Query-Oriented Summarization of RDF Graphs

Résumés orientés requêtes de graphes RDF

Šejla Čebirić
INRIA, France
sejla.cebiric@inria.fr

François Goasdoué
U. Rennes 1 & INRIA, France
fg@irisa.fr

Ioana Manolescu
INRIA, France
ioana.manolescu@inria.fr

RÉSUMÉ

RDF est le modèle de données du W3C, fondé sur les graphes, pour les applications du Web Sémantique. Nous étudions le problème du résumé de graphes RDF : étant donné un graphe RDF G, trouver un graphe RDF H_G résumant G aussi précisément que possible, tout en étant si possible plusieurs ordres de magnitude plus petit que le graphe original. Nos résumés sont destinés à aider l'exploration de graphes RDF, ainsi que la formulation et l'optimisation de requêtes.

Nous proposons quatre sortes de résumé de graphe RDF, obtenus comme des quotients de graphes dont les relations d'équivalence reflètent la similarité entre noeuds vis-à-vis de leurs types ou connexions. Nous étudions aussi s'ils possèdent les propriétés formelles de représentativité (H_G devrait représenter autant d'information de G que possible) et de précision (H_G devrait éviter, autant que possible, de refléter des informations qui ne sont pas dans G). Enfin, nous présentons des expériences faites sur plusieurs graphes RDF synthétiques ou issus d'applications réelles.

ABSTRACT

The Resource Description Framework (RDF) is the W3C's graph data model for Semantic Web applications. We study the problem of RDF graph summarization: given an input RDF graph G, find an RDF graph H_G which summarizes G as accurately as possible, while being possibly orders of magnitude smaller than the original graph. Summaries are aimed as a help for RDF graph exploration, as well as query formulation and optimization.

We devise four kinds of RDF graph summaries obtained as quotient graphs, with equivalence relations reflecting the similarity between nodes w.r.t. their types or connections. We also study whether they enjoy the formal properties of representativeness (H_G should represent as much information about G as possible) and accuracy (H_G should avoid, to the possible extent, reflecting information that is not in G). Finally, we report the experiments we made on several synthetic and real-life RDF graphs.

1. INTRODUCTION

The Resource Description Framework (RDF) is a graph-based data model promoted by the W3C as the standard for Semantic Web applications. Its associated query language is SPARQL. RDF graphs are often *large* and *varied*, produced in a variety of contexts, e.g., scientific applications, social or online media, government data etc. They are *heterogeneous*, i.e., resources described in an RDF graph may have very different sets of properties. An RDF resource may have: no types, one or several types (which may or may not be related to each other). *RDF Schema* (RDFS) information may optionally be attached to an RDF graph, to enhance the description of its resources. Such statements also entail that in an RDF graph, some data is **implicit**. According to the W3C RDF and SPARQL specification, **the semantics of an RDF graph comprises both its explicit and implicit data**; in particular, SPARQL query answers must be computed *reflecting both the explicit and implicit data*. These features make RDF graphs complex, both structurally and conceptually. It is intrinsically hard to get familiar with a new RDF dataset, especially if an RDF schema is sparse or not available at all.

In this work, we study the problem of *RDF summarization*, that is: given an input RDF graph G, find an RDF graph H_G which *summarizes* G *as accurately as possible, while being possibly orders of magnitude smaller* than the original graph. Such a summary can be used in a variety of contexts: to help an RDF application designer get acquainted with a new dataset, as a first-level user interface, or as a support for query optimization as typically used in semi-structured graph data management [10] etc. Our approach is *query-oriented*, i.e., a summary should enable static analysis and help formulating and optimizing queries; for instance, querying a summary of a graph should reflect whether the query has some answers against this graph, or finding a simpler way to formulate the query etc. Our approach is the first semi-structured data summarization approach focused on *partially explicit, partially implicit* RDF graphs.

In the sequel, Section 2 recalls the basics of the RDF data, schema and queries, and sets the requirements for our query-oriented RDF summaries. We then introduce a general concept of RDF summary in Section 3, then decline it into several distinct brands, in Section 4 and 5. We formally establish the properties of these summaries with respect to the representation of the graph they derive from, and analyze the complexity of building them; in all cases, this complexity is in the low-degree polynomials in the number of graph edges. We have fully implemented these summarization procedures; we describe our implementation and report experimental results, before discussing related works. *For space reasons, proofs for this paper's claims are delegated to [5].* Finally, as a picture is worth a thousand words, graphical representations of sample summaries can be found at: `https://team.inria.fr/cedar/projects/rdfsummary`.

2. PRELIMINARIES

We introduce RDF graphs and queries in Section 2.1, and re-

BDA 2016, 15 au 18 Novembre, Poitiers, France.

Assertion	Triple	Relational notation
Class	s rdf:type o	o(s)
Property	s p o	p(s, o)

Constraint	Triple	OWA interpretation
Subclass	s $\prec_{sc}$ o	s $\subseteq$ o
Subproperty	s $\prec_{sp}$ o	s $\subseteq$ o
Domain typing	s $\hookleftarrow_d$ o	Π_{domain}(s) $\subseteq$ o
Range typing	s $\hookrightarrow_r$ o	Π_{range}(s) $\subseteq$ o

Figure 1: RDF (top) & RDFS (bottom) statements.

quirements for our RDF summaries in Section 2.2.

2.1 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 [21], using uniform resource identifiers (URIs), typed or un-typed literals (constants) and blank nodes (unknown URIs or literals). Blank nodes are essential features of RDF allowing to support *unknown URI/literal tokens*. These are conceptually similar to the labelled nulls or variables used in incomplete relational databases [1], as shown in [8].

Notations. We use s, p, and o in triples as placeholders. Literals are shown as strings between quotes, e.g., "$string$".

Figure 1 (top) shows how to use triples to describe resources, that is, to express class (unary relation) and property (binary relation) assertions. The RDF standard [21] provides 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 G shown below describes a book, identified by doi_1: its author (a blank node _:b_1 related to the author name), title and date of publication.

$$\mathtt{G} = \begin{array}{l}\{\mathtt{doi}_1 \text{ rdf:type } \mathtt{Book},\ \mathtt{doi}_1\ \mathtt{writtenBy}\ \mathtt{\_}{:}b_1,\\ \ \mathtt{doi}_1\ \mathtt{hasTitle}\ \text{“}\mathtt{Le\ Port\ des\ Brumes}\text{”},\\ \ \mathtt{\_}{:}b_1\ \mathtt{hasName}\ \text{“}\mathtt{G.\ Simenon}\text{”},\\ \ \mathtt{doi}_1\ \mathtt{publishedIn}\ \text{“}\mathtt{1932}\text{”}\}\end{array}$$

RDF Schema. A valuable feature of RDF is RDF Schema (RDFS) that allows enhancing the descriptions in RDF graphs. RDFS triples declare *semantic constraints* between the classes and the properties used in those graphs.

Figure 1 (bottom) shows the four types of constraints we consider, and how to express them through triples; for concision, we use the symbols $\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_d$ and $\hookrightarrow_r$ to denote the standard properties used in subclass, subproperty, domain and range constraints, respectively. *Domain* and *range* denote respectively the first and second attribute of every property. The RDFS constraints (Figure 1) are interpreted under the open-world assumption (OWA) [1]. For instance, given two relations R_1, R_2, the OWA interpretation of the constraint $R_1 \subseteq R_2$ is: any tuple t in the relation R_1 *is considered as being also in the relation* R_2 (the inclusion constraint *propagates* t *to* R_2). More specifically, when working with the RDF data model, if the triples hasFriend rdfs:domain Person and Anne hasFriend Marie hold in the graph, then so does the triple Anne rdf:type Person. The latter is due to the domain constraint in Figure 1.

Implicit triples are an important RDF feature, considered part of the RDF graph even though they are not explicitly present in it, e.g., Anne rdf:type Person above. 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 of the entailment sequence, the triples previously entailed are also taken into account).

For instance, assume that the before-mentioned RDF graph G is extended with the following constraints.

- books are publications:
  Book rdfs:subClassOf Publication
- writing something means being an author:
  writtenBy rdfs:subPropertyOf hasAuthor
- books are written by people:
  writtenBy rdfs:domain Book
  writtenBy rdfs:range Person

These constraints lead to the following implicit triples being part of the RDF graph:

$\mathtt{doi}_1$ rdf:type Publication $\mathtt{doi}_1$ hasAuthor _:b_1
writtenBy $\hookleftarrow_d$ Publication _:b_1 rdf:type Person

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 $\mathtt{G}^\infty$ defined as the fixed-point 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 $\in \mathtt{G}^\infty$.

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* $\mathtt{G}^\infty$, since *the semantics of an RDF graph is its saturation* [21].

Well-behaved RDF graphs. We say an RDF graph is *well-behaved* if (i) no class appears in the property position; (ii) no class has properties other than the rdf:type and than those of RDF Schema. While not part of the RDF specification, these constraints reflect common-sense data design decisions, and hold in all datasets we experimented with. We assume all graphs are all-behaved.

