(n)$ is in $\mathtt{G}_{/\equiv}$ iff $f(n)$ is in $(\mathtt{G}_{/\equiv})^{\infty}$.

Therefore, every $\mathtt{G}^{\infty}$ node n maps the $f(n)$ $(\mathtt{G}_{/\equiv})^{\infty}$ 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 $f(n_1)\ p\ f(n_2)$ is an edge in $\mathtt{G}_{/\equiv}$, by definition of an RDF summary. Finally, if $f(n_1)\ p\ f(n_2)$ is an edge in $\mathtt{G}_{/\equiv}$, then $f(n_1)\ p\ f(n_2)$ is also an edge in $(\mathtt{G}_{/\equiv})^{\infty}$.
- $n_1\ p'\ n_2$ is an edge in $\mathtt{G}$, and $p'\ \prec_{sp}\ p$ is in $\mathtt{S}_{\mathtt{G}}^{\infty}$, 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 $f(n_1)\ p'\ f(n_2)$ is an edge in $(\mathtt{G}_{/\equiv})^{\infty}$, hence $f(n_1)\ p\ f(n_2)$ 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 (Property 1).

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

- $n_1\ \tau\ c$ is an edge in $\mathtt{G}$. This holds iff $f(n_1)\ \tau\ c$ is an edge in $\mathtt{G}_{/\equiv}$, by definition of an RDF summary (recall that $f(c) = c$ for classes). Finally, if $f(n_1)\ \tau\ c$ is an edge in $\mathtt{G}_{/\equiv}$, then $f(n_1)\ \tau\ c$ is also an edge in $(\mathtt{G}_{/\equiv})^{\infty}$.
- $n_1\ p\ n_2$ is an edge in $\mathtt{G}$ and $p\ \hookleftarrow_d\ c$ (or $p\ \hookrightarrow_r\ c$) is in $\mathtt{S}_{\mathtt{G}}^{\infty}$, thus $n_1\ \tau\ c$ is produced by saturation in $\mathtt{G}^{\infty}$. In this case, we show similarly as above that $f(n_1)\ p\ f(n_2)$ is an edge in $(\mathtt{G}_{/\equiv})^{\infty}$, hence $f(n_1)\ \tau\ c$ 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 (Property 1).

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

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

4.3 Bisimulation-based RDF summarization

We now instantiate our summarization framework. We start by defining *RDF FW (forward), BW (backward) and FB bisimilarity*, based on Definition 6:

Definition 9. (FW RDF BISIMILARITY) *Forward (FW) RDF bisimilarity is an RDF node equivalence relation such that two data nodes $n_1, n_2 \in \mathtt{G}$ are FW-bisimilar, denoted $n_1 \equiv_{\mathtt{fw}} n_2$, iff (i) for every $\mathtt{G}$ data triple of the form $n_1\ \mathtt{p}\ m_1$ there exists a $\mathtt{G}$ data triple of the form $n_2\ \mathtt{p}\ m_2$ such that $m_1 \equiv_{\mathtt{fw}} m_2$, and conversely (ii) for every $\mathtt{G}$ data triple of the form $n_2\ \mathtt{p}\ m_2$ there exists a $\mathtt{G}$ data triple of the form $n_1\ \mathtt{p}\ m_1$ such that $m_1 \equiv_{\mathtt{fw}} m_2$.*

Similarly, a *backward (BW) RDF bisimilarity*, noted $\equiv_{\mathtt{bw}}$, is defined by inverting the triples' subject and object positions in the definition above. Further, a *forward and backward (FB) RDF bisimilarity* between data nodes, noted $\equiv_{\mathtt{fb}}$, is defined by nodes that are both FW and BW similar. In Figure 4, each of the nodes A, B, p_1, p_2, p_3 and p_4 is $\equiv_{\mathtt{fw}}$, $\equiv_{\mathtt{bw}}$, and $\equiv_{\mathtt{fb}}$ to itself (these are all RDF equivalence relations, thus they are reflexive), and none of them is bisimilar to any other node, since they are class and property nodes, and Definition 6 rules this out. Further, u_1 is $\equiv_{\mathtt{fw}}$, $\equiv_{\mathtt{bw}}$, and $\equiv_{\mathtt{fb}}$ to u_3, as in $\mathtt{G}$, they both have the properties p_1 and τ and no incoming properties; similarly u_2 is equivalent to u_4. From the $\equiv_{\mathtt{fw}}$ perspective only, u_2, u_4 and u_6 are all equivalent as they have no outgoing edges. Bounded versions of these RDF bisimilarities could also be easily defined as discussed in Section 4.1 for their simple (non-RDF) counterparts.

So far, our equivalence notions do not exploit the possible types which may be attached to RDF nodes in a graph. To do so, we define:

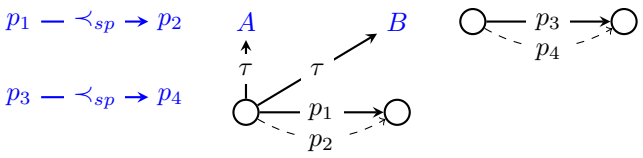


Figure 5: RDF summary through $\equiv_{\mathtt{fb}}$ of the graph in Figure 4.

Definition 10. (TYPED EQUIVALENCE) Typed equivalence, *noted* $\equiv_{\mathtt{T}}$*, is an RDF node equivalence relation defined as follows: two data nodes* n_1 *and* n_2 *are type-equivalent, noted* $n_1 \equiv_{\mathtt{T}} n_2$*, iff they have exactly the same set of types in* G*, which is non-empty.*

In Figure 4, $u_1 \not\equiv_{\mathtt{T}} u_3$ as they have different types. A node in an RDF graph may have no type; such a node is equivalent to itself (again, due to the reflexiveness of all RDF equivalence relations, including $\equiv_{\mathtt{T}}$), but not equivalent to any other node.

Clearly, each RDF node equivalence relation leads to a partition over the data nodes in G. Thus, any RDF node equivalence (in particular, $\equiv_{\mathtt{fw}}$, $\equiv_{\mathtt{bw}}$, $\equiv_{\mathtt{fb}}$ or $\equiv_{\mathtt{T}}$ above) leads to a (possibly distinct) quotient summary, denoted $\mathtt{G}_{/\equiv_{\mathtt{fw}}}$, $\mathtt{G}_{/\equiv_{\mathtt{bw}}}$, $\mathtt{G}_{/\equiv_{\mathtt{fb}}}$ and $\mathtt{G}_{/\equiv_{\mathtt{T}}}$. For illustration, Figure 5 shows the summary $\mathtt{G}_{/\equiv_{\mathtt{fb}}}$ of the RDF graph G in Figure 4. Since all nodes in that G are untyped, its summary $\mathtt{G}_{/\equiv_{\mathtt{T}}}$ has the same number of nodes as G. This shows that summarization through $\equiv_{\mathtt{T}}$ *alone* may fail to reduce the graph size.

To exploit *type information when available, and structure similarity when types are lacking*, we define:

Definition 11. (TYPED BISIMILARITY SUMMARIES) *Let* $\equiv_{\mathtt{U,fw}}$ *(untyped FW equivalence) be an RDF node equivalence relation that holds between two nodes* n_1, n_2 *iff (i)* n_1, n_2 *have no types in* G *and (ii)* $n_1 \equiv_{\mathtt{fw}} n_2$*. The* typed FW summary *of* G*, denoted* $\mathtt{G}_{/\equiv_{\mathtt{T,fw}}}$ *is defined as:* $(\mathtt{G}_{/\equiv_{\mathtt{T}}})_{/\equiv_{\mathtt{U,fw}}}$.
The typed BW summary*, noted* $\mathtt{G}_{/\equiv_{\mathtt{T,bw}}}$*, is similarly defined by replacing* fw *with* bw *in the above, while for the* typed FB summary*, noted* $\mathtt{G}_{/\equiv_{\mathtt{T,fb}}}$*,* fw *is replaced with* fb.

Let us follow for instance the construction of the typed FW summary. First, during summarization through $\equiv_{\mathtt{T}}$, typed nodes from G are grouped according to their types, while untyped G nodes and edges between them are simply copied into an isomorphic summary subgraph. Then, we summarize again, applying $\equiv_{\mathtt{fw}}$ only to the untyped nodes of $\mathtt{G}_{/\equiv_{\mathtt{T}}}$. Overall, the process gives *priority to types*, i.e., G nodes are summarized into a same node as soon as they have the same types in G, regardless of the surrounding graph structure; and second, for G nodes lacking types, we use structural bisimilarity to decide equivalence.

Figure 8 illustrates $\equiv_{\mathtt{T,bw}}$ summarization. As all data nodes are untyped in G, u_1, u_4 are summarized together in $\mathtt{G}_{\equiv_{\mathtt{T,bw}}}$. In contrast, in $\mathtt{G}^{\infty}$, u_1 is typed and u_4 is not, therefore in $(\mathtt{G}^{\infty})_{\equiv_{\mathtt{T,bw}}}$ they are represented by different nodes.

5 Efficient summarization shortcuts

Given that the semantics of G is $\mathtt{G}^{\infty}$, an RBGP* representative summary should reflect both the explicit and the implicit triples of G. A straightforward way to obtain $(\mathtt{G}^{\infty})_{/\equiv}$ is compute $\mathtt{G}^{\infty}$ and then summarize it. This is not directly possible when one does not have the right to add triples to G, and when it is possible, it may be time and space-consuming. Further, it has to be maintained when data or schema triples in G change.

To avoid going through the saturation step, we identify a method (below) which guarantees that we can build RBGP* representative summaries efficiently:

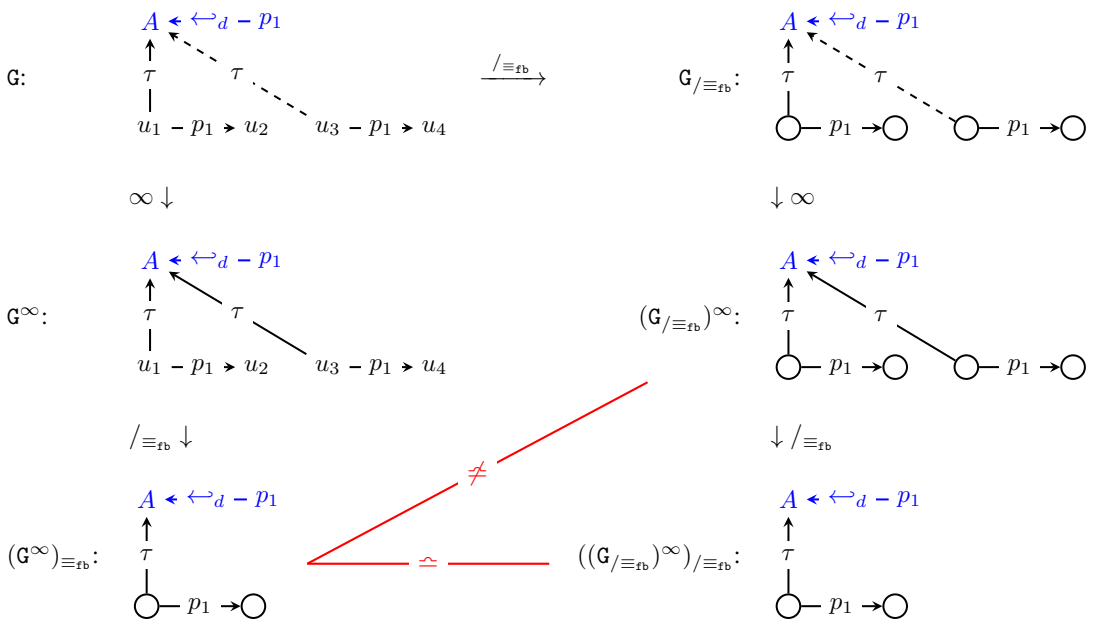


Figure 6: Shortcut example for RDF summarization through $\equiv_{\mathrm{fb}}$.

Definition 12. (SHORTCUT) *Summarization through the RDF node equivalence relation $\equiv$ admits a shortcut iff for any RDF graph $\mathtt{G}$, $(\mathtt{G}^{\infty})_{/\equiv} \simeq ((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ holds, where $\simeq$ denotes a strong isomorphism (Definition 8).*

The efficient method (or shortcut) to build $(\mathtt{G}^{\infty})_{/\equiv}$ introduced above is: summarize $\mathtt{G}$; saturate the result; then summarize it again. The shortcut leads to a graph whose saturation is strongly isomorphic to that of $(\mathtt{G}^{\infty})_{/\equiv}$. The two may differ in the exact URIs of the summary *data* nodes (depicted as blank circles in this paper's examples); these are just *representatives* of $\mathtt{G}$ node groups, and their exact URIs do not matter. The summary structure matters a lot, as illustrated in the Introduction; this is preserved by $\simeq$. Thus, essentially, *a shortcut allows to obtain (an equivalent to) the summary of the saturated $\mathtt{G}$ without saturating it*.

Figure 6 illustrates a shortcut for an $\equiv_{\mathrm{fb}}$-based summary $\mathtt{G}_{/\equiv_{\mathrm{fb}}}$ of an RDF graph $\mathtt{G}$: $(\mathtt{G}^{\infty})_{/\equiv_{\mathrm{fb}}} \simeq ((\mathtt{G}_{/\equiv_{\mathrm{fb}}})^{\infty})_{/\equiv_{\mathrm{fb}}}$ holds. Note that the last summarization step is necessary here, because $(\mathtt{G}^{\infty})_{/\equiv_{\mathrm{fb}}} \simeq (\mathtt{G}_{/\equiv_{\mathrm{fb}}})^{\infty}$ *does not* hold.

The next theorem is the major result of this work. It establishes a sufficient condition on an RDF node equivalence relation for the existence of a shortcut. To be able to state the theorem, we introduce the following:

Representation function f By the summary definition, to every node in $\mathtt{G}$ corresponds exactly one node in the summary $\mathtt{G}_{\equiv}$. We call *representation function* and denote $f_{\equiv}$ (or simply f, when this does not cause confusion) the function associating a summary node to each $\mathtt{G}$ node; we say $f(n)$ *represents* n in the summary. For instance, for the graph $\mathtt{G}$ in Figure 4 and its summary in Figure 5, $f(u_1) = f(u_3)$ is the node having the types A and B in Figure 5.

