RDF Graph Summarization Based on Approximate Patterns

Mussab Zneika, Claudio Lucchese, Dan Vodislav, Dimitris Kotzinos

RDF Graph Summarization Based on Approximate Patterns

Mussab Zneika[1], Claudio Lucchese[2], Dan Vodislav[1], and Dimitris Kotzinos[1]

[1]ETIS Lab (ENSEA, UCP, CNRS UMR 8051)
Pontoise, France
[2]ISTI-CNR,
Pisa, Italy
Mussab.Zneika@ensea.fr,Claudio.Lucchese@isti.cnr.it,
Dan.Vodislav@u-cergy.fr,Dimitrios.Kotzinos@u-cergy.fr

Abstract. The Linked Open Data (LOD) cloud brings together information described in RDF and stored on the web in (possibly distributed) RDF Knowledge Bases (KBs). The data in these KBs are not necessarily described by a known schema and many times it is extremely time consuming to query all the interlinked KBs in order to acquire the necessary information. But even when the KB schema is known, we need actually to know which parts of the schema are used. We solve this problem by summarizing large RDF KBs using top-K approximate RDF graph patterns, which we transform to an RDF schema that describes the contents of the KB. This schema describes accurately the KB, even more accurately than an existing schema because it describes the actually used schema, which corresponds to the existing data. We add information on the number of various instances of the patterns, thus allowing the query to estimate the expected results. That way we can then query the RDF graph summary to identify whether the necessary information is present and if it is present in significant numbers whether to be included in a federated query result.

Keywords: RDF graph summary, approximate patterns, RDF query, Linked Open Data, federated query

1 Introduction

The amount of RDF (Resource Description Framework, www.w3.org/RDF/) data available on the semantic web is increasing fast both in size and complexity, e.g. more than 1000 datasets are now published as part of the Linked Open Data (LOD) cloud, which contains more than 62 billion RDF triples, forming big and complex RDF data graphs. It is also well established that the size and the complexity of the RDF data graph have a direct impact on the evaluation of the RDF queries expressed against these data graphs. There are cases, especially on the LOD cloud, where we observe that a query against an RDF Knowledge Base (KB) might retrieve no results at the end because either (a) the association between the different RDF KBs is weak (based only on a few associative links) or

(b) there is an association at the schema level that has never been instantiated at the actual data level. The bigger and more complex the RDF KBs involved are, the more costly this operation will be, without giving any useful results at the end. So it is useful to know before evaluating a complex query towards an actual KB both the structure and the size of the content of the KB. This means that we need to know the main associations among the different "types" of data stored and statistical information (mainly counts) for the instances that can be classified under them.

By creating summaries of the RDF KBs, we allow the user or the system to decide whether or not to post a query, since (s)he knows whether information is present or not. 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 significant results. We need to compute the summaries only once and update them only after significant changes to the KB. Given the (linked) nature of LOD KBs this will speed up the processing of queries in both centralized and distributed settings. Moreover, this would allow working and posting queries towards many RDF KBs that carry none at all or only partial schema information. By applying RDF summarization techniques, we can extract, at least, a subset of the schema information (that should represent quite well at least the main types of instances stored in the KB and their relationships) and thus facilitate the query building for the end users with the additional benefit of categorizing the contents of the KB based on the summary. We can envision similar benefits when KBs are using mixed vocabularies to describe their content. In all these cases we can use the RDF summary to concisely describe the data in the RDF KB. Thus in this work we study the problem of LOD/RDF graph summarization that is: *given an input RDF graph (that might extending itself over multiple RDF stores and might link different datasets), find the summary graph which reduces its size, while preserving the original inherent structure and correctly categorizing the instances included in the KB.*

Two main categories of graph summarization efforts have been proposed in the literature to this date and are discussed in more detail in Section 5 of this paper: (1) *aggregation and grouping approaches* [11], which are based on grouping the nodes of input RDF graph G into clusters/groups based on the similarity of attributes' values and neighborhood relationships associated with nodes of G and (2) *structural extraction approaches* [4, 6] which are based on extracting some kind of schema where the summary graph is obtained based on an equivalence relation on the RDF data graph G, where a node represents an equivalence class on nodes of G. To the best to our knowledge, few of these approaches are concentrating on RDF KBs and only one of them [4] is capable of producing RDF schema as result, which would allow the use of RDF tools (e.g. SPARQL) to query the summary. Our approach provides comparable or better results in most cases.