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

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:

- $\mathtt{S_G}$, called the **schema** component, is the set of all G triples whose properties are $\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_d$ or $\hookrightarrow_r$;
- $\mathtt{T_G}$, called the **type** component, is the set of the τ triples from G;
- $\mathtt{D_G}$, called the **data** component, holds all the remaining triples of G.

Note that each of $\mathtt{D_G}$, $\mathtt{S_G}$, and $\mathtt{T_G}$ is an RDF graph by itself.

The graph-based representation of an RDF graph. As per the RDF specification [21], *the set of nodes* of an RDF graph is the set of subjects and objects of triples in the graph. Further, we define:

(i) a *data node* as any URI or literal occurring as a subject or object in $\mathtt{D_G}$, or as a subject in $\mathtt{T_G}$; (ii) a *class node* as a URI appearing in the object position of triples from $\mathtt{T_G}$, and (iii) a *property node* as a URI appearing in the subject or object position of $\prec_{sp}$ triples, or in the subject position of $\hookleftarrow_d$ or $\hookrightarrow_r$ triples from $\mathtt{S_G}$. The *edges* of an RDF graph correspond to its triples, where the subject/object is the edge source/target node, and the property is the edge label.

Size and cardinality notations. We denote by $|\mathtt{G}|_n$ the number of nodes in a graph $\mathtt{G}$, and by $|\mathtt{G}|_e$ its number of edges. Further, for a given attribute $x \in \{\mathtt{s}, \mathtt{p}, \mathtt{o}\}$ and graph $\mathtt{G}$, we note $|\mathtt{G}|^0_x$ the number of distinct values of the attribute x within $\mathtt{G}$. For instance, $|\mathtt{D_G}|^0_\mathtt{p}$ 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 [15]. 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 $\{t_1, \ldots, t_\alpha\}$ are triple patterns; the query head variables $\bar{x}$ are called *distinguished variables*, and are a subset of the variables 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 may lead to an incomplete answer; the complete answer is obtained by evaluating q against $\mathtt{G}^\infty$. For instance, the query below asks for name of the author of "Le Port des Brumes":

$$q(x_3) \text{ :- } x_1 \ \mathtt{hasAuthor}\ x_2,\ x_2\ \mathtt{hasName}\ x_3$$
$$x_1\ \mathtt{hasTitle}\ \text{“}\mathtt{Le\ Port\ des\ Brumes}''$$

Its answer against the explicit and implicit triples of our sample graph is: $q(\mathtt{G}^\infty) = \{\langle \text{“G. Simenon”} \rangle\}$. Note that evaluating q only against $\mathtt{G}$ leads to the empty answer, which is obviously incomplete.

2.2 RDF Summary Requirements

We assume that *the summary* $\mathtt{H_G}$ *of an RDF graph* $\mathtt{G}$ *is an RDF graph itself*. Further, we require summaries to satisfy the following conditions:

Schema independence It must be possible to summarize $\mathtt{G}$ whether or not it has a schema.

Semantic completeness The summary of $\mathtt{G}$ must reflect not only the explicit data, but also the implicit one, given that the semantic of $\mathtt{G}$ is its saturation.

Tolerance to heterogeneity The summary should enable the recognition of "similar" resources despite some amount of heterogeneity in their properties and/or types.

The following properties are of a more quantitative nature:

Compactness The summary should be typically smaller than the RDF graph, ideally by orders of magnitude.

Representativeness The summary should preserve as much information about $\mathtt{G}$ as possible[1].

[1]A clear trade-off exists between compactness and representativeness; we present concrete proposals for such a trade-off later on.

Accuracy The summary should avoid, to the extent possible, reflecting data that does not exist in $\mathtt{G}$.

Criteria for representativeness and accuracy. Our query-oriented RDF graph summarization leads us to the following criteria. For *representativeness*, queries with results on $\mathtt{G}$ should also have results on the summary. Symmetrically, for *accuracy*, a query that can be matched on the summary, should also be matched on the RDF graph itself.

To formalize our criteria, we use $\mathcal{Q}$ to denote an *RDF query language (dialect)*; a concrete choice of such a dialect will shortly follow.

DEFINITION 1. (QUERY-BASED REPRESENTATIVENESS) *Let* $\mathtt{G}$ *be any RDF graph.* $\mathtt{H_G}$ *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* $q(\mathtt{H}^\infty_\mathtt{G}) \neq \emptyset$.

Note that *several graphs may have the same summary*, which corresponds to the intuition that a summary loses *some* of the information from the original graph. If two RDF graphs differ only with respect to such information, they have the same summary. We term *inverse set* of a summary $\mathtt{H_G}$, the set of all RDF graphs whose summary is $\mathtt{H_G}$. This leads to the accuracy criterion:

DEFINITION 2. (QUERY-BASED ACCURACY) *Let* $\mathtt{G}$ *be any RDF graph,* $\mathtt{H_G}$ *its summary, and* $\mathcal{G}$ *the inverse set of* $\mathtt{H_G}$*. The summary* $\mathtt{H_G}$ *is* $\mathcal{Q}$-accurate *if for any query* $q \in \mathcal{Q}$ *such that* $q(\mathtt{H}^\infty_\mathtt{G}) \neq \emptyset$*, 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*.

For the sake of compactness, we decide that the (voluminous) set of literals, along with subject and object URIs for non-τ triples from $\mathtt{G}$ should not appear in $\mathtt{H_G}$. However, given that property URIs are often specified in SPARQL queries [2], and that there are typically far less distinct property URIs than there are distinct subject or object URIs [22], property URIs should be preserved by the summary. This leads us to the following SPARQL dialect:

DEFINITION 3. (RELATIONAL BGP QUERY) *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.*

A sample RBGP is:

$$q(x_1, x_3) \text{ :- } x_1\ \tau\ \text{“}\mathtt{Book}''\ x_1\ \mathtt{author}\ x_2,\ x_2\ \mathtt{reviewed}\ x_3$$

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

3. RDF SUMMARIZATION

Let $\mathtt{G} = \langle \mathtt{D_G}, \mathtt{S_G}, \mathtt{T_G} \rangle$ be an RDF graph. For illustration, we will rely on the graph in Figure 2; here and in the sequel, we show class nodes in purple boxes. Further, $\mathtt{T_G}$ triples appear as purple arrows, $\mathtt{D_G}$ consists of the triples shown in black, while $\mathtt{S_G}$ is empty.

We recall a classical definition from graph theory:

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* $E \subseteq V \times V \times A$ *where each edge carries a label from* A*. Let* $\equiv \subseteq V \times V$ *be an equivalence relation over the nodes of* V.

The quotient graph of G over $\equiv$*, denoted* $G_\equiv$*, 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 nodes* $n_1 \in S_1$ *and* $n_2 \in S_2$ *such that the edge* $n_1 \xrightarrow{a} n_2 \in E$.

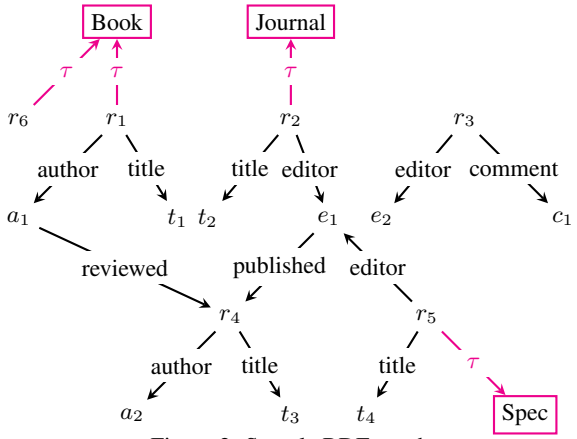

Figure 2: Sample RDF graph.

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 Figure 2, the data properties are: author, title, editor, comment, reviewed, and published, which for the sake of brevity we will denote as a, t, e, c, r and p.

Summarization approach and node equivalence. We will define summaries through quotient graphs, using various equivalence relations. To establish such relations, we first examine the data properties of graph nodes; we identify interesting relations between these properties (Section 3.1) and use these to define *node equivalence relations based on their data properties* in Section 3.2, where we also introduce a simple *node equivalence relation based on node types*. In Section 3.3, we formalize RDF summaries, and study some of their important properties.

3.1 Data property relationships and cliques

We start by considering *relations between data properties* in a graph $\mathtt{G}$. The simplest relationship is co-occurrence, when a resource is the source and/or target of two data properties. We generalize this into:

DEFINITION 5. (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 one of the following conditions holds: (i) a resource in* $\mathtt{G}$ *has 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 *has* $\mathtt{p}_1$ *and* $\mathtt{p}_3$*, and moreover* $\mathtt{p}_3$ *and* $\mathtt{p}_2$ *are source-related.*

2. $\mathtt{p}_1, \mathtt{p}_2 \in \mathtt{G}$ *are* target-related *in the same conditions as above (replacing in (i) "has a property" by "is the value of a property" and in (ii) "source" by "target").*

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

For illustration, consider the sample graph in Figure 2. 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. Properties r and p are target-related due to r_4. The source cliques are:

$$SC_1 = \{a, t, e, c\}; SC_2 = \{r\}; SC_3 = \{p\};$$

whereas the target cliques are:

$$TC_1 = \{a\}; TC_2 = \{t\}; TC_3 = \{e\}; TC_4 = \{c\}; TC_5 = \{r, p\}$$

r	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	$\emptyset$

	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

	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 source clique can be viewed as "all the data properties of resources of the same kind"; in the above example, it makes sense to group together properties corresponding to various kinds of publications, such as author, title, editor, and comment. Similarly, a target clique comprises "the data properties of which same-kind resources are values".

