Semantic Web 1 (2016) 1–5
IOS Press

Quality Metrics For RDF Graph Summarization

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Mussab Zneika [a] Dan Vodislav [a] and Dimitris Kotzinos [a]
[a] *ETIS, UMR 8051 Paris Seine University, University of Cergy-Pontoise, ENSEA, CNRS*
E-mail: Mussab.Zneika@ensea.fr, Dan.Vodislav@u-cergy.fr,Dimitrios.Kotzinos@u-cergy.fr

Abstract. RDF Graph Summarization pertains to the process of extracting concise but meaningful summaries from RDF Knowledge Bases (KBs) representing as close as possible the actual contents of the KB both in terms of structure and data. RDF Summarization allows for better exploration and visualization of the underlying RDF graphs, optimization of queries or query evaluation in multiple steps, better understanding of connections in Linked Datasets and many other applications. In the literature, there are efforts reported presenting algorithms for extracting summaries from RDF KBs. These efforts though provide different results while applied on the same KB, thus a way to compare the produced summaries and decide on their quality and best-fitness for specific tasks, in the form of a quality framework, is necessary. So in this work, we propose a comprehensive Quality Framework for RDF Graph Summarization that would allow a better, deeper and more complete understanding of the quality of the different summaries and facilitate their comparison. We work at two levels: the level of the ideal summary of the KB that could be provided by an expert user and the level of the instances contained by the KB. For the first level, we are computing how close the proposed summary is to the ideal solution (when this is available) by defining and computing its precision, recall and F-measure against the ideal solution. For the second level, we are computing if the existing instances are covered (i.e. can be retrieved) and in what degree by the proposed summary. Again we define and compute its precision, recall and F-measure against the data contained in the original KB. We also compute the connectivity of the proposed summary compared to the ideal one, since in many cases (like, e.g., when we want to query) this is an important factor and in general in RDF, datasets that are linked within are usually used. We use our quality framework to test the results of three of the best RDF Graph Summarization algorithms, when summarizing different (in terms of content) and diverse (in terms of total size and number of instances, classes and predicates) KBs and we present comparative results for them. We conclude this work by discussing these results and the suitability of the proposed quality framework in order to get useful insights for the quality of the presented results.

Keywords: Quality framework; Quality metrics; RDF Summarization; Linked Open Data; RDF Query processing

1. Introduction

RDF has become one of the major standards in describing and publishing data, establishing what we call the Semantic Web. Thus, the amount of RDF data available increases fast both in size and complexity, making the appearance of RDF Knowledge Bases (KBs) with millions or even billions of triples something usual. Given that RDF is built on the promise of linking together relevant datasets or KBs and with the appearance of the Linked Open Data (LOD) cloud, we can now query KBs (both standalone or distributed) with millions or billions of triples altogether. This increased size and complexity of RDF KBs has a direct impact on the evaluation of the RDF queries we express against these RDF KBs. Especially on the LOD cloud, we observe that a query against a big, complex, interlinked and distributed RDF KB might retrieve no results at the end because either the association between the different RDF KBs is weak (is based only on a few associative links) or there is an association

at the schema level that has never been instantiated at the actual data level. Moreover, a lot of these RDF KBs carry none at all or only partial schema information (mainly contain instances built and described separately). Additionally, in the LOD cloud the number of KBs which do not use the full schema or they use multiple schemas is increased due to the absence of the schema information which describes the interlinks between the datasets and the combinatorial way of mixing vocabularies.

One way to address the concerns described above is by creating summaries of the RDF KBs. Thus we allow the user or the system to decide whether or not to post a query, since she knows whether information is present or not based on the summary. This would provide significant cost savings in processing time since we will substitute queries on complex RDF KBs with queries first on the summaries (on much simpler structures with no instances) and then with queries only towards the KBs that we know will produce some useful results. Graph summarization techniques would allow the creation of a concise representation of the KB regardless of the existence or not of schema information in the KB. Actually, the summary will represent the actual situation in the KB, namely should capture the *existing/used* classes and relationships by the instances and not what the schema proposes (and might have never been used). This should facilitate the query building for the end users with the additional benefit of exploring the contents of the KB based on the summary. This is true regardless if we use heterogeneous or homogeneous, linked or not, standalone or distributed KBs. In all these cases we can use the RDF summary to concisely describe the data in the RDF KB and possibly add useful information for the RDF graph queries, like the distribution and the number of instances for each involved entity.

In the literature we can find various efforts proposing summarization techniques for RDF graphs. These techniques, presented briefly in section 3, come from various scientific backgrounds ranging from generic graph summarization to explicit RDF graph summarization. While all promise that they provide correct, concise and well-built summaries so far has been very little effort into addressing in a comprehensive and coherent way the problem of evaluating these summaries against different criteria and have some mathematical metrics to describe the quality of the results. Only sparse efforts have been reported, usually tailored to a specific method or algorithm. So with this paper, we aim to cover the gap that exists in the literature and provide a comprehensive Quality Framework for RDF Graph Summarization that would allow a better, deeper and more complete understanding of the quality of the different summaries and facilitate their comparison. We propose to take into account the possibility to compare the summary against two levels of information possibly available for a RDF KB. In the case where an ideal summary is available, either because it has been proposed by a human expert or because we can assume that an existing schema represents perfectly the data graph, we compare the summary provided by the algorithms with it and use similarity measures to compute its precision and recall against the ideal summary. If this is not available or usually in addition to it, we compute the percentage of the instances represented by the summary (including both class and property instances). This provides us with the understanding of how well the summary covers the KB. Moreover we introduce a metric to cover the coherency dimension if the problem, i.e. how well connected the computed summary graph is. One can combine at the end the two overall metrics or use them independently. In order to validate the proposed quality metrics, we evaluated three of the most promising RDF graph summarization algorithms and report on the quality of their results over different datasets with diverse characteristics. We should note here that the proposed Quality Framework is independent of any of the algorithms evaluated but it is suitable in providing a common ground to compare them.

This is why we could summarize our contribution as presenting a quality framework that:

- Evaluates the quality of RDF Graph Summaries, where a combined effort is made to summarize, while preserving existing important semantics, basic structure and coherence;
- Works at different levels, both trying to understand the comparison of the two summaries (ideal and computed) at the schema and the instance levels, while previous approaches were mainly dealing with one level (which corresponds to the instance level in our approach);
- Provides novel customized definitions for precision and recall for summaries, thus allowing better capturing of the quality of the results âĂŞ so we go beyond the standard property and recall definitions;
- Adds the discussion on the connectivity of the computed summary and tries to promote summaries that are more connected. This is quite cru-

cial if we want to later on query the summary using standard RDF tools.

So, the proposed framework allows for understanding the quality of the different summaries at different levels. The users can pick the metrics that better fit to the task for which they need to pick a summary.

The paper is structured as follows: Section 2 introduces some of the foundations of RDF and RDFS, which are useful for defining later on some concepts in our work; Section 3 provides a review of the existing works around quality metrics in graph summarization; while Section 4 presents our proposed Quality Metrics for RDF Graph Summaries. Section 5 presents the three of the most promising RDF Graph Summarization algorithms in the literature that are compared using the proposed Quality Framework in Section 6, where the extensive experiments performed in order to validate the appropriateness of the proposed metrics are reported. We then conclude our paper in section 7.

2. Preliminaries

2.1. RDF

As per the W3C standards, the RDF data model represents data on the Web as a set of triples of the form (s, p, o), expressing the fact that for the subject s, the value of the property p is the object/value o. RDF data can also be represented as a labeled directed graph in which entities (subjects/objects) are represented as nodes and property instances (expressed by the triples) as labeled directed edges. RDF datasets are usually accompanied with a RDF Schema[1], which provides a data-modeling vocabulary for RDF data. RDF Schema (RDFS) defines a set of classes for declaring the resource types and a set of properties for declaring the resource relationships and attributes. RDF Schema describes relations between classes and properties, but could also be represented as a directed labeled graph, where the labeled nodes represent the classes and the labeled edges represent properties relating class instances.

Let C, P, I and L be the sets of *class* Universal Resource Identifiers (URIs), *property* URIs, *instance* URIs and *literal* values respectively, and let T be a set of RDFS standard properties (*rdfs:range*, *rdfs:domain*, *rdf:type*, *rdfs:subClassOf*, etc.). The concepts of RDF schemas and instances can be formalized as follows.

[1] https://www.w3.org/TR/2004/REC-rdf-schema-20040210/

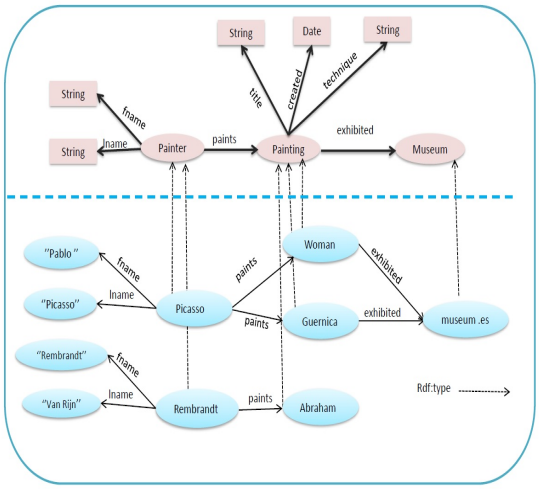


Fig. 1.: RDF Schema and data graphs

Definition 1 ***(RDF schema graph).*** *An* RDF schema graph $G_s = (N_s, E_s, \lambda_s, C, P, T)$ *is a directed labeled graph where:*

- N_s *is the set of nodes, representing classes and properties.*
- $E_s \subseteq \{(x, \alpha, y) | \, x \in N_s, \alpha \in T, y \in N_s\}$ *is the set of labeled edges.*
- $\lambda_s : N_s \longrightarrow C \cup P$ *is an injective node labeling function that maps nodes of* N_s *to class and property URIs.*

We note $\lambda_e : E_s \longrightarrow T$ *the edge labeling function that associates to each edge* $(x, \alpha, y) \in E_s$ *the RDFS standard property URI* $\alpha \in T$.

Definition 2 ***(RDF data graph).*** *An* RDF data graph $G_i = (N_i, E_i, \lambda_i, I, P, L, C)$ *is a directed labeled graph where:*

- N_i *is the set of nodes, representing instances, literals and* class *URIs .*
- $E_i \subseteq \{(x, \alpha, y) | \, x \in N_i, \alpha \in P, y \in N_i\}$ *is the set of labeled edges.*
- $\lambda_i : N_i \longrightarrow I \cup L \cup C$ *is a node labeling function that maps nodes of* N_i *to instance URIs,* class *URIs or literals.*

We note $\lambda_{ei} : E_i \longrightarrow P$ *the edge labeling function that associates to each edge* $(x, \alpha, y) \in E_i$ *the property URI* $\alpha \in P$.

Example 1 *The upper part of Figure 1 shows a visualization of an RDF schema graph example for the*

cultural domain, representing only class nodes, while properties are illustrated as edges between classes. For example, the class Painter denotes the set of instances which represent painter entities, while property paints relates class Painter instances to class Painting instances. The lower part of Fig. 1 depicts an instance (data) graph building on this schema. This graph represents 6 different resources. For example the resource Picasso is an instance of the Painter class having properties fname, lname and paints.

Type edges. Edges labeled with *rdf:type* in the RDF data graph explicitly describe the type (class) of an instance, e.g. dashed edges in Fig. 1, where for instance *Picasso* is declared to be a *Painter*. We will note in the following the type edge label with τ. For an instance $x \in N_i$, we define $Types(x) = \{\lambda_i(y) | (x, \tau, y) \in E_i\}$ to be the set of types related to the node x via an explicit type edge definition, e.g., *Types*(*Picasso*)= $\{Painter\}$, while *Types*(*Guernica*)= $\{Painting\}$.

Properties. We denote by $Properties(x) = \{\alpha : \forall(x, \alpha, y) \in E_i : \alpha \neq \tau \wedge \lambda_i(y) \in I \wedge x \in N_i\}$, a set of labels of the non-Type edges which associate the node x with a set of entity nodes(nodes labeled by *instance* URIs).

Attributes. We denote by $Attributes(x) = \{\alpha : \forall(x, \alpha, y) \in E_i : \alpha \neq \tau \wedge \lambda_i(y) \in L \wedge x \in N_i\}$ a set of labels of the non-Type edges which associate the node x with a set of literal nodes(nodes labeled by *literal* values) ,

Example 2 *The set of properties associated with Picasso node in our example are* $\{paints\}$*, while the set of attributes of Picasso node are* $\{fname, lname\}$.

Definition 3 ***(Class Instances)*** *We denote by* $instances(c \in C) = \{\lambda_i(x) : \forall(x, \tau, y) \in E_i : y = c\}$ *a set of labels of the nodes which are associated to the node c (represent the class) via a typed edge* τ*, or in other words the set of resources(subjects) belonging to the class c.*

Definition 4 ***Property Instances.*** *We denote by* $instances(p \in P) = \{\lambda_i(x) : \forall(x, \alpha, y) \in E_i : \alpha = p\}$ *a set of labels of the nodes which are associated to other nodes via the property p, or in other words, is the set of resources (subjects) having the property p.*

Example 3 *The set of instances of the class Painting in our example are* $\{Woman, Guernica, Abraham\}$*, while the set of instances of the property exhibited (which is one of the Painting class's properties) are* $\{Woman, Guernica\}$

Patternid	classes *C*	Properties *Pr*	Instances *Ins*	*Sup*
1	Painter	fname, lname, paints	Picasso, Rembrandt	2
2	Painting	exhibited	Woman, Guernica	2
3	Painting	-	Abraham	1
3	Museum	-	museum .es	1

Table 1: Knowledge patterns example (computed based on the bisimilarity relation)

2.2. Knowledge patterns

A knowledge pattern (or simply pattern from now on) characterizes a set of instances in an RDF data graph that share a common set of types and a common set of properties. More precisely:

Definition 5 ***(Knowledge Pattern)*** *A knowledge pattern KP in an RDF data graph is a quad* (Cl, Pr, Ins, SUP)*, where* $Cl = \{c_1, c_2, ..., c_n\} \subseteq C$ *is a set of classes,* $Pr = \{Pr_1, Pr_2, ..., Pr_m\} \subseteq P$ *is a set of properties,* $Ins \subseteq I$ *is the set of instances that have all the types of Cl and all the properties of Pr, and* $SUP = |Ins|$ *is called the support of the knowledge pattern in the RDF data graph (i.e. the number of instances that have all types and all properties).*

We introduce the term knowledge pattern because it is not sure that all summarization algorithms will produce something that can be necessarily defined as an RDF class or RDF property and also because we want to differentiate from the classes/properties of the ideal summary when we compare the two.