Thus in this paper, we address the problem of creating RDF summaries of LOD/RDF graphs that is: given an input RDF graph, find the summary graph which reduces its size, while preserving the original inherent structure and correctly categorizing the instances included in the KB. The contribution of our work is a novel solution into summarizing semantic LOD/RDF graphs, where our summary graph is a RDF graph itself so that we can post simplified queries towards the summarizations and not the original graphs and exploit also the statistical information about the structure of a the RDF input graph which are included to our summary graph like the number of class and property instances per pattern, so as to decide whether or not to post a query to a specific RDF KB, our solution is based on mining top-k approximate graph patterns [13]. In summary, our solution is responding to all the requirements by extracting the best approximate RDF graph patterns, construct a summary RDF schema out of them and thus concisely describe the RDF input data. We offer the following features:

- The summary is a RDF graph itself, which allows us to post simplified queries towards the summarizations using the same techniques (e.g. SPARQL).
- Statistical information like the number of class and property instances per pattern is included in our summary graph, which allows us to estimate a query's expected results' size towards the original graph.
- The summary is much smaller than the original RDF graph, contains all the important concepts and their relationships based on the number of instances.
- Schema independence: it summarizes the RDF input graphs regardless of having or not RDFS triples (this means that we do not require or assume any schema information).
- Heterogeneity independence: it summarizes the RDF graphs whether they are carrying heterogeneous or homogeneous information.

In the sequel, Section 2 recalls the some of the foundations of RDF and RDFS, which are useful for defining some concepts in our work and are used to define both the schema and the queries asked against any RDF graph; section 2 also sets the requirements for calculating RDF summaries. Section 3 describes our approach for RDF graph summarization and describes both the pre-processing of the data and the post processing of the results in order to construct a summary that is also a valid RDFS. Section 4 presents our preliminary experiments while Section 5 presents related work. We then conclude our paper in section 6.

2 Preliminaries

In this section, we give basic terminology used in this work about the RDF data, schema and queries. We then formulate the problem this work addresses.

The RDF data model is the standard model for representing data on the Web in terms of triples of the form (s, p, o), explaining that the subject s has the property p, and the value of that property p is the object o. Each triple denotes a binary relationship between two entities. For example, the triple $(X, painted, Z)$

denotes a relationship between an entity represented by X (e.g., a painter) and another entity represented by Z (e.g., a painting). The intuitive way to view a collection of RDF data statements is to represent them as a labeled directed graph in which entities are represented as nodes and named relationships as labeled directed edges. These RDF data statements are usually accompanied with a schema called RDF Schema which provides a data-modeling vocabulary for RDF data. RDF Schema provides concepts for declaring and describing the resource types (called classes) (e.g. Painter) and the resource relationship and attributes (called properties) (e.g. paints). RDF Schema can also be represented as a directed labeled graph where the labeled nodes represent the names of classes and the labeled edges the name of relations and properties. Some definitions are given below to define and explain the RDF schema graph and the RDF instance Graph. 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, rdfs:subPropertyOf, rdfs:label, rdfs:comment }.The concepts of RDF schemas and instances can be formalized as follows.