It is easy to see that the set of source (or target) property cliques is a partition over the data properties of $\mathtt{G}$. Further, if a resource $r \in \mathtt{G}$ 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 2 are shown in Table 1.

DEFINITION 6. (PROPERTY DISTANCE IN A CLIQUE) *The* distance *between two data properties* $\mathtt{p}, \mathtt{p}'$ *in a source clique* SC *is:*

- 0 *if there exists a resource in* $\mathtt{G}$ *having them both;*
- *otherwise, the smallest integer* n *such that* $\mathtt{G}$ *holds resources* $r_0, r_2, \ldots, r_n \in \mathtt{G}$ *and data properties* $\mathtt{p}_1, \ldots, \mathtt{p}_n$ *such that* r_0 *has* $\mathtt{p}$ *and* $\mathtt{p}_1$*,* r_1 *has* $\mathtt{p}_1$ *and* $\mathtt{p}_2$*, . . .,* r_n *has* $\mathtt{p}_n$ *and* $\mathtt{p}'$*.*

For instance, in our sample graph, the distance between a and t is 0 since r_1 has them both. The distance between a and e is 1, while the distance between a and c is 2.

It is easy to see that $\mathtt{p}$ and $\mathtt{p}'$ are at distance n for $n > 0$ iff there exists a resource having $\mathtt{p}$ 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 $\mathtt{G}$; when moving from $\mathtt{G}$ to $\mathtt{G}^\infty$, resources may have more data properties due to the $\prec_{sp}$ constraint, thus possibly different cliques. The following important property characterizes the relationship between the cliques of $\mathtt{G}$ and $\mathtt{G}^\infty$:

LEMMA 1 (SATURATION VS. PROPERTY CLIQUES). *Let* C*,* C_1*,* C_2 *be distinct source (or target) cliques corresponding to* $\mathtt{G}$*.*

1. *There exists exactly one source (resp. target) clique* C^∞ *corresponding to* $\mathtt{G}^\infty$ *such that* $C \subseteq C^\infty$*.*

2. *We call* saturated clique *and denote* C^+ *the set of properties comprising all the properties in* C *and all their generalizations (superproperties). 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. *Let* $\mathtt{p}_1, \mathtt{p}_2$ *be two data properties in different source (or target) cliques of* $\mathtt{G}$*, namely* 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 *source cliques of* $\mathtt{G}$*,* $k \geq 0$*, denoted* $D_1, D_2, \ldots, D_k$ *such that:*

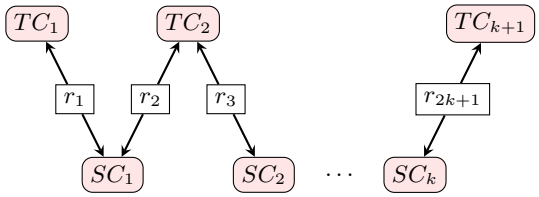

Figure 3: Alternating sequence of shared cliques between weakly related resources $r_1, r_2, \ldots, r_{2k+1}$.

$$C_1^+ \cap D_1^+ \neq \emptyset,$$
$$D_1^+ \cap D_2^+ \neq \emptyset, \ldots, D_{k-1}^+ \cap D_k^+ \neq \emptyset, D_k^+ \cap C_2^+ \neq \emptyset.$$

3.2 Equivalence relations among graph nodes

We use the above source and target cliques to define two notions of equivalence between data nodes of a graph:

DEFINITION 7. (STRONG AND WEAK EQUIVALENCE) *Two data nodes of* $\mathtt{G}$ *are* strongly related*, denoted* $n_1 \equiv_{\mathtt{S}} n_2$*, iff they have the same source* and *target cliques.*

Two nodes are weakly related*, denoted* $n_1 \equiv_{\mathtt{W}} n_2$*, iff they have either a same non-empty source* or *target cliques, or they are both weakly related to another data node of* $\mathtt{G}$.

Observe that strong relatedness implies weak relatedness.

In Figure 2, the resources r_1, r_2, r_3, r_5 are *strongly* related to each other, as well as t_1, t_2, t_3, t_4. Moreover, $r_1, ..., r_5$ are *weakly* related 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, resources can also be weakly related through an *alternating sequence of source and target cliques*, as illustrated in Figure 3.

If one does not wish to consider data properties as a basis for node equivalence, a simple equivalence based on node types is:

DEFINITION 8. (TYPE-BASED EQUIVALENCE) *Two nodes* n_1, n_2 *of* $\mathtt{G}$ *are type-equivalent, denoted* $n_1 \equiv_{\mathtt{T}} n_2$*, iff (i) both* n_1 *and* n_2 *have types in* $\mathtt{G}$ *and (ii) they have exactly the same set of types.*

Recall that some or even all nodes in an RDF graph may lack types; all such nodes are equivalent from the viewpoint of $\equiv_{\mathtt{T}}$.

Clearly, each equivalence relation defines a partition over the data nodes in $\mathtt{G}$.

3.3 RDF summaries

We define RDF summaries based on graph quotients:

DEFINITION 9. (RDF SUMMARY) *Given an RDF graph* $\mathtt{G} = \langle \mathtt{D_G}, \mathtt{S_G}, \mathtt{T_G} \rangle$ *and an equivalence relation* $\equiv$ *among the set of nodes of* $\mathtt{G}$*, a summary of* $\mathtt{G}$ *is an RDF graph* $\mathtt{H_G} = \langle \mathtt{D_H}, \mathtt{S_H}, \mathtt{T_H} \rangle$ *whose triples are defined as follows:*

SCH $\mathtt{H_G}$ *has the same schema triples as* $\mathtt{G}$*:* $\mathtt{S_H} = \mathtt{S_G}$*;*

TYP+DAT $\mathtt{T_H} \cup \mathtt{D_H}$ *is the quotient of* $\mathtt{T_G} \cup \mathtt{D_G}$ *by* $\equiv$*:* $\mathtt{T_H}$ *is the set of* τ *triples in* $(\mathtt{T_H} \cup \mathtt{D_H})_{\equiv}$*, while* $\mathtt{D_H}$ *holds the remaining triples.*

This definition *preserves the RDF schema unchanged*. We do this, first, because a schema (when present) provides a valuable and typically compact set of constraints describing the data semantics; and second, because we want the summary to enable, on a smaller graph, the kinds of reasoning possible on the original graph – in particular, as we shall see, as a means to achieve summary completeness (Section 2.2). Further, since the summary is an RDF graph, *all its nodes must be labeled by URIs or literals*; in particular, the subjects in $\mathtt{T_H}$ and $\mathtt{D_H}$ must be URIs.

We find that:

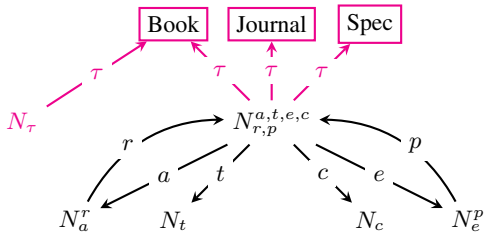

Figure 4: Weak summary of the RDF graph in Figure 2.

PROPOSITION 1. (SUMMARY REPRESENTATIVENESS) *An RDF summary* $\mathtt{H}$ *is RBGP-representative.*

The proof [5] is based on building from every embedding ϕ of a query q into $\mathtt{G}^\infty$, an embedding from q into $\mathtt{H_G^\infty}$, thus establishing the non-emptiness of $q(\mathtt{H_G^\infty})$.

We are interested in summaries having the following property:

DEFINITION 10. (FIXPOINT PROPERTY) *A summary* $\mathtt{H}$ *has the* fixpoint property *iff for any graph* $\mathtt{G}$*,* $\mathtt{H_{H_G}} = \mathtt{H_G}$ *holds.*

Intuitively, the fixpoint property expresses the fact that a summary cannot be summarized further, i.e., $\mathtt{H_G}$ is its own summary. This is a desirable property as we wish our summaries to be as compact as possible, while satisfying our requirements. It turns out that our summaries enjoy this property, as they are quotient graphs:

PROPOSITION 2. (SUMMARY FIXPOINT) *An RDF summary has the fixpoint property.*

Moreover, the fixpoint property of our summaries has another immediate good consequence, as a summary is obviously a summary of itself:

PROPOSITION 3. (ACCURACY) *An RDF summary is accurate.*

4. WEAK-EQUIVALENCE SUMMARIES

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

4.1 Weak summary

Our simplest summary is solely based on weak equivalence:

DEFINITION 11. (WEAK SUMMARY) *The weak summary of the graph* $\mathtt{G}$*, denoted* $\mathtt{W_G}$*, is its quotient graph w.r.t. the weak equivalence relation* $\equiv_{\mathtt{W}}$.