Pattern Instances. We denote by $instances(pa) = Ins$ a set of the original KB resources having the same set of the properties/types of the pattern pa, or in other words is the set of bindings for the $?a$ variable over the RDF data graph in the following SPARQL-like conjunctive pattern: $\{<?a, \tau, c_1 >, <?a, \tau, c_2 >, ..., <?a, \tau, c_n >, <?a, Pr_1, ?b_1 >, <?a, Pr_2, ?b_2 >, ..., <?a, Pr_m, ?b_m >\}$, e.g. instances(p2)= { Woman, Guernica }

Example 4 *Table 1 shows possible patterns which can be extracted from the RDF instance graph depicted in Figure 1 based on a forward bisimilarity relation.*

2.3. RDF Summary Graph

In order to be able to properly address the problem of creating RDF summaries of LOD/RDF graphs we need to define what an RDF Summary is. We follow

the Definition 5 of the Knowledge Pattern and we define the summary as a set of the Knowledge Patterns that the algorithms compute. The proposed Quality Framework will help the user understand which proposed summary will better represent the original KB in terms of structure, coverage and connectivity. So, the Summary graph is defined as follows:

Definition 6 ***(Summary graph)***
Let $G = G_s \bigcup G_i$ *be an RDF graph, including both schema and instance information. The* RDF Summary Graph $SG = \{C, P, I\}$ of G *is a graph consisting of a set of Knowledge patterns* Π*, where* $pa \in \Pi$ *and for which:*

- $C = \bigcup_{pa \in \Pi} pa.Cl$ *the set of classes of* SG*;*
- $P = \bigcup_{pa \in \Pi} pa.Pr$ *the set of properties of* SG*;*
- $I = \bigcup_{pa \in \Pi} pa.Ins$ *the set of the instances represented by the summary.*

2.4. Bisimilarity Relation

Bisimilarity in a directed labeled graph is an Equivalence Relation defined on a set of nodes N, such that two nodes (u,v) are *bisimilar* if and only if the set of ougoing edges of u is equal to the set of outgoing edges of v and also, all successor nodes of u and v must be bisimilar (in other words, the outgoing paths of u and v are similar). We call the *bisimilarity* relation when defined based on outgoing paths, Forward (FW) Bisimilarity, and when it is based on incoming paths, Backward (BW) Bisimilarity. More on bisimilarity can be found at [19]. In the example presented in Table 1 we can notice that *Woman* and *Guernica* are grouped together by an algorithm that is based on bisimilaruty, while *Abraham* is missing an outgoing link (exhibited) and thus is grouped separately.

3. Related work

RDF graph summarization has been intensively studied, with various approaches and techniques proposed to summarize the RDF graphs, which could be grouped into four main categories:

1. *Aggregation and grouping approaches* [31,32, 33,36,26], which are based on grouping the nodes of an input RDF graph G into clusters/groups based on the similarity of the attribute values and on the neighborhood relationships associated with nodes of G.
2. *Structural extraction approaches* [12,23,14,22, 15,30,35,20,21,9,24,25,29], which define an equivalence relation on the nodes of the RDF data graph G, usually based on the set of incident graph paths. This allows extracting a form of schema for G by representing the equivalence classes of nodes of G as nodes in the summary graph, characterized by the set of incident paths of each class.
3. *Logical compression approaches* [17,18], which are based on compressing the RDF datasets by generating a set of logical rules from the dataset and removing triples that can be inferred from these rules. The summary graph is then represented by a compressed graph and set of logical decompression rules, with the drawback that such approaches do not produce RDF graphs as summaries.
4. *Pattern-mining-based approaches* [37,16,38], which are based on extracting frequent patterns from the RDF graph, then composing them to build an approximated summary graph.

Typically, the RDF summarization methods proposed so far do not address in depth the problem of the quality of the produced RDF summaries. A noticeable exception is the work in [11], which proposes a model for evaluating the precision of the graph summary, compared to a gold standard summary which is a forward and backward bisimulation summary. The main idea of the precision model is based on counting the edges or paths that exist in the summary and/or in the gold summary graph. The precision of a summary is evaluated in the standard way, based on the number of true positives (the number of edges existing in the summary and in the input graph) and false positives (the number of invalid edges and paths existing in the summary but not in the input graph). The first limitation of this quality model [11] is that it works only with the summaries generated by an algorithm that uses a bisimulation relation. Similarly to our quality framework, they consider the precision at instance level, i.e how many of summary class/property instances are correctly matched in the original KB. Unlike to our work, this work does not consider the recall at the instance level, because it claims that the way summarization algorithms work, does not allow them to miss any instance. But this is not always correct, e.g. the approximate RDF summarization algorithms like [37,38] might miss a lot of instances. As it is well-known, the precision alone cannot accurately assess the qual-

ity, since a high precision can be achieved at the expense of a poor recall by returning only few (even if correct) common paths. Additionally and unlike our work, this model does not consider at all the quality of the summary at the schema level, e.g. what if one class/property of the ideal summary is missing or an extra one is added or a property is assigned to the wrong class. In all these cases, the result will be the same, while it is obvious that it should not. Finally, [11] is missing completely any notion of evaluating the connectivity of the final summarization result.

One more effort, [13], addressing the quality of hierarchical dataset summaries is reported in the literature. The hierarchical dataset summary is based on the grouping of the entities in the KB using their types and the values of their attributes. The quality of a given/computed hierarchical grouping of entities is based on three metrics: (1) the weighted average coverage of the hierarchical grouping, i.e. the average percentage of the entities of the original graph that are covered by each group in the summary; (2) the average cohesion of the hierarchical grouping where the cohesion of a subgroup measures the extent to which the entities in it form a united whole; and (3) the height of a hierarchical grouping, i.e. the number of edges on a longest path between the root and a leaf. The main limitation of this approach is that it works only with the hierarchical dataset summaries, since metrics like the cohesion of the hierarchical groups or the height of the hierarchy cannot be computed in other cases. Moreover, the proposed groupings provide a summary that can be used for a quick inspection of the KB but cannot be queried by any of the standard semantic query languages. On the other hand and similarly to our quality framework, [13] considers the recall (named coverage) at instance level, i.e how many of the instances of the original KB are correctly covered by the summary concepts. Contrary to our work, this model does not consider at all the quality of the summary at the schema level. Notions from [13] can also be found in the current paper, where algorithms like [37,38] that rely on approximation get penalized if they approximate too much, in fact loosing the cohesion of the instances represented by the computed knowledge patterns.

Besides that, only few efforts have been reported in the literature addressing the quality of the schema summarization methods in general [34,28,10], i.e. the quality of the RDF schema that can be obtained through RDF summarization. The quality of the RDF schema summary in [28] is based on expert ground truth and is calculated as the ratio of the number of classes identified both by the expert users and the summarization tool over the total number of classes in the summary. The main limitation of this approach is that it uses a Boolean match of classes and fails to take into account similarity between classes when classes are close but not exactly the same as in the ground truth or when classes are represented by more than one class in the summary. Works in schema matching (e.g. [34]) are also using to some extend similar metrics like recall, precision, F1-Measure commonly used in Information Retrieval, but are not relevant to our work since even if we consider an RDF graph summary as an RDF schema, we are not interested in matching its classes and properties one by one, since as stated above this binary view of the summary results does not offer much in the quality discussion. Additionally these works do not take into account issues like the size of the summary.

To the best of our knowledge, this is the first effort in the literature to provide a comprehensive Quality Framework for RDF Graph Summarization, independent of the type and specific results of the algorithms used and the size, type and content of the KBs. We provide metrics that help us understand not only if this is a valid summary but also if a summary is better than another in terms of the specified quality characteristics. And we can do this by assessing information, if available, both at schema and instance levels.

4. Quality Assessment Model

In this section we present a comprehensive and coherent way to measure the quality of RDF summaries produced by any algorithm that summarises RDF graphs. The framework is independent of the way algorithms work and makes no assumptions on the type or structure neither of the input nor of the final results, besides being expressed in RDF; this is required in order to guarantee the validity of the result but can be easily extended to other cases of semantic summarisation, like for graphs expressed in OWL or Description Logics. In order to achieve this, we work at two levels:

- *schema level*, where if an ideal summary exists, the summary is compared with it by computing the precision and recall for *each* class and its neighbourhood (properties and attributes having as domain that class) of the produced summary against the ideal one; we also compute the pre-

Measure	What it indicates	How it is computed
$SchemaRecall(c, \Pi)$	Schema recall of a class c over the set of patterns Π.	Divide the number of relevant class's properties that are reported in Π on the total number class's properties.
$SchemaRec_{ClassAll}$	Overall schema class recall.	Compute the mean of the various $SchemaRecall(c, \Pi)$ for all the classes c of the ground-truth Schema S.
$Sim(pa, c)$	Similarity between a class c and a pattern pa.	Divide the number of common properties between the class c and the pattern pa on the total number of pa propertiespa.
$Nps(c)$	The number of patterns that represent the class c	Count all the patterns having $Sim(pa, c)$>0.
$SchemaPrec(c, \Pi)$	Schema class precision of the class c over the set of patterns Π.	Sum the $sim(pa, c)$ for all the patterns of Π.
$SchemaPrec_{ClassAll}$	Overall schema class precision.	Compute the mean of the various class precision values $SchemaPrec(c, \Pi)$ for all the retrieved classes of the ground-truth Schema S.
$SchemaF1_c$	Schema class F-Measure.	Combine the $SchemaPrec_{ClassAll}$ and $SchemaRec_{ClassAll}$ using the standard formula of the F-Measure.
$SchemaRec_{PropertyAll}$	Overall Schema property recall.	Divide the number of relevant properties extracted by the summary on the total number of properties in the ground truth schema.
$SchemaF1_p$	Schema property F-Measure.	Combine the $SchemaPrec_{PropertyAll}$ and $SchemaRec_{PropertyAll}$ using the standard formula of the F-Measure
$SchemaF1$	Overall schema F-measure.	Combine the class schema F-Measure $SchemaF1_c$ and property schema F-Measure $SchemaF1_p$.

Table 2: Summary description of the proposed Schema Measures

cision and recall of the whole summary against the ideal one. The first will capture the quality of the summary at the local (class) level, while the second will give us the overall quality in terms of classes' and properties/attributes' precision and recall.

– *instance level*, where the coverage that the summary provides for class and property instances is calculated, i.e. how many instances will be retrieved if we query the whole summary graph. We use again precision and recall against the contents of the original KB.

At the end, a metric is presented that provides an indication of the quality of the graph summary by measuring whether or not the summary is a connected graph. Ideally, a summary should be a connected graph but this also depends on the actual data stored in the Knowledge Base. Thus a disconnected graph could be an indication of the data quality in the KB and not necessarily a problem of the summarization process. Nevertheless, we present it here as another indicator of the quality process, especially if the summary is compared with an ideal one, but for the reason mentioned before we avoid to combine it with the rest of the presented metrics. Finally, we discuss some results that combine these metrics and interpret their meaning.

4.1. Quality Model in the presence of an ideal summary (schema level)

In this section we present our quality assessment framework to evaluate the quality of an RDF graph summary against a ground truth summary **(S)** (e.g. one provided by an expert). We measure how close the proposed summary is to the ground truth summary by computing its precision and recall against this ground truth. We suggest that we compute both the precision and recall at the class and at the property level and at the overall summary level. Table 2 gives us a summary description of the schema-level proposed measures.

Precision and Recall for classes We present here the recall and the precision measures for the classes of the detected patterns against a ground truth summary S. We first introduce the recall over the classes which is the fraction of relevant classes that are reported in the summary. Given a set of knowledge patterns Π (as defined in Section 2.1 and referred commonly as patterns from now on) and a set of classes $C \in S$, we start by defining the recall of a class $c \in C$ over the set of patterns Π as the fraction of relevant class's properties (namely properties that have this class as their domain) that are reported in Π, we denote it by schema class recall $SchemaRec(c, \Pi)$:

$$SchemaRecall(c, \Pi) = \frac{|\bigcup_{pa \in \Pi} (A(c) \cap A(pa))|}{|A(c)|} \quad (1)$$

The A(pa) is the set of properties and attributes involved in the pattern pa, and the A(c) is the set of properties and attributes of the ideal class c. Thus, the overall summary recall using the classes $SchemaRec_{ClassAll}$ is computed as the mean of the various schema class recall $SchemaRecall(c, \Pi)$ for all the classes c of the ground-truth Schema S.

$$SchemaRec_{ClassAll} = \frac{1}{|C|} \sum_{c \in C} SchemaRecall(c, \Pi) \tag{2}$$

The precision is the fraction of retrieved classes and properties of the summary that are relevant. If a knowledge pattern of a summary carries a *typeof* link then this pattern is relevant to a specific class if the typeof points to this class, if not this is not relevant to this class. If no *typeof* information exists then we use the available properties and attributes to evaluate the similarity between a class and a pattern. Thus we define the L(c, pa) function to capture this information and we add this to the similarity function.

$$L(c, pa) = \begin{cases} 1, & if \text{typeof(pa)} = c \text{ or typeof(pa)} = \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

The similarity between a class c in the ideal summary and a pattern pa $Sim(pa, c)$ in the computed summary is defined as the number of common properties between class c and pattern pa divided on the total number of the properties of the patterns pa:

$$Sim(pa, c) = L(pa, c) * \frac{|A(c) \cap A(pa)|}{|A(pa)|} \tag{4}$$

Given that a class might be represented by more than one knowledge patterns, depending on the algorithm used, we are interested in introducing a way to penalize cases where this happens, thus favoring smaller summaries over bigger ones. We achieve this by introducing a weight function that allows us to reduce the similarity value if this is based on consuming multiple patterns. Thus we introduce the following exponential function, which uses coefficient a to allow variations if needed in the future, and is chosen based on experimental evaluation of the functions that could provide us with a smooth decay in similarity as patterns' number increases. The $Nps(c)$ is the number of patterns that represent the class c and $\alpha \in [1, 10]$.