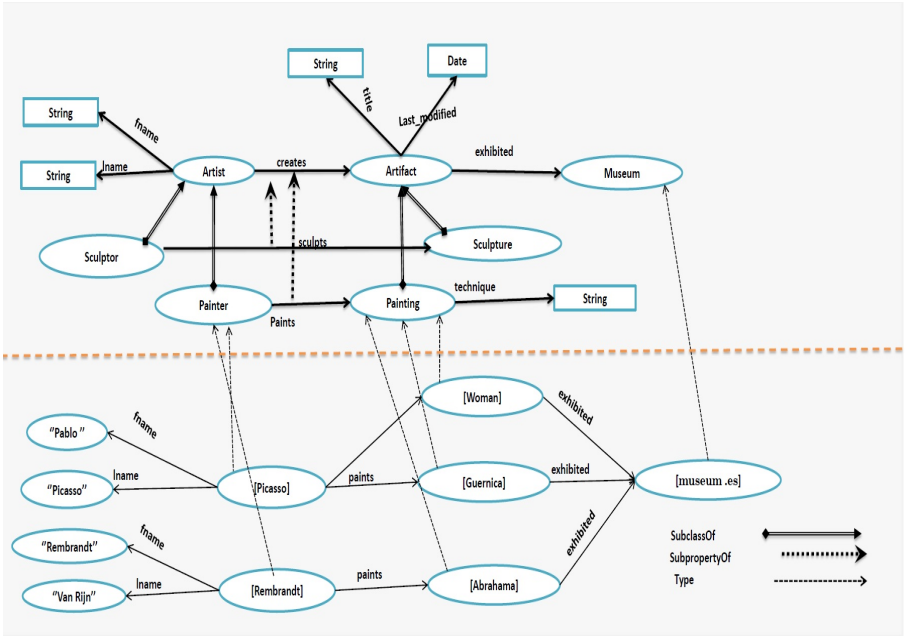

Fig. 1. RDF Schema and instance graphs

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

- N_s *is the set of nodes.*
- $E_s \subseteq \{(x, \alpha, y) | e : x \in N_s, \alpha \in T, y \in N_s\}$ *is the set of labelled edges.*
- $\lambda_s : N_s \longrightarrow C \cup P$ *is a injective node labeling function that maps nodes of* N_s *to class and property URIs, such that* $\lambda_s(n) \in C \cup P$ *for any* $n \in N_s$.
- $\lambda_e : E_s \longrightarrow T$ *is a injective edge labeling function that maps edges of* E_s *to RDFS standard property URIs included in* T*, such that* $\lambda_e(e) \in T$ *for any* $e \in E_s$.

Example 1. The upper part of Fig. 1 shows a visualization example of an RDF schema graph which describes the cultural domain. For example, the class *Artist* denotes the set of resources which represent artists' entities, while the class *Artifact* denotes the set of resources which represent artifacts' entities. Note that properties serve to represent characteristics of resources as well as relationships between resources. For example the properties *fname,lname* represent the first name and the last name of an artist respectively, while property *creates* denotes that instances of the class *Artist* are related to instances of the class *Artifact* by a *create* relationship. Both classes and properties support inheritance, e.g., the class *Painter* is a subclass of *Artist* class while the property *paints* is sub-property of *creates* property.

Definition 2 (RDF data graph). *An* RDF instance graph *or* RDF data graph $G_i = (N_i, E_i, \lambda_i, \lambda_{ei}, I, P, L)$ G_s *is a directed labeled graph where:*

- N_i *is the set of nodes.*
- $E_i \subseteq \{(x, \alpha, y) : x \in N_i, \alpha \in P, y \in N_i\}$ *is the set of labelled edges.*
- $\lambda_i : N_i \longrightarrow I \cup L$ *is a node labelling function that maps nodes of* G_i *to instance URIs or literals, respectively such that* $\lambda_i(n) \in I \cup L$ *for any* $n \in N_i$.
- $\lambda_{ei} : E_i \longrightarrow P$ *is a injective edge labeling function that maps edges of* E_i *to property URIs, such that* $\lambda_{ei}(e) \in P$ *for any* $e \in E_i$.

Example 2. The lower part of Fig. 1 depicts an instance graph building on the schema information explained in the Example 1, where the dashed arrows denote a member of relationships from instances to classes. This graph represents 6 different resources. The resource [Picasso] (we use [X] to denote that X is an instance of some textitclass) is an instance of the *Painter* class (part of the RDF Schema defined earlier) having two properties *fname* and *lname* with values of type *String* and two properties *paints* with value the resources [Woman] and [Guernica]. The resource [Rembrandt] is also described as an instance of the *Painter* class having two properties *fname* and *lname* with string value but it has only one property *paints* with value the resource [Abrahama]. [Abrahama], [Woman] and [Guernica] resources are described as instances of *Painting* class having *exhibited* property with value the resource [museum.es] which is described as an instance of the *Museum* class.