Theorem 1 (Sufficient condition for the existence of shortcuts)**.** *Given an RDF node equivalence relation $\equiv$, and an RDF graph $\mathtt{G}$, let $\mathtt{G}_{/\equiv}$ be its summary and $f_{\equiv}$ the corresponding representation function from $\mathtt{G}$ nodes to $\mathtt{G}_{/\equiv}$ nodes.*

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 $(\mathtt{G}^{\infty})_{/\equiv} \simeq ((\mathtt{G}_{/\equiv})^{\infty})_{/\equiv}$ holds.

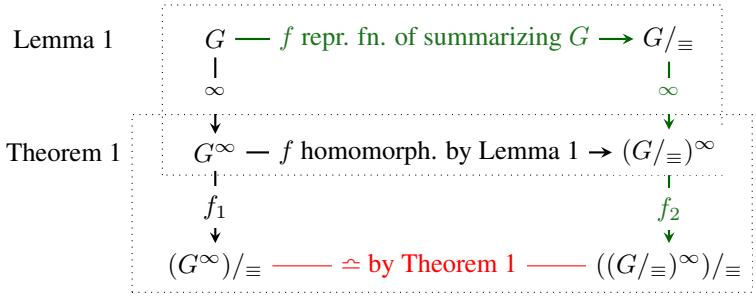


Figure 7: Illustration for Lemma 1 and Theorem 1.

The proof relies on the following lemma, which states a crucial structural property relating $\mathtt{G}$, $\mathtt{G}^\infty$ and the representation function f (see the upper part of Figure 7):

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

Lemma 1 is important as it connects RDF saturation (vertical ∞-labeled edges in Figure 7) to the representation of RDF nodes by corresponding summary nodes (black horizontal edges). The lemma states that the representation function associated to the summarization of $\mathtt{G}$ defines an homomorphism from the saturation of $\mathtt{G}$ to that of its summary. Note how this homomorphism (central horizontal edge in Figure 7) closes a rectangle materializing two paths from $\mathtt{G}$ to $(\mathtt{G}^\infty)_{/\equiv}$: one consists of the first two steps of the shortcut (Definition 12), while the other requires saturating $\mathtt{G}$.

The Lemma is crucial to Theorem 1's proof, as it allows establishing for any RDF node equivalence relation enjoying the Theorem's condition, that: (i) there exists a *bijection* from the nodes of $(\mathtt{G}^\infty)_{/\equiv}$ to those of $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, which maps each $(\mathtt{G}^\infty)_{/\equiv}$ node representing the set of equivalent $\mathtt{G}^\infty$ nodes $n_1 \equiv \cdots \equiv n_\alpha$ to the $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ node representing the set of equivalent $\mathtt{G}^\infty$ nodes $f(n_1) \equiv \cdots \equiv f(n_\alpha)$; and (ii) this bijection further defines a $\simeq$*-isomorphism* from the summary of $\mathtt{G}^\infty$ to that of its homomorphism $(\mathtt{G}_{/\equiv})^\infty$ through f. The existence of such a $\simeq$ establishes that the shortcut identified in Definition 12 holds, as illustrated in Figure 7 by the bottom red edge showing that the shortcut path made of green edges leads from $\mathtt{G}$ to (a graph $\simeq$-isomorphic to) $(\mathtt{G}^\infty)_{/\equiv}$, namely $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$.

Importantly, *not all RDF equivalence relations admit a shortcut*, as Figure 8 shows. From $\mathtt{G}$, the top row traces the two-step computation of $(\mathtt{G}^\infty)_{/\equiv_{\mathtt{T,bw}}}$, while moving down to the bottom row we see the three-step shortcut to $((\mathtt{G}_{/\equiv_{\mathtt{T,bw}}})^\infty)_{/\equiv_{\mathtt{T,bw}}}$. The shortcut does not lead to (a graph $\simeq$ to) the $\equiv_{\mathtt{T,bw}}$ summary of $\mathtt{G}^\infty$, because u_1 and u_4 are kept separate in the top row, while they are fused on the bottom one.

However, for RDF summaries based (only) on RDF bisimilarity relations, we readily establish based on Theorem 1:

Theorem 2. *Summarization through the RDF equivalence relations $\equiv_{\mathtt{fw}}$, $\equiv_{\mathtt{bw}}$ and $\equiv_{\mathtt{fb}}$ admits a shortcut.*

The condition identified in Theorem 1 is sufficient; finding a necessary (and sufficient) condition is currently open.

6 Data property cliques and clique-based equivalence

This section sets the first step for the novel contributions of this paper: the novel notion of property cliques (Section 6.1), followed by two new RDF node equivalence relations (Section 6.2) on which we build our summaries.

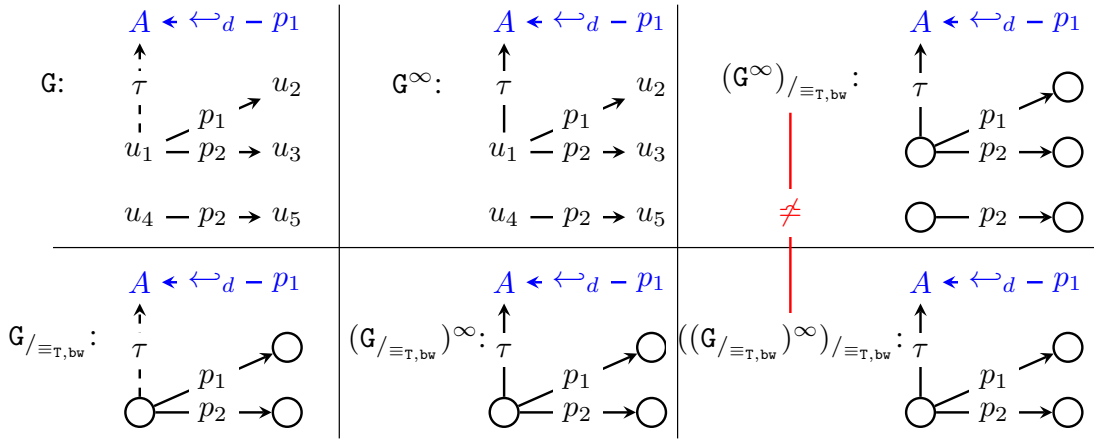


Figure 8: Sample typed bisimilarity summarization through $\equiv_{\mathtt{T,bw}}$.

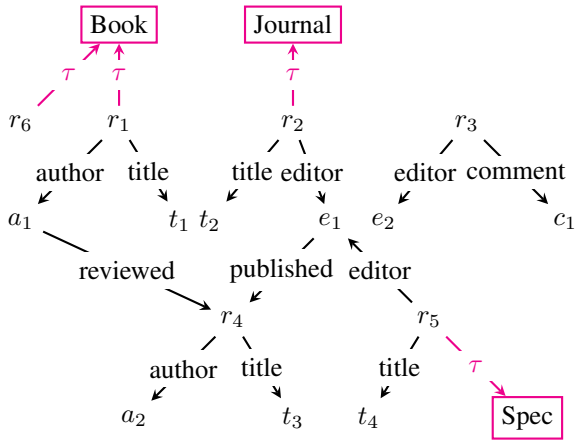


Figure 9: RDF graph used for clique-based summarization.

6.1 Data property cliques

The simplest relation between two data properties in a graph is *co-occurrence*, when a resource is the source and/or target of two data properties. However, in heterogeneous RDF graphs, two properties $\mathtt{p}_1$, $\mathtt{p}_2$ may co-occur on some nodes, while other nodes may have $\mathtt{p}_1$ or only $\mathtt{p}_2$, while such nodes have yet other properties $\mathtt{p}_3$, $\mathtt{p}_4$ etc. Thus, we introduce a more general and flexible notion crucial to this work, as follows:

Definition 13. (PROPERTY RELATIONS AND CLIQUES) *Let $\mathtt{p}_1$, $\mathtt{p}_2$ be two data properties in $\mathtt{G}$:*

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

node	r_1	r_2	r_3	r_4	r_5
$SC(r)$	SC_1	SC_1	SC_1	SC_1	SC_1
$TC(r)$	$\emptyset$	$\emptyset$	$\emptyset$	TC_5	$\emptyset$
node	a_1	t_1	t_2	e_1	e_2
$SC(r)$	SC_2	$\emptyset$	$\emptyset$	SC_3	$\emptyset$
$TC(r)$	TC_1	TC_2	TC_2	TC_3	TC_3
node	c_1	t_4	a_2	t_3	r_6
$SC(r)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$TC(r)$	TC_4	TC_2	TC_1	TC_2	$\emptyset$

Table 1: Source and target cliques of the sample RDF graph.

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

For illustration, consider the sample graph in Figure 9. Here, properties a and t are source-related due to r_1 (condition (i) in the definition). Similarly, t and e are source-related due to r_2; consequently, a and e are source-related (condition (ii)). Properties r and p are target-related due to r_4. The non-empty source cliques are $SC_1 = \{a, t, e, c\}$, $SC_2 = \{r\}$ and $SC_3 = \{p\}$, whereas the non-empty target cliques are $TC_1 = \{a\}$, $TC_2 = \{t\}$, $TC_3 = \{e\}$, $TC_4 = \{c\}$ and $TC_5 = \{r, p\}$.

A source clique can be seen as "all the data properties of same-kind resources"; above, it makes sense to group together properties corresponding to various kinds of publications, such as author, title, editor, and comment, *despite the heterogeneity exhibited by the presence/absence of these properties* on the RDF graph nodes. Similarly, a target clique comprises "the data properties of which same-kind resources are values".

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 resource $r \in$ G has data properties, they are all in the same source clique; similarly, all the properties of which r is a value are in the same target clique. This allows us to refer to *the source (or target) clique of* r, denoted $SC(r)$ and $TC(r)$. If r is not the value of any property (respectively, has no property), we consider the target (respectively, source) clique of r to be $\emptyset$.

The target and source cliques of the resources in the graph shown in Figure 9 are shown in Table 1.

Definition 14. (PROPERTY DISTANCE IN A CLIQUE) *The* distance *between two data properties* $\mathtt{p}, \mathtt{p}'$ *in a source (resp. target) clique is:*

- 0 *if there exists a resource in* G *that is the subject (resp. object) of both;*
- *otherwise, the smallest integer* n *such that* G *holds resources* $r_0, \ldots, r_n \in$ G *and data properties* $\mathtt{p}_1, \ldots, \mathtt{p}_n$ *such that* r_0 *is the subject (resp. object) of* p *and* $\mathtt{p}_1$*,* r_1 *is the subject (resp. object) of* $\mathtt{p}_1$ *and* $\mathtt{p}_2$*, ...,* r_n *is the subject (resp. object) of* $\mathtt{p}_n$ *and* $\mathtt{p}'$*.*

In Figure 9, the distance between a and t is 0 since r_1 has both. The distance between a and e is 1, while the distance between a and c is 2. Clearly, p and $\mathtt{p}'$ are at distance n for $n > 0$ iff a resource has both p and $\mathtt{p}''$, and further $\mathtt{p}''$ is at distance $n - 1$ from $\mathtt{p}'$.

Source and target cliques are defined w.r.t. a given graph G. What is the *impact of saturating* G *on the cliques*? In $\mathtt{G}^\infty$, every G resource has all the data properties it had in G, therefore two data properties belonging to a G clique are also in the same clique of $\mathtt{G}^\infty$. Further, if the schema of G comprises $\prec_{sp}$ constraints, a resource may have in $\mathtt{G}^\infty$ a data property that it did not have in G. In turn, this leads to $\mathtt{G}^\infty$ cliques which "subsume and fuse" several G cliques into one.

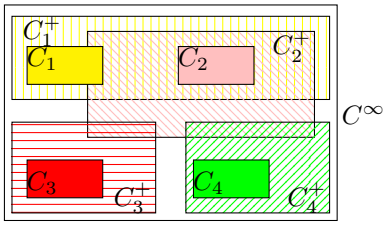


Figure 10: Sample cliques of G, their saturations, and their enclosing clique in $\mathtt{G}^\infty$.

For a given clique C of G, we call *saturated clique* and denote C^+ the of all the properties in C and all their generalizations (superproperties). Observe that C^+ reflects only C and the schema of G; in particular, it does not reflect the data properties shared by resources in $\mathtt{G}^\infty$, and thus in general C^+ is *not* a clique of $\mathtt{G}^\infty$. The following Lemma characterizes the relationships between cliques of G, their saturated versions C^+, and cliques of $\mathtt{G}^\infty$:

Lemma 2 (Saturation vs. property cliques)**.** *Let C, C_1, C_2 be distinct non-empty source (or target) cliques of* G*.*

1. *There exists exactly one source (resp. target) clique C^∞ corresponding to $\mathtt{G}^\infty$ 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* G*. Further,* the above saturated cliques are all connected through intersections*: 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* 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^∞ corresponding to $\mathtt{G}^\infty$ if and only if there exist k non-empty source (resp. target) cliques of* 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 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 p iff there exist some $\mathtt{p}_1 \in C_1$ and $\mathtt{p}_2 \in C_2$ which are specializations of the same p (one, but not both, may also be 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 p, thus these two properties are in the same $\mathtt{G}^\infty$ clique. Similarly, r_2 has $\mathtt{p}_2$ and p, which ensures that 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 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 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 p in $\mathtt{G}^\infty$. Then, in G, r has a property $\mathtt{p}'$ which is either p, or is such that

$\mathtt{p}' \prec_{sp} \mathtt{p}$ 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 $\mathtt{p}' \prec_{sp} \mathtt{p}$.

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.

□

The lemma is crucial as it states the relationship between the cliques (thus, the summaries) of $\mathtt{G}$ and $\mathtt{G}^\infty$; our goal is to build the summary of the latter *without* having access to it, but just access to $\mathtt{G}$. Figure 10 illustrates the lemma. The clique C^∞ of $\mathtt{G}^\infty$ encloses the cliques C_1, C_2, C_3, C_4 of $\mathtt{G}$ (Lemma item 1). C_2^+ intersects all of C_1^+, C_3^+ and C_4^+, thus they are all in the same clique C^∞ (item 2), which is the union of C_1^+, C_2^+, C_3^+ and C_4^+ (item 3). A chain of intersecting cliques connects e.g., C_1^+ through C_2^+ to C_4^+. Item 4 is not illustrated, as individual properties do not show; it follows quite directly from item 3.