It follows from the definition of quotient graphs that for each $\mathtt{G}$ node equivalence set w.r.t. $\equiv_{\mathtt{W}}$, there is exactly one node in $\mathtt{W_G}$. Further, note that the partition of the set of nodes of $\mathtt{G}$ into sets according to $\equiv_{\mathtt{W}}$ is also *a partition of* $\mathtt{G}$ *data properties at the source*, that is: in $\mathtt{G}$, the sources of edges labeled with a given data property $\mathtt{p}$ are all weakly equivalent. For instance, in Figure 2, the sources of all "editor" edges are in the weakly equivalence class $\{r_1, r_2, r_3, r_4, r_5\}$. By the same reasoning, the $\equiv_{\mathtt{W}}$ partition over the set of nodes of $\mathtt{G}$ induces a *partition of the source cliques of* $\mathtt{G}$; and by a symmetrical reasoning, it introduces also a *partition over the target cliques of* $\mathtt{G}$. In Figure 2, 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 nodes in* $\mathtt{G}$ *one can associate through a bijection, a set of* $\mathtt{G}$ *source cliques, and a set of* $\mathtt{G}$ *target cliques*. We shall refer to the node $n_S \in \mathtt{W_G}$ representing S as:

$$n_S = N(\underset{n \in S}{\cup} TC(n), \underset{n \in S}{\cup} SC(n))$$

where N, termed *representation function*, is any injective function taking as input two sets of URIs (respectively, a set of target data properties, and a set of source data properties), and returning a new

URI. Without loss of generality, we will use N to denote a concrete representation function which assigns URIs to nodes in quotient (RDF) graphs.

Notations. Whenever this simplifies reading, we may use N_r to denote the weak summary node representing a certain resource $r \in S$. Similarly, we may use N_{SC}^{TC} to denote $N(TC, SC)$. Further, for simplicity, we will mostly *omit the set delimiters* when showing TC and SC, and omit one such set altogether if it is empty.

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 2 is shown in Figure 4. Its nodes are:

- $N_{r,p}^{a,t,e,c}$ 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_a^r for the set $\{a_1, a_2\}$;
- N_t for the relatedness partition set $\{t_1, t_2, t_3, t_4\}$;
- N_e^p for the set $\{e_1, e_2\}$;
- N_c for the set $\{c_1\}$.

The edges from $N_{r,p}^{a,t,e,c}$ to N_a and N_t are due to r_1 and item **DAT** of the summary definition (Definition 9); the edge to N_e^p is due to r_2 and r_3; the edge to N_c is due to r_3. The edge from N_a^r to $N_{r,p}^{a,t,e,c}$ is due to a_1, and the edge from N_e^p to $N_{r,p}^{a,t,e,c}$ is due to e_1. The τ edges outgoing $N_{r,p}^{a,t,e,c}$ 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:

PROPOSITION 4. (UNIQUE DATA PROPERTIES) *Each* $\mathtt{G}$ *data property appears exactly once in* $\mathtt{W_G}$.

Importantly, the above Property 4 warrants that the number of data edges in $\mathtt{W_G}$ is exactly $|\mathtt{D_G}|_\mathtt{p}^0$, the number of distinct data properties in $\mathtt{G}$. Thus, its number of data nodes is at most $2|\mathtt{D_G}|_\mathtt{p}^0$. The number of type triples in $\mathtt{W_G}$ is bound by $min(|\mathtt{T_G}|_e, 2|\mathtt{D_G}|_\mathtt{p}^0 * |\mathtt{T_G}|_\mathtt{o}^0|)$: the latter corresponds to the case when every data node in $\mathtt{W_G}$ is of every type in $\mathtt{T_G}$.

The next important property of the weak summary is:

PROPOSITION 5. (WEAK COMPLETENESS) *For a given graph* $\mathtt{G}$, $\mathtt{W_{G^\infty}} = \mathtt{W}_{(\mathtt{W_G})^\infty}$ *holds.*

This property is important, as it gives a mean to compute $\mathtt{W_{G^\infty}}$ without saturating $\mathtt{G}$, but only summarizing $\mathtt{G}$, then saturating the smaller (typically by several orders of magnitude) $\mathtt{W_G}$, and applying again weak summarization. This is exemplified in Figure 5, which traces the transformation of a graph $\mathtt{G}$ on one hand, into $\mathtt{W_G}$, $(\mathtt{W_G})^\infty$, and then $\mathtt{W}_{(\mathtt{W_G})^\infty}$; and on the other hand, from $\mathtt{G}$ to $\mathtt{G^\infty}$ then $\mathtt{W_{G^\infty}}$. The graph at the bottom right in the figure illustrates the equality stated by the proposition.

The proof of Proposition 5 requires the following crucial lemma:

LEMMA 2 (SHARED CLIQUES VS. SUMMARIZATION). *If two resources* r', r'' *share a target clique in* $\mathtt{G^\infty}$, *their nodes* $N_{r'}, N_{r''}$ *representing them, share a target clique in* $(\mathtt{W_G})^\infty$.

Figure 5 illustrates weak sumary completeness on an example; to avoid clutter, summary nodes are shown as unlabeled circles.

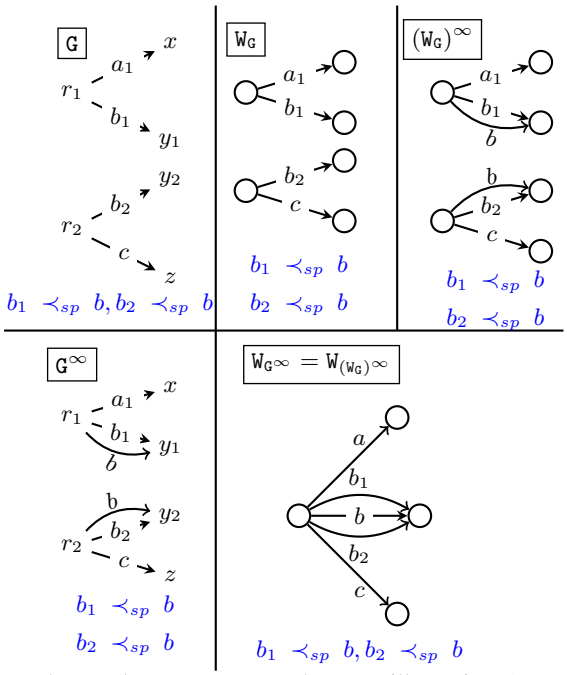

Figure 5: Weak summary completeness illustration (summary nodes shown as unlabeled circles).

4.2 Typed weak summary

In this section, we introduce a summary based on the weak relatedness notion but which gives a strong preeminence to type information: it *represents together nodes having the same set of RDF types*, and quotients the data triples only for nodes having no types.

Formally, given a graph $\mathtt{G}$, we term *typed resources* of $\mathtt{G}$ and denote $\mathtt{TR_G}$ the set of subjects of $\mathtt{T_G}$ (type) triples. We denote *untyped resources* of $\mathtt{G}$ the set $\mathtt{UN_G}$ of subjects or objects of $\mathtt{D_G}$ triples, which have no types. $\mathtt{UN_G}$ may comprise URIs and/or literals. The *untyped data graph* of $\mathtt{G}$, denoted $\mathtt{UD_G}$, is the subset of $\mathtt{D_G}$ triples whose subject and object belong to $\mathtt{UN_G}$.

We start by introducing two helper notions:

DEFINITION 12. (TYPE-BASED SUMMARY) *The type-based summary of* $\mathtt{G}$, *denoted* $\mathtt{T_G}$, *is the summary of* $\mathtt{G}$ *through the* $\equiv_\mathtt{T}$ *equivalence relation.*

This summary groups together typed resources which have the same non-empty set of types, and copies each untyped $\mathtt{G}$ node, since none of them is equivalent to any other node. We assume available an *injective* function C which, given as input a set X of (class) URIs, returns a URI $C(X)$, and given an empty set of URIs, returns a new URI on every call. An easy bijection holds between the $\equiv_\mathtt{T}$ equivalence classes of typed resources in $\mathtt{G}$, and their respective sets of types. Thus, every node in the summary $\mathtt{T_G}$ can be seen as obtained through a call to $C(X)$, where X is the set of types of the resources in an equivalence class wrt $\equiv_\mathtt{T}$. Figure 6 illustrates a typed summary.

DEFINITION 13. (U-WEAK EQUIVALENCE AND SUMMARY) *Two nodes in a graph* $\mathtt{G}$ *are* untyped-weak equivalent, *denoted* $n_1 \equiv_\mathtt{UW} n_2$, *iff* n_1 *and* n_2 *have no type in* $\mathtt{G}$ *and* $n_1 \equiv_\mathtt{W} n_2$.

The untyped-weak summary of $\mathtt{G}$, *denoted* $\mathtt{UW_G}$, *is its sumary through* $\equiv_\mathtt{UW}$.