Definition 3 (Type Edge). *We define* Type Edge *the edge with rdf:type label, which is typically used to define a* type *of which the node is an instance of,*

e.g., the dashed edge type *in Fig. 1 declares that the node Picasso is a Painter. We denote the type edge with* (x, τ, y)*. Let* $Types(x) = \{\lambda_i(y) : \forall(x, \tau, y) \in E_i \wedge x \in N_i\}$ *be the set of nodes' labels related to the node* x *via an explicit type edge definition, e.g., the* $Types(Picasso) = \{Painter\}$*, while* $Types(Guernica) = \{Painting\}$*.*

Definition 4 (Properties). *We define as* $Properties(x) = \{\alpha : \forall(x, \alpha, y) \in E_i : \alpha \neq \tau \wedge \lambda_i(y) \in I \wedge x \in N_i\}$ *the set of labels of the non-Type edges which associate the node* x *with a set of entity nodes(nodes labeled by* instance *URIs).*

Definition 5 (Attributes). *We define as* $Attributes(x) = \{\alpha : \forall(x, \alpha, y) \in E_i : \alpha \neq \tau \wedge \lambda_i(y) \in L \wedge x \in N_i\}$ *the 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 3. 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 6 (RDF graph pattern). *An RDF graph pattern* $G_P = (N_P, E_P, \lambda_P, \beta, P)$ *is a connected edge-labeled directed graph where:*

- N_P *is a set of nodes;*
- $E_P \subseteq E_S$*;*
- $\lambda_P : E_P \longrightarrow P$ *and for* $e \in E_P, \lambda_P(e) = \lambda_s(e)$*;*
- $\beta : N_P \longrightarrow N$ *maps nodes to the set of natural numbers.*

Example 4. The pattern $\{1 \longrightarrow paints \longrightarrow 2 \longrightarrow exhibited \longrightarrow 3\}$ has adequate support in the instance graph shown in the bottom part of Fig. 1, which means that we can find an adequate number of instances and instance relationships or properties in the corresponding part of the RDF data graph that could be represented by this pattern.

2.1 RDF Summary Requirements

Given the above definitions, we are interested in extracting a summary graph having the following characteristics:

- The summary is a RDF graph: The summary graph should be a RDF graph itself, which allows us to post simplified queries towards the summarizations using the same languages or techniques (e.g. SPARQL).
- The size of the Summary: The volume of a graph is the numbers of its edges and nodes. Reducing the volume of a summary comes with a price, that of reduced precision of the summary. Thus the summary graph should:
  - Be smaller than the original RDF graph.
  - Contain all the important information.
  - Report the most representative nodes (classes) and edges (properties).
  - Be schema independent: It must be possible to summarize the RDF graphs whether or not they have associated RDFS triples.

We are also interested in working towards specifying the quality of the summary. An example of this is identifying the summary's precision, i.e. errors in summary that can be e.g. invalid edges or path(s), which do not exist in the actual data graph. The precision model should account for the paths that exist in summary but not in data graph.

3 RDF summarization

We present in this section our approach of RDF graph summarization, which is based on extracting the smallest set of approximate graph patterns (as provied in [13]) that best describe the input dataset, where the quality of the description is measured by an information theoretic cost function. We use a modified version of the PANDA$^+$ algorithm presented in [13], which uses a greedy strategy to identify the smallest set of patterns that best optimize the given cost function. The PANDA$^+$ algorithm normally stops producing further patterns when the cost function of a new patterns' set is larger than the corresponding noise reduction. It also allows the users to fix a value k to control the number of extracted patterns. Since PANDA$^+$ is using a binary matrix to represent the instances participation in a property (column), one of the challenges that we faced was how to map the RDF KB to this binary matrix while preserving the semantics of this KB and in addition producing always a valid RDF graph as a result. Our approach works in three independent steps that are described below and are visualized in Fig.2.

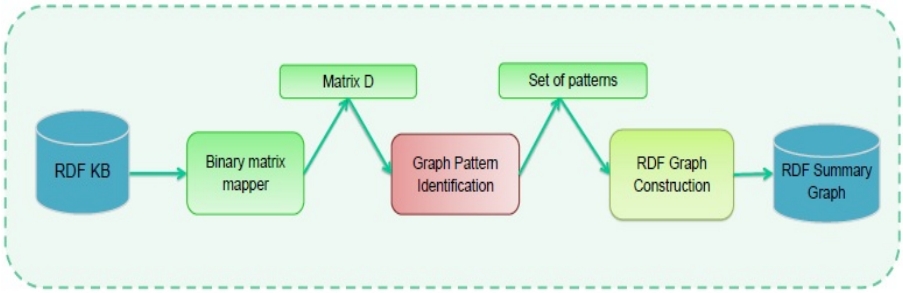

Fig. 2. Our RDF graph summarization approach

3.1 Binary Matrix Mapper

We transform the RDF graph into a binary matrix D, where the rows represent the subjects and the columns represent the predicates. We preserve the semantics of the information by capturing distinct types (if present), all attributes and properties. In order to capture both the subject and the object of