6.2 Clique-based RDF node equivalences

Building on property cliques, we define the following equivalences:

Definition 15. (STRONG AND WEAK EQUIVALENCE) Strong *(denoted $\equiv_{\mathtt{S}}$) and* weak *(denoted $\equiv_{\mathtt{W}}$)* equivalences *are RDF node equivalence relations. Two data nodes of* $\mathtt{G}$ *are* strongly equivalent*, denoted $n_1 \equiv_{\mathtt{S}} n_2$, iff they have the same source* and *target cliques.*

Two data nodes 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* $\mathtt{G}$.

Observe that strong equivalence implies weak equivalence.

In Figure 9, the resources r_1, r_2, r_3, r_5 are *strongly* equivalent to each other, as well as t_1, t_2, t_3, t_4. Moreover, $r_1, ..., r_5$ are *weakly* equivalent to each other due to their common source clique SC_1, as well as t_1, t_2, t_3, t_4 due to their common target clique; the same holds for a_1 and a_2, and separately for e_1 and e_2. In general, weakly equivalent resources can be connected as exemplified in Figure 11, by an *alternating sequence of source and target cliques*.

7 Weak-equivalence summaries

In this section, we explore summaries based on the weak equivalence $\equiv_{\mathtt{W}}$ of graph nodes.

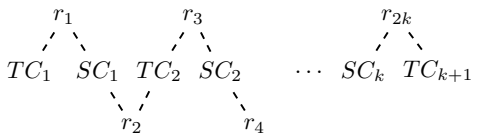


Figure 11: Weakly equivalent resources of G and their cliques.

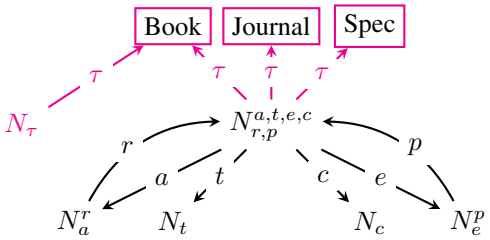


Figure 12: Weak summary of the RDF graph in Figure 9.

7.1 Weak summary

Our first new summary is solely based on weak equivalence:

Definition 16. (WEAK SUMMARY) *The* weak summary *of the graph* G*, denoted* $\mathtt{G}_{/\mathtt{W}}$*, is its quotient graph w.r.t. the weak equivalence relation* $\equiv_{\mathtt{W}}$.

For each set of $\equiv_{\mathtt{W}}$-equivalent G nodes, there is exactly one node in $\mathtt{G}_{/\mathtt{W}}$. Note that the partition of G nodes into $\equiv_{\mathtt{W}}$ equivalence classes is also *a partition of* G *data properties at the source*, that is: the sources of G edges labeled with a given data property p are all weakly equivalent. For instance, in Figure 9, the sources of all "editor" edges are in the weak equivalence class $\{r_1, r_2, r_3, r_4, r_5\}$.

The $\equiv_{\mathtt{W}}$ partition over the set of nodes of G also induces a *partition of the source cliques of* G. Indeed, if $n_1 \equiv_{\mathtt{W}} n_2$, their source cliques are connected through a chain of alternating cliques as sketched in Figure 11; conversely, such a chain, by definition, only connects weakly related resources. By a symmetrical reasoning, the weak equivalence classes of G also lead to a *partition over the target cliques of* G. For instance, in Figure 9, to the equivalent resource set $\{r_1, \ldots, r_5\}$ corresponds the set of source cliques $\{SC_1\}$ and the set of target cliques $\{TC_5\}$. Thus, *to each set* S *of weakly equivalent* G *nodes one can associate through a bijection, a set of* G *source cliques, and a set of* G *target cliques*. This leads to:

Definition 17. (SET REPRESENTATIVE) *Let* N *be any injective function taking as input two sets of URIs, and returning a new URI. Let* S *be an equivalence class of* G *data nodes w.r.t.* $\equiv_{\mathtt{W}}$*. The weak summary node representing all* S *nodes, also called the* set representative *of* S *in the summary, is:* $N(\bigcup_{r\in S} TC(r), \bigcup_{r\in S} SC(r))$.

In the above, N is called on: the set of data properties from all the target cliques of S nodes; and the set of data properties from all the source cliques of S nodes. We will use N to denote any function which assigns URIs to nodes in quotient (RDF) graphs.

Notations. We use N_r to denote the weak summary node representing a resource $r \in S$, and N^{TC}_{SC} to denote $N(TC, SC)$. For simplicity, we will mostly *omit the set delimiters* when showing TC and SC, and omit one such set altogether if it is empty.

For any resource $r \in \mathtt{G}$ which has types and a non-empty source or target clique, its types are carried to N_r in the weak summary.

In the particular case where a data resource $r \in \mathtt{G}$ is neither the source nor the target of data properties, i.e., $TC(r) = SC(r) = \emptyset$ (thus r can only appear in τ triples), r is represented by $N(\emptyset, \emptyset)$ which we

denote N_τ in the sequel. Observe that if a resource r such that $TC(r) = SC(r) = \emptyset$ has types, the weak summary carries the respective types to N_τ.

The weak summary of the graph in Figure 9 is shown in Figure 12. Its nodes are: $N^{a,t,e,c}_{r,p}$ for the relatedness partition set $\{r_1, \ldots, r_5\}$. The target properties of this node are $TC(r_4)$ since the other nodes have empty target clique; the source properties are those in $SC(r_1)$ which is also the source clique of all the other resources in the set; N^r_a for the set $\{a_1, a_2\}$; N_t for the relatedness partition set $\{t_1, t_2, t_3, t_4\}$ etc. The edges from $N^{a,t,e,c}_{r,p}$ to N_a and N_t copy the outgoing edges of r_1, represented by $N^{a,t,e,c}_{r,p}$; the edge to N^p_e is due to r_2 and r_3; the edge to N_c is due to r_3. The edge from N^r_a to $N^{a,t,e,c}_{r,p}$ is due to a_1, and the edge from N^p_e to $N^{a,t,e,c}_{r,p}$ is due to e_1. The τ edges outgoing $N^{a,t,e,c}_{r,p}$ are due to the resources r_1, r_2 and r_3; the creation of N_τ (shown in purple font) is due to the node r_6 in the original graph.

The weak summary has the following important properties:

Property 3. (UNIQUE DATA PROPERTIES) *Each* $\mathtt{G}$ *data property appears exactly once in* $\mathtt{G}_{/\mathtt{W}}$.

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 $\mathtt{p}$ existed, let TC be the target clique it belongs to. By the definition of the weak summary, n_1 was created by a call to $N(UTC_1, USC_1)$ such that TC is included in the union of target cliques UTC_1. Similarly, n_2 was created by a call to $N(UTC_2, USC_2)$ such that TC is included in the union of target cliques UTC_2. Then, $UTC_1 \cap UTC_2 \supseteq TC \neq \emptyset$, which contradicts the fact that different equivalence classes of $\mathtt{G}$ 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 $\mathtt{G}$ also appears in the summary. Thus, there is exactly one $\mathtt{p}$-labeled edge in $\mathtt{G}_{/\mathtt{W}}$ for every data property in $\mathtt{G}$. □

Importantly, the above Property 3 warrants that the number of data edges in $\mathtt{G}_{/\mathtt{W}}$ is exactly $|\mathtt{D}_\mathtt{G}|^0_\mathtt{p}$, the number of distinct data properties in $\mathtt{G}$. Thus, its number of data nodes is at most $2|\mathtt{D}_\mathtt{G}|^0_\mathtt{p}$. The number of type triples in $\mathtt{G}_{/\mathtt{W}}$ is bound by $min(|\mathtt{T}_\mathtt{G}|_e, 2|\mathtt{D}_\mathtt{G}|^0_\mathtt{p} * |\mathtt{T}_\mathtt{G}|^0_\mathtt{o}|)$: the latter corresponds to the case when every data node in $\mathtt{G}_{/\mathtt{W}}$ is of every type in $\mathtt{T}_\mathtt{G}$.

The next important property enjoyed by weak summaries is the existence of *shortcuts*, in the sense of Definition 12. Figure 13 exemplifies this, by tracing the transformation of a graph $\mathtt{G}$ on one hand, into $\mathtt{G}_{/\mathtt{W}}$, $(\mathtt{G}_{/\mathtt{W}})^\infty$, and then $((\mathtt{G}_{/\mathtt{W}})^\infty)_{/\mathtt{W}}$; and on the other hand, from $\mathtt{G}$ to $\mathtt{G}^\infty$ then $(\mathtt{G}^\infty)_{/\mathtt{W}}$. The graph at the bottom right in the figure is at the same time $((\mathtt{G}_{/\mathtt{W}})^\infty)_{/\mathtt{W}}$ and $(\mathtt{G}^\infty)_{/\mathtt{W}}$.

Showing that $\mathtt{G}_\mathtt{W}$ admits a shortcut is rather involved. For this, we rely on the next Proposition:

Property 4. (SAME CLIQUES-W) $\mathtt{G}^\infty$ *and* $(\mathtt{G}_{/\mathtt{W}})^\infty$ *have identical source clique sets, and identical target cliques sets. Further, any node* n *in* $\mathtt{G}^\infty$ *has the source clique* SC *and target clique* TC *iff* $f(n)$ *in* $(\mathtt{G}_{/\mathtt{W}})^\infty$ *has the source clique* SC *and target clique* TC*, with* f *the representation function (recall Proposition 1).*

Proposition 4 is established based on Proposition 1, and on the following Lemma:

Lemma 3. *Data properties are target-related (resp. source-related) in* $(\mathtt{G}_{/\mathtt{W}})^\infty$ *iff they are target-related (resp. source-related) in* $\mathtt{G}^\infty$.

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 a same target clique $TC^\infty_\mathtt{W}$ in $(\mathtt{G}_{/\mathtt{W}})^\infty$.

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

- For every $1 \leq j \leq m$:

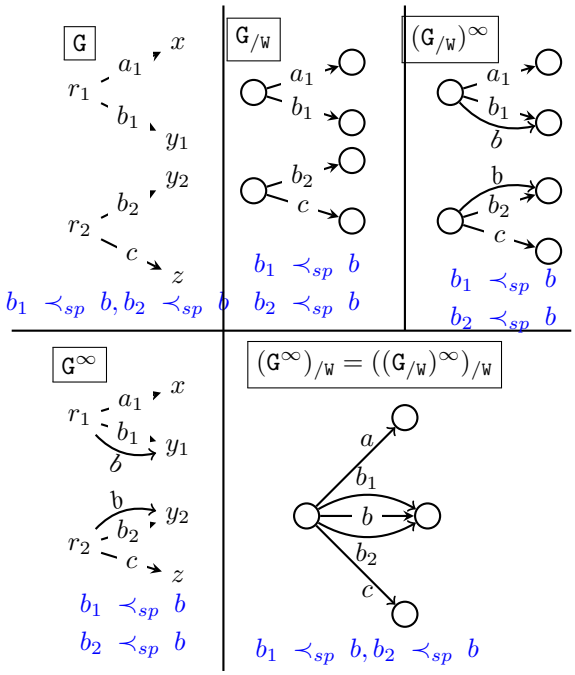


Figure 13: Weak summary shortcut illustration (summary nodes shown as unlabeled circles).

  - $TC^j_{\mathtt{W}}$ 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 $TC^j_{\mathtt{W}}$. Thus, the properties in $TC^j_{\mathtt{W}}$ are target-related in $\mathtt{G}$.
  - Thus, in $\mathtt{G}^\infty$, also, the properties in $TC^j_{\mathtt{W}}$ are target-related.
  - From this and the definition of a saturated graph and of a saturated target clique, it follows that the properties from $(TC^j_{\mathtt{W}})^+$ are target-related in $\mathtt{G}^\infty$.

- Further, still by Lemma 2, point 4, each $(TC^j_{\mathtt{W}})^+$ intersects at least another $(TC^l_{\mathtt{W}})^+$ for $1 \leq l \neq j < m$, thus the target properties in all the $(TC^j_{\mathtt{W}})^+$ 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 $TC^\infty_{\mathtt{W}}$.

"If": if data properties are target-related in $\mathtt{G}^\infty$, then they belong to a 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 Proposition 1, each image of $f(n_i) = N(\cup_{r \in S_{n_i}} TC(r), \cup_{r \in S_{n_i}} SC(r))$ 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 $(\mathtt{G}_{/\mathtt{W}})^\infty$. □

The existence of a shortcut for $\mathtt{W}$-summaries, stated by Theorem 3 below, is then shown based on Proposition 4 by proving that $\mathtt{W}$-summaries enjoy the sufficient condition stated by Theorem 1.

Theorem 3 ($\mathtt{W}$-completeness)**.** *Weak summaries are complete.*

Proof. We show that $\mathtt{W}$ summaries enjoy the sufficient condition for completenes stated in Theorem 1. That is, 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 Proposition 1), let us prove that: $n_1 \equiv_{\mathtt{W}} n_2$ in $\mathtt{G}^\infty$ iff $f(n_1) \equiv_{\mathtt{W}} f(n_2)$ in $(\mathtt{G}_{/\equiv_{\mathtt{W}}})^\infty$.

Two summary nodes are weakly-equivalent if they have the same non-empty source or target clique. The claim to prove hence immediately follows from Property 4, which guarantees that n_1 and $f(n_1)$, resp. n_2 and $f(n_2)$, have the same source and target cliques. □

Weak summary size. The size of the weak summary is at most that of the original graph. This bound is reached if all nodes have distinct source and target properties. Moreover, Property 3 entails that the number of data edges in $\mathtt{G}_{/\mathtt{W}}$ is exactly the number of distinct data properties in G. Thus, its number of data nodes is at most twice this number. The number of type triples in $\mathtt{G}_{/\mathtt{W}}$ is bound by the smallest between the number of type edges in G.