Untyped weak equivalence restricts weak equivalence to untyped resources only. The untyped-weak summary $\mathtt{UW_G}$ summarizes $\mathtt{UD_G}$ in weak fashion, and leaves all typed resources untouched. We can now define:

DEFINITION 14. (TYPED WEAK SUMMARY) *The typed weak summary* $\mathtt{TW_G}$ *of an RDF graph* $\mathtt{G}$ *is the untyped-weak summary of the type-based summary of* $\mathtt{G}$, *namely* $\mathtt{UW}_{\mathtt{T_G}}$.

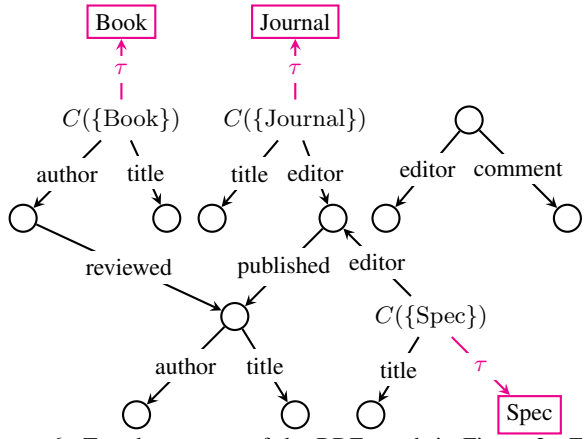

Figure 6: Typed summary of the RDF graph in Figure 2. Each circle is a distinct node whose URI results from calling $C(\emptyset)$.

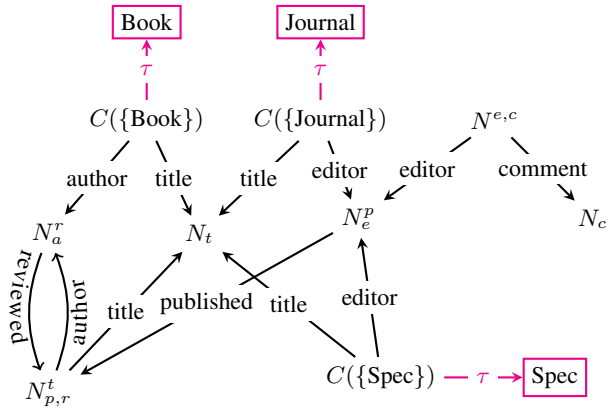

Figure 7: Typed weak summary example: $\mathtt{TW}_G = \mathtt{UW}_{T_G}$ for the T_G in Figure 6.

For example, Figure 7 shows the typed weak summary of the RDF graph in Figure 2. Note how distinct types lead to different nodes in the summary.

PROPOSITION 6. (TYPED WEAK FIXPOINT) *The typed weak summary (Definition 14) has the fixpoint property.*

PROPOSITION 7. (TYPED WEAK NON-COMPLETENESS) *There exists an RDF graph* G *such that* $\mathtt{TW}_{G^\infty} \neq \mathtt{TW}_{(\mathtt{TW}_G)^\infty}$ *holds.*

More generally, in the presence of domain ($\hookleftarrow_d$) or range ($\hookrightarrow_r$) RDF schema rules, one cannot compute $\mathtt{TW}_{G^\infty}$ from G because the typed weak summary represents typed resources differently from the untyped ones. The $\hookleftarrow_d$ and $\hookrightarrow_r$ schema rules may turn untyped resources into typed ones, thus leading to divergent representations of the data nodes of G in $\mathtt{TW}_G$ and respectively $\mathtt{TW}_{G^\infty}$.

5. STRONG EQUIVALENCE SUMMARIES

In this section, we discuss summary alternatives based on the strong equivalence $\equiv_S$ between graph nodes. Strong summaries (Section 5.1) are mainly based on nodes' data properties; the typed strong summary (Section 5.2) gives preeminence to types.

5.1 Strong summary

RDF data nodes represented by the same strong summary node have *the same source clique and the same target clique*. Formally, based on the N function introduced in Section 4.1, we define:

DEFINITION 15. (STRONG SUMMARY) *The* strong summary f_S *is an RDF summary based on the strong equivalence* $\equiv_S$.

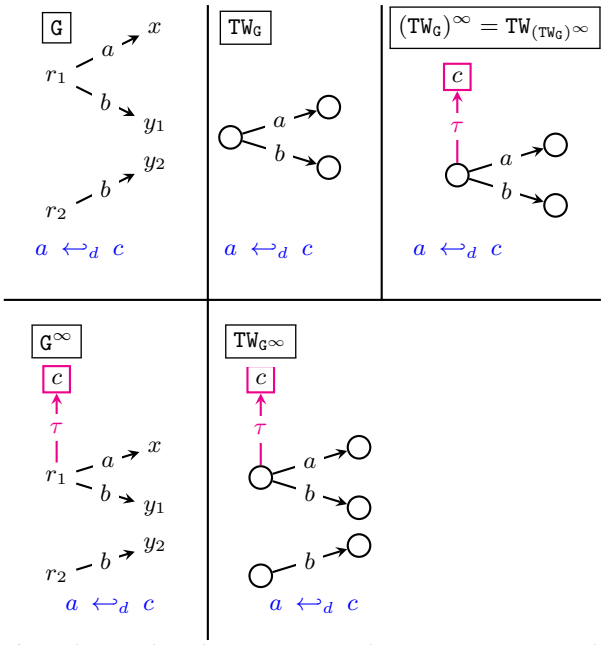

Figure 8: Typed weak summary completeness counter-example.

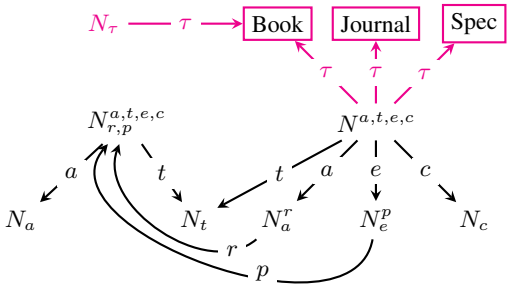

Figure 9: Strong summary of the RDF graph in Figure 2.

Recall from the definition of $\equiv_S$ 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 G, and a node in the strong summary. To follow through this intuition, in this section, we denote by N_{SC}^{TC} the node representing the set of $\equiv_S$-equivalent nodes of G having the target clique TC and the source clique SC.

For example, the strong summary of the graph of Figure 2 is shown in Figure 9. The strong summary comprises e.g., the nodes:

- $N(\emptyset, SC_1) = N^{a,t,e,c}$, for r_1, r_2, r_3 and r_5;
- $N(TC_5, SC_1) = N_{r,p}^{a,t,e,c}$, for r_4;
- $N(TC_1, SC_2) = N_r^a$, for a_1

Compared with the weak summary (Figure 4), the strong summary refines (splits) the weak summary node $N_{r,p}^{a,t,e,c}$ into two nodes, $N_{r,p}^{a,t,e,c}$ and $N^{a,t,e,c}$, because the resources from G having the output clique $\{a, t, e, c\}$ have either an empty target clique, or the target clique $\{r, p\}$. As a consequence, a strong summary may have several edges with the same label: in Figure 9, an a-labeled edge exits $N_{r,p}^{a,t,e,c}$ and another one exits $N^{a,t,e,c}$. In contrast, in a weak summary, only one edge can have a given label (Property 4).

As a result, the number of data nodes in the data component D_{S_G} is bound by $min(|D_G|_n, (|D_G|_e^0)^2)$ (recall the notations introduced in

Section 2.1). Indeed, $\mathtt{S_G}$ 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 $|\mathtt{D_G}|_e^0$. By a similar reasoning, the number of data triples in $\mathtt{S_G}$ is bound by $min(|\mathtt{D_G}|_e, (|\mathtt{D_G}|_e^0)^4)$. In the worst case, $\mathtt{T_{S_G}}$ has as many nodes (and triples) as $\mathtt{T_G}$; $\mathtt{S_{S_G}}$ is identical to $\mathtt{S_G}$.