a property, we create two columns for each property. The first column captures the instance that is the subject (belongs to the domain of the property), while the second one(we call it *reverse property*) captures the instance that is the object (belongs to the range of the property),eg. for the property paints we create two columns(paints,R_paints) see Table 1 where the column paints captures the participation of an instance as subject $\{Picasso, Rembrant\}$ while the column R_paints captures the participation of an instance as object $\{Woman, Guernica, Abrahama\}$. We extend the RDF URI information by adding a label to represent the different predicates carrying this information into the patterns. This label is of the following form: Usage prefix and the RDF URI element label where these two parts are concatenated with a forward slash ("/"), where the usage prefix is T for type, P for property and R for reverse properties. This matrix is defined in the following way:

$$D(i;j) = \begin{cases} 1, & \text{the i-th URI (as defined in RDF) has j-typeof or is j-property's} \\ & \text{domain/range or is j-attribute's domain} \\ 0, & \text{otherwise} \end{cases}$$

Example 5. Table 1 shows the mapped binary matrix D for the RDF graph depicted in Fig.1. This matrix consists of 9 columns and 6 rows, where the columns represent 2 distinct attributes(fname,lname), 2 distinct properties (paints, exhibited), 2 distinct reverse proprieties (Reverse_paints, Reverse_exhibted), 3 distinct types (Painter(c), Painting(c),Museum(c)). In order to distinguish between the types/classes and the properties/attributes at the visualization level , we use Y(c) to denote that Y is type/class. The rows represent the 6 distinct subjects (Picasso, RembrantvanRijn, Woman, Guernica, Abraham,museum.es), e.g. D(1,1)=D(1,3)=D(1,4)= D(1,5)=1 because Picasso, who is described in the first row, is an instance of Painting class and has (lname, fname) attributes and paints properties respectively, while D(1,6)=0 because Picasso does not have the exhibited property.

Table 1. The mapped binary matrix D for the RDF instance graph depicted in Fig.1

	Painter(c)	Painting(c)	lname	fname	paints	exhibited	R_paints	R_exhibited	Museum(c)
Picasso	1	0	1	1	1	0	0	0	0
Rembrant	1	0	1	1	1	0	0	0	0
Woman	0	1	0	0	0	0	1	0	0
Guernica	0	1	0	0	0	1	1	0	0
Abraham	0	1	0	0	0	1	1	0	0
museum.es	0	0	0	0	0	0	0	1	1

Note here that our experiments so far (please see next section) provide indication that the algorithm works adequately well even in the absence of any schema information, or in other words no schema information is required for the algorithm to work adequately well.

3.2 Graph Pattern Identification

We aim at creating a summary of the input RDF graph by finding patterns in the binary matrix produced in the previous step (see Table 1). By patterns, we mean properties (columns) that occur (are marked with 1) either completely or partially (and thus approximately) in several subjects (rows). This problem is known in the data mining community as *approximate pattern mining*. This is an alternative approach to pattern enumeration. It aims at discovering the set of k patterns that best *describe*, or *model*, the input data. Algorithms differ in the formalization of the concept of *dataset description*. The quality of a description is measured internally with some cost function, and the top-k mining task is casted into the optimization of such cost. In most of such formulations, the problem is demonstrated to be NP-hard, and therefore greedy strategies are adopted. Moreover in our case, it is important that we also manage to preserve or extract some meaningful semantics from the KB, so the problem has an additional level of complexity, which is partially handled in the next step where an RDF graph is constructed from the extracted patterns.

Example 6. Table 2 shows possible patterns which can be extracted from the mapped binary matrix depicted in Table 1. The first column represents the *pattern id*. The second column represents the predicates included in a pattern and the third column represents the number of subjects per pattern, e.g., the pattern P1 denotes that there are three subjects belong to the *Painting* class and have {*exhibited*} an outgoing attribute and{*paints*} an incoming attribute. It should be noted here that since approximate patterns are computed having a subject classified under a pattern, as already explained, does not necessarily mean that in the KB this subject carries necessarily all the properties. This one reason why the information on which subjects are classified under which pattern is not carried along in the extracted schema.