7.2 Typed weak summary

In this section, we introduce a variant of the W-summary, based on the simple idea that RDF type information, when available, can be used to consider equivalent resources having the same *set of types* (recall that an RDF node may have 0, 1 or more types, not necessarily related). This summary represents all G nodes having the exact same types together, while for untyped resources, it reverts to the quotient by the weak equivalence relation $\equiv_{\mathtt{W}}$. Such type-conditioned summarization can be seen as applying two consecutive quotients to G. We formalize this as follows:

Definition 18. (TYPED EQUIVALENCE) Typed equivalence*, noted $\equiv_{\mathtt{T}}$, is an RDF node equivalence relation defined as follows: two data nodes n_1 and n_2 are type-equivalent, noted $n_1 \equiv_{\mathtt{T}} n_2$, iff they have exactly the same set of types in* G*, which is non-empty.*

Definition 19. (TYPED WEAK SUMMARY) *Let $\equiv_{\mathtt{UW}}$ (*untyped weak equivalence*) be an RDF node equivalence relation that holds between two nodes n_1, n_2 iff (i) n_1, n_2 have no types in* G *and (ii) $n_1 \equiv_{\mathtt{W}} n_2$. The* typed weak summary $\mathtt{G}_{/\mathtt{TW}}$ *of an RDF graph* G *is the untyped-weak summary of the type-based summary of* G*, namely $(\mathtt{G}_{/\mathtt{T}})_{/\mathtt{UW}}$.

Untyped weak equivalence is the weak equivalence $\equiv_{\mathtt{W}}$ restricted to untyped resources only. The typed-weak summary $\mathtt{G}_{/\mathtt{UW}}$ treats the G triples whose subject and object lack types exactly like $\mathtt{G}_{/\mathtt{W}}$, and groups typed resources by their types. For example, Figure 14 shows the typed weak summary of the RDF graph in Figure 9. It represents all editors with a single node, all titles with a single node etc., like $\mathtt{G}_{/\mathtt{W}}$ (Figure 12). However, in contrast with $\mathtt{G}_{/\mathtt{W}}$, $\mathtt{G}_{/\mathtt{TW}}$ represents publications by three distinct nodes, depending on their types.

Property 5 (No shortcut for $\mathtt{G}_{\mathtt{TW}}$)**.** *Typed weak summaries do not admit a shortcut.*

Figure 15 shows a counter-example. In G and $\mathtt{G}_{/\mathtt{TW}}$, resources are untyped; a typed resource only appears due to saturation and the constraint $a \hookleftarrow_d c$. Thus, in $\mathtt{G}_{/\mathtt{TW}}$, one (untyped) node represents all data property subjects; in $(\mathtt{G}_{/\mathtt{TW}})^{\infty}$, this (single) node gains a type, and in $((\mathtt{G}_{/\mathtt{TW}})^{\infty})_{/\mathtt{TW}}$, it is represented by a single node. In contrast, in $\mathtt{G}^{\infty}$, one resource is typed and the other one isn't, leading to distinct nodes in $(\mathtt{G}^{\infty})_{/\mathtt{TW}}$. This cannot be isomorphic with the result obtained from G if one starts by TW summarization, thus by Definition 12, $\mathtt{G}_{/\mathtt{TW}}$ does not admit a shortcut.

Typed weak summary size. The size of the typed weak summary is at most that of the original graph. This bound is reached if all nodes have distinct types.

8 Strong equivalence summaries

We now present summaries based on the strong RDF node equivalence. Strong summaries (Section 8.1) rely on it alone, while typed strong summaries (Section 8.2) give preeminence to types.

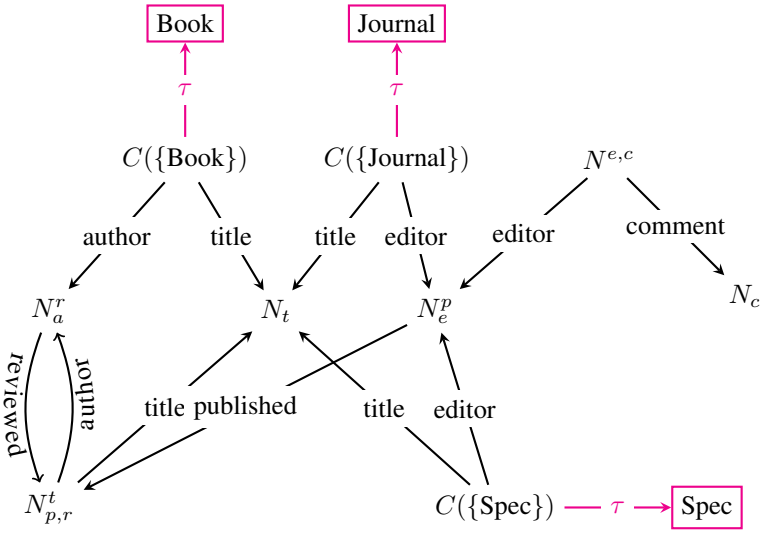


Figure 14: Typed weak summary $\mathtt{G}_{/\mathtt{TW}}$ of the graph $\mathtt{G}$ in Figure 9.

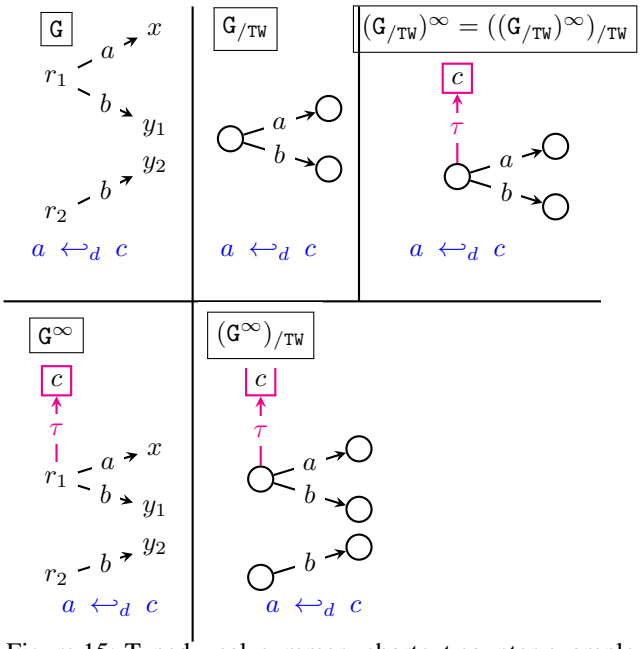


Figure 15: Typed weak summary shortcut counter-example.

8.1 Strong summary

Definition 20. (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* $\equiv_{\mathtt{S}}$.

Recall from the definition of $\equiv_{\mathtt{S}}$ that only data nodes having the same source clique and the same target clique are equivalent in this sense. Thus, it is easy to establish a bijection between any pair (source clique, target clique) of a data node in $\mathtt{G}$, and a node in the strong summary. To follow through this

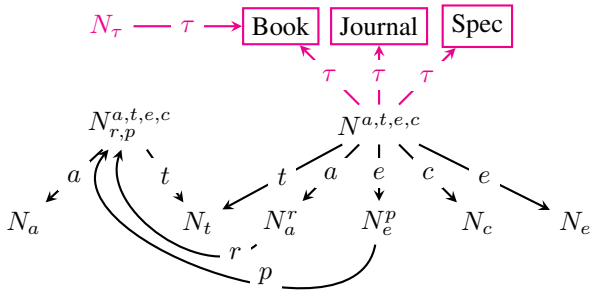


Figure 16: Strong summary of the RDF graph in Figure 9.

intuition, in this section, we denote by N^{TC}_{SC} the node representing the set of $\equiv_{\mathtt{S}}$-equivalent nodes of $\mathtt{G}$ having the target clique TC and the source clique SC.

For example, the strong summary of the graph of Figure 9 is shown in Figure 16. The strong summary comprises the nodes:

- $N(\emptyset, SC_1) = N^{a,t,e,c}$, for r_1, r_2, r_3 and r_5;
- $N(TC_5, SC_1) = N^{a,t,e,c}_{r,p}$, for r_4;
- $N(TC_1, SC_2) = N^a_r$, for a_1;
- $N(TC_2, \emptyset) = N_t$, for t_1, t_2, t_3 and t_4;
- $N(TC_3, SC_3) = N^p_e$, for e_1;
- $N(TC_3, \emptyset) = N_e$, for e_2;
- $N(TC_4, \emptyset) = N_c$, for c_1;
- $N(TC_1, \emptyset) = N_a$, for a_2;
- $N(\emptyset, \emptyset) = N_\tau$ for r_6;
- Book, Journal, and Spec are copied from the schema of $\mathtt{G}$.

As a result, the number of data nodes in the data component $\mathtt{D}_{\mathtt{S}_\mathtt{G}}$ is bound by $min(|\mathtt{D}_\mathtt{G}|_n, (|\mathtt{D}_\mathtt{G}|^0_e)^2)$ (recall the notations introduced in Section 2). Indeed, $\mathtt{S}_\mathtt{G}$ cannot have more data nodes than $\mathtt{G}$; also, it cannot have more nodes than the number of source cliques times the number of target cliques, and each of these is upper-bounded by $|\mathtt{D}_\mathtt{G}|^0_e$. By a similar reasoning, the number of data triples in $\mathtt{S}_\mathtt{G}$ is bound by $min(|\mathtt{D}_\mathtt{G}|_e, (|\mathtt{D}_\mathtt{G}|^0_e)^4)$. In the worst case, $\mathtt{T}_{\mathtt{S}_\mathtt{G}}$ has as many nodes (and triples) as $\mathtt{T}_\mathtt{G}$; $\mathtt{S}_{\mathtt{S}_\mathtt{G}}$ is identical to $\mathtt{S}_\mathtt{G}$.

Similarly to weak summaries, strong summaries admit a shortcut (Definition 12); this is our last major formal result. We prove this based on counterparts of statements established for $\mathtt{G}_\mathtt{W}$:

Property 6. (SAME CLIQUES-S) *$\mathtt{G}^\infty$ and $(\mathtt{G}_{/\mathtt{S}})^\infty$ have identical source clique sets, and identical target cliques sets. Further, any node n in $\mathtt{G}^\infty$ has the source clique SC and target clique TC iff $f(n)$ in $(\mathtt{G}_{/\mathtt{S}})^\infty$ has the source clique SC and target clique TC, with f the representation function (recall Proposition 1).*

The above Proposition follows directly from Proposition 1 and the following Lemma:

Lemma 4. *Data properties are target-related (resp. source-related) in* $(\mathtt{G}_{/\mathtt{S}})^{\infty}$ *iff they are target-related (resp. source-related) in* $\mathtt{G}^{\infty}$.

Proof. "If": if data properties are target-related in $\mathtt{G}^{\infty}$, then they belong to a 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 Proposition 1, each image of $f(n_i) = N(TC(n_i), SC(n_i))$ 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 a 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 (recall Definition 14) which all have the target clique $TC^{\mathtt{S},\infty}$. In $\mathtt{G}_{/\mathtt{S}}$, each such r_{i+1} has a target clique $TC_i^{\mathtt{S}} \subseteq TC^{\mathtt{S},\infty}$, moreover each r_{i+1} results from a set of $\mathtt{G}$ nodes $n_{i+1}^j, j \geq 1$, which by definition of a strong RDF summary, have all the source clique $TC_i^{\mathtt{S}}$. Hence, every such n_{i+1}^j 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. □

Based on Proposition 6, we prove that $\mathtt{G}_{\mathtt{S}}$ enjoys the sufficient condition stated by Theorem 1:

Theorem 4 ($\mathtt{G}_{\mathtt{S}}$ shortcut)**.** *Strong summaries admit a shortcut.*

Proof. We show that $\mathtt{S}$ summaries enjoy the sufficient shortcut condition stated in Theorem 1. That is, given two nodes n_1, n_2 in $\mathtt{G}^{\infty}$, f the representation homomorphism corresponding to the weak-equivalence relation $\equiv_{\mathtt{S}}$, and $f(n_1)$, $f(n_2)$ the images of n_1, n_2 in $(\mathtt{G}_{/\mathtt{S}})^{\infty}$ through f (recall Proposition 1), let us prove that: $n_1 \equiv_{\mathtt{S}} n_2$ in $\mathtt{G}^{\infty}$ iff $f(n_1) \equiv_{\mathtt{S}} f(n_2)$ in $(\mathtt{G}_{/\equiv_{\mathtt{S}}})^{\infty}$.
Two summary nodes are strongly-equivalent if they have the same non-empty source or target clique. The claim to prove hence immediately follows from Property 4, which guarantees that n_1 and $f(n_1)$, resp. n_2 and $f(n_2)$, have the same source and target cliques. □

Figure 17 illustrates this on an example.

Strong summary size. The number of data nodes in the data component $\mathtt{D}_{\mathtt{S}_{\mathtt{G}}}$ is bound by $min(|\mathtt{D}_{\mathtt{G}}|_n, (|\mathtt{D}_{\mathtt{G}}|_e^0)^2)$ (recall the notations introduced in Section 2). Indeed, $\mathtt{S}_{\mathtt{G}}$ cannot have more data nodes than $\mathtt{G}$; also, it cannot have more nodes than the number of source cliques times the number of target cliques, and each of these is upper-bounded by $|\mathtt{D}_{\mathtt{G}}|_e^0$. By a similar reasoning, the number of data triples in $\mathtt{S}_{\mathtt{G}}$ is bound by $min(|\mathtt{D}_{\mathtt{G}}|_e, (|\mathtt{D}_{\mathtt{G}}|_e^0)^4)$. In the worst case, $\mathtt{T}_{\mathtt{S}_{\mathtt{G}}}$ has as many nodes (and triples) as $\mathtt{T}_{\mathtt{G}}$; $\mathtt{S}_{\mathtt{S}_{\mathtt{G}}}$ is identical to $\mathtt{S}_{\mathtt{G}}$.

8.2 Typed strong summary

The summary presented here is the counterpart of the typed weak one from Section 7.2, but based on strong equivalence.

Definition 21. (TYPED STRONG SUMMARY) *Let* $\equiv_{\mathtt{US}}$ *(*untyped strong equivalence*) be an RDF node equivalence relation that holds between two nodes* n_1, n_2 *iff (i)* n_1, n_2 *have no types in* $\mathtt{G}$ *and (ii)* $n_1 \equiv_{\mathtt{S}} n_2$. *The* typed strong summary $\mathtt{G}_{/\mathtt{TS}}$ *of an RDF graph* $\mathtt{G}$ *is defined as:* $(\mathtt{G}_{/\mathtt{T}})_{/\mathtt{US}}$.