We prove in [5] that:

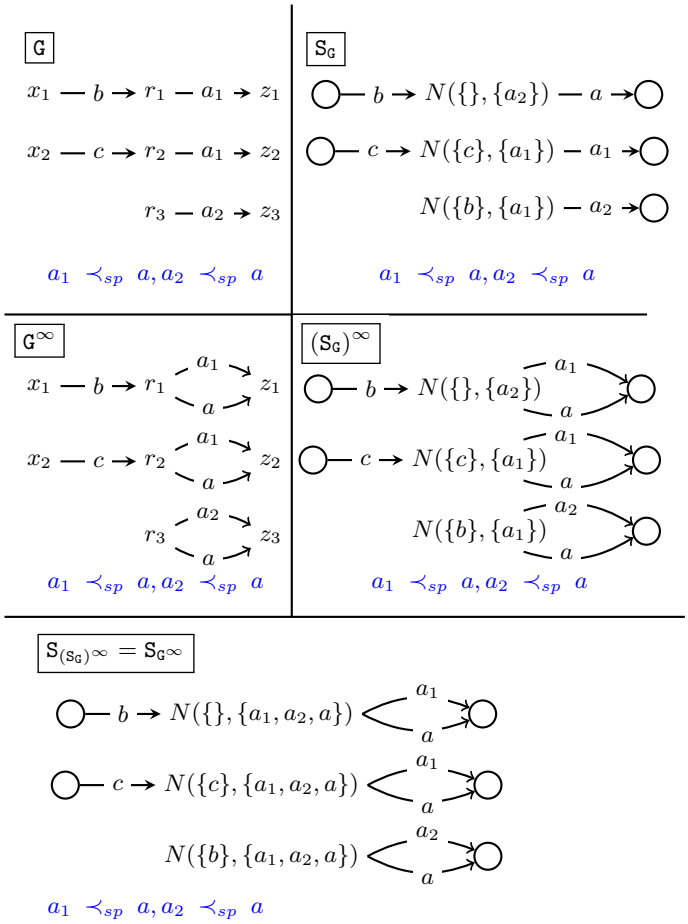

Figure 10: Illustration of the strong completeness statement (some summary nodes are shown as unlabeled circles).

PROPOSITION 8. (STRONG COMPLETENESS) *For a given graph* $\mathtt{G}$, $\mathtt{S_G}^\infty = \mathtt{S}_{(\mathtt{S_G})^\infty}$ *holds.*

5.2 Typed strong summary

The summary presented here is the counterpart of the typed weak one from Section 4.2, but based on strong relatedness. Relying on the notions of untyped resources $\mathtt{UN_G}$, untyped data graph $\mathtt{UD_G}$ and the class set representation function $C(X)$ introduced there, we similarly define:

DEFINITION 16. (U-STRONG EQUIVALENCE AND SUMMARY) *Two nodes in a graph* $\mathtt{G}$ *are* untyped-strong equivalent*, denoted* $n_1 \equiv_{\mathtt{US}} n_2$*, iff* n_1 *and* n_2 *have no type in* $\mathtt{G}$ *and* $n_1 \equiv_{\mathtt{S}} n_2$.

The untyped-strong summary of $\mathtt{G}$*, denoted* $\mathtt{US_G}$*, is its sumary through* $\equiv_{\mathtt{US}}$.

Untyped strong equivalence restricts strong equivalence to untyped resources only. The untyped-strong summary $\mathtt{US_G}$ summarizes $\mathtt{UD_G}$ in strong fashion, and leaves all typed resources untouched. We can now define:

DEFINITION 17. (TYPED STRONG SUMMARY) *The* typed strong summary $\mathtt{TS_G}$ *of an RDF graph* $\mathtt{G}$ *is the untyped-strong summary of the type-based summary of* $\mathtt{G}$*, namely* $\mathtt{US}_{\mathtt{T_G}}$.

In our example, it turns out that the type-strong summary of the RDF graph in Figure 2 coincides with the type-weak one shown in Figure 7. As can be easily seen from their definition, 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.

PROPOSITION 9. (TYPED STRONG FIXPOINT) *The typed strong summary (Definition 17) has the fixpoint property.*

The proof follows again by from the fixpoint property of the type-based summary, and that of the untyped strong one.

PROPOSITION 10. (TYPED STRONG NON-COMPLETENESS) *There exists an RDF graph* $\mathtt{G}$ *such that* $\mathtt{TS_G}^\infty \neq \mathtt{TS}_{(\mathtt{TS_G})^\infty}$ *holds.*

More generally, in the presence of domain ($\hookleftarrow_d$) or range ($\hookrightarrow_r$) RDF schema rules, one cannot compute $\mathtt{TS_G}^\infty$ from $\mathtt{G}$ because the typed weak summary represents typed resources differently from the untyped ones. The $\hookleftarrow_d$ and $\hookrightarrow_r$ schema rules may turn untyped resources into typed ones, thus leading to divergent representations of the data nodes of $\mathtt{G}$ in $\mathtt{TS_G}$ and respectively $\mathtt{TS_G}^\infty$.

6. DATA STRUCTURES & ALGORITHMS

We implemented our summarization tool in Java 1.8 (approximately 15.000 lines of code). It issues SQL queries to the underlying RDF data store implemented in PostgreSQL, and builds the respective summaries based on the results of these queries. We chose a simple, generic, yet robust storage of RDF graphs within the PostgreSQL v9.3.2 relational database server.

The triples are loaded from a file to the triples table in Postgres through the Postgres `COPY` command (currently, only files in n-triples format are supported). The triples table comprises three columns, for subject, property and object. We encode the triples table and subsequently work only with the *integer representation of the input RDF graph*. Operating on integers instead of strings provides for savings both in processing time and memory. For each resource from $\mathtt{G}$, the dictionary table in Postgres stores its unique integer value.

The encoded triples table is split into a table of data triples and a table of type triples, where each row is assigned its sequence number.

We rely on the Jena RDF API [23] to parse the RDFS triples. Typically there are much less schema triples compared to instance triples, so we load the schema directly to memory, after which we encode and store it to a separate Postgres table.

6.1 Data structures

Summary. Similar to the input graph, we work with *an integer representation of the summary resources*, i.e., each URI in the summary is represented by an integer during the summary construction; recall that the summary does not contain literals, and represents literals from the input graph by summary nodes given fresh URIs. We decode the summary to retrieve the property URIs after it is fully built.

Currently, our summary graphs are built in memory, based on the Trove library[2], which improves on the performance of java.util

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

Object-based collections w.r.t. time and memory by using primitives instead of wrapper classes (`int` in place of `Integer` etc.). However, our algorithms can be adapted to manipulate more scalable (disk-based and/or distributed) data structures, and we are currently working in that direction.

Data nodes. The representation of each data node (whether typed or untyped) from $\mathtt{G}$ with exactly one data node in $\mathtt{H_G}$ is stored in a `TIntIntMap` map, denoted `rd`. Further, a `TIntObjectMap`$\langle$`TIntSet`$\rangle$ multi-map contains for each data node in $\mathtt{H_G}$ a set of represented data nodes from $\mathtt{G}$; we denote it `dr`.

Data node class set. A typed data node has exactly one (explicit) class set, which we store in a `TIntObjectMap`$\langle$`TIntSet`$\rangle$, denoted `dcls`. Observe that this in fact models the summary type edges; each key in the map represents a subject of a type triple and the object is the class from its respective class set.

Data node properties. For mapping data properties to their sources (targets, respectively) we use a `TIntIntMap`, denoted `dpSrc` (`dpTarg`, respectively). For each source (target) data node, a set of its adjacent data properties is stored in a `TIntObjectMap`$\langle$`TIntSet`$\rangle$, denoted `srcDps` (`targDps`). These structures will be used in weak and typed weak summaries, to facilitate merging of data nodes. Observe that in $\mathtt{TW_G}$ only untyped data nodes may be merged, so the typed data nodes of $\mathtt{TW_G}$ will not be stored in these structures. On the other hand, during the summarization of data triples in $\mathtt{W_G}$ all of them are untyped at that point since the typing is yet unknown, so all the data nodes attached to some data properties in $\mathtt{W_G}$ will be stored in the described structures.[3]

Data edges/triples. We denote with `dtp` the mapping of data properties to data triple(s) they appear in.
Weak: A `TIntObjectMap` maps each data property from $\mathtt{W_G}$ to a summary data triple represented by a `DataTriple` Java object. Observe that a distinct data property may appear in only one data triple in $\mathtt{W_G}$.
Typed weak & strong: A Trove map `TIntObjectMap`$\langle$`Collection`$\langle$`DataTriple`$\rangle\rangle$ stores, for each data property from $\mathtt{TW_G}$ and $\mathtt{S_G}$, the collection of data triples it appears in. Thus, a data property may appear in multiple data triples of $\mathtt{TW_G}$ and $\mathtt{S_G}$, depending on the type of the subject and object of the considered data triple.

Representing class sets. Any class set from $\mathtt{G}$ is represented by exactly one data node in $\mathtt{TW_G}$ and this information is stored in a `TObjectIntMap`$\langle$`TIntSet`$\rangle$, denoted `clsd`.

Cliques. All source/target property cliques of a strong summary are maintained in two `List`$\langle$`TIntSet`$\rangle$ lists, denoted `sc` and `tc`, respectively.
Clique IDs: As an integer ID of each clique, we take its index in the respective lists (`sc` and `tc`), incremented by one.

Mapping $\mathtt{G}$ data nodes to cliques. Two `TIntIntMap` maps store the mapping of each $\mathtt{G}$ subject to the ID of its source clique, and of each $\mathtt{G}$ object to the ID of its target clique. These two maps are denoted `sToSc` and `oToTc`, respectively.

Mapping cliques to $\mathtt{S_G}$'s data nodes. Finally, each clique is represented by a data node in $\mathtt{S_G}$ and this mapping of a clique ID to its representative data node is stored in two `TIntObjectMap`$\langle$`TIntSet`$\rangle$ maps, denoted `scToSrc` and `tcToTarg`.