We define the T(c, pa) function to capture if a pattern pa can be used to represent the class c; this function returns 1 if the similarity function between the pattern pa and c is bigger than zero (so the pattern covers some of the elements that define the class) and zero otherwise.

$$T(c, pa) = \begin{cases} 1, & if Sim(pa, c) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Based on the T(c, pa) function, the number of patterns $Nps(c)$ that represent the class is defined as follows:

$$Nps(c) = \sum_{pa \in \Pi} T(c, pa) \tag{6}$$

$$W(c) = e^{1 - \sqrt[\alpha]{Nps(c)}} \tag{7}$$

Based on this weight function we define the class precision metric for every pattern pa in the computed summary and every class c in the ground truth summary as follows:

$$SchemaPrec(c, \Pi) = W(c) * \frac{\sum_{pa \in \Pi} Sim(pa, c)}{Nps(c)} \tag{8}$$

Thus, we define the schema class precision $SchemaPrec_{ClassAll}$ as the mean of the various class precision values $SchemaPrec(c, \Pi)$ for all the classes of the ground-truth Schema S.

$$SchemaPrec_{ClassAll} = \frac{\sum_{c \in C} SchemaPrec(c, \Pi)}{|C1|} \tag{9}$$

where C1 $\subseteq$ C is the list of all the ground truth's retrieved classes, or in other words, is the list of the ground truth's classes for which $SchemaPrec(c, \Pi) > 0$.

However, neither precision nor recall alone can accurately assess the match quality. In particular, recall can easily be maximized at the expense of a poor precision by returning as many correspondences as possible. On the other side, a high precision can be achieved at the expense of a poor recall by returning only few (correct) correspondences. Hence it is necessary to consider both measures and and express this through a combined measure; we use the F-Measure for this

purpose, namely $SchemaF1_c$:

$$SchemaF1_c = 2 * \frac{SchemaPrec_{ClassAll} * SchemaRec_{ClassAll}}{SchemaPrec_{ClassAll} + SchemaRec_{ClassAll}} \quad (10)$$

Precision and Recall for properties The overall recall at the property level, namely $SchemaRec_{PropertyAll}$ is computed as the ratio between the number of common properties extracted by the summary and the ones in the ground truth summary divided by the number of properties in the ground truth summary:

$$SchemaRec_{PropertyAll} = \frac{|\bigcup_{pa \in \Pi} A(pa) \cap \bigcup_{c \in C} A(c)|}{|\bigcup_{c \in C} A(c)|} \quad (11)$$

We note that the schema precision at the property level in our experiments is always equal to 1 (see Section 6), which means that in our examples there are no false positives for properties. Summarization algorithms do not invent new properties but they might report some properties that are not present in the ground truth summary. So, precision for properties namely $SchemaPre_{PropertyAll}$, is computed as the ratio between the number of common properties between the extracted summary and the number of properties existing in the ground truth summary and is as follows:

$$SchemaPrec_{PropertyAll} = \frac{|\bigcup_{pa \in \Pi} A(pa) \cap \bigcup_{c \in C} A(c)|}{|\bigcup_{pa \in \Pi} A(pa)|} \quad (12)$$

Thus, the F-Measure for the schema properties, namely $SchemaF1_p$ will be calculated as:

$$SchemaF1_p = 2 * \frac{SchemaPrec_{PropertyAll} * SchemaRec_{PropertyAll}}{SchemaPrec_{PropertyAll} + SchemaRec_{PropertyAll}} \quad (13)$$

Overall Schema level F-measure After defining the individual metrics for the class schema F-Measure $SchemaF1_c$ and property schema F-Measure $SchemaF1_p$, we can define the combined overall schema F-measure $SchemaF1$ as the weighted harmonic mean of the class schema F-Measure and property schema F-Measure :

$$SchemaF1 = \beta * SchemaF1_p + (1-\beta) * SchemaF1_c \quad (14)$$

where the weight $\beta \in [0,1]$. The overall schema F-measure provides a better insight on the combination of the number of classes found by the summarization algorithm and the overall number of properties discovered. The metrics used to compute precision and recall at schema class level include (all) the properties discovered (equations (1), (4) and (11), (12) respectively). But by penalizing the expression of a class by more than one patterns while computing the schema class F-measure, the quality of the results of the summarization algorithms towards the properties gets blurred and is also penalized, which should not be the case. So, we use the schema property recall and precision to recover the notion of quality on property discovery in all cases for the whole schema, so algorithms that will discover all or most of the properties will get acknowledged, even if they use multiple knowledge patterns to do that. Even in the case of not having multiple patterns representing a class the computations for the schema property recall and precision are not redundant because they capture different aspects of the summary's quality, since the overall schema class level precision and recall is an average and thus not the same as the overall property level precision and recall. So the first one tells us how much of the semantics of the classes is recovered in the summary, while the second tells us how many of the overall schema properties are present regardless of where they belong.

Connectivity One more important aspect that we need to consider, is the connectivity of the summary, i.e is the summary a connected graph? So, we propose a new metric to measure how many disconnected graphs exist in the summary and what percentage of the classes in the ground truth they represent. The connectivity of a summary graph G_s Con(G_s) is defined as the number of the connected components(independent subgraphs) of the summary graph divided on the number of the connected components(independent subgraphs) of the ground truth.

$$Con(G_s) = \frac{number of connected components of the summary}{number of connected components of the ground truth} \quad (15)$$

We compute the number of connected components for the summary (and in the same manner for the ground truth) using the breadth-first search algorithm, where given a particular node n, we will find the entire connected component containing n (and no more) before returning. To find all the connected components of a summary (or the ground truth) graph, we loop through the nodes, starting a new breadth-first search, whenever the loop reaches a node that has not already been included in a previously found connected component. This metric gives an indication of the con-

Measure	What it indicates	How it is computed
$instances(c)$	The list of class c instances.	——-
$instances(p)$	The list of subjects which have the property p.	——-
$instances(pa)$	The list of covered class instances by the pattern pa.	—-
$instances(\Pi)$	The list of class instances covered by the set of patternsΠ.	—-
$instances(D)$	The list of all class instances of original KB D.	—-
$Cov_c(c, pa)$	The list of the class instances which are represented by a pattern pa.	Get the $instances(pa)$ if the pattern pa is relevant to the class c or $\emptyset$ otherwise.
$instances(c, \Pi)$	The total number of class instances that are reported by a set of patterns Π representing the class c.	Sum the $\lvert Cov_c(c, pa)\rvert$ for all the patterns of the Π.
$InstancePrec(c, \Pi)$	The instance class precision of a class c over the set of patterns Π.	Divide the number of original instances of the class c reported in Π on $instances(c, \Pi)$.
$InstancePrec_{ClassAll}$	Overall instance class precision.	The mean of the various $InstancePrec(c, \Pi)$ for all the classes of the ground-truth Schema S.
$InstanceF1_c$	Instance class F-Measure.	Combine the $InstancePrec_{ClassAll}$ and $SchemaRec_{ClassAll}$ using the standard formula of the F-Measure.
$Cov_p(p, pa)$	The list of the original property instances which are successfully represented by a pattern pa.	Get the $instances(p)$ if the property p is reported in the pattern pa or get $\emptyset$ otherwise.
$instances(p, \Pi)$	The list of the original property p instances that are successfully covered by a set of patterns Π.	The Union of the $Cov_p(p, pa)$ for all the in Π.
$InstanceRec(p, \Pi)$	The instance property recall.	Divide $\lvert\Pi\ instances(p, \Pi)\rvert$ on $instances(p)$.
$nstanceRec_{PropertyAll}$	Overall recall at the instance property lebel	Weighted mean of the various $InstanceRec(p, \Pi)$ for all the properties of the ground-truth.
$InstancePrec(p, \Pi)$,	The precision of a property p in P over the set of patterns Π.	
$InstancePrec_{PropertyAll}$	Overall instance property precision	Mean of the various $InstanceRec(p, \Pi)$ for all the covered properties of the ground-truth.
$InstanceF1_p$:	Instance property F-Measure	Combine the $InstancePrec_{PropertyAll}$ and $InstanceaRec_{PropertyAll}$ using the standard formula of the F-Measure.
$InstanceF1$	Overall instance F-measure .	Combine the class Instance F-Measure $InstanceF1_c$ and property Instance F-Measure $InstanceF1_p$.

Table 3: Summary Description of the proposed Instance Measures

nectivity of a generated summary. If it is 1, it shows that the summary is a graph connected as well as the ground truth graph, but if it is bigger than 1 it means that the summary is more disconnected than desired. The higher the connectivity, the more the links that are missing between the classes of the computed graph compared to the ground truth; this could even capture correctly a completely disconnected summary graph. This metric allows us to penalize (if needed) disconnected (compared to the ground truth) summary graphs and allows for progressive linear penalties. It is also theoretically possible that the summary graph will be more connected than the ground truth graph, this will give us values less than 1. The value of the connectivity can tend to but will never reach 0.

4.2. Quality Model At the Instance Level

We measure the quality with regard to the instances by introducing the notion of the coverage of the instances of the original KB, i.e. how many of the original class and property instances are successfully represented by the computed RDF summary graph (e.g. can be retrieved in the case of a SPARQL query). This requires computing both the precision and recall at the class instance and at the property instance levels. Table 3 gives us a summary description of the proposed instance level metrics.

Precision and Recall for class instances The overall recall at the instance class level is the total number of the class instances represented by the computed summary divided on the total number of instances of the original KB D.

$$InstanceRec_{ClassAll} = \frac{|instances(\Pi)|}{|instances(D)|} \tag{16}$$

The class $instances(\Pi)$ is the list of instances covered by the set of patterns Π, $instances(D)$ is the list of all instances of the original KB D. To avoid the problem of overlapping of instances in several patterns which will cause the over-coverage, we calculate the $instances(\Pi)$, $instances(D)$ as follows:

$$instances(\Pi) = \bigcup_{pa \in \Pi} instances(pa) \tag{17}$$

$$instances(D) = \bigcup_{c \in C} instances(c) \quad (18)$$

The $instances(pa)$ denotes the list of covered instances by the pattern *pa* and the $instances(c)$ denotes the list of instances of the type *c* in the original KB D.

We denote by $Cov_c(c, pa)$, the list of the class instances which are represented by a pattern *pa*:

$$Cov_c(c, pa) = \begin{cases} instances(pa), & if L(c, pa) = 1 \\ \emptyset, & \text{otherwise} \end{cases} \quad (19)$$

Thus, we can define the total number of class instances $instances(c, \Pi)$ that are reported by a set of patterns Π representing the class c as:

$$instances(c, \Pi) = \sum_{pa \in \Pi} |Cov_c(c, pa)| \quad (20)$$

We define $InstancePrec(c, \Pi)$ the instance precision of a class c in C over the set of patterns Π as follows:

$$InstancePrec(c, \Pi) = \frac{|instances(c) \cap instances(c, \Pi)|}{|instances(c, \Pi)|} \quad (21)$$

Thus, we define the overall instance class precision denoted by $InstancePrec_{ClassAll}$ as the weighted mean of the various $InstancePrec(c, \Pi)$ for all the retrieved classes:

$$InstancePrec_{ClassAll} = \sum_{c \in C} wi(c) * InstancePrec(c, \Pi) \quad (22)$$

The *wi(c)* is the weight of a class c and it measures the percentage of class instances of the class c with respect to the total number of class instances in the KB. This is used to weight in the importance of the specific class in terms of the number of instances it "represents"; so the more instances it "represents" the bigger the weight. It is defined as the number of instances of class c in the KB $instances(c)$ compared to the total number of class instances in the KB $instances(D)$.

$$wi(c) = \frac{instances(c)}{instance(D)} \quad (23)$$

The overall instance class recall and the overall instance class precision are combined by the instance class F-Measure, namely $InstanceF1_c$:

$$InstanceF1_c = 2 * \frac{InstancePrec_{ClassAll} * InstanceRec_{ClassAll}}{InstancePrec_{ClassAll} + InstancePrec_{ClassAll}} \quad (24)$$

Precision and Recall at Property Level The $Cov(p, pa)$ represents the list of the original property instances which are successfully represented by a pattern pa:

$$Cov_p(p, pa) = \begin{cases} instances(pa), & if p \in pa \\ \emptyset, & \text{otherwise} \end{cases} \quad (25)$$

We denote by the $instances(p, \Pi)$ the list of the original property instances that are successfully covered by a set of patterns Π:

$$Instance(p, \Pi)) = \bigcup_{pa \in \Pi} (Cov_p(p, pa) \cap instances(p)) \quad (26)$$

The *instances(p)* denotes the list of original instances which have the property p in original KB D. Thus, the instance property recall $InstanceRec(p, \Pi)$ defined as:

$$InstanceRec(p, \Pi) = \frac{|instances(p, \Pi) \cap instances(p)|}{|instances(p)|} \quad (27)$$

The overall recall at the instance property level $InstanceRec_{PropertyAll}$ is computed as the weighted mean of the various instance property recall $InstanceRec$ for all the properties of the ground-truth.

$$InstanceRec_{PropertyAll} = \sum_{p \in P} wi(p) * InstanceRec(p, \Pi) \quad (28)$$

The *wi(p)* is the weight of the property p and it measures the percentage of instances of a property p with respect to the total number of property instances in the KB. It is defined as the number of instances of property *p* in the KB $instances(p)$ compared to the total number of property instances in the KB. Again the idea here is to capture the important properties by weighting in the number of property instances each one represents.

$$wi(p) = \frac{instances(p)}{\sum_{p1 \in P} instances(p1)} \quad (29)$$

We define $InstancePrec(p, \Pi)$, the precision of a property p in P over the set of patterns Π as follows:

$$InstancePrec(p, \Pi) = \frac{|instances(p) \cap instances(p, \Pi)|}{|instances(p, \Pi)|} \quad (30)$$

Thus, we define the overall instance precision for property instances denoted by $InstancePrec_{PropertyAll}$ as the mean of the various $InstancePrec(c, \Pi)$ for all the properties of the ground-truth Schema S:

$$InstancePrec_{PropertyAll} = \frac{\sum_{p \in P} InstancePrec(p, \Pi)}{|P1|} \quad (31)$$

where P1 $\subseteq$ P is the list of retrieved properties, or in other words the list of properties having $InstancePrec(p, \Pi) > 0$. The overall instance recall and the overall instance precision for property instances are combined by the instance class F-Measure, namely $SchemaF1_c$:

$$InstanceF1_p = 2 * \frac{InstancePrec_{PropertyAll} * InstanceRec_{PropertyAll}}{InstancePrec_{PropertyAll} + InstancePrec_{PropertyAll}} \quad (32)$$

Thus, the overall instance F-measure $InstanceF1$ is obtained by combining the overall instance schema F-Measure $InstanceF1_c$ and overall property instance F-Measure $InstanceF1_p$.

$$InstanceF1 = \beta * InstanceF1_p + (1-\beta) * InstanceF1_c \quad (33)$$

where the weight $\beta \in [0, 1]$. The overall instance F-measure can be viewed as a compromise between overall class instance F-Measure and overall property instance F-Measure. It is high only when both overall class and property instance F-Measure are high. It is equivalent to the class instance F-Measure when $\beta = 0$ and to the property instance F-Measure when $\beta = 1$.