Table 2. Extracted patterns example

ID	Pattern	Correspondence class
P1	Painting(c),exhibited, revers_paint	3
P2	Painter(c),paints, fname, lname	2
P3	Museum(c)	1

Firstly we introduce some notation. Without loss of generality we refer to a *binary* matrix $\mathcal{D} \in \{0,1\}^{N \times M}$ as a *transactional dataset* of N transactions and M items, where $\mathcal{D}(i,j) = 1$ if the $j-$th item occurs in the $i-$th transaction, and $\mathcal{D}(i,j) = 0$ otherwise. An *approximate pattern* P identifies two sets of items/transactions, and is denoted by a pair of binary vectors $P = \langle P_I, P_T \rangle$, where $P_I \in \{0,1\}^M$ and $P_T \in \{0,1\}^N$. The outer product $P_T \cdot P_I^{\mathsf{T}} \in \{0,1\}^{N \times M}$ of the two binary vectors identifies a sub-matrix of $\mathcal{D}$. We say that the occurrence (i,j) is covered by P *iff* $i \in P_T$ and $j \in P_I$.

The quality of a set of patterns $\Pi = \{P_1, \ldots, P_{|\Pi|}\}$ depends on how well they match the given dataset $\mathcal{D}$. We account for the mismatches with a *noise matrix* $\mathcal{N} \in \{0,1\}^{N \times M}$ defined as:

$$\mathcal{N} \;=\; \bigvee_{P \in \Pi} (P_T \cdot P_I^{\mathsf{T}}) \quad \veebar \; \mathcal{D}. \tag{1}$$

where $\vee$ and $\veebar$ are respectively the element-wise *logical or* and *xor* operators. The matrix $\mathcal{N}$ encompasses those occurrences $\mathcal{D}(i,j) = 1$ which are not covered by any pattern in Π (*false negatives*), as well as those $\mathcal{D}(i,j) = 0$ which are incorrectly covered by any of the patterns in Π (*false positives*).

Approximate Top-k Pattern Discovery requires to find a small set of patterns Π that minimizes the noise matrix $\mathcal{N}$. More formally:

Problem 1 (Approximate Top-k Pattern Discovery). Given a binary dataset $\mathcal{D} \in \{0,1\}^{N \times M}$ and an integer k, find the pattern set $\overline{\Pi}_k$, $\left|\overline{\Pi}_k\right| \leq k$, that minimizes a *cost function* $J(\Pi_k, \mathcal{N})$:

$$\overline{\Pi}_k = \operatorname*{argmin}_{\Pi_k} J(\Pi_k, \mathcal{N}). \tag{2}$$

Different approaches proposed different cost functions which are tackled with specific greedy strategies. In addition, it is usually possible to specify additional *parameters*, whose purpose is to make the pattern set $\overline{\Pi}_k$ subject to some *constraints*, such as the minimum frequency of a pattern (i.e., the number of its transactions), or the maximum amount of false positives tolerated in each pattern.

In this work, we adopted the state-of-the-art PaNDa$^+$ algorithm [13] to extract relevant patterns from the binary dataset resulting from a transformation of the original RDF graph.

PaNDa$^+$ adopts a greedy strategy by exploiting a two-stage heuristics to iteratively select a new pattern: (a) discover a noise-less pattern that covers the yet uncovered 1-bits of $\mathcal{D}$, and (b) extend it to form a good approximate pattern, thus allowing some false positives to occur within the pattern. It is discussed also in section5 that PaNDa$^+$ is considered the state of the art for the approximate pattern mining algorithms.

PaNDa$^+$ greedily optimizes the following cost function:

$$J^+(\Pi_k, \mathcal{N}, \gamma_{\mathcal{N}}, \gamma_P, \rho) \;=\; \gamma_{\mathcal{N}}(\mathcal{N}) \;+\; \rho \cdot \sum_{P \in \Pi_k} \gamma_P(P) \tag{3}$$

where $\mathcal{N}$ is the noise matrix, $\gamma_{\mathcal{N}}$ and γ_P are user defined functions measuring the cost of the noise and patterns descriptions respectively, and $\rho \geq 0$ works as a regularization factor weighting the relative importance of the patterns cost.