Untyped strong equivalence restricts strong equivalence to untyped resources only. The summary $\mathtt{G}_{/\mathtt{TS}}$ summarizes untyped data resources in strong fashion, and groups all typed resources by their types. In our example, it turns out that $\mathtt{G}_{/\mathtt{TS}}$ for the RDF graph in Figure 9 coincides with the $\mathtt{G}_{/\mathtt{TW}}$ one shown in Figure 14. As can be easily seen from their definitions, the type-weak and type-strong summaries behave identically on the triples involving typed resources; on the untyped ones, the difference is of the same nature as the difference between the strong and weak summaries.

Property 7 (No shortcut for $\mathtt{G}_{/\mathtt{TS}}$)**.** *Typed strong summaries do not admit a shortcut.*

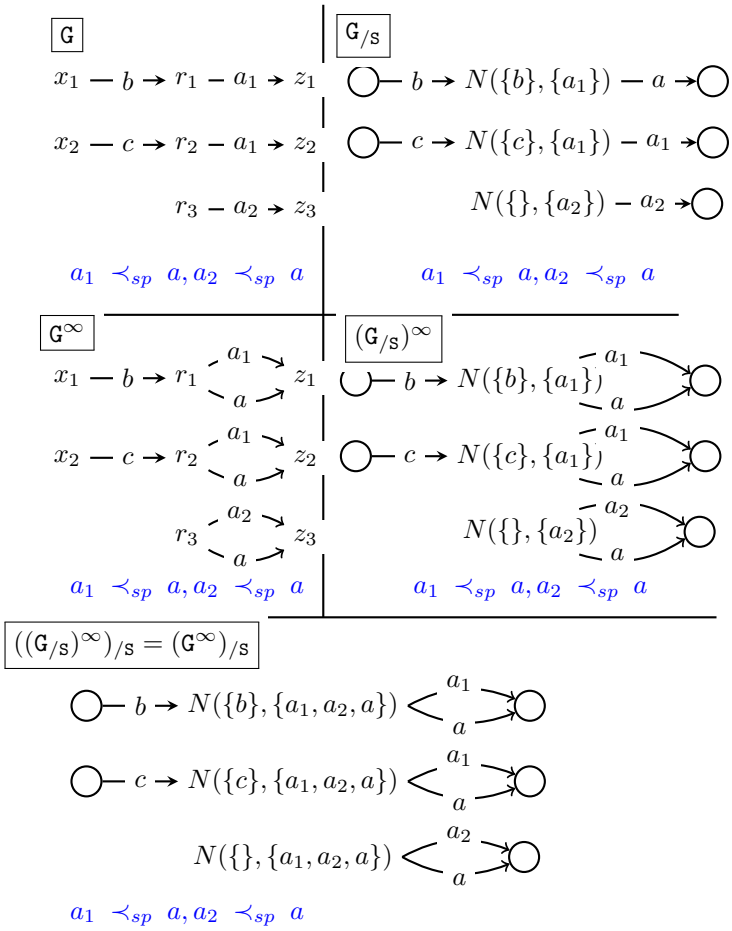


Figure 17: Illustration of the strong completeness statement (some summary nodes are shown as unlabeled circles).

More generally, in the presence of domain ($\hookleftarrow_d$) or range ($\hookrightarrow_r$) RDF schema constraints, one cannot compute $(\mathtt{G}^{\infty})_{/\mathtt{TS}}$ from $\mathtt{G}$ because the $\mathtt{TS}$ summary represents typed resources differently from the untyped ones. The $\hookleftarrow_d$ and $\hookrightarrow_r$ constraints may turn untyped resources into typed ones, thus leading to divergent representations of the data nodes of $\mathtt{G}$ in $\mathtt{G}_{/\mathtt{TS}}$ and respectively $(\mathtt{G}^{\infty})_{/\mathtt{TS}}$. Thus, one cannot build from $\mathtt{G}$, without saturating it, a graph isomorphic to $(\mathtt{G}^{\infty})_{/\mathtt{TS}}$.

Typed strong summary size. Similarly to the type weak summary, the size is at most that of the original graph. This bound is also reached if all nodes have distincts types.

9 Summarization algorithms

We have implemented a summarization tool in Java 1.8 (approx. 34.000 lines) which, given a graph $\mathtt{G}$, builds $\mathtt{G}_{/\mathtt{W}}$, $\mathtt{G}_{/\mathtt{TW}}$, $\mathtt{G}_{/\mathtt{S}}$ and $\mathtt{G}_{/\mathtt{TS}}$ as well as the bisimulation-based RDF summaries $\mathtt{G}_{/\mathtt{fw}}$, $\mathtt{G}_{/\mathtt{bw}}$ and $\mathtt{G}_{/\mathtt{fb}}$.

9.1 Bisimulation summaries

We follow the algorithm from [20] for summarizing the data triples, and add our specific treatment of schema and type triples to obtain representative summaries. $\mathtt{G}$ is initially stored in PostresSQL, in a dictionary-encoded triple table. The bisimulation summary itself is stored in MapDB, allowing to build the summary in memory, backed by disk storage. Inspired by [24], we implement the signature table as a Trie, which we choose to keep in memory.

Algorithm 1 shows how to generate the summary $\mathtt{G}_{\equiv_{\mathtt{fb}}}$ of an RDF graph $\mathtt{G}$. $N^D_{\mathtt{G}}$ and $E_{\mathtt{G}}$ denote the data nodes of $\mathtt{G}$ and the edges of $\mathtt{G}$, while the data nodes, respectively edges of the summary are $N^D_{\mathtt{fb}}$ and $E_{\mathtt{fb}}$. The algorithm is iterative, first representing all $\mathtt{G}$ data nodes by a single summary node, and gradually replacing this initial representation function f_0 by f_1, f_2 etc. until stationarity is reached; $f_k(n)$ is the representation function of a node n in round k. In each round k, we compute the k-th *signature* of a data node n from based on data triples of which n is a subject and/or object, and represent n in round k by a summary node corresponding to this signature. Thus, there is a forward component ($FW_k(n)$) and a backward component ($BW_k(n)$) of the signature, each of which also reflects the current representation of the node at the other end of the triple. The k-th signature of n concatenates $f_{k-1}(n)$, $FW_k(n)$ and $BW_k(n)$. To build $\mathtt{G}_{\equiv_{\mathtt{fw}}}$ or $\mathtt{G}_{\equiv_{\mathtt{bw}}}$, the signature should use only FW_n, or only BW_n, respectively.

9.2 Clique summaries

The clique summaries are built mostly in memory and also using a few SQL queries. The algorithms for building $\mathtt{G}_{\mathtt{W}}$ and $\mathtt{G}_{\mathtt{TW}}$ are significantly different, and more efficient, than what could be designed by simply following the summary definitions; we outline them below. $\mathtt{G}_{/\mathtt{S}}$ and $\mathtt{G}_{/\mathtt{TS}}$ are built in a fashion closer to the definitions [6]. Below, we first introduce the data structures used by the algorithms.

9.3 Data structures

Currently, our summary graphs are built in memory, based on the Trove library[4] providing efficient collection data structures. We represent $\mathtt{G}$ URIs by integers, and store them in a set of map/ multi-map data structures, listed in Figure 18, together with the summaries which use them. All structures are not needed by all summaries, but many of them are used repeatedly. We describe them below, underlined for visibility.

$\underline{\mathtt{rd}}$ maps $\mathtt{G}$ data nodes to their the $\mathtt{G}_{/\equiv}$ representative; the $\underline{\mathtt{dr}}$ multimap stores the reverse mapping (summary node to all $\mathtt{G}$ node it represents). We call *class set* a set of types attached to a same resource

[4] http://trove.starlight-systems.com/

Input : RDF graph $\mathtt{G} = \langle \mathtt{D_G}, \mathtt{S_G}, \mathtt{T_G} \rangle$
Output: Forward and backward bisimulation summary $\mathtt{G}_{\equiv_{\mathtt{fb}}}$
$f_0(n) \leftarrow 0$ for $\forall n \in N^D_{\mathtt{G}}$; $seed \leftarrow 1$; $sigTable \leftarrow \{\}$; $k \leftarrow 0$;
repeat
  $k \leftarrow k + 1$
  for $n \in N^D_{\mathtt{G}}$ **do**
    $f_k(n) \leftarrow$ getRepresentative(n, $\mathtt{D_G}$, k)
  end
until $f_k(n) = f_{k-1}(n)$ *for all* $n \in N^D_{\mathtt{G}}$;
$N^D_{\mathtt{fb}} \leftarrow \{f_k(n) \mid n \in N^D_{\mathtt{G}}\}$; $E_{\mathtt{fb}} \leftarrow \{(f_k(n), f_k(n')) \mid (n, n') \in E_{\mathtt{G}}\}$;
Function getRepresentative($n, \mathtt{D_G}$, k)
  $FW_k(n) = \{(p, f_{k-1}(o)) \mid n\ p\ o \in \mathtt{D_G}\}$, $BW_k(n) = \{(p, f_{k-1}(s)) \mid s\ p\ n \in \mathtt{D_G}\}$
  Sort FW_n and BW_n by their first, then by their second component
  $sig_k(n) = f_{k-1}(n) \| FW_k(n) \| BW_k(n)$ *// $\|$ denotes concatenation*
  $repr \leftarrow sigTable.get(sig_k(n))$ *// look up representative of n in round k*
  if $repr = \perp$ **then**
    $repr \leftarrow seed$; $sigTable.put(sig_k(n), repr)$; $seed \leftarrow seed + 1$
  end
  return $repr$;
endFunction

Algorithm 1: Bisimulation summarization algorithm for $\mathtt{G}_{\equiv_{\mathtt{fb}}}$ based on [20].

	Weak	Typed Weak	Strong	Typed Strong
rd, dr	✓	✓	✓	✓
dcls	✓	✓	✓	✓
clsd		✓		✓
dpSrc, dpTarg	✓	✓		
srcDps, targDps	✓	✓		
dtp	✓	✓	✓	✓
sc, tc			✓	✓
sToSc, oToTc			✓	✓
scToSrc, tcToTarg			✓	✓

Figure 18: Overview of the data structures for different summaries.

in G; to each non-empty class set corresponds exactly one node in $\mathtt{G}_{/\mathtt{TW}}$ and $\mathtt{G}_{/\mathtt{TS}}$. clsd associates the corresponding summary node to each class set, while dcls stores the class set of every (typed) data node from G. In a weak summary $\mathtt{G}_{/\mathtt{W}}$, every data property appears only once (Property 3); in $\mathtt{G}_{/\mathtt{TW}}$, appears *at most once with an untyped source and untyped target* (it may appear several times e.g., with subjects of different types and/or with typed vs. untyped subjects, see the four *title* edges in the $\mathtt{G}_{/\mathtt{TW}}$ summary in Figure 14). The maps dcls, dpSrc and dpTarg associate to a data property, its source and target (in $\mathtt{G}_{\mathtt{W}}$), respectively, its (possible) single untyped source and untyped target (in $\mathtt{G}_{\mathtt{TW}}$). Conversely, for each G data node (for $\mathtt{G}_{\mathtt{W}}$), respectively, for each *untyped* G data node ($\mathtt{G}_{/\mathtt{TW}}$), the sets of data properties it is a source, respectively target of are stored in the maps srcDps (targDps). dtp maps every data property to the summary triple(s) it appears in. Each source and target clique is assigned an integer ID, and the IDs are stored in two lists, sc and tc. sToSc and oToTc associate to each data node, the ID of its source respectively target clique. Finally, for each source (target) clique ID, the $\mathtt{G}_{/\mathtt{S}}$ (and $\mathtt{G}_{/\mathtt{TS}}$) nodes having that clique are stored in scToSrc and tcToTarg.

Building $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{TW}}$ requires a single pass over the data. We create candidate summary nodes as

Input : Data triples table $\mathtt{D_G}$, summary $\mathtt{G_{/W}}$
Output: Data triples represented in $\mathtt{G_{/W}}$
foreach s p o *in* $\mathtt{D_G}$ **do**
  $src \leftarrow$ getSource(s, p, $\mathtt{G_{/W}}$); $targ \leftarrow$ getTarget(o, p, $\mathtt{G_{/W}}$)
  // getTarget may have modified src and vice-versa
  $src \leftarrow$ getSource(s, p, $\mathtt{G_{/W}}$); $targ \leftarrow$ getTarget(o, p, $\mathtt{G_{/W}}$)
  if *!existsDataTriple(*$\mathtt{G_{/W}}$*,* src*,* p*,* $targ$*)* **then**
    createDataTriple($\mathtt{G_{/W}}$, src, p, $targ$)
  end
end
Procedure createDataTriple*(*$\mathtt{G_{/W}}$*,* src*,* p*,* $targ$*)*
  dtp.put(p, src p $targ$)
  dpSrc.put(p, src); srcDps.put(src, p)
  dpTarg.put(p, $targ$); targDps.put($targ$, p)
endProcedure

Algorithm 2: Summarizing data triples in $\mathtt{G_{/W}}$

we go over the G triples, and record the association between G nodes and the respective summary nodes in the maps described above, gradually gaining knowledge about the data properties that each G node is a source and target of. When nodes share a source or target property, their summary representative nodes are fused (again based on Property 3). Building $\mathtt{G_{/S}}$ and $\mathtt{G_{/TS}}$ requires a first pass to compute the cliques, and a second to build a summary node for each (source clique, target clique) pair.

Summarizing data triples in $\mathtt{G_{/W}}$. Algorithm 2 shows how we summarize data triples in a weak summary. The methods $getSource$ and $getTarget$ implement the representation functions, which map data nodes from G to data nodes in $\mathtt{G_{/W}}$. These methods create the respective nodes the first time they are called for a specific data property; on subsequent calls, the respective nodes are returned.

Algorithm 3 shows how we map the subjects of data triples in G to data nodes in $\mathtt{G_{/W}}$. After the method $getSource$ has been executed, the source node of the property p, denoted src_u, and the node representing the subject s, denoted src_s must be the same. If neither src_u nor src_s exist yet, src will be a new data node representing s (line 3). In the cases when one of the nodes exist and the other does not, we simply use the existing node as src (lines 3-3). Moreover, if src_s had not existed, it means that s was unrepresented, so we assign src as its representative (line 3). On the other hand, if both src_u and src_s exist and they are the same, it does not matter which one we choose as src (line 3). When they differ, we have to merge them (line 3). The $mergeDataNodes$ method replaces the node with less edges. The remaining node becomes the source/target of all the edges of the replaced node, and it is assigned to represent all the resources represented by the replaced node. Therefore, this method updates the replaced node in any of the maps rd, dr, dpSrc, srcDps, dpTarg, targDps, with the remaining node. Effectively, merging data nodes that are attached to common properties *gradually builds property cliques*. The method $getTarget$ in Algorithm 2 is very similar to $getSource$, the only difference being in passing the object o instead of the subject, and working with untyped property targets instead of sources.