We need also to make one last point for the computation of the instance-level metrics, for the case when our KB contains no schema information. In this case and in order to make the instance level class precision and recall computable we need to annotate the KB with $typeof(class)$ so as to be able to compute the metrics presented above. If not, we will declare the instance level class precision and recall uncomputable but we will be able to continue the quality assessment using the rest of the metrics, including property precision and recall at the instance level. This demonstrates that the proposed Quality Framework will work under all circumstances.

5. Representative Algorithms for validating the Quality Framework

5.1. Algorithms' description

As we have already mentioned in section 3, the RDF graph summarization algorithms could be grouped into four main categories. Based on the results reported in the literature we have chosen three of the most well performing RDF graph summarization algorithms [20,12,37] according to their authors. Our selection of these algorithms was also based on specific properties and features that they demonstrate: (a) they do not require the presence of RDF schema (triples) in order to work properly, (b) they work on both homo- and heterogeneous KBs, (c) they provide statistical information about the available data (which can be used to estimate a query's expected results' size), and (d) they provide a summary graph that is considerably smaller than the original graph.

ExpLOD [20,21] is a RDF graph summarization algorithm and tool that produces summary graphs for specific aspects of an RDF dataset, like class or predicate usage. The summary graph is computed over the RDF graph based on a forward bisimulation that creates group nodes based on classes and predicates. Two nodes *v* and *u* are bisimilar if they have the same set of types and properties. The generated summaries contain metadata about the structure of the RDF graph, like the sets of used RDF classes and properties. Some statistics like the number of instances per class or per property are aggregated with this structural information. The ExpLOD summaries are extracted by partition refinement algorithms or alternatively via SPARQL query where the summary graph is a labeled graph with unlabeled edges. The advantage of ExpLOD approach is that its generated summaries show a dataset's structure as homo- or heterogeneous as it may be. The big disadvantage is the need for transforming the original RDF KB into a ExpLOD graph which is an unlabeled edges graph, where for each triple in RDF KB it generates a node for the subject, node for the object and a unique node for the predicate. Then an edge is drawn from the subject node to the predicate node and other edge from the predicate node to the object node. This process requires the materialization of the whole dataset and this can be limiting in cases of large KBs.

The second limitation is that the created summary is not necessarily a RDF graph itself.

Campinas et al [12] are creating their own RDF summarization graph, whose nodes represent a subset of the original nodes based on their types or used predicates. This summary graph is generated by the following mechanism: (1) extract the types and predicates for each node in the original graph; (2) group the nodes which share the same set of types into the same node summary where two nodes, one of type A and one of types A and B, will end up in different disjoint summary nodes; (3) group based on attributes only if a node does not have a class definition. Like ExpLOD, a summary node is created for each combination of classes, i.e., two nodes, one of type A and one of types A and B, will end up in different disjoint summary nodes. Some statistics like the number of instances per class or the number of property instances are aggregated with this summary graph. Unlike ExpLOD, the summary nodes are not further partitioned based on their interlinks (properties), i.e., two nodes of type A, one has a, b and c properties and one has a and d properties will end up in the same summary node. Unlike ExpLOD, their summary graph is a RDF graph which makes it compatible for storing at RDF databases and queried by SPARQL.

Zneika et al. [37,38] present an approach for RDF graph summarization based on mining a set of approximate graph patterns is presented. It aims at extracting the best approximate RDF graph patterns that describe the input dataset and it works in three independent steps that are described below.

Binary Matrix Mapper: Transform the RDF graph into a binary matrix, where the rows represent the subjects and the columns represent the predicates. They preserve the semantics of the information by capturing distinct types (if present), all attributes and properties (capturing property participation both as subject and object for an instance).

Graph Pattern Identification: The binary matrix created in previous step is used in a calibrated version of the PaNDa+ [27] algorithm, which allows to experiment with different cost functions while retrieving the best approximate RDF graph patterns. Each extracted pattern identifies a set of subjects (rows) all having approximately the same properties (cols). The patterns are extracted so as to minimize errors and to maximize the coverage (i.e. provide a richer description) of the input data. A pattern thus encompasses a set of concepts (type, property, attribute) of the RDF dataset, holding at the same time information about the number of instances that support this set of concepts.

Constructing the RDF summary graph: A process, which reconstructs the summary as a valid RDF graph using the extracted patterns is applied at the end. The process exploits information already embedded in the binary matrix and constructs a valid RDF schema to summarize the KB.

5.2. Algorithms' implementation

We implemented the three algorithms used in the experiments hereafter ourselves. The implementations of two of the algorithms (ExpLOD and Camplinas et al.) were not available from the original authors so we had to implement them ourselves in Java, based on the corresponding papers. We validated the implementation running tests with the datasets described in the original papers. Since we were getting the same results we are quite confident that the implementations are correct. Given also that performance benchmarking is out of scope of this current work, we did not have to deal with any kind of extreme optimizations.

5.3. Working Example

In an effort to better explain the way our quality assessment framework works and captures the differences among the different summaries we provide a working example. We have created an artificial dataset containing information about music artists and their productions. Figure 2 shows a visualization example of the RDF instance graph of this dataset. We have 3000 resources describing the music-artists and all of them have the name and made properties, while only 2500 resources have the rdf:type property, 2049 resources have the homepage property, 2850 have the img property, 50 resources have the biography property. We can also notice that we have 5000 resources describing the records and all of them have the date, image, track and maker properties, while 4995 resources have the title property and only 28 resources have the description property. There are also 45000 resources describing the tracks and all of them have the rdf:type, title, track-number and available-as properties, while only 5 resources have the olga property (used to link a track to a Document for tracking in the On-Line Guitar Archive). These tracks are available as a Playlist or/and as ED2K formats. Figure 3 shows an ideal summary for this dataset as was suggested by an expert.

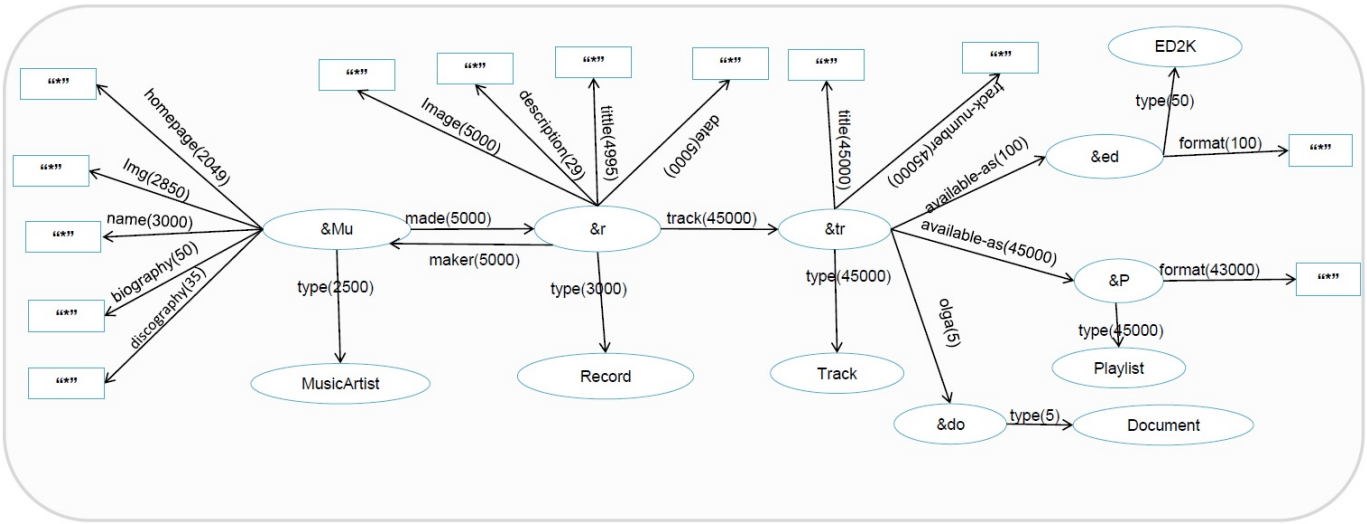


Fig. 2.: An artificial dataset about music artists and their productions

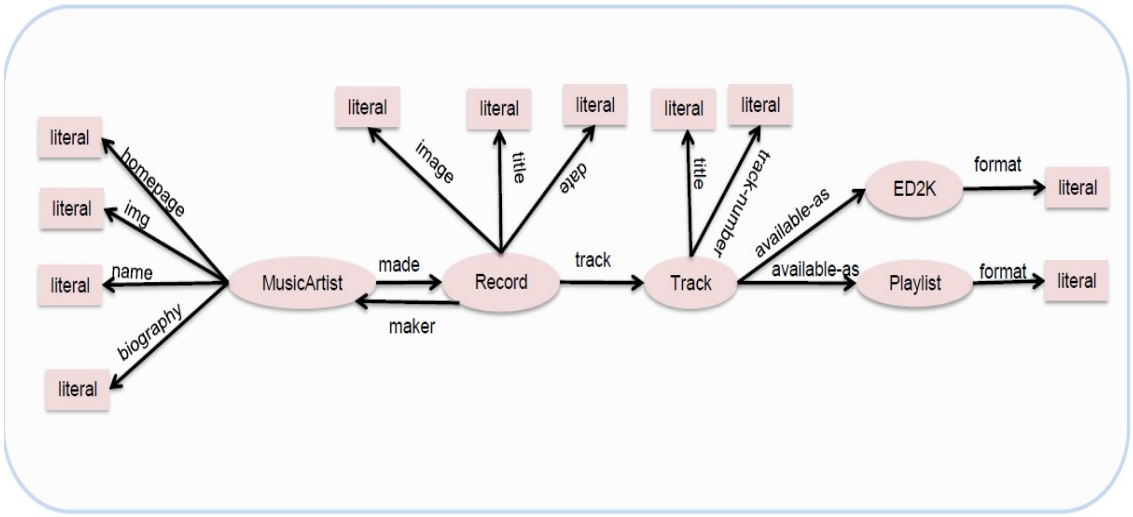


Fig. 3.: The ideal summary of the dataset depicted in Figure 2

Tables 4, 5 and 6 present the three RDF summaries generated using the three discussed algorithms: ExpLOD, Campinas et al and Zneika et al respectively. The first column shows the pattern id, the second shows the predicates involved in the pattern, while the third column shows the corresponding ideal summary class for a pattern. The last column shows the number of instances per pattern. The Figures 4, 5 and 6 are a visualization representing for three RDF summaries generated using ExpLOD, Campinas et al and Zneika et al. receptively.

5.3.1. Schema-level metrics

Here we calculate the precision for the MusicArtist class for the three summaries. We start by the ExpLOD summary described in table 4, Sim(Pa1,MusicArtist)=1, because all the properties of the pattern Pa1 are properties of the MusicArtist in the ideal summary. Actually for each $Pa \in \{Pa1, Pa2, Pa3, Pa4, Pa6, Pa7\}$ Sim(Pa,MusicArtist)=1 for the same reason. Concerning the pattern Pa5, it has 6 properties, 5 of which are properties of MusicArtist that are included in the ideal summary. But the pattern Pa5 has also chosen

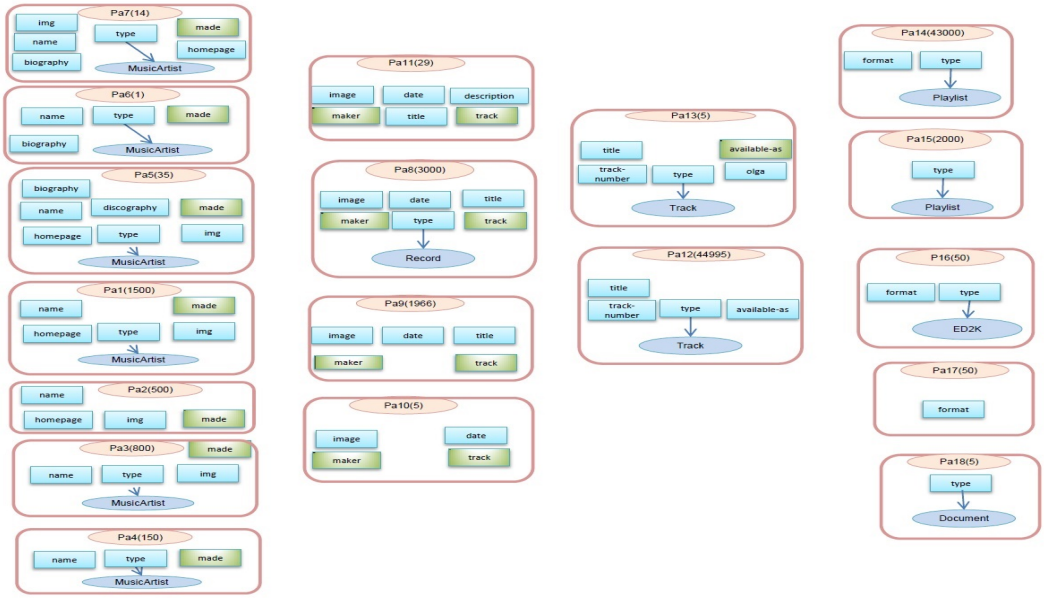


Fig. 4.: The ExpLOD Summary of the dataset depicted in Figure 2

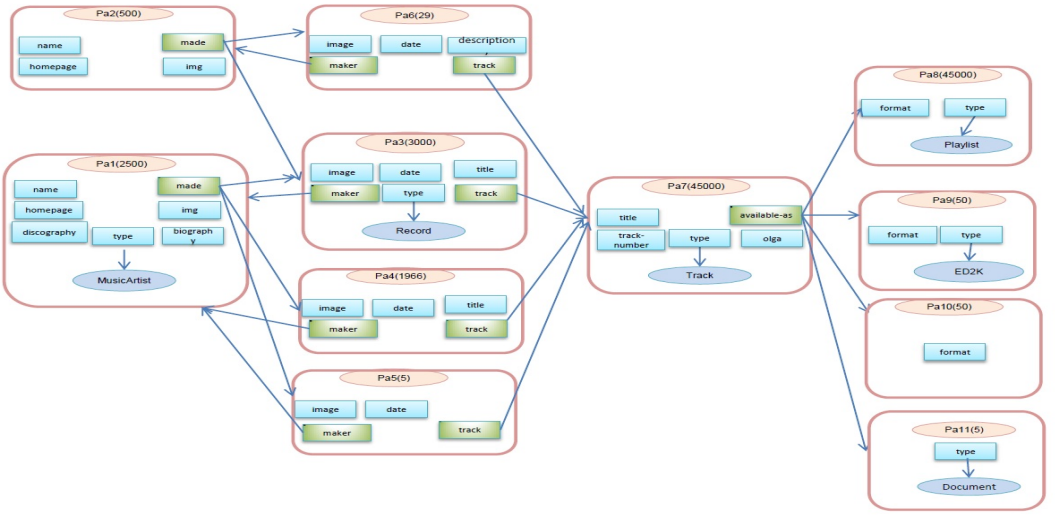


Fig. 5.: The Campinas et al Summary of the dataset depicted in Figure 2

the discography property, which is not included in the ideal summary. That makes the Sim(Pa5,MusicArtist) = $\frac{5}{6}$. Any other pattern Pa in the table has $Sim(MusicArtist, Pa) = 0$, because it has a different *typeof* and there are no common properties between these patterns and the MusicArtist class. So the Nps(MusicArtist)=7, and with the $\alpha = 3$ then $W(MusicArtist) = e^{1-\sqrt[3]{7}} = 0.40$. Hence, the precision of the patterns corresponding to the MusicArtist class is: $SchemaPrec(MusicArtist, \Pi) = 0.40 * \frac{1+1+1+1+0.83+1+1}{7} = 0.39$.