[3]In both cases, as the choice of the structures shows, there can be only one untyped source and one untyped target data node per distinct data property, while any untyped data node may be attached to multiple data properties.

Algorithm 1 Summarizing data triples in $\mathtt{W_G}$

Input: Data triples table $\mathtt{D_G}$, the summary $\mathtt{W_G}$
Output: Data triples represented in $\mathtt{W_G}$

```
1: data ← EVAL(SELECT s, p, o FROM D_G)
2: for each s p o in data do
3:     src ← GETSOURCE(s, p, W_G)
4:     targ ← GETTARGET(o, p, W_G)
5:     // GETTARGET may have modified src and vice-versa
6:     src ← GETSOURCE(s, p, W_G)
7:     targ ← GETTARGET(o, p, W_G)
8:     if !EXISTSDATATRIPLE(W_G, src, p, targ) then
9:         CREATEDATATRIPLE(W_G, src, p, targ)
10:    end if
11: end for
12: procedure CREATEDATATRIPLE(W_G, src, p, targ)
13:    dtp.put(p, src p targ)
14:    dpSrc.put(p, src), srcDps.put(src, p)
15:    dpTarg.put(p, targ), targDps.put(targ, p)
16: end procedure
```

For the typed strong summary cliques are computed only for untyped data nodes.

Dictionary. The Postgres dictionary table comprises pairs of integer keys and string values, for all encoded resources from $\mathtt{G}$ and $\mathtt{H_G}$. The dictionary lookup is necessary on two occasions: (i) when splitting the encoded triples table into encoded data and type tables, we need the integer keys for RDF type and RDFS properties; (ii) to decode the summary after summarization we need the string values for all class and property keys, and possibly for decoding the resources represented by each summary data node. For (i) we load only the necessary entries from the dictionary table, while in (ii) we perform the summary decoding completely in Postgres (via joins with the dictionary table).

6.2 Algorithms

To build the weak and strong summaries, we summarize the data triples and then the type triples. In the typed weak summary though, we first summarize the type triples and then the data triples.

Data triples are read one by one, their subject and object are represented with source and target data nodes, possibly unifying the source and target nodes based on the information newly found, so as to ensure the existence of an homomorphism from $\mathtt{G}$ into the summaries (and in particular, that a given $\mathtt{G}$ node is consistently mapped to the same summary node, even though it may participate to several triples scattered over the graph).

We show below the algorithms for computing the weak summary $\mathtt{W_G}$; the algorithms corresponding to the typed weak, strong, and typed strong summaries appear in [5].

Summarizing data triples in $\mathtt{W_G}$. Algorithm 1 shows the procedure for summarizing data triples in a weak summary. The methods GETSOURCE( ) and GETTARGET( ) implement the representation functions, which map data nodes from $\mathtt{G}$ to data nodes in $\mathtt{W_G}$.

Algorithm 2 shows how we map the subjects of data triples in $\mathtt{G}$ to data nodes in $\mathtt{W_G}$. 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 5). In the cases when one of the nodes exist and the other does not, we simply use the existing node as src (lines 6-10). Moreover, if src_s had not existed, it means that `s` was unrepresented, so we assign src as its representative (line 7). On

Algorithm 2 Representing the subject s of a data property p in G with a data node in W_G

Input: s, p, W_G
Output: Data triples represented in W_G
Variables:
src_u - data node representing an untyped source of p
src_s - data node representing a (possibly typed) resource s

```
1: procedure GETSOURCE(s, p, W_G)
2:     src_u ← dpSrc.get(p)
3:     src_s ← rd.get(s)
4:     if src_u =⊥ ∧ src_s =⊥ then
5:         return CREATEDATANODE(W_G, s)
6:     else if src_u ≠⊥ ∧ src_s =⊥ then
7:         rd.put(s, src), dr.put(src, s)
8:         return src_u
9:     else if src_u =⊥ ∧ src_s ≠⊥ then
10:        return src_s
11:    else if src_u ≠⊥ ∧ src_s ≠⊥ then
12:        if src_s = src_u then
13:            return src_s
14:        else
15:            return MERGEDATANODES(W_G, src_s, src_u)
16:        end if
17:    end if
18: end procedure
19: procedure CREATEDATANODE(H_G, r)
20:    d ← NEWINTEGER( )
21:    rd.put(r, d), dr.put(d, r)
22:    return d
23: end procedure
```

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 (lines 12-13). When they differ, we have to merge them (line 15).

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 1 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.

Summarizing type triples in W_G. Algorithm 3 shows the implementation of weak summarization of type triples. First, we retrieve subjects and classes of all type triples (line 1) and we try to represent each resulting pair (lines 4-9) as follows. We look up the data node src representing s (line 14). 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 (line 14) and we add the class to the class set of src (line 18).

Alternatively, if src is empty in line 15, we know that s is *typed-only*, so we add it to the list of typed-only resources, and its class to the list of typed-only classes and leave it unrepresented for the time being (lines 6-8).

When all subjects attached to some data properties in G have been represented, if there are any typed-only resources, we represent

Algorithm 3 Summarizing type triples in W_G

Input: Type triples table T_G, the summary W_G
Output: Type triples represented in W_G

```
1: typ ← EVAL(SELECT s, c FROM T_G)
2: toRes ← []; toCls ← []
3: for each (s, c) in typ do
4:     repr ← REPRESENTTYPETRIPLE(W_G, s, c)
5:     if repr = false then
6:         toRes.add(s), toCls.add(c)
7:     end if
8: end for
9: if toRes.size > 0 then
10:    REPRESENTTYPEDONLY(W_G, toRes, toCls)
11: end if
12: procedure REPRESENTTYPETRIPLE(W_G, s, c)
13:    d ← rd.get(s)
14:    if d =⊥ then
15:        return false
16:    end if
17:    cls_d ← dcls.get(d), cls_d.add(d, c)
18:    return true
19: end procedure
20: procedure REPRESENTTYPEDONLY(W_G, toRes, toCls)
21:    d ← NEWINTEGER( )
22:    for each r in toRes do
23:        rd.put(r, d), dr.put(d, r)
24:    end for
25:    cls_d ← dcls.get(d)
26:    for each c in toCls do
27:        cls_d.add(c)
28:    end for
29: end procedure
```

them as well (line 11). The procedure REPRESENTTYPEDONLY (line 21) creates a single data node d representing all resources from the list of typed-only resources and having *all* the types of typed-only resources.

Algorithm complexity. Our algorithms incur I/O costs linear in the size of G (which we read from the disk-based database). They also evaluate several joins queries through the RDBMS. In general, the complexity of the evaluation plan depends on the concrete data, disk and memory settings etc., in our experiments the I/O cost dominated the join evaluation costs, and the overall algorithm complexity remains linear (with different constant factors for the different algorithms). This assumes all our data structures (Section 6.1) fit in memory. Our complexity analysis details can be found in [5].

7. EXPERIMENTAL RESULTS

We ran the summary building tool on a machine with an Intel Xeon X5647 Processor (2.93GHz, 8 CPUs, 4 Cores per CPU) and 16Gb of memory.

The PostgreSQL version 9.3.2 was configured with 4 GB of shared buffers, 64 MB working memory, checkpoint segments of size 256 MB. We have run the summarization tool for various JDBC fetch sizes and dataset sizes and have settled on 100,000 as the optimal fetch size for our hardware setting. The experiments were run with 16 GB of RAM assigned to the Java Virtual Machine.

We generated the weak, strong, typed weak and typed strong summary for The Berlin SPARQL Benchmark [3] (BSBM) dataset of various sizes. Figure 11 reports the number of data nodes and the number of all the nodes, respectively, in the four summaries.

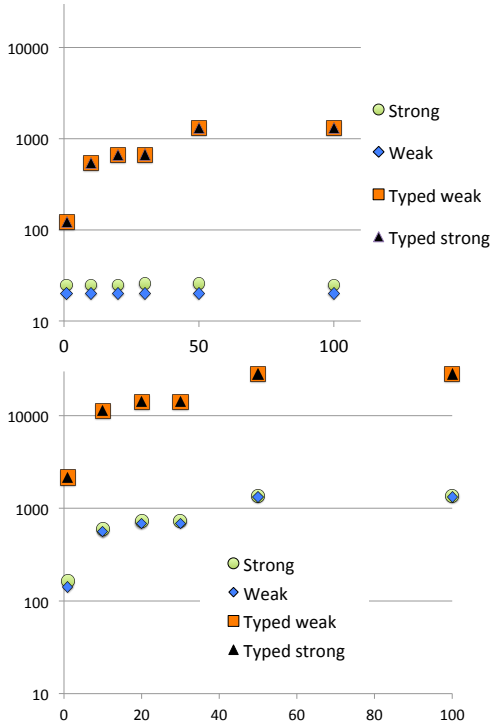

Figure 11: The numbers of data nodes (top) and all nodes (bottom) in BSBM summaries.