Algorithm 4 outlines the summarization of type triples in $\mathtt{G_{/W}}$. It iterates over the subjects and classes of all type triples and tries to represent each resulting pair as follows. We look up the summary data node src representing s. If s is attached to any data property in G, this means s has already been represented when summarizing data triples and we assign its representative data node to src and we add the class to the class set of src. Otherwise, s has some types but no data property, thus we record it as a typed-only resource. Once all type triples whose subjects also had some data property have been represented, if there are any typed-only resources, we represent them as well. The procedure representTypedOnly creates the single data node N_τ representing all resources from the list of typed-only resources and having *all* the types of typed-only resources (recall Section 7.1).

Input : s, p, $\mathtt{G}_{/\mathtt{W}}$
Output: Data triples represented in $\mathtt{G}_{/\mathtt{W}}$
Variables: src_u - data node representing an untyped source of p;
src_s - data node representing a (possibly typed) resource s
Procedure getSource(s, p, $\mathtt{G}_{/\mathtt{W}}$)
  $src_u \leftarrow \mathtt{dpSrc}.get(\mathtt{p})$; $src_s \leftarrow \mathtt{rd}.get(\mathtt{s})$
  if $src_u = \perp \wedge src_s = \perp$ **then**
    return createDataNode($\mathtt{G}_{/\mathtt{W}}$, s)
  else if $src_u \neq \perp \wedge src_s = \perp$ **then**
    $\mathtt{rd}.put(\mathtt{s}, src)$, $\mathtt{dr}.put(src, \mathtt{s})$
    return src_u
  else if $src_u = \perp \wedge src_s \neq \perp$ **then**
    return src_s
  else
    if $src_s = src_u$ **then**
      return src_s
    else
      return mergeDataNodes($\mathtt{G}_{/\mathtt{W}}$, src_s, src_u)
    end
  end
endProcedure
Procedure createDataNode($\mathtt{G}_{/\equiv}$, r)
  $d \leftarrow$ newInteger( ); $\mathtt{rd}.put(r, d)$; $\mathtt{dr}.put(d, r)$
  return d
endProcedure

Algorithm 3: Representing the subject s of a data property p in G with a data node in $\mathtt{G}_{/\mathtt{W}}$

9.3.1 Typed weak summmary

Summarizing data triples in $\mathtt{G}_{/\mathtt{TW}}$. The basic procedure for summarizing data triples in $\mathtt{G}_{/\mathtt{TW}}$ (Algorithm 5) is very similar to the one of weak presented in Algorithm 2, with the difference that the entries are stored to the maps only if src and $targ$ respectively, are untyped[5]. Further, the mapping of data nodes in $\mathtt{G}_{/\mathtt{TW}}$ is similar to the one of weak summary; they diverge when both src_s and src_u exist. *The decision on merging data nodes depends on the typing of src_s.* If src_s is typed, or it is untyped but already the same as src_u, we choose src_s as src (lines 5-5). On the other hand, if the two nodes are both untyped and different from each other, we must merge them (line 5). Therefore, in the typed weak summary only untyped data nodes may be merged, while the weak allows for the merging of typed data nodes as well. More precisely, since in the weak summary the data triples are represented *before* the type triples, the typing information is not yet available - the merged nodes may or may not be typed. However, in the typed weak summary the type triples are represented first and taken into account during the summarization of data triples to build a finer-grained summary.

Summarizing type triples in $\mathtt{G}_{/\mathtt{TW}}$. Algorithm 6 shows how we summarize type triples in the typed weak summary. For each distinct subject we retrieve its class set (line 6) and the representative data node of the class set (line 6). Observe that in a typed weak summary there is one data node *per distinct class set* which represents all subjects having the class set. If such a data node already exists for the given class set, we simply store it as the representative of the current subject (line 6). Otherwise, we create a new data node d, which represents s and has the same class set as s (line 6). Further, d is stored as the representative of its class set (line 6).

[5]Since in $\mathtt{G}_{/\mathtt{TW}}$ all typed data nodes are created before untyped ones and the assigned values are sequential, the method $isTyped(d)$ is as simple as checking whether d is less than or equal to the highest typed data node.

Input : Type triples table $\mathtt{T_G}$, the summary $\mathtt{G_{/W}}$
Output: Type triples represented in $\mathtt{G_{/W}}$
$toRes \leftarrow [\,]$; $toCls \leftarrow [\,]$
foreach $(\mathtt{s}, \mathtt{c})$ *in* $\mathtt{T_G}$ **do**
  $repr \leftarrow$ representTypeTriple($\mathtt{G_{/W}}$, s, c)
  if $repr = false$ **then**
    $toRes.add(\mathtt{s})$, $toCls.add(\mathtt{c})$
  end
end
if $toRes.size > 0$ **then**
  representTypedOnly($\mathtt{G_{/W}}$, $toRes$, $toCls$)
end
Procedure representTypeTriple($\mathtt{G_{/W}}$, s, c)
  $d \leftarrow \mathtt{rd}.get(\mathtt{s})$
  if $d = \perp$ **then**
    return false
  end
  $cls_d \leftarrow \mathtt{dcls}.get(d)$, $cls_d.add(d, \mathtt{c})$
  return true
endProcedure
Procedure representTypedOnly($\mathtt{G_{/W}}$, $toRes$, $toCls$)
  $N_\tau \leftarrow$ newInteger( )
  foreach r *in* $toRes$ **do**
    $\mathtt{rd}.put(r, N_\tau)$, $\mathtt{dr}.put(N_\tau, r)$
  end
  $cls_d \leftarrow \mathtt{dcls}.get(N_\tau)$
  foreach c *in* $toCls$ **do**
    $cls_d.add(\mathtt{c})$
  end
endProcedure

Algorithm 4: Summarizing type triples in $\mathtt{G_{/W}}$

Summarizing schema. Prior to summarization, the encoded schema table was created. Since the schema of G and $\mathtt{G}_{/\equiv}$ is the same, no action is needed. **JDBC**. The results of all queries issued to Postgres are fetched through JDBC. A JDBC parameter that can have an impact on the summarization time is the fetch size. Larger fetch size is better because there are less real connections to the database to get all the results. But too large a fetch size will make you wait until that many results are available. The optimal fetch size will vary depending on the hardware setting.

9.3.2 Strong summary

Summarizing data triples in $\mathtt{G_{/S}}$. Instead of building the cliques gradually as in the weak summary, in Algorithm 7 we compute all source and target cliques at the very beginning (lines 1-2). Then in lines 3-15, for each data node d in G, a representative data node d_{repr} is assigned in $\mathtt{G_{/S}}$. This data node d_{repr} must have the exact same source and target clique as d, and there can be at most one such node for any two source and target cliques. Finally, for each data triple s p o in $\mathtt{D_G}$, we retrieve the representative data node of s, denoted src, and of o, denoted $targ$, forming a data triple src p $targ$ in $\mathtt{G_{/S}}$. The method for creating the data triple is the same as for the weak summary in Algorithm 2.

The procedure buildSourceCliques($\mathtt{D_G}$, $\mathtt{G_{/S}}$) invokes the procedure computeSourceClique($\mathtt{G_{/S}}$, s, $P_\mathtt{s}$) shown in Algorithm 8 for each subject in $\mathtt{D_G}$ and the set of data properties of which it is the source.

Input : The subject s of the data triple, the property p of s, the summary $\mathtt{G}_{/\mathtt{TW}}$
Output: Data triples represented in $\mathtt{G}_{/\mathtt{TW}}$
Procedure getSource*(*s, p, $\mathtt{G}_{/\mathtt{TW}}$*)*
  $src_u \leftarrow \mathtt{dpSrc}.get(\mathtt{p})$; $src_s \leftarrow \mathtt{rd}.get(\mathtt{s})$
  if $src_u = \perp \wedge src_s = \perp$ **then**
    return createDataNode($\mathtt{G}_{/\mathtt{TW}}$, s)
  else if $src_u \neq \perp \wedge src_s = \perp$ **then**
    $\mathtt{rd}.put(\mathtt{s}, src)$, $\mathtt{dr}.put(src, \mathtt{s})$
    return src_u
  else if $src_u = \perp \wedge src_s \neq \perp$ **then**
    return src_s
  else
    if *isTyped(*src_s*)* $\vee$ $(src_s = src_u)$ **then**
      return src_s
    else
      return mergeDataNodes($\mathtt{G}_{/\mathtt{TW}}$, src_s, src_u)
    end
  end
endProcedure

Algorithm 5: Representing the subject of a data property in G with a data node in $\mathtt{G}_{/\mathtt{TW}}$

Input : Type triples table $\mathtt{T}_\mathtt{G}$, the summary $\mathtt{G}_{/\mathtt{TW}}$
Output: Type triples represented in $\mathtt{G}_{/\mathtt{TW}}$
$typ \leftarrow$ eval(SELECT s, c FROM $\mathtt{T}_\mathtt{G}$ ORDER BY s)
foreach *distinct* s *in* typ **do**
  $cls_\mathtt{s} \leftarrow \mathtt{dcls}.get(\mathtt{s})$
  $d \leftarrow \mathtt{clsd}.get(cls_\mathtt{s})$
  if $d \neq \perp$ **then**
    $\mathtt{rd}.put(\mathtt{s}, d)$, $\mathtt{dr}.put(d, \mathtt{s})$
  else
    $d \leftarrow$ createDataNode$\mathtt{G}_{/\mathtt{TW}}$, $cls_\mathtt{s}$, s
    $\mathtt{clsd}.put(cls_\mathtt{s}, d)$
  end
end
Procedure createDataNode*(*$\mathtt{G}_{/\mathtt{TW}}$*,* r*,* cls_r*)*
  $d \leftarrow$ newInteger , $\mathtt{rd}.put(r, d)$, $\mathtt{dr}.put(d, r)$, $\mathtt{dcls}.put(d, cls_r)$
  return d
endProcedure

Algorithm 6: Summarizing type triples in $\mathtt{G}_{/\mathtt{TW}}$

Input : Data triples table $\mathtt{D_G}$, the summary $\mathtt{G_{/S}}$
Output: Data triples represented in $\mathtt{G_{/S}}$
Variables: $scId$, $tcId$ - source (target) clique ID; D_{SC}, D_{TC} - the set of source (target) data nodes in $\mathtt{G_{/S}}$ representing the source clique $scId$ (target clique $tcId$)
buildSourceCliques($\mathtt{D_G}$, $\mathtt{G_{/S}}$)
buildTargetCliques($\mathtt{D_G}$, $\mathtt{G_{/S}}$)
foreach *distinct data node* d *in* $\mathtt{D_G}$ **do**
  $scId \leftarrow \mathtt{sToSc}.get(d)$, $tcId \leftarrow \mathtt{oToTc}.get(d)$
  $D_{SC} \leftarrow \mathtt{scToSrc}.get(scId)$, $D_{TC} \leftarrow \mathtt{tcToTarg}.get(tcId)$
  if $D_{SC} \cap D_{TC} \neq \emptyset$ **then**
    $\mathtt{rd}.put(r, d_{repr})$, $\mathtt{dr}.put(d_{repr}, r)$
  else
    $d_{repr} \leftarrow$ createDataNode($\mathtt{G_{/S}}$, d)
    $\mathtt{scToSrc}.put(scId, d_{repr})$, $\mathtt{tcToTarg}.put(tcId, d_{repr})$
  end
end
foreach s p o *in* $\mathtt{D_G}$ **do**
  $src \leftarrow \mathtt{rd}.get(\mathtt{s})$, $targ \leftarrow \mathtt{rd}.get(\mathtt{o})$
  if *!existsDataTriple(*$\mathtt{G_{/S}}$*,* src*,* p*,* $targ$*)* **then**
    createDataTriple($\mathtt{G_{/S}}$, src, p, $targ$)
  end
end

Algorithm 7: Summarizing data triples in $\mathtt{G_{/S}}$

First, in Algorithm 8 (lines 8-8) we try to find an existing source clique, denoted $srcClq$, that shares at least one property with $P_\mathtt{s}$. If such a clique exists we simply add to it all the properties from $P_\mathtt{s}$ (line 8). Otherwise, $srcClq$ is a new set with all the properties of $P_\mathtt{s}$, and we add it to the list of all source cliques sc in $\mathtt{G_{/S}}$ (lines 8-8). Finally, $srcClq$ is assigned to s in sToSc (line 8).

The procedures for building target cliques are very similar to these described for the source cliques, so we omit the discussion.

Summarizing type triples in $\mathtt{G_{/S}}$. The types in $\mathtt{G_{/S}}$ are summarized in the same manner as for the $\mathtt{W_G}$, described in Algorithm 4.

Property 8. (ALGORITHM CORRECTNESS) *Given an RDF graph* G*: Algorithms 2 and 4, together, build the weak summary* $\mathtt{G_{/W}}$*. Algorithms 5 and 6 build the typed weak summary* $\mathtt{G_{/TW}}$*. Algorithm 7 shows how to build the strong summary* $\mathtt{S_G}$*; building* $\mathtt{TS_G}$ *is very similar.*

Algorithm complexity. The complexity of building $\mathtt{G_{/W}}$ and $\mathtt{G_{/TW}}$ is dominated by the cost of merging summary nodes, in the worst case, once per triple in G, leading to $O(|\mathtt{G}| \cdot \mathtt{DPG})$, where DPG is the number of distinct data properties in G. To build $\mathtt{G_{/S}}$ and $\mathtt{G_{/TS}}$, we actually compute the cliques. First, we find the distinct data properties of every distinct data node in G, at a cost of $O(\mathtt{N} \cdot \log(\mathtt{N}) + |\mathtt{G}|)$, where N is the number of data nodes in G. Then, for each data node n, we examine all the cliques known so far, to see if some should be fused due to the data properties of n: in the worst case we may do DPG fusions at a cost of DPG each, thus a total cost of $\mathtt{DPG}^2$. Further, if there are fusions, the associations between the data nodes previously visited and their cliques need to be updated. A (very pessimistic) upper bound for the number of nodes concerned is N. Thus, the complexity of this stage is in $O(\mathtt{N}(\mathtt{N} + \mathtt{DPG}^2))$; this dominates the complexity of the first step and thus dictates the complexity of building $\mathtt{G_{/S}}$ and $\mathtt{G_{/TS}}$. As our experiments show next, the actual scale-up is much better, and close to linear in $|\mathtt{G}|$.