Depending on the parameters of the J^+, PaNDa$^+$ can greedily optimize several families of cost functions, including the ones proposed by other state-of-the-art algorithms [25, 15, 16, 12]. In this work, inspired by the MDL principle

[19] we used $\gamma_{\mathcal{N}}(\mathcal{N}) = \mathsf{enc}(\mathcal{N})$, $\gamma_P(P) = \mathsf{enc}(P)$ and $\rho = 1$, where $\mathsf{enc}(\cdot)$ is the optimal encoding cost.

PANDA$^+$ extracts patterns iteratively, and each pattern is grown greedily by adding new items and checking those transactions that approximately include those items. Rather than considering all the possible exponential combinations of items, these are sorted to maximize the probability of generating large cores, and processed one at the time without backtracking. We mention two sorting strategies: (a) by frequency of an item in the full dataset, and (b) by the average frequency of every pair of items including the given item (named *charm* by [26]).

Differently from other algorithms, PANDA$^+$ allows to define two maximum noise thresholds $\epsilon_r, \epsilon_c \in [0, 1]$ which bound the ratio of *false positive*, row- and column-wise respectively, of each extracted pattern. Finally, it also allows to tune via the parameter ρ the relative importance of the patterns simplicity versus the amount of noise induced.

These features make PANDA$^+$ a very flexible tool for approximate pattern mining extraction and allow us to include some RDF related knowledge in the algorithm so that the computations will benefit from that.

3.3 Constructing the RDF summary graph

We have implemented a process, which reconstructs the summary as a valid RDF graph using the extracted patterns. For each pattern, we start by generating a node labeled by a URI (minted from a hash function), then we add an attribute with the *bc:extent* label representing the number of instances for this pattern. Then and for each item involved in this pattern, we use the labels generated in 3.1 to understand its type. So depending on whether it is:

- a property: We generate a direct edge from the node representing the pattern containing this property to the node representing the pattern containing the reverse property.
- an attribute: We generate a direct edge to a newly generated node labeled by a URI (g from a hash function).
- Type: We generate a direct edge labeled with *RDF:type* label to the newly generated node labeled with the RDFS label of this type.

The process exploits RDF-related information already embedded in the binary matrix (e.g. property X range links) and tries to construct a valid RDF schema to represent the KB. This schema is enriched with statistical information since the algorithm returns for each pattern the number of instances it corresponds to.

Example 7. Fig.3 shows the constructed RDF summary graph for the set of patterns depicted in Table 2. The names of the patterns (using their pattern-ids (e.g. P1, P2, etc.) are not showed here) but we can easily, even visually, observe that we have recovered the original schema minus the *subclassof* and *subpropertyof* relationships, which we do not deal with at this stage of the work. In this example we also do not capture the *superclasses* but this is due to the fact that they are not explicitly instantiated in the KB.

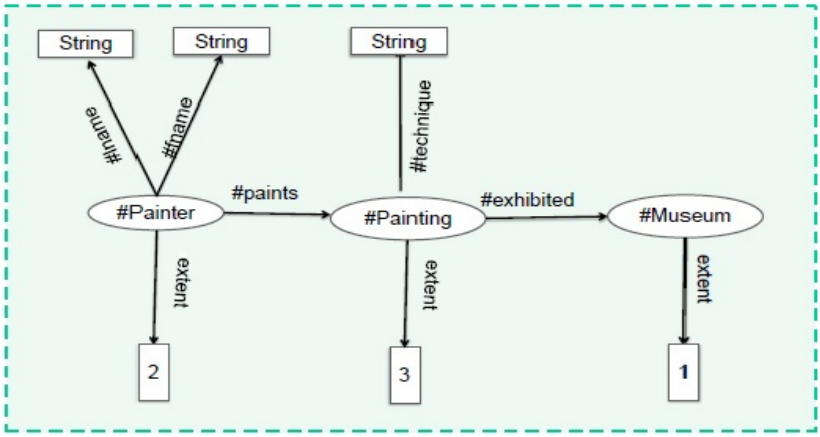

Fig. 3. RDF Summary graph for the set of patterns depicted in Table 2

4 Experiments

In this section, we give an evaluation of our RDF graph summarization approach using the real-world Jamendo[1] dataset. Jamendo is a real dataset from the LOD cloud containing information about music artists and their productions, since it is an online distributor of the production of independent music artists. The data focus on record authorship, release and distribution over internet channels. Its data representation relies on the Music Ontology[2] and parts of FOAF, Dublin Core, Event, Timeline and Tags ontologies. This dataset is interlinked with the Geonames[3] and the Musicbrainz[4] datasets. It consists of 1,047,837 triples, which are classified under 11 classes and are using 25 properties. The schema information about the Jamendo dataset is reported in Fig 4. We evaluate our approach for the following two cases:

- Fully typed data: Where each instance of this dataset has at least one typeof link/property.
- Untyped Data: Where none of the datasets subjects/objects or properties has a defined type (we explicitly deleted all of them).