Now let us take the Campinas et al. summary described in Table 5. In this table we can see that we have two patterns Pa1 and Pa2 represent the Musi-

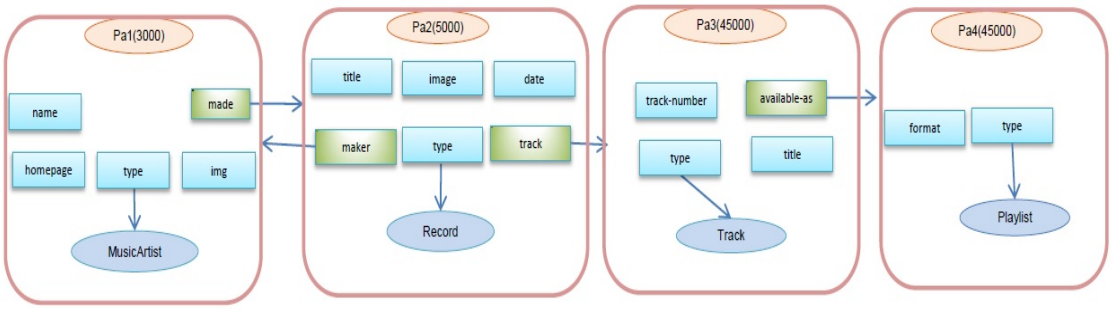


Fig. 6.: The Zneika et al Summary of the dataset depicted in Figure 2

cArtist class, so $Nps(Musicartist) = 2$, thus the weight: $W(MusicArist) = e^{1-\sqrt[3]{2}} = 0.77$. The first pattern Pa1 has 6 properties where 5 of these 6 properties are properties of MusicArtist in the ideal summary. But it has chosen the discography property, too, which is not included in the ideal summary. That makes Sim(MusicArtist,Pa1)=$\frac{5}{6}$. Respectively, Pa2 has Sim(Musicartist,Pa2)=1, since all of its properties are included in the ideal summary. From all above we conclude the precision of Campinas et al.: $SchemaPrec(MusicArtist, \Pi) = 0.77 * \frac{1+0.83}{2} = 0.70$.

Now let us compute the precision of the MusicArtist for the Zneika et al. summary depicted in Table 6, Sim(Pa1,MusicArtist)=1, because all the properties of the pattern Pa1 are properties of the MusicArtist in the ideal summary. Any other pattern Pa in the table 6 has Sim(Musicartist,Pa)=0, because it has a different typeof link and no common properties exist between each one of these patterns and the MusicArtist class. So $Nps(MusicArtist) = 1$, and keeping $\alpha = 3$ then $W(MusicArist) = e^{1-\sqrt[3]{1}} = 1$. Hence, the precision of class MusicArtist is: $SchemaPrec(MusicArtist, \Pi) = 1 * \frac{1}{1} = 1$.

Following the same procedure, we can calculate the precision for each class in the set of classes of the ideal summary; these results are reported in table 7a. We should also note that the class Document, which is reported in the summaries of the ExpLod and Campinas et al., is not a class in the ideal summary.

Table 7b shows the values of the recall for the list of ideal summary classes. We can note that for ExpLOD and Campinas et al, all recall values are 1, as their patterns cover all the properties in the ideal summary. While for Zneika et al, the recall for the MusicArtist is 0.8, because pattern Pa1, which represents the MusicArtist class, does not cover the biography property, so its recall equals 4 properties over 5 in the ideal summary, $SchemaRec(MusicArtist, \Pi) = \frac{4}{5} = 0.80$.

To calculate the schema-level property precision, we notice that each one of the ExpLOD and Campinas et al. has 16 properties, 13 of these 16 are included in the ideal summary, the other three: discography, description, and ogla are not. That makes the property precision for each one of these two summaries $SchemaPrec_{PropertyAll} = \frac{13}{16} = 0.81$. The properties reported by the Zneika et al summary are all included in the ideal summary, thus its precision is 1.

Concerning the recall at the property level, ExpLOD and Campinas et al. recall equals 1, as they included all the properties in the ideal summary, while Zneika et al missed one property which is biography, so its recall is $SchemaRec_{PropertyAll} \frac{12}{13} = 0.92$.

5.3.2. *Instance-level metrics*

Table 9b shows the values of the recall for the list of distinct properties of the dataset depicted in Figure 2. We can note that for ExpLOD and Campinas et al, all recall values are 1, as their patterns cover all the property instances of the datasets. While for Zneika et al, the property instance recall values for the biography, discography and description are 0, because these properties are completely missing from Zneika el. summary.

While Table 9a shows the values of the property instance precision. We can note that for ExpLOD , all precision values are 1, as its patterns described in Table 4 are correctly identified all the property instances of the datasets. For the example, for the property homepage having 2049 instances in original dataset, you can see that it is included in 4 patterns $\{Pa1, Pa2, Pa5, Pa7\}$, thus |*Instance(biography,*

ID	Pattern	corresponding class	Instance number
Pa1	MusicArtist(c), name, img, homepage, made	MusicArtist	1500
Pa2	name, img, homepage, made	MusicArtist	500
Pa3	MusicArtist(c), name, img, made	MusicArtist	800
Pa4	MusicArtist(c), name, made	MusicArtist	150
Pa5	MusicArtist(c), name, img, homepage, made, biography, discography	MusicArtist	35
Pa6	MusicArtist(c), name, made, biography	MusicArtist	1
Pa7	MusicArtist(c), name, made, img, homepage, biography	MusicArtist	14
Pa8	Record(c), image, title, date, maker, track	Record	3000
Pa9	image, title, date, maker, track	Record	1966
Pa10	image, date, maker, track	Record	5
Pa11	image, description, title, date, maker, track	Record	29
Pa12	Track(c), title, track-number, available-as	Track	44995
Pa13	Track(c), title, track-number, available-as, olga	Track	5
Pa14	Playlist(c), format	Playlist	43000
Pa15	Playlist(c)	Playlist 2000	
Pa16	ED2K(c), format	ED2K 50	
Pa17	format	–	50
Pa18	Document	Document	5

Table 4: ExpLOD summary for the dataset depicted in Figure 2

ID	Pattern	corresponding class	Instance number
Pa1	MusicArtist(c), name, img, homepage, made, biography, discography	MusicArtist	2500
Pa2	name, img, homepage, made	MusicArtist	500
Pa3	Record(c), image, title, date, maker, track	Record	3000
Pa4	image, title, date, maker, track	Record	1966
Pa5	image, date, maker, track	Record	5
Pa6	image, description, title, date, maker	Record	29
Pa7	Track(c), title, track-number, available-as, olga	Track	45000
Pa8	Playlist(c), format	Playlist	45000
Pa8	ED2K(c), format	ED2K	50
Pa9	format	-	50
Pa10	Document	Document	5

Table 5: Campinas et al summary for the dataset depicted in Figure 2

ID	Pattern	corresponding class	Instance number
Pa1	MusicArtist(c), name, img, homepage, made	MusicArtist	3000
Pa2	Record(c), image, title, date , maker, track	Record	5000
Pa3	Track(c), title, track-number, available-as	Track	45000
Pa4	Playlist(c), format	Playlist	45000

Table 6: Zneika el al summary for the dataset depicted in Figure 2

	ExpLod	Campinas et al	Zneika et al
$SchemaPrec(MusicArtist, \Pi)$	0.39	0.70	1
$SchemaPrec(Record, \Pi)$	0.52	0.52	1
$SchemaPrec(Track, \Pi)$	0.67	0.80	1
$SchemaPrec(Playlist, \Pi)$	0.64	0.64	1
$SchemaPrec(ED2K, \Pi)$	0.77	0.77	-
$SchemaPrec_{ClassAll}$	0.60	0.69	1

(a) Schema Precision at Class level

	ExpLod	Campinas et al	Zneika et al
$SchemaRec(MusicArtist, \Pi)$	1	1	0.80
$SchemaRec(Record, \Pi)$	1	1	1
$SchemaRec(Track, \Pi)$	1	1	1
$SchemaRrc(Playlist, \Pi)$	1	1	1
$SchemaRec(ED2K, \Pi)$	1	1	0
$SchemaRec_{ClassAll}$	1	1	0.76

(b) Schema Recall at Class level

Table 7: Schema Metrics at Class level

	ExpLod	Campinas et al	Zneika et al
$SchemaPrec_{PropertyAll}$	0.81	0.81	1

(a) Schema Precision at Property level

	ExpLod	Campinas et al	Zneika et al
$SchemaRec_{PropertyAll}$	1	1	0.92

(b) Schema Recall at property level

Table 8: Schema Metrics at Property level

Π)| = *1500+500+35+14 = 2049*. Hence, the *InstancePrec(homepage,*Π*)* = $\frac{2049}{2049} = 1$. Following the same procedure, we can find that all property precision values are 1 for the Explod summary.

Now let us try to compute the instance precision value for the homepage property for the Campinas et al. summary described in Table 5. From this table 5 we can note that this property is included in the patterns Pa1 and Pa2, thus $|Instance(hompage, \Pi)| = 2500 + 500$. Hence,the InstancePrec(homepage,Π)=$\frac{2049}{3000} = 0.68$.

Now let us take the Zneika el al. summary described in table 6. In this table we can see that only the pattern Pa1 has the homepage property. Thus $|Instance(hompage, \Pi)| = 3000$. Hence,the InstancePrec(homepage,Π)=$\frac{2049}{3000} = 0.68$.

Following the same procedure, we can calculate the instance property precision for all the dataset properties ; these results are reported in Table 9a.

On the other hand, the results for the class precision and recall at the instance level in this example is always equal to 1 or almost 1 (since in one case only a few class instances are missing) and thus their computation provides no further insights for this example. This is why, the corresponding tables were omitted.

5.3.3. Connectivity

Table 10 reports the connectivity metric values for the summaries produced by the three discussed algorithms. It shows that the ExpLod has a value of 6 for this metric because its summary ends up with 6 separate components while the ideal summary depicted in Figure 3 has exactly one connected component. This value means the ExpLOD provides a disconnected summary. The two other algorithms report a value of 1, which means that these two algorithm provide a summary as connected as the ideal one (one connected component in this case).

6. Experiments

In this section, we compare the quality of the generated summaries of the three RDF graph summarization approaches covered in section 5. We implemented these three approaches in Java 1.8 using the Nxparser[2] API to parse the RDF triples. All the experiments ran on a Intel(R) Core(i5) Opteron 2.5 GHz server with 16 GB of RAM (of which 14 GB was assigned to the Java Virtual Machine), running Windows 7. Section 6.1 describes the datasets considered in the experiments. Section 6.2 gives a quality evaluation of the created summaries based on the three discussed approaches and using the metrics described in section 4.

6.1. Datasets

Table 11 shows the datasets from the LOD cloud that are considered for the experiments. The first seven columns show the following information about each dataset: its name, the number of triples it contains, and the number of instances, classes, predicates, properties and attributes. The eighth column shows the class instance distribution metric which provides an indication on how instances are spread across the classes and it is defined as the standard deviation (SD) in the number of instances per class. When the number of class instances per class in a dataset is quite close then the standard deviation is small; while, when there are considerable differences, the standard deviation will be relatively large. The ninth column shows the property instance distribution metric which provides an indication on how instances are spread across the properties and it is also defined as standard deviation (SD) in the number of instances per property.

The main goal of our datasets selection is to use real-world datasets from diverse domains with different size (number of triples) and with different numbers of classes (and class instances) and properties (and properties instances). We are also interested in the distribution of the data which might indicate if the structure of the KB or the size of the represented knowledge could affect the quality of the generated summaries. So we have datasets from 270 thousand (Jpeel) to 263 million triples (Lobid), from one (Bank2) to 53 unique

[2]Nxparser: https://github.com/nxparser/nxparser

	ExpLod	Campinas et al	Zneika et al
$InstancePrec(name, \Pi)$	1	1	1
$InstancePrec(img, \Pi)$	1	0.95	0.95
$InstancePrec(homepage, \Pi)$	1	0.68	0.68
$InstancePrec(made, \Pi)$	1	1	1
$InstancePrec(biography, \Pi)$	1	0.02	-
$InstancePrec(discography, \Pi)$	1	0.01	-
$InstancePrec(image, \Pi)$	1	1	1
$InstancePrec(title, \Pi)$	1	1	0.999
$InstancePrec(date, \Pi)$	1	1	1
$InstancePrec(maker, \Pi)$	1	1	1
$InstancePrec(track, \Pi)$	1	1	1
$InstancePrec(descrption, \Pi)$	1	1	-
$InstancePrec(track - number, \Pi)$	1	1	1
$InstancePrec(available - as, \Pi)$	1	1	1
$InstancePrec(olga, \Pi)$	1	0.0001	-
$InstancePrec(format, \Pi)$	1	1	1
$InstancePrec_{PropertyAll}$	1	0.80	0.88

(a) Instance Precision at Property level

	ExpLod	Campinas et al	Zneika et al
$InstanceRec(name, \Pi)$	1	1	1
$InstanceRec(img, \Pi)$	1	1	1
$InstanceRec(homepage, \Pi)$	1	1	1
$InstanceRec(made, \Pi)$	1	1	1
$InstanceRec(biography, \Pi)$	1	1	0
$InstanceRec(discography, \Pi)$	1	1	0
$InstanceRec(image, \Pi)$	1	1	1
$InstanceRec(title, \Pi)$	1	1	1
$InstanceRec(date, \Pi)$	1	1	1
$InstanceRec(maker, \Pi)$	1	1	1
$InstanceRec(track, \Pi)$	1	1	1
$InstanceRec(descrption, \Pi)$	1	1	0
$InstanceRec(track - number, \Pi)$	1	1	1
$InstanceRec(available - as, \Pi)$	1	1	1
$InstanceRec(olga, \Pi)$	1	1	0
$InstanceRec(format, \Pi)$	1	1	1
$InstancePrec_{PropertyAll}$	1	1	0.99

(b) Instance Recall at Property level

Table 9: Instance Metrics at Property level

s	ExpLod	Campinas et al	Zneika et al
Connectivity	6	1	1

Table 10: Connectivity

classes (LinkedMDB), from about 76 thousand(Jpeel) to about 18 million unique instances/entities and from 12 to 222 predicates. These datasets range from being very homogeneous (the Bank dataset where all subjects have the same list of attributes and properties) to being very heterogeneous (LinkedMDB where the attributes and properties are very heterogeneous across types). The diversity of the datasets can help us to understand better how the selected approaches work in different situations and thus validate that the proposed quality metrics will capture the different behaviours correctly.