Computing accuracy loss. We developed a tool which enumerates summary subgraphs and for each subgraph, asks an existential SQL query to check if the subgraph has matches in the data. Our algorithm

Input : The summary $\mathtt{G}_{/\mathtt{S}}$, subject $\mathtt{s}$ from $\mathtt{D}_{\mathtt{G}}$, the set of data properties $P_{\mathtt{s}}$ of which $\mathtt{s}$ is the source
Output: $\mathtt{sc}$ list and $\mathtt{sToSc}$ map of $\mathtt{G}_{/\mathtt{S}}$ populated
Procedure computeSourceClique*(*$\mathtt{G}_{/\mathtt{S}}$, $\mathtt{s}$, $P_{\mathtt{s}}$*)*

```
srcClq ←⊥
foreach c in sc do
    if c ∩ P_s ≠ ∅ then
        srcClq ← c
        break;
    end
end
if srcClq ≠⊥ then
    srcClq.addAll(P_s)
else
    srcClq.addAll(P_s)
    sc.add(srcClq)
end
sToSc.put(s, sc.indexOfsrcClq+1)
```

endProcedure

Algorithm 8: Computing source cliques in $\mathtt{G}_{/\mathtt{S}}$

enumerates smallest subgraphs first, and re-uses the knowledge that a certain subgraph has no matches in $\mathtt{G}$ to avoid asking queries that correspond to larger graphs comprising the empty one (as we can ascertain the larger graph has no matches, either).

10 Experiments

We report here on experiments based on our algorithms described in Section 9.

Experiments settings. We stored $\mathtt{G}$ in PostgreSQL v9.6 (encoding URIs and literals as integers for efficiency), while the bisimulation summary is stored in Apache MapDB (`www.mapdb.org`). We used a server with an Intel Xeon CPU E5-2640 v4 @2.40GHz and 124 GB of memory; Postgres had 30 GB of shared buffers and 640 MB working memory. The JVM had 70 GB of RAM.

Datasets. We used *synthetic* RDF graphs (BSBM [2] and LUBM [14] benchmark data) as well as *real-life* graphs: INSEE Geo geographic data (`http://rdf.insee.fr/geo/index.html`) and DBLP. Similar experiment results on other popular datasets appear in [6]. The numbers of data triples, respectively type triples in $\mathtt{G}^{\infty}$ (denoted Gsat data and Gsat type, respectively) appear at the top of Figure 19; note the logarithmic vertical axis. We do not show the $\mathtt{G}^{\infty}$ schema size as it is much smaller (in the hundreds) in all cases.

Baseline: bisimulation summaries. To compare our work, we picked *bisimulation-based RDF summaries* to instantiate in our RDF summarization framework. We chose these as a comparison baseline because (i) they are *the only representative summaries (Definition 1) taking into account the semantics and types of RDF, that we are aware of, beyond the four summaries introduced here*; (ii) bisimulation summaries have been extensively studied in the literature. We omit $\mathtt{G}_{/\mathtt{bw}}$ results for space limitations and because they are between those of $\mathtt{G}_{/\mathtt{fb}}$ and $\mathtt{G}_{/\mathtt{fw}}$ (see below).

Bisimulation summarization and shortcut experiments. We built $\mathtt{G}_{\mathtt{fb}}$, $\mathtt{G}_{\mathtt{fw}}$ and $\mathtt{G}_{\mathtt{bw}}$ for synthetic [2, 14] and real[6] RDF graphs, outlined in Table 2; $|\cdot|$ denotes the size (number of triples) of an RDF graph. We found that $\mathtt{G}_{/\equiv_{\mathtt{fb}}}$ can be almost as large as $\mathtt{G}$, e.g., for BSBM 1M, summarization reduces the number

[6]INSEE Geographic dataset, available at: `http://rdf.insee.fr/geo/index.html`

G	$\|G\|$	$\|G^\infty\|$	$\|G_{/\equiv_{fw}}\|$	$\|(G^\infty)_{/\equiv_{fw}}\|$	$\|S_G\|$	t (s)	t_S (s)	Speed-up
BSBM 1M	1,000,708	1,009,138	7,124	8,076	150	47.95	44.71	**6.75%**
BSBM 10M	10,538,484	10,628,484	16,883	18,833	584	487.54	446.87	**8.34%**
LUBM 1M	1,227,868	1,388,296	1,989	1,989	124	69.81	62.77	**10%**
LUBM 10M	11,990,183	13,562,978	25,002	25,002	124	679.29	599.39	**11.76%**
INSEE Geo	369,042	1,126,239	12,524	26,954	281	145.14	67.59	**53.43%**

Table 2: Summarization experiments.

of triples by only 0.02%! As $G_{/\equiv_{fw}}$ is often much smaller than G, we only report on experiments with $G_{/\equiv_{fw}}$. In Table 2, t is the time to saturate then summarize G, while t_S is the time to compute the same result following the shortcut (green path in Figure 7). The shortcut is faster in all cases, and significantly so (53.43%) for the INSEE dataset where G^∞ is much larger than G; recall also that the shortcut is the only option when one cannot saturate G. Further, we see that $(G^\infty)_{/\equiv_{fw}}$ is 41 to 697 times smaller than G^∞, which confirms that the summary is much easier to handle than G; interactive graph visualization techniques and/or querying (e.g., in the example from the Introduction, ask "find the politicians owning companies") allow to determine the usefulness of G based on its representative summary $(G^\infty)_{/\equiv}$.

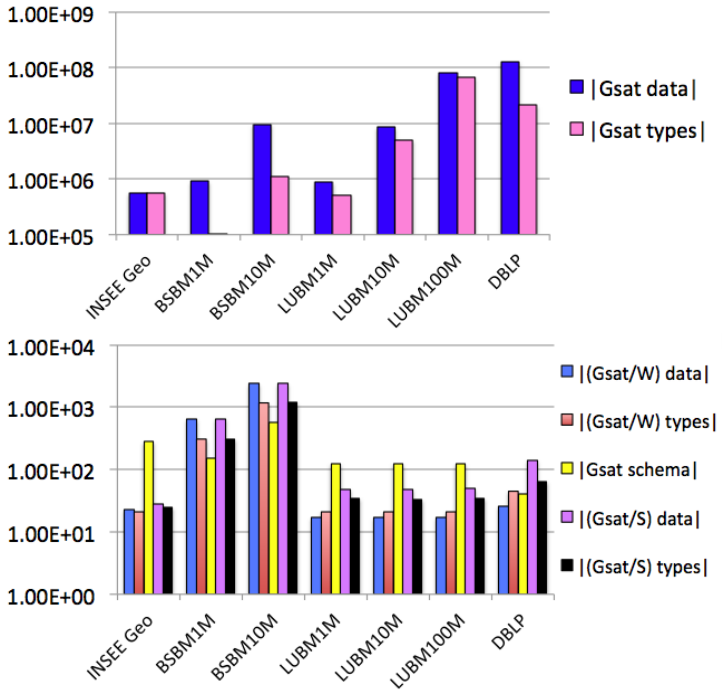


Figure 19: Top: data and type triple counts in G^∞ for various RDF graphs; bottom: data, type and schema triple counts in $(G^\infty)_{/W}$ and $(G^\infty)_{/S}$ for these graphs.

Summaries size. Figure 19 (bottom) shows the number of data, types, and schema triples for $(G^\infty)_{/W}$ and $(G^\infty)_{/S}$ (the size of the saturated schema is the same). We define the **compression factor** $cf_\equiv$ as $|G|/|G_{/\equiv}|$, the ratio between the numbers of triples in G and in the summary. In our experiments, the $G_{/TW}$ is very close or equal to the size of $G_{/W}$, while the $G_{/TS}$ is close to that of $G_{/S}$. Compression factors for the same graphs as above, for our weak and strong summaries as well as for G_{fb} and G_{fw}, also appear in Figure 20. Figure 20 shows that the compression factor for our four summaries is at least 500, often above 10^3, and up to $2 \cdot 10^6$ for DBLP dataset; in our experiments, cf_W was typically the highest. In contrast, since $\equiv_{fb}$ is a very strong condition, $G_{/fb}$ summarizes very little (cf_{fb} rounds to 1 up to 3). Since $\equiv_{fw}$ is

Graph G and $\mathtt{G}_{/\equiv}$	$cf_{\equiv}$	t (s)	t_S (s)	Speed-up
INSEE Geo, $\mathtt{G}_{/\mathtt{W}}$	1,186	21.42	0.66	**96.92%**
INSEE Geo, $\mathtt{G}_{/\mathtt{S}}$	1,164	22.60	1.70	**92.48%**
INSEE Geo, $\mathtt{G}_{/\mathtt{fw}}$	29	148.83	67.76	**54.47%**
INSEE Geo, $\mathtt{G}_{/\mathtt{fb}}$	3	310.54	270.03	**13.04%**
BSBM1M, $\mathtt{G}_{/\mathtt{W}}$	956	3.02	1.30	**56.95%**
BSBM1M, $\mathtt{G}_{/\mathtt{S}}$	937	5.51	3.53	**35.94%**
BSBM1M, $\mathtt{G}_{/\mathtt{fw}}$	140	61.68	58.71	**5.81%**
BSBM1M, $\mathtt{G}_{/\mathtt{fb}}$	1	332.95	628.23	**-88.68%**
LUBM1M, $\mathtt{G}_{/\mathtt{W}}$	7579	7.56	1.39	**81.61%**
LUBM1M, $\mathtt{G}_{/\mathtt{S}}$	5903	9.49	3.11	**67.23%**
LUBM1M, $\mathtt{G}_{/\mathtt{fw}}$	617	86.21	78.74	**8.66%**
LUBM1M, $\mathtt{G}_{/\mathtt{fb}}$	1	314.82	621.14	**-97.29%**
LUBM100M, $\mathtt{G}_{/\mathtt{W}}$	705,897	1633.94	144.65	**91.15%**
LUBM100M, $\mathtt{G}_{/\mathtt{S}}$	547,155	1810.26	557.95	**69.18%**
DBLP, $\mathtt{G}_{/\mathtt{W}}$	2,123,767	1180.04	238.96	**79.75%**
DBLP, $\mathtt{G}_{/\mathtt{S}}$	731,978	1325.06	630.89	**52.39%**
DBLP, $\mathtt{G}_{/\mathtt{fw}}$	574	20521.23	19363.34	**5.64%**

Figure 20: Compression factors, time to generate $(\mathtt{G}^{\infty})_{/\mathtt{W}}$ and $(\mathtt{G}^{\infty})_{/\mathtt{S}}$ directly and through the shortcut for a variety of RDF graphs, and the shortcut speed-up.

a weaker condition, $\mathtt{G}_{/\mathtt{fw}}$ compresses more ($cf_{\mathtt{fw}}$ between 29 and 617), but still much less than our four summaries.

A more detailed look at BSBM summarization is provided in Figure 21, which shows node and edge counts for summaries of BSBM graphs of various sizes. The horizontal axis is labeled in input triples; the vertical axis is in log scale. We plot the number of class nodes of G next to the summary data nodes, and the number of schema edges next to the edge counts. The figure shows that $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{S}}$ strongly summarize graph structure, while $\mathtt{G}_{/\mathtt{TW}}$ and $\mathtt{G}_{/\mathtt{TS}}$ which isolate typed from untyped data nodes lead to larger, more complex summaries.

Clique summarization time and scale-up. Figure 22 shows the average time (in seconds, 3 executions) to build clique summaries; they go up to 800 seconds for 100M triples. Times for the LUBM and BSBM synthetic data are also plotted in the lower part of the figure: both axes are in logarithmic scale, the horizontal axis labeled in $|\mathtt{G}|$ triples). The graphs show that summarization time grows almost linearly, and the time to build $\mathtt{G}_{/\mathtt{W}}$ grows fastest. This is because building $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{S}}$ maximizes the number of nodes for which cliques are analyzed and fused; in contrast, when building $\mathtt{G}_{/\mathtt{TW}}$ and $\mathtt{G}_{/\mathtt{TS}}$, *the cliques of typed data nodes are never constructed (directly or indirectly)*. $\mathtt{G}_{/\mathtt{S}}$ and $\mathtt{G}_{/\mathtt{TS}}$ often take longer to build than $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{TW}}$; this is because of the extra effort needed to build the cliques, while $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{TW}}$ use a more direct, faster method. We find summarization times acceptable, also considering that this is an off-line task; if speed-up is important, our algorithms can easily be parallelized to take advantage of a parallel processor (as in [20]) or a parallel computing framework such as Spark.

Shortcut experiments. Figure 20 also shows the time to build $(\mathtt{G}^{\infty})_{/\equiv}$ in two ways: (i) saturate G then summarize, denoted t, and (ii) summarize G, saturate the summary, and summarize again, denoted t_S. We define the shortcut **speed-up** as $(t - t_S)/t$ and report it also in the figure. The speed-up is highest for $\mathtt{G}_{/\mathtt{W}}$ (up to almost 97%) and $\mathtt{G}_{/\mathtt{S}}$ (up to 97%): this is a direct consequence of their good compression. Indeed, t includes the time to summarize $\mathtt{G}^{\infty}$, while t_S includes the time to summarize $(\mathtt{G}_{/\equiv})^{\infty}$. If the latter is much smaller, t_S is smaller and the speed-up is positive. The much more modest compression factor of $\mathtt{G}_{/\mathtt{fw}}$ translates into a modest speed-up (5% to 13%), while for $\mathtt{G}_{/\mathtt{fb}}$, the speed-up is negative! This is because with a basically unnoticeable compression through $\equiv_{\mathtt{fb}}$, there is no time gain to summarize $(\mathtt{G}_{/\mathtt{fb}})^{\infty}$ instead of $\mathtt{G}^{\infty}$. Thus, the overhead of the *three* shortcut steps (instead of two on the other path) is not offset, that is, $t < t_S$.