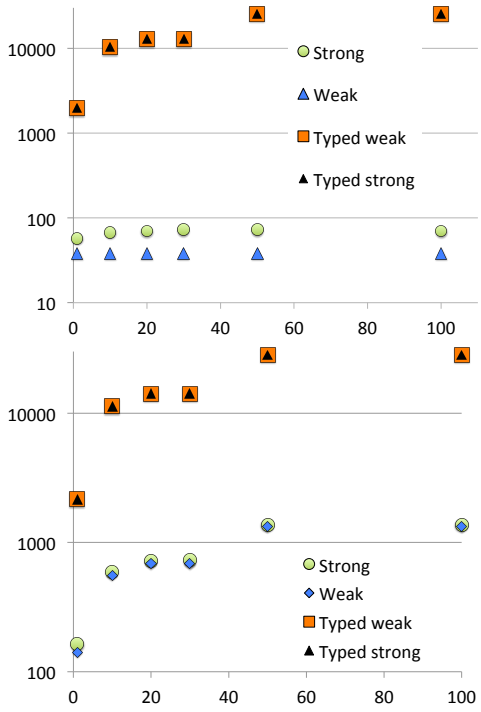

Figure 12: The numbers of data edges (top) and all edges (bottom) in BSBM summaries.

The horizontal axis is numbered in millions of triples in the input, while the vertical axis is in logarithmic scale. The number series are ordered so as to improve readability; observe that $\mathtt{S_G}$ and $\mathtt{W_G}$ numbers are very close to each other, and so are the $\mathtt{TW_G}$ and $\mathtt{TS_G}$ numbers (the respective dots sometimes overlap). This shows that in practice there is not a big difference between sharing an input/output clique directly, as nodes are required to do in order to be summarized together in $\mathtt{S_G}$, and through a chain of cliques, as it is the case in $\mathtt{W_G}$. Isolating typed data nodes from the others, as we do in $\mathtt{TW_G}$ and $\mathtt{TS_G}$, does have a strong impact, multiplying the number of data nodes by a factor between 5 and 50. Further, for BSBM data, the number of class nodes (the difference between the two numbers recorded in 11) is significantly higher than the number of data nodes (by a factor of 5 up to 60) for the strong and weak summaries, demonstrating the very high summarization power (reduction in data nodes) of these summaries. The number of class nodes goes approximately from 100 to 1300 for all summary sizes.

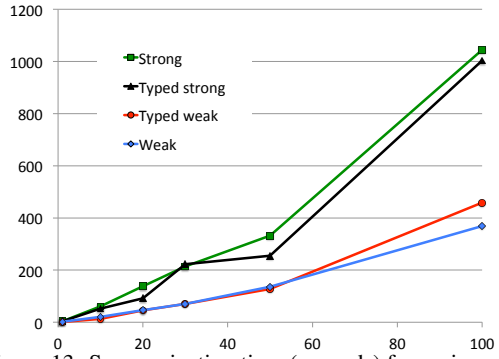

Figure 13: Summarization time (seconds) for various data sizes

Figure 12 presents the number of data edges, and the number of total edges in each summary. The figure confirms that the summaries $\mathtt{W_G}$ and $\mathtt{S_G}$, which we call *type-first* summaries (to highlight the importance they give to RDF types) behave similarly, while the typed ones are most complex but again similar to each other. The overall number of edges remains very moderate (at most 28210) remains very small compared to the data size (10 to 100 million triples); thus, the summary occupies at most 0.028 of the data size, and in the best case, only 2.8×10^{-4} of the data size.

Figure 13 depicts the time needed to build the summaries using our Postgres-based algorithms. The weak and strong summaries, are built in at most 8 minutes, whereas $\mathtt{TS_G}$ takes up to 1000 s (16 minutes) and $\mathtt{TW_G}$ up to 32 minutes. We find these times acceptable for a centralized implementation based on a DBMS, especially considering that summarization is an off-line task. The building time increases with the data size, and is higher for the strong and typed strong summaries than for the other two. This is because building strong summaries also requires actually computing the cliques, whereas for the weak ones, this is not needed.

Similar summary size and construction time metrics for other popular RDF datasets can be found in [5].

8. RELATED WORK

OEM and XML summaries. Summaries have long been studied in the context of semistructured data. Dataguides [10] were introduced to summarize semistructured OEM graphs, similar to RDF, but assumed to have a "root" node, from which all others are accessible; this may not hold for RDF. A Dataguide features *exactly once each path in the original graph*, and *each Dataguide path corresponds to one path in the graph*. This allows for *several* distinct dataguides, whose construction from the graph is shown [17] to amount to constructing a deterministic finite automaton out of a non-deterministic one; Dataguides construction has worst-case exponential time complexity, thus is not in general feasible. An algo-

rithm is provided for building *strong* Dataguides, used as a basis for indexing. The 1-index [16] groups together OEM or XML nodes that are reachable by exactly the same set of paths. Later works focused on indexes for supporting XML path queries [6, 12], or path-based XML summarization into graphs [7]. All these works differ from ours, because the input is a tree or DAG and/or because it lacks types and implicit information.

Graph summarization. Graph summarization has been very intensively studied, in particular through mining or clustering; large-scale graph processing is also a hot topic. A large number of works build on the idea of Dataguides for graph data, oftentimes referred to as *structural indexes*, which bear a similarity to graph summaries, both being a reduced version of the input graph and collapsing nodes based on some common attributes.

Our focus is on *RDF graphs* with *implicit* data, for which we devised *query-oriented* summaries, which are RDF graphs themselves and may be computed on a variety of platforms.

Graph *cores* have been studied in [9]. A graph core `C` for a given graph `G` is a graph such that an isomorphism exists between `G` and `C`, and `C` is the smallest graph with this property. Our summaries are not cores of the incoming graph `G`, since we cannot guarantee a homomorphism from either summary to the graph `G`. In exchange, both summary versions we consider can be built in polynomial time in the size of `G`, while computing the core is much harder.

Bisimulation-based approaches group nodes by the similarity of their neighborhood [14, 19]. In [14] bisimulation is used to summarize graphs from the LOD cloud, focusing on the distribution of classes and properties across LOD sources. The resulting *resource-oriented* summary comprises unlabeled edges. [19] utilizes bisimulation to construct a structural index from structure patterns exhibiting certain edge labels that comprise paths of some maximum length. The main problem with bisimulation is that as the size of the neighborhood increases, the size of bisimulation grows exponentially and can be as large as the input graph. Thus, as we aim for both complete and compact summaries, bisimulation is not a good fit.

To overcome the problem with bisimulation, [11] suggests locality-based summaries, generated by a graph partitioning algorithm. Nonetheless, a reduction of the input RDF graph is necessary which is achieved by removing triples having properties with literal values, thus resulting in an incomplete summary. Recall that we wish to preserve queries comprising properties with literal values as well.

A *triple-oriented* structural index for RDF data is built in [18] as a non-RDF graph, where a node comprises a set of triples forming a data partition, while an edge describes the way in which triples from its adjacent nodes join. [20] proposes a tree RDF index, storing regions defined by center vertices, limited to property paths and built assuming that the input RDF graph is saturated.

In [13], RDF classes are inferred based on the common properties of resources. Thus, only common source patterns are analyzed, while the common targets and property paths are not considered. Further, the rdf:type properties that a dataset may comprise are simply ignored.

[4] explores alternative RDF summaries w.r.t. graph homomorphism and trades precision for efficiency in computing them.

Summarizing implicit data is not considered in these works.

9. CONCLUSION

In this work we proposed four kinds of summaries, derived based on node similarity w.r.t. their types or connections. We offered a formal study of the desirable summary properties, notably the representativeness and accuracy, which illustrate the trade-offs between different kinds of summaries. Moreover, this is the first work to focus on partially explicit, partially implicit RDF graphs. The experiments on several synthetic and real-life RDF datasets confirm the practical feasibility of our centralized implementation. Future work will focus on improving scalability by leveraging a massively parallel platform such as Spark. Additionally, we are interested in devising advanced visualizations of our RDF summaries.

10. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An empirical study of real-world SPARQL queries. *CoRR*, abs/1103.5043, 2011.

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

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

[5] Š. Čebirić, F. Goasdoué, and I. Manolescu. Query-Oriented Summarization of RDF Graphs. Research Report RR-8920, INRIA Saclay ; Université Rennes 1, Sept. 2016.

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

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

[8] F. Goasdoué, I. Manolescu, and A. Roatiş. Efficient query answering against dynamic RDF databases. In *EDBT*, 2013.

[9] C. Godsil and G. Royle. *Algebraic graph theory*. Springer-Verlag, 2001.

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

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

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

[13] K. Kellou-Menouer and Z. Kedad. A clustering based approach for type discovery in RDF data sources. In Extraction et Gestion des Connaissances, 2015.

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

[15] D. Lanti, M. Rezk, G. Xiao, and D. Calvanese. The NPD benchmark: Reality check for OBDA systems. In *EDBT*, 2015.

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

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

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

[19] T. Tran, G. Ladwig, and S. Rudolph. Managing structured and semistructured RDF data using structure indexes. *IEEE Trans. Knowl. Data Eng.*, 25(9):2076–2089, 2013.

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

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

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

[23] Jena. `http://jena.sourceforge.net`.