6.2. Evaluation Results

In this section, we discuss the quality results of the RDF graph summarization approaches covered in section 5, evaluated over all the datasets described in Table 11 for the following two cases:

- Typed Dataset: the KB contains schema information, like definition of classes and properties and more importantly a significant number of instances of a dataset have at least one typeof link/property.
- Untyped Dataset: there is no schema information in the KB and more importantly *none* of the datasets subjects/objects or properties has a defined type (we explicitly checked and deleted all of them).

The distinction for the experimentation is important because there are algorithms that try to exploit schema related information (mainly typeof links) in order to gain insights for the structure of the KB. While, where ever available using this information could be valuable, we would like to test the summarization algorithms in cases when this information is not available, too. With that we can validate that the proposed Quality Framework will correctly capture the differences in the results and will correctly identify, for example, algorithms that work well in both cases.

6.2.1. Implementation of the Quality Framework

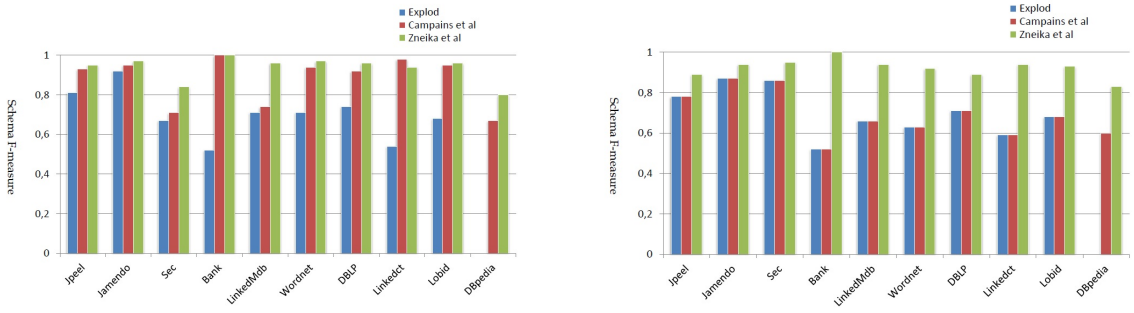
We implemented our Quality Framework as a software that takes as input the results of any RDF Graph Summarization algorithm and the ideal summary and computes the different metrics that are required to capture the quality of the results at the different levels described earlier. It outputs the values for the different metrics in an automated fashion and allows to compute F-measures where applicable. In principle it can be used to compare the quality of any summary against an ideal one or to understand how close two summaries are to one another. It is implemented in Java and we plan to make it available as open source software.

We describe the different steps applied in the form of algorithmic pseudocode that allows to track the computations taking place at the different levels that

Dataset	Triples	Instances	Classes	Predicates	properties	attributes	Class instance distribution		Property instance distribution	
							Mean	SD	Mean	SD
Jpeel [4]	271,369	76,229	9	26	14	12	8,449	8,289.61	9,374.48	15,988.21
Jamendo [3]	1,047,950	335,925	11	25	14	11	20,542	19,622.08	34,633.48	59,458.62
Sec [a]	1,813,135	460,446	5	12	3	9	66,861.8	41,233.64	144,041.83	63,388.13
linkedMDB [6]	6,148,121	694, 400	53	222	153	69	13,971	37,368.26	24,758.70	80,271.76
Bank [1]	7,348,860	200,429	1	33	0	33	200,429	0	197,065.61	4,786.98
Wordnet [8]	8,574,807	647,215	5	63	55	8	129,147	69,768.22	59,947.92	113,775.88
DBLP [2]	41,802,523	5,942,858	10	19	9	10	497,153.9	971,029.76	538,837.42	805,531.71
Linkedct [5]	49,084,152	5,364,776	30	121	44	77	178,826	217,293.64	214,010.65	218,145.29
Lobid [7]	263,215,517	17,854,885	24	104	40	64	663,355.26	996,359.95	661,974.82	979,956.84
DBpedia[b]	438,336,517	3,769,926	436	1894	919	975	129,248	188,372.89	86,136.66	227,632.07

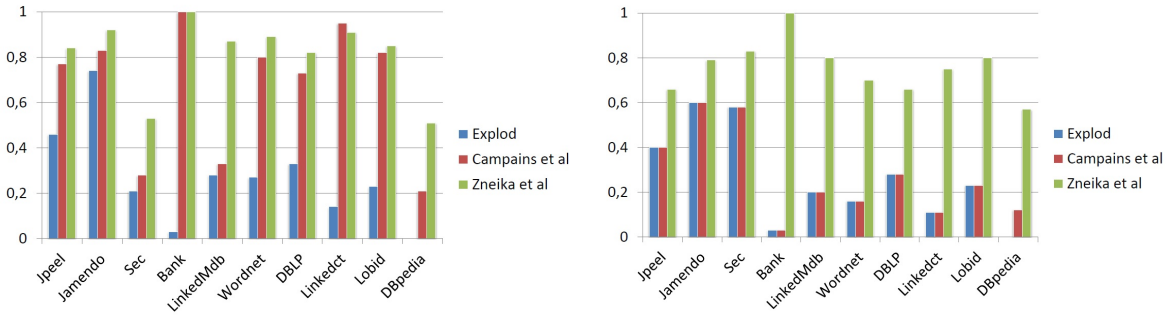
Table 11: Descriptive statistics of the datasets

[a]U.S. SEC data: http://www.govtrack.us/data/misc/sec.n3.gz
[b]http://wiki.dbpedia.org/data-set-38



(a) Typed datasets (b) Untyped datasets

Fig. 7.: F-Measure results for typed/untyped presented datasets at the schema Level



(a) Typed datasets (b) Untyped Datasets

Fig. 8.: Class precision results for typed/untyped presented datasets at the schema Level

are Quality Framework operates. The pseudocode of Algorithm 1 gives an overview of our implementation of the computations at the schema level. The function which computes the schema class recall is shown in Algorithm 2, while the one, which computes the schema class precision, is shown in Algorithm 3. The function which computes the schema property precision and recall is shown Algorithm 4. In the same manner, the pseudocode in Algorithm 5 gives an overview of the computations at the Instance level. The func-

Algorithm 1. Schema Level Metrics
INPUT: Set of knowledge patterns $\Pi = \{Pa_i : i : 1.....N\}$, ideal summary S=$\{C, P, I\}$ where C, P, and I are Set of classes, properties and instances, α and β.
OUTPUT: Rec_c schema class Recall, $Prec_c$ schema class precision , Rec_p schema property recall , $Prec_p$ schema property precision, F_c Schema class F-Measure , F_p Schema property F-Measure and SchemaF1 overall schema F-Measure .

Begin
$Rec_c \leftarrow$ Schema-Class-Recall(C,Π)
$Prec_c \leftarrow$ Schema-Class-Precision(C,Π, α)
$F_c \leftarrow \frac{Prec_c * Rec_c}{Prec_c + Rec_c}$
$Prec_p, Rrec_p \leftarrow$Schema-Property-Recall-Precision(C,Π)
$F_p \leftarrow \frac{Prec_p * Rec_p}{Prec_p + Rec_p}$
$SchemaF1 \leftarrow \beta * F_p + (1-\beta) * F_c$
End

Algorithm 2. Function Schema Class Recall

function SCHEMA-CLASS-RECALL(C,Π)
  $Rec_c \leftarrow 0$ ▷ schema class recall
  for each c ∈ C **do**
    $ListA \leftarrow \emptyset$ ▷ the list of common properties of c and Π
    for each pa ∈ Π **do**
      $ListA \leftarrow ListA \cup (A(c) \cap A(pa))$
      ▷ where A(c), A(Pa) are the set properties of pa and c
    end for
    rec $\leftarrow \frac{|ListA|}{|A(c)|}$
    $Rec_c \leftarrow Rec_c + rec$
  end for
  $Rec_c \leftarrow \frac{Rec_c}{|C|}$
  return Rec_c
end function

tion which computes the instance class recall is shown in Algorithm 6, while the one, which computes the instance class precision, is shown Algorithm 7. The function which computes the instance property precision and recall is shown in Algorithm 8.

6.2.2. Results for schema level metrics

Table 13 reports the precision, recall and F-Measure values at the schema level for classes and properties of the generated RDF summaries over the set of datasets depicted in table 11 for the *typed* and *untyped* cases. The left part of Table 13 shows the results for the *typed* used datasets while the right part shows the results for *untyped* used datasets. The Figures 7 and 8 are a flow chart representing for The overall schema F-Measure

Algorithm 3. Function Schema Class Precision

function SCHEMA-CLASS-PRECISION(C,Π, α)
  $Prec_c \leftarrow 0$ ▷ the Schema class precision
  for each c ∈ C **do**
    Nps $\leftarrow$ 0
    prec $\leftarrow$ 0
    for each Pa ∈ Π **do**
      compute the similarity Sim(pa,c) using the equation (4)
      prec $\leftarrow prec + Sim(pa, c)$
      if Sim(pa,c)>0 **then**
        Nps $\leftarrow Nps + 1$
      end if
    end for
    W $\leftarrow e^{1-\sqrt[q]{Nps}}$
    Prec $\leftarrow w * \frac{prec}{Nps}$
    $Prec_c \leftarrow Prec_c + Prec$
  end for
  $Prec_c \leftarrow \frac{Prec_c}{|C|}$
  Return $Prec_c$
end function

Algorithm 4. Function Schema Property Precision and Recall

function SCHEMA-PROPERTY-RECALL-PRECISION(C,Π)
  $Rec_p \leftarrow 0$ ▷ the Schema property recall
  $Prec_p \leftarrow 0$ ▷ the Schema property precision
  $ListA \leftarrow \emptyset$ ▷ all the properties involved in C
  $ListB \leftarrow \emptyset$ ▷ all the properties involved in Π
  for each c ∈ C **do**
    $ListA \leftarrow (ListA \cup (A(c))$
  end for
  for each pa ∈ Π **do**
    $ListB \leftarrow (ListB \cup (A(pa))$
  end for
  $ListC \leftarrow (ListA \cap ListB)$ ▷ the common properties between ListA and ListB
  $Rec_p \leftarrow \frac{|ListC|}{|ListA|}$
  $Prec_p \leftarrow \frac{|ListC|}{|ListB|}$
  return $Rec_p, Prec_p$
end function

and the class precision metrics values receptively that carries more visualization details.

We can note from Table 13 that the schema property recall, schema property precision and the schema property F-Measure, reported in columns R_p, P_p and F_p respectively, are always equal to 1 for the ExpLod and the Campinas et al algorithms over all the presented datasets. The same is true for the schema class recall reported in column R_c. We can also note from the right part of the 13 that the values of the previously mentioned measures are equal to 1. This is because the Ex-

Algorithm 5. Instance Level Metrics
INPUT: Set of knowledge patterns $\Pi = \{Pa_i : i : 1.....N\}$, Ideal summary S and β.
OUTPUT: $InsRec_c$ Instance class Recall, $InsPrec_c$ Instance class precision , $InsRec_p$ Instance property recall, $InsPrec_p$ Instance property precision, $InsF_c$ Instance class F-Measure, $InsF_p$ Instance property F-Measure and InstanceF1 overall Instance F-Measure.