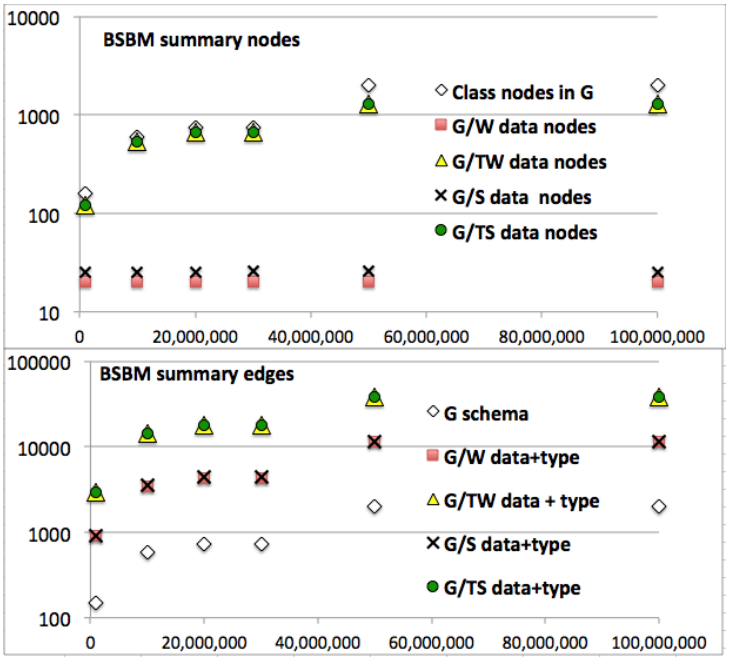


Figure 21: Summary node counts (top) and edge counts (bottom) for BSBM graphs.

Experiment conclusion. Our experiments have shown that our four summaries can be built efficiently, and they reduce graph sizes by factors hundreds up to $2 \cdot 10^6$; they are *two to three orders of magnitude more compact than bisimulation-based summaries*; $\mathtt{G}_{/\mathtt{W}}$ was *the most compact and representative RDF summary*, and the other three summaries we introduced are close. Finally, among the summaries which admit a shortcut ($\mathtt{G}_{/\mathtt{W}}$, $\mathtt{G}_{/\mathtt{S}}$, $\mathtt{G}_{/\mathtt{fb}}$ and $\mathtt{G}_{/\mathtt{fw}}$), *the shortcut speed-up is up to 96% for* $\mathtt{G}_{/\mathtt{W}}$, trailed by $\mathtt{G}_{/\mathtt{S}}$, while for $\mathtt{G}_{/\mathtt{fw}}$ the speed-up is small and for $\mathtt{G}_{/\mathtt{fb}}$ it is negative (the shortcut has no interest here). Despite their polynomial complexity, in practice, our algorithms scale basically linearly. Overall, if summary size or building time are a critical concern, we recommend using $\mathtt{G}_{/\mathtt{W}}$; otherwise, $\mathtt{G}_{/\mathtt{S}}$ may give finer-granularity information.

11 Related Work

Graph summarization has a long history and many applications.

Non-RDF graph summaries ignore graph semantics, thus representativeness, a main benefit of our summaries, is not guaranteed. Dataguides [13] (including "strong" ones) are not quotients, as a graph node may be represented by *more than one* dataguide node, thus a dataguide may be larger than the original graph, and its construction has worst-case exponential time complexity in the size of $\mathtt{G}$. [25, 9, 17, 11, 24] build quotient summaries based on (sometimes unidirectional, sometimes k-bounded) bisimulation, which, as shown in Section 4.1, prevents representativeness. Further, we observed and confirmed by our experiments in the preceding section, that such summaries do not achieved significant graph compression. [12] considers reachability and graph pattern queries on labeled graphs, and builds *answer-preserving summaries* for such queries; this property subsumes representativeness. However, their query semantics is not based on graph homomorphisms, which underlie SPARQL and the dialects we use, but on *bounded graph simulation*. Under this semantics, answering a query becomes P (instead of NP), at the price of not preserving the query structure (i.e., joins), which our representativeness preserves (Proposition 1).

More distant from our work, [8] introduces an *aggregation* framework for OLAP on labeled graphs, whereas we focus on representing graph strucuture and semantics. [7] builds a set of randomized sum-

Dataset	$\|G\|$	$G_{/W}$	$G_{/S}$	$G_{/TW}$	$G_{/TS}$
John Peel	271k	0.33	1.37	0.43	0.98
INSEE Geo	369k	0.58	1.65	0.55	1.50
DBpedia	8M	9.73	25.85	6.82	21.80
DBLP	150M	434.88	636.56	175.87	12412.49
LinkedCT	42M	74.55	314.75	88.73	252.38
LinkedMDB	6.14M	1.12	3.49	0.90	3.06
LUBM	1M	1.34	3.30	1.33	3.20
LUBM	10M	13.92	28.27	10.42	31.14
LUBM	100M	221.62	403.97	132.89	457.63
WatDiv	10M	14.78	22.77	8.33	21.02

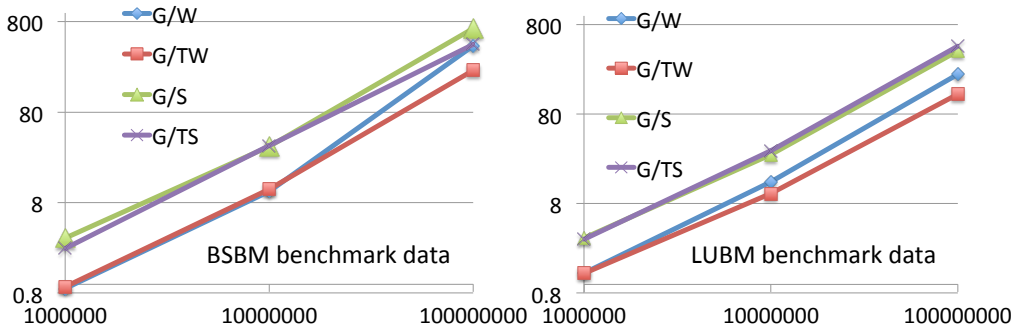


Figure 22: Summarization times (s) and their scale-up with the input size, for various RDF datasets.

maries to be *mined* instead of the original graph for better performance, with guaranteed bounds on the information loss. Graph compression with bounded "error" (number of edges to be added as "corrections" after decompression, to retrieve the original graph) is studied in [26, 18]. Quantitative summarization, where nodes and edges are summarized by their counts, is the focus of [22]. [23] surveys many other quantitative, mining-oriented graph sampling and summarization methods.

RDF summarization [10, 30, 31] study bisimulation-based RDF quotient summaries, in particular providing efficient parallel summarization algorithms [10] and showing they are representative for data-only queries [30]. However, these summaries ignore the special role type and schema triples play in RDF, and therefore cannot guarantee representativeness for the implicit data and type-aware queries. We have included bisimulation-based RDF summaries in our framework, and showed they admit our shortcut.

[28] introduces a *simulation* RDF quotient based on *triple (not node) equivalence*. (Intuitively, simulation is a unidirectional "half" of bisimulation, e.g., replace "iff" by "if" in the bisimilarity condition). For compactness, they bound the simulation; this (and other aspects of their proposal) prevents representativeness.

Other proposals are based on clustering [15] user-defined aggregation rules [29], mining [7], and identification of frequent subtrees [36]; they are not representative in our sense either. Finally, some summaries record structural or statistical aspects about an RDF graph: [19] uses forward bisimulation to characterize usage frequency of classes and properties from some namespaces, while [27] computes statistics reflecting data property frequence, the most specific types of properties' subjects/objects etc. None of the abovementioned RDF summary proposals ensures representativeness.

We had demonstrated [4] and (informally) presented $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{TW}}$ in [3], with procedural definitions (not as quotients); unfortunately, the shortcut described there was incorrect. In this work, we present the first formalization of $\mathtt{G}_{/\mathtt{W}}$, $\mathtt{G}_{/\mathtt{TW}}$, and introduce the novel $\mathtt{G}_{/\mathtt{S}}$ and $\mathtt{G}_{/\mathtt{TS}}$, based on the summarization framework described in [6, 5]. Further, we show that $\mathtt{G}_{/\mathtt{S}}$, $\mathtt{G}_{/\mathtt{W}}$ admit our shortcut, and experimentally demonstrate that these are several orders of magnitude more compact than their comparable (representative) RDF summaries, namely $\mathtt{G}_{/\mathtt{fw}}$ and $\mathtt{G}_{/\mathtt{fb}}$; the shortcut also translates into dramatic speed-up factors for summarizing $\mathtt{G}^{\infty}$. This demonstrates their strong theoretical properties and practical interest.

12 Conclusion

We studied the problem of building summaries of RDF graphs, accounting for their structure and semantics; in particular, we focused on *quotient* graph summaries, widely studied in the literature. We defined a framework for representative summarization of RDF graph, based on RDF equivalence relations carefully crafted to reflect the particular role schema and types play in RDF. For efficiency, and to obtain the complete summary $(\mathtt{G}^{\infty})_{/\equiv}$ even when we are not able to saturate $\mathtt{G}$, we exhibit a shortcut (Theorem 1) and provide a sufficient condition for the shortcut to hold on a given RDF summary. Further, we introduced a new concept of *property cliques* out of which we builds four novel*clique summaries*, two of which (as well as bisimulation-based ones) can be *efficiently* computed through our shortcut method. All our clique summaries are *representative* w.r.t. a large and useful dialect of SPARQL; they can be built efficiently and achieve RDF graph compression factors of hundreds to $2 \cdot 10^6$; the highest values are reached for real-life datasets. Future work will improve scalability by leveraging a massively parallel platform such as Spark. We are also interested in devising advanced visualizations of our RDF summaries.

References

[1] RDF Summary project website: https://team.inria.fr/cedar/projects/rdfsummary, 2017.

[2] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[3] Š. Čebirić, F. Goasdoué, and I. Manolescu. Query-oriented summarization of RDF graphs. In *BICOD*, 2015.

[4] Š. Čebirić, F. Goasdoué, and I. Manolescu. Query-oriented summarization of RDF graphs (demonstration). *PVLDB*, 8(12), 2015.

[5] Š. Čebirić, F. Goasdoué, and I. Manolescu. Efficient representative summarization of RDF graphs. Submitted for publication, 2017.

[6] Š. Čebirić, F. Goasdoué, and I. Manolescu. Query-Oriented Summarization of RDF Graphs. Research Report RR-8920, INRIA, 2017. https://hal.inria.fr/hal-01325900/file/RR.pdf.

[7] C. Chen, C. X. Lin, M. Fredrikson, M. Christodorescu, X. Yan, and J. Han. Mining graph patterns efficiently via randomized summaries. *PVLDB*, 2(1), 2009.

[8] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu. Graph OLAP: towards online analytical processing on graphs. In *ICDM*, 2008.

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

[10] M. P. Consens, V. Fionda, S. Khatchadourian, and G. Pirrò. S+EPPs: Construct and explore bisimulation summaries + optimize navigational queries; all on existing SPARQL systems (demonstration). *PVLDB*, 8(12), 2015.

[11] M. P. Consens, R. J. Miller, F. Rizzolo, and A. A. Vaisman. Exploring XML web collections with DescribeX. *ACM TWeb*, 4(3), 2010.

[12] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *SIGMOD*, 2012.

[13] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB*, 1997.

[14] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3), 2005.

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

[16] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.

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

[18] K. Khan, W. Nawaz, and Y. Lee. Set-based approximate approach for lossless graph summarization. *Computing*, 97(12):1185–1207, 2015.

[19] S. Khatchadourian and M. P. Consens. ExpLOD: Summary-based exploration of interlinking and RDF usage in the linked open data cloud. In *ESWC*, 2010.

[20] S. Khatchadourian and M. P. Consens. Constructing bisimulation summaries on a multi-core graph processing framework. In *GRADES*, 2015.

[21] W. Le, F. Li, A. Kementsietsidis, and S. Duan. Scalable keyword search on large RDF data. *IEEE TKDE*, 26(11), 2014.

[22] K. LeFevre and E. Terzi. GraSS: Graph structure summarization. In *SDM*, 2010.

[23] S.-D. Lin, M.-Y. Yeh, and C.-T. Li. Sampling and summarization for social networks (tutorial), 2013.

[24] Y. Luo, G. H. L. Fletcher, J. Hidders, Y. Wu, and P. D. Bra. External memory k-bisimulation reduction of big graphs. In *CIKM*, 2013.

[25] T. Milo and D. Suciu. Index structures for path expressions. In *ICDT*, 1999.

[26] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD*, 2008.

[27] M. Palmonari, A. Rula, R. Porrini, A. Maurino, B. Spahiu, and V. Ferme. ABSTAT: linked data summaries with abstraction and statistics. In *ESWC Workshops*, 2015.

[28] F. Picalausa, Y. Luo, G. H. L. Fletcher, J. Hidders, and S. Vansummeren. A structural approach to indexing triples. In *ESWC*, 2012.

[29] M. Rudolf, M. Paradies, C. Bornhövd, and W. Lehner. SynopSys: large graph analytics in the SAP HANA database through summarization. In *GRADES*, 2013.

[30] A. Schätzle, A. Neu, G. Lausen, and M. Przyjaciel-Zablocki. Large-scale bisimulation of RDF graphs. In *SWIM*, 2013.

[31] T. Tran, G. Ladwig, and S. Rudolph. Managing structured and semistructured RDF data using structure indexes. *IEEE TKDE*, 25(9), 2013.

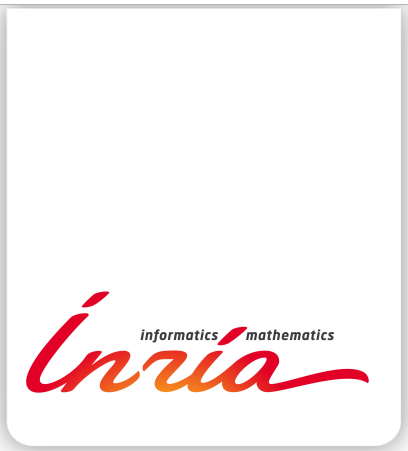
[32] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *ICDE*, 2009.

[33] W3C. Resource description framework. http://www.w3.org/RDF/.

[34] W3C. SPARQL 1.1 query language. http://www.w3.org/TR/sparql11-query/, March 2013.

[35] http://gromgull.net/blog/2010/09/btc2010-basic-stats, 2010.

[36] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: Tree + delta >= graph. In *VLDB*, 2007.



RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399