1: **Begin**
2: $InsRec_c \leftarrow$ Instance-Class-Recall(C,Π)
3: $InstPrec_c \leftarrow$ Instance-Class-Precision(C,Π, α)
4: $InsF_c \leftarrow \frac{InsPrec_c * InsRec_c}{InsPrec_c + InsRec_c}$
5: $InsPrec_p, InsRrec_p \leftarrow$ Instance-Property-Recall-Precision(C,Π)
6: $InsF_p \leftarrow \frac{InsPrec_p * InsRec_p}{InsPrec_p + InsRec_p}$
7: $InstanceF1 \leftarrow \beta * InsF_p + (1-\beta) * InsF_c$
8: **End**

Algorithm 6. Function Instance Class Recall

1: **function** INSTANCE-CLASS-RECALL(C,Π)
2: $InsRec_c \leftarrow 0$
3: $Instances_\Pi \leftarrow \emptyset$ ▷ list of all the class instances reported in Π
4: $Instances_C \leftarrow \emptyset$ ▷ list of all the class instances involved in D
5: **for** each c ∈ C **do**
6: $Instances_C \leftarrow (Instances_C \cup instances(c))$
7: **end for**
8: **for** each pa ∈ Π **do**
9: $Instances_\Pi \leftarrow (Instances_\Pi \cup (instances(pa))$
10: **end for**
11: $InsRec_c \leftarrow \frac{|Instances_\Pi|}{|Instances_C|}$ ▷ $InsRec_c$ Instance class recall
12: **return** $InsRec_c$
13: **end function**

pLod and Campinas et al algorithms depend on the notion of the forward bisimulation that groups the original nodes based on classes and/or predicates, hence they are no missed properties or types (and of course nothing new is added), thus the schema class recall values will be always 1 for the ExpLod and the Campinas et al. A predicates-based grouping is necessary for the Campinas et al algorithm when the entities' nodes do not have a class definition, hence they are no missed properties for the *untyped* case, which explains why the values for these measures have not changed for the *untyped* datasets. This also explains why we have the same measures' values for the ExpLod and Campinas et al for the *untyped* datasets. For Zneika et al algorithm, although it depends on the approximation type selected, if we exclude the linkedct dataset the values

Algorithm 7. Function Instance Class Precision

1: **function** INSTANCE-CLASS-PRECISION(C,Π)
2: $InsPrec_c \leftarrow 0$
3: **for** each c ∈ C **do**
4: $Cov_c \leftarrow 0$
5: CovList ← ∅
6: prec ← 0
7: **for** each pa ∈ Π **do**
8: **if** L(pa,c)=1 **then**
9: $Cov_c \leftarrow Cov_c + |instances(pa)|$
10: CovList ← $CovList \cup (instances(c) \cap instances(pa))$ ▷ the list of class c instances reported in π
11: **end if**
12: **end for**
13: $InsPrec_c \leftarrow InsPrec_c + \frac{CovList}{Cov_c}$
14: **end for**
15: $InsPrec_c \leftarrow \frac{InsPrec_c}{|C1|}$
16: Return $InsPrec_c$
17: **end function**

Algorithm 8. Function Instance Property Precision and Recall

1: **function** SCHEMA-CLASS-PRECISION(P,Π)
2: $InsRec_p \leftarrow 0$
3: $InsPrec_p \leftarrow 0$
4: **for** each p ∈ P **do**
5: CovList ← ∅
6: $Cov_p \leftarrow 0$
7: **for** each pa ∈ Π **do**
8: **if** p ∈ pa **then**
9: CovList ← $CovList \cup (instances(p) \cap instances(pa))$
10: $Cov_p \leftarrow Cov_p + |instances(pa)|$
11: **end if**
12: **end for**
13: $InsRec_p \leftarrow InsRec_p + \frac{CovList}{|instances(p)|}$
14: $InsPrec_p \leftarrow Prec_p + \frac{CovList}{cov_p}$
15: **end for**
16: $InsRec_p \leftarrow \frac{InsRec_p}{|P|}$
17: $InsPrec_p \leftarrow \frac{InsPrec_p}{|P1|}$
18: Return $InsPrec_p, InsRec_p$
19: **end function**

for measures mentioned previously are also equal to 1 for the *typed* an *untyped* datasets, which means that the algorithm successfully summarizes the KBs, despite the fact that by construction the algorithm uses approximate pattern mining to detect the classes and properties available and thus some could have been possibly missed.

Another notable observation from the Table 13g and the Figure 7b, is that for the Bank dataset and for the

overall schema F-Measure the perfect value (equal to 1) is reported for the Zneika et al and Campinas et al algorithms. This is because the Bank dataset is a fully *typed* and homogeneous dataset(each subject of this dataset has at least one typeof link/property) and as we explained earlier, the Campinas et al algorithm groups the original nodes based only on their types when types exist, hence they are no missed or added properties in this case.

For the Sec dataset, the table 13e shows that the values of schema class precision reported in column P_c and depicted in Figure 8a are low for the three discussed algorithms. This is because that the ground truth schema of the Sec dataset contains a lot of inheritance relationships and as none of three discussed algorithms deals with inheritance, the three algorithms end up with a lot of overlapping patterns (some properties which belong to the subclasses are assigned to the patterns which represent the superclasses).

Tables 13s,13t report metrics values at the schema level for the generated RDF summaries of the Campinas et al and Zneika et al algorithms over the DBpedia dataset for the *typed* and *untyped* cases. We do not report results for the ExpLOd algorithm because ExpLOD's implementation was bound to datasets that fit in main memory and DBpedia could not fit in main memory. We notice from these tables that the values of the schema class precision reported in column P_c are low for the Zneika el al and Campinas et al summaries. This is because the DBpedia KB contains a lot of entities/resources having multiple classes/types and a lot of classes carry subsumption (inheritance) relationships. Actually, on average an entity has four types associated with it, and as apparently none of the two mentioned algorithms deals adequately with multiple classification, the two algorithms end up with a lot of overlapping patterns (some properties which belong to class A are assigned to the patterns which represent class B in the multiple classification case or some properties which belong to the subclasses are assigned to the patterns which represent the superclasses in inheritance case). An additional reason to have a poorer precision for the Campinas et al summary is that the type definitions are missing of a quite large number of DBpedia KB's instances. As already discussed in this case, the Campinas et al groups the nodes based on the properties and this makes it generating a summary where a lot of the ideal summary classes are represented by several knowledge patterns.

Table 13 shows well that algorithms like ExpLod do not provide quality summaries in extreme cases like the Bank dataset (where we have only one class) or in heterogeneous datasets like LinkedMDB, Linkedct and DBLP, where they report very low class precision values, because instances of the same class in these cases have quite different properties and they cannot be grouped together by ExpLod. This is because the ExpLod algorithm depends on the notion of the forward bisimulation [19] that groups the original nodes based on the existence of common typeof and property links. In other words, two nodes v and u are bisimilar and will end-up in the same equivalent class (pattern) if they have *exactly* the same set of types and properties. Thus, it might generate a summary where many ideal summary classes are represented by several knowledge patterns. For example, in the Bank dataset case, which contains only one class in the ideal summary, ExpLOD generated 79 knowledge patterns. And as we mentioned in section 4.1 we have included in our framework a way to penalize these cases by introducing the W(c) exponential function (see equation 7). Table 13 and Figures 8 and 7 also demonstrate that the Zneika et al algorithm gives better results, when compared with the other two algorithms, over all the presented datasets, and it showcases that it works well with heterogeneous datasets like the LinkedMdb, unlike the ExpLod and Campinas et al that give a low class precision with the heterogeneous datasets.

By comparing the results for the *typed* datasets case depicted in Figure 8a and the *untyped* datasets depicted in Figure 8b. We can easily observe that the behavior of Zneika et al and ExpLod algorithms in the case of the *untyped* cases is the same as in the case of the *typed* datasets, which means that the quality of the summary is not affected by the presence (or not) of schema information in the KB. While we can easily observe the significant impact the absence of typeof schema information had for the Campinas et al algorithm.

The discussion so far provides some insights on how we can use the proposed Quality Framework to assess the quality of the summaries produced by the different algorithms. Since we are looking at comparing the quality of the computed summary to a ground truth summary provided by an expert in general we can observe that:

- the summarization algorithms usually capture correctly the properties involved in the data but miss at different levels (and for different reasons) some of the classes. The Quality Framework provides enough resolution to really identify the algorithms that provide a better summary in turn

of the classes reported and the quality of this report (e.g. are all properties reported, is the class present as one entity in the computed summary, etc.).
- the summarization algorithms do not capture well cases where the data are multiply classified or where there are quite widespread subsumption relationships.
- the summarization algorithms could have quite a few differences when reporting on the contents of the KB and the quality of the summaries could greatly vary and this is mostly because of the differences in the precision of reporting the classes in the summary, including penalizing verbose descriptions (like those reported by Explod). So actually we can capture even fine differences where for example a single class in the ground truth is represented by two in the computed summary.

6.2.3. Results for instance level metrics

Table 14 reports the precision, the recall and the F-Measure of RDF summaries at the instance level, based on the same datasets and algorithms as before. The left part of Table 13 shows the results for the *typed* datasets while the right part shows the results for *untyped* datasets. For each dataset, we report the precision, the recall and the F-measure values at class and property level. We note that ExpLod produces the best results (actually perfect ones, always 1) since it is not missing any property or class instance because ExpLod works by grouping of even two instances if they have the same set of attributes and types, thus does not add any false positives. We can also note that the instance class precision and the instance recall precision reported in columns P_c and R_c are always equal to 1 for Campinas et al algorithm over all the presented datasets, while the property instance precision reported in column P_p is low in most presented datasets. This is because the Campinas et al algorithm works by grouping of two instances if they have the same set types, thus it does not add any false positives at the class level but maybe it will assign some properties to subjects/instances which do not actually have these properties at the KB (false positive at the property level). This explain why it is important to take into consideration quality metrics at the property and class level.

Table 14 shows also that the behavior of Zneika et al and ExpLod algorithms in the case of the *untyped* datasets is the same or approximately the same as in the case of the *typed* datasets, which means that the quality of the summary with regard to the coverage of the instances is not affected by the presence (or not) of schema information in the KB for these two algorithms. On the other hand, we can easily observe the great positive impact left by the absence of typeof schema information for the Campinas et al algorithm.

Also, tables 14s,14t report metrics values at the instance level for the generated RDF summaries of the Campinas et al and Zneika et al algorithms over the DBpedia dataset for the *typed* and *untyped* cases respectively. From the table 14t, we can note that Campinas et al produces the perfect results since it is not missing any property or class instance because for the untyped case, Campinas et al works by grouping of instances if they have the same set of properties, regardless of how many they are; thus does not add any false positives. On the contrary, the table 14s shows the Campinas et al produces a very poor value for the instance level property precision reported in column P_p because with the presence of the class definition for the entities in the KB, works by grouping of instances based only on the types they carry and ignores e.g. how many they are. Thus with a very heterogeneous KB like DBpedia, the Campinas el al algorithm ends up with a lot of extra property instances since for all the properties the same number of property instances is assumed, since the algorithm looks only at the type information.

From this discussion, we can observe that the summarization algorithms provide results of good quality when the coverage of the instances in the KB is concerned. The proposed quality metrics clearly show that relying only on this metric is not adequate to judge the quality of a summary since a lot of the algorithms report perfect scores in all measures. But still we have cases where we can distinguish the quality among the results based on the instances covered by the computed summary, especially when algorithms use approximative methods to compute the summary (one algorithm in our case). It is worth noting here that our Quality Framework can capture both under-coverage (when not all instances are represented in the final result) and over-coverage (when some instances are represented more than once or some fictitious instances are included) of instances. With the metrics at the instance level we can capture these fine differences for covering correctly or not and how much the instance in the KB.

Dataset	ExpLod	Campinas et al	Zneika et al
Jpeel	25	1	1
Jamendo	31	1	1
Sec	6	1	1
LinkedMDB	8464	1	1
Bank	11	1	1
Wordnet	778	1	1
DBLP	108	1	1
Linkedct	5699	1	1
Lobid	9786	1	1

Table 12: Connectivity Metric results

6.2.4. *Results combining schema- and instance-level metrics*

By comparing the results in both cases, it becomes clear why it is important, to take into consideration quality metrics that capture information both at the instance and the conceptual level. Otherwise behaviors like the one demonstrated by ExpLod cannot be captured and summaries that are flawed might be indistinguishable from better ones. Overall, we could argue that the Quality Framework introduced in section 4 is adequate for capturing the fine differences in quality of the summaries produced by the three algorithms. We can also see that with a closer look at the results we can gain or verify insights on how specific algorithms work and the quality of the summaries they produce.

One final metric to be considered is whether the final graph is connected or not and appears as more than one connected components. This might mean that the summarization algorithm while captures correctly the important properties and classes in the KB fails to provide at the end a connected graph. This is important because this might signify whether the summary graph is usable or not for answering for example SPARQL queries. Table 12 reports the connectivity metric values for the summaries produced by the three discussed algorithms over all the datasets described in table 11.It shows that the ExpLod has always high values for this metric which means it provides a disconnected summary, while the two others have always 1, which means that these two algorithms provide a fully connected summary.

So measuring the quality at the schema level, the instance level and the connected components of the graph can give us a detailed view of the strengths and weaknesses of a summary and decide whether to use it or not depending on the potential use and application. We avoided combining all the measures together because this might blur the final picture. The idea is not to necessarily prove an algorithm as better or worse (we can do this to a great extend through the different *F-measures*) but mainly to help the user understand the different qualities of the summaries and choose the best one for the different needs of the diverse use cases.

7. Conclusions and Future Work

In this paper, we introduced a quality framework by defining a set of metrics, that can be used to comprehensively evaluate any RDF summarization algorithm that is reported in the literature. The metrics proposed are independent of the algorithm, the KB (thus the data) and the existence or not of schema information within the KB. The proposed Quality Framework proposed in the paper captures correctly various desirable properties of the original KB. So, it accounts for:

- the conciseness of the summary by:
  - Penalizing the verboseness in the form of multiple patterns representing a single class in the ideal summary
  - Capturing the similarity of the different patterns or groups created by the summarization algorithm with the corresponding ideal summary parts, even if this similarity is not 100%
- the connectedness of the summary by:
  - Introducing a metric on the connectivity of the summary, thus prioritizing connected summaries against not so connected ones
- the comprehensibility of the summary by:
  - Covering the schema part and thus understanding how good a summary is at the structural level
  - Covering the instance part and thus understanding how good a summary is at covering the instances that are in the KB

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.46	0.63	1	1	1	0.81
Campinas et al	1	0.77	0.87	1	1	1	0.93
Zneika et al	1	0.84	0.90	1	1	1	0.95

(a) Typed Jpeel

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.40	0.57	1	1	1	0.78
Campinas et al	1	0.40	0.57	1	1	1	0.78
Zneika et al	1	0.66	0.79	1	1	1	0.89

(b) Untyped Jpeel

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.74	0.85	1	1	1	0.92
Campinas et al	1	0.83	0.90	1	1	1	0.95
Zneika et al	1	0.92	0.95	1	1	1	0.97

(c) Typed Jamendo

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.60	0.75	1	1	1	0.87
Campinas et al	1	0.60	0.75	1	1	1	0.87
Zneika et al	1	0.79	0.88	1	1	1	0.94

(d) Untyped Jamendo

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.21	0.34	1	1	1	0.67
Campinas et al	1	0.28	0.43	1	1	1	0.71
Zneika et al	1	0.53	0.69	1	1	1	0.84

(e) Typed Sec

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.58	0.73	1	1	1	0.86
Campinas et al	1	0.58	0.73	1	1	1	0.86
Zneika et al	1	0.83	0.90	1	1	1	0.95

(f) Untyped Sec

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.03	0.05	1	1	1	0.52
Campinas et al	1	1	1	1	1	1	1
Zneika et al	1	1	1	1	1	1	1

(g) Typed Bank

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.03	0.05	1	1	1	0.52
Campinas et al	1	0.03	0.05	1	1	1	0.52
Zneika et al	1	1	1	1	1	1	1

(h) Untype Bank

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.28	0.43	1	1	1	0.71
Campinas et al	1	0.33	0.49	1	1	1	0.74
Zneika et al	1	0.87	0.93	1	1	1	0.96

(i) Typed LinkedMDB

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.20	0.33	1	1	1	0.66
Campinas et al	1	0.20	0.33	1	1	1	0.66
Zneika et al	1	0.80	0.89	1	1	1	0.94

(j) Untyped LinkedMDB

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.27	0.42	1	1	1	0.71
Campinas et al	1	0.80	0.88	1	1	1	0.94
Zneika et al	1	0.89	0.94	1	1	1	0.97

(k) Typed Wordnet

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.16	0.27	1	1	1	0.63
Campinas et al	1	0.16	0.27	1	1	1	0.63
Zneika et al	1	0.70	0.85	1	1	1	0.92

(l) Untyped Wordnet

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.33	0.49	1	1	1	0.74
Campinas et al	1	0.73	0.84	1	1	1	0.92
Zneika et al	1	0.82	0.90	1	1	1	0.96

(m) Typed DBLP

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.28	0.43	1	1	1	0.71
Campinas et al	1	0.28	0.43	1	1	1	0.71
Zneika et al	1	0.66	0.79	1	1	1	0.89

(n) Untyped DBLP

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.14	0.09	1	1	1	0.54
Campinas et al	1	0.95	0.97	1	1	1	0.98
Zneika et al	0.95	0.91	0.92	0.93	1	0.96	0.94

(o) Typed Linkedct

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.11	0.19	1	1	1	0.59
Campinas et al	1	0.11	0.19	1	1	1	0.59
Zneika et al	1	0.75	0.85	1	1	1	0.94

(p) Untyped Linkedct

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.23	0.37	1	1	1	0.68
Campinas et al	1	0.82	0.90	1	1	1	0.95
Zneika et al	1	0.85	0.91	1	1	1	0.96

(q) Typed Lobid

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.23	0.37	1	1	1	0.68
Campinas et al	1	0.23	0.37	1	1	1	0.68
Zneika et al	1	0.80	0.87	1	1	1	0.93

(r) Untyped Lobid

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
Campinas et al	1	0.21	0.34	1	1	1	0.67
Zneika et al	0.92	0.51	0.65	0.93	1	0.96	0.80

(s) Typed DBpedia

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
Campinas et al	1	0.12	0.21	1	1	1	0.60
Zneika et al	0.91	0.57	0.70	0.95	1	0.97	0.83

(t) Untyped DBpedia

Table 13: Precision, Recall and F-Measure at the Schema level. The R_c column reports the schema class Recall $SchemaRec_{ClassAll}$. The P_c column reports the schema class precision $SchemaPrec_{ClassAll}$. The $F1_c$ reports the schema class F-measure $SchemaF1_c$. The R_p column reports the schema property Recall $SchemaRec_{PropertyAll}$. The P_p coulmn reports the schema property precision $SchemaPrec_{PropertyAll}$. The $F1_p$ column reports the schema property F-measure $SchemaF1_p$. The F1 column reports the overall schema F-Measure $SchemaF1$

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.33	0.49	0.74
Zneika et al	0.99	0.96	0.97	0.99	0.95	0.97	0.97

(a) Typed Jpeel

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
Zneika et al	0.99	0.96	0.97	0.99	0.95	0.97	0.97

(b) Untyped Jpeel

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.49	0.65	0.82
Zneika et al	1	0.98	0.99	1	0.98	0.99	0.99

(c) Typed Jamendo

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
Zneika et al	1	0.98	0.99	1	0.98	0.99	0.99

(d) Untyped Jamendo

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.92	0.95	0.97
Zneika et al	1	1	1	1	1	1	1

(e) Typed Sec

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
Zneika et al	1	1	1	1	1	1	1

(f) Untyped Sec

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.97	0.98	0.99
Zneika et al	1	1	1	1	0.97	0.98	0.99

(g) Typed Bank

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
Zneika et al	1	1	1	1	0.97	0.98	0.99

(h) Untyped Bank

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.08	0.14	0.57
Zneika et al	1	0.93	0.96	1	0.73	0.84	0.89

(i) Typed LinkedMDB

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
Zneika et al	1	0.93	0.96	1	0.73	0.84	0.89

(j) Untyped LinkedMDB

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.32	0.48	0.74
Zneika et al	1	0.80	0.88	1	0.82	0.90	0.89

(k) Typed Wordnet

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
Zneika et al	1	0.93	0.96	1	0.73	0.84	0.89

(l) Untyped Wordnet

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.64	0.78	0.89
Zneika et al	1	0.82	0.90	1	0.71	0.83	0.86

(m) Typed DBLP

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.79	0.88	0.94
Zneika et al	1	1	1	1	0.96	0.98	0.99

(n) Untyped DBLP

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.79	0.88	0.94
Zneika et al	1	1	1	1	0.96	0.98	0.99

(o) Typed Linkedct

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
Zneika et al	1	0.93	0.96	1	0.73	0.84	0.89

(p) Untyped Linkedct

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.37	0.54	0.77
Zneika et al	1	0.91	0.95	1	0.86	0.92	0.935

(q) Typed Lobid

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
Zneika et al	1	0.89	0.94	1	0.77	0.88	0.91

(r) Untyped Lobid

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
Campinas et	1	1	1	1	0.06	0.36	0.68
Zneika et al	0.92	0.73	0.81	0.89	0.61	0.72	0.76

(s) Typed DBpedia

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
Campinas et al	1	1	1	1	1	1	1
Zneika et al	1	0.91	0.95	1	0.86	0.92	0.93

(t) Untyped DBpedia

Table 14: Precision, Recall and F-Measure at the instance level. The R_c column reports the instance class Recall $InstanceRec_{ClassAll}$. The P_c column reports the instance class precision $InstancePrec_{ClassAll}$. The $F1_c$ column reports the instance class F-measure $InstanceF1_c$. The R_p column reports the instance property Recall $InstanceRec_{PropertyAll}$. The P_p column reports the instance property precision $InstancePrec_{PropertyAll}$. The $F1_p$ column reports the instance property F-measure $InstanceF1_p$. The F1 column reports the overall instance F-Measure $InstanceF1$

  - Understanding how well connected the summary and thus the content of the KB is
  - Capturing subtle differences in the result summary, like the omission of just one property or the approximation over the number of instances that allows the user to really understand why and where there is a problem

- the overall quality of the summary so that it can be compared with other summaries by combining the different metrics like precision, recall, F-measure at different levels with connectedness in order to allow for the overall comparison, while the different metrics still provide a more detailed idea on where there are problems with a computed summary.

We made a big effort on validating that the proposed Quality Framework correctly captures the differences present in different summaries by evaluating three different algorithms (that work in substantially different ways) over ten different and diverse datasets, showcasing that indeed the different aspects are correctly captured in terms of quality and that the results are easily matched towards the status of the KB.

To the best of our knowledge, the literature does not report any other effort that tries to capture the quality properties of RDF graph summaries both at the concept (schema) and instance level in a complete and comprehensive way. The experiments showed that using the proposed set of metrics we are able now to compare the quality at different levels of the RDF summaries produced by different algorithms found in the literature, applied on different and diverse datasets and extract useful insights for their suitability for various tasks.

We plan to extend this work by applying the framework to Linked Data sources where quality results might be different for each part of the linked datasets. We would like to explore both theoretically and experimentally whether there are ways to provide consolidated quality metrics treating the linked KBs as one, which will go beyond simply averaging the individual quality results. We would also like to use the framework to assess the quality of the results of more algorithms, in order to validate experimentally its suitability.

References

[1] About World Bank Linked Data: World Bank Finances. http://worldbank.270a.info/about.html#about-datasets/. Accessed: 2017-03-30.

[2] DBLP Bibliography Database in RDF Datahub. https://datahub.io/dataset/fu-berlin-dblp. Accessed: 2017-03-30.

[3] Jamendo DBTune home,. http://dbtune.org/jamendo. Accessed: 2017-03-30.

[4] John Peel DBTune home,. http://dbtune.org/bbc/peel. Accessed: 2017-03-30.

[5] LinkedCT Datahub. https://datahub.io/dataset/linkedct/. Accessed: 2017-03-30.

[6] LinkedMDB home,. http://www.linkedmdb.org/. Accessed: 2017-03-30.

[7] lobid-Bibliographic Resources. https://datahub.io/dataset/lobid-resources. Accessed: 2017-03-30.

[8] WordNet RDF home. http://wordnet-rdf.princeton.edu. Accessed: 2017-03-30.

[9] Anas Alzogbi and Georg Lausen. Similar structures inside rdf-graphs. In *LDOW*, 2013.

[10] Samur Araujo, Jan Hidders, Arjen P de Vries, and Daniel Schwabe. Serimi: resource description similarity, rdf instance matching and interlinking. In *Proceedings of the 6th International Conference on Ontology Matching-Volume 814*, pages 246–247. CEUR-WS. org, 2011.

[11] Stéphane Campinas, Renaud Delbru, and Giovanni Tummarello. Efficiency and precision trade-offs in graph summary algorithms. In *Proceedings of the 17th International Database Engineering & Applications Symposium*, pages 38–47. ACM, 2013.

[12] Stephane Campinas, Thomas E Perry, Diego Ceccarelli, Renaud Delbru, and Giovanni Tummarello. Introducing rdf graph summary with application to assisted sparql formulation. In *Database and Expert Systems Applications (DEXA), 2012 23rd International Workshop on*, pages 261–266. IEEE, 2012.

[13] Gong Cheng, Cheng Jin, and Yuzhong Qu. HIEDS: A generic and efficient approach to hierarchical dataset summarization. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3705–3711, 2016.

[14] Mariano P Consens, Valeria Fionda, Shahan Khatchadourian, and Giuseppe Pirro. S+ epps: construct and explore bisimulation summaries, plus optimize navigational queries; all on existing sparql systems. *Proceedings of the VLDB Endowment*, 8(12):2028–2031, 2015.

[15] Marek Dudás, Vojtech Svátek, and Jindrich Mynarz. Dataset summary visualization with lodsight. In *The Semantic Web: ESWC 2015 Satellite Events - ESWC 2015 Satellite Events Portorož, Slovenia, May 31 - June 4, 2015, Revised Selected Papers*, pages 36–40, 2015.

[16] François Goasdoué and Ioana Manolescu. Query-oriented summarization of rdf graphs. *Proceedings of the VLDB Endowment*, 8(12), 2015.

[17] Amit Krishna Joshi, Pascal Hitzler, and Guozhu Dong. Towards logical linked data compression. In *Proceedings of the Joint Workshop on Large and Heterogeneous Data and Quantitative Formalization in the Semantic Web, LHD+ SemQuant2012, at the 11th International Semantic Web Conference, ISWC2012*. Citeseer, 2012.

[18] Amit Krishna Joshi, Pascal Hitzler, and Guozhu Dong. Logical linked data compression. In *Extended Semantic Web Conference*, pages 170–184. Springer, 2013.

[19] Raghav Kaushik, Philip Bohannon, Jeffrey F Naughton, and Henry F Korth. Covering indexes for branching path queries. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 133–144. ACM, 2002.

[20] Shahan Khatchadourian and Mariano Consens. Explod: Summary-based exploration of interlinking and rdf usage in the linked open data cloud. *The Semantic Web: Research and Applications*, pages 272–287, 2010.

[21] Shahan Khatchadourian and Mariano P Consens. Exploring rdf usage and interlinking in the linked open data cloud using explod. In *LDOW*, 2010.

[22] Shahan Khatchadourian and Mariano P Consens. Understanding billions of triples with usage summaries. *Semantic Web Challenge*, 2011.

[23] Shahan Khatchadourian and Mariano P Consens. Constructing bisimulation summaries on a multi-core graph processing framework. In *Proceedings of the GRADES'15*, page 8. ACM, 2015.

[24] Mathias Konrath, Thomas Gottron, and Ansgar Scherp. Schemex–web-scale indexed schema extraction of linked open data. *Semantic Web Challenge, Submission to the Billion Triple Track*, pages 52–58, 2011.

[25] Mathias Konrath, Thomas Gottron, Steffen Staab, and Ansgar Scherp. Schemex–efficient construction of a data catalogue by stream-based indexing of linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 16:52–58, 2012.

[26] Amine Louati, Marie-Aude Aufaure, Yves Lechevallier, and France Chatenay-Malabry. Graph aggregation: Application to social networks. In *HDSDA*, pages 157–177, 2011.

[27] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. A unifying framework for mining approximate top-k binary patterns. *IEEE TKDE*, 26:2900–2913, 2014.

[28] Carlos Eduardo Pires, Paulo Sousa, Zoubida Kedad, and Ana Carolina Salgado. Summarizing ontology-based schemas in pdms. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 239–244. IEEE, 2010.

[29] Alexander Schätzle, Antony Neu, Georg Lausen, and Martin Przyjaciel-Zablocki. Large-scale bisimulation of rdf graphs. In *Proceedings of the Fifth Workshop on Semantic Web Information Management*, page 1. ACM, 2013.

[30] Blerina Spahiu, Riccardo Porrini, Matteo Palmonari, Anisa Rula, and Andrea Maurino. ABSTAT: ontology-driven linked data summaries with pattern minimalization. In *The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers*, pages 381–395, 2016.

[31] Yan Sun, Kongfa Hu, Zhipeng Lu, Li Zhao, and Ling Chen. A graph summarization algorithm based on rfid logistics. *Physics Procedia*, 24:1707–1714, 2012.

[32] Yuanyuan Tian, Richard A Hankins, and Jignesh M Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 567–580. ACM, 2008.

[33] Yuanyuan Tian and Jignesh M Patel. Interactive graph summarization. In *Link Mining: Models, Algorithms, and Applications*, pages 389–409. Springer, 2010.

[34] Zhichun Wang. A semi-supervised learning approach for ontology matching. In *Chinese Semantic Web and Web Science Conference*, pages 17–28. Springer, 2014.

[35] Haiwei Zhang, Yuanyuan Duan, Xiaojie Yuan, and Ying Zhang. Assg: adaptive structural summary for rdf graph data. ISWC, 2014.

[36] Ning Zhang, Yuanyuan Tian, and Jignesh M Patel. Discovery-driven graph summarization. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 880–891. IEEE, 2010.

[37] Mussab Zneika, Claudio Lucchese, Dan Vodislav, and Dimitris Kotzinos. Rdf graph summarization based on approximate patterns. In *International Workshop on Information Search, Integration, and Personalization*, pages 69–87. Springer, 2015.

[38] Mussab Zneika, Claudio Lucchese, Dan Vodislav, and Dimitris Kotzinos. Summarizing linked data rdf graphs using approximate graph pattern mining. In *EDBT 2016*, pages 684–685, 2016.