Table 3 shows the results of applying the PANDA$^+$ with the charm sorting and typed Xor Cost function parameters (which are briefly explained in section 3.2) on the fully typed Jamendo dataset. The first column shows the pattern id, the second shows the predicates involved in the pattern, while the third column shows the number of instances per pattern. The last column shows

[1] http://dbtune.org/jamendo/
[2] http://musicontology.com/
[3] http://www.geonames.org/ontology
[4] http://musicbrainz.org/

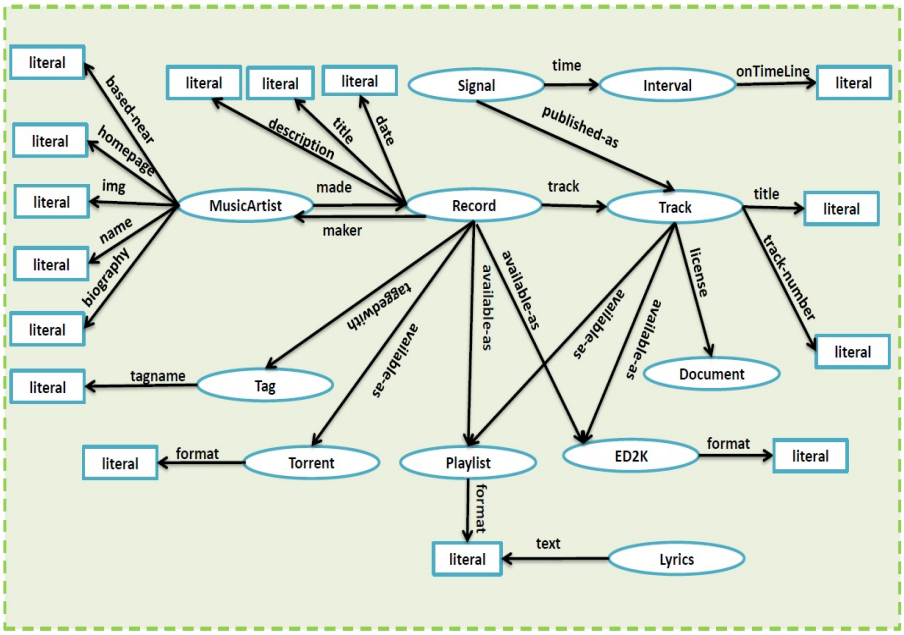

Fig. 4. Schema information about the Jamendo dataset

the corresponding class for a pattern. We have 15 patterns: P1 represents the Playlist class and the properties that have this class as domain or range, P2 represents the Track class and the properties that have this class as domain or range, P3 represents the Signal class and the properties that have this class as domain or range, P4 represent the Interval class and the properties that have this class as domain or range, P5 represents the Record class and the properties that have this class as domain or range, P6 represents the Tag class and the properties that have this class as domain or range, P7 represents Lyrics class and the properties that have this class as domain or range , P8 represents MusicArtist class and the properties that have this class as domain or range, P9 represents MusicArtist class also, P10 represents Document class and the properties that have this class as domain or range, P14 represents Torrent class, and P15 represents ED2k class. We can note that we have two patterns P8 and P9 represent the class MusicArtist. The pattern 8 represents it and its following properties(name,homepage,biography,based-near,img) while the pattern P9 represents it and its following properties(name,made). We have this case because at the level of data we have 3346 instances of MusicArtist have the following properties (name,homepage,biography,based-near,img), while only 159 instances having (name,made. Our Post-processing in this case to merge these two patterns and replace them by one pattern.

Table 3. PANDA$^+$ with fully typed Jamendo dataset

ID	Pattern	Extent	corresponding class
P1	Playlist(c),Reverse_available-as,format	102804	Playlist
P2	Track(c),available-as,title,license,track-number,Reverse_published-as,Reverse_track	45634	Track
P3	Signal(c),published-as,Reverse-recorded-as,time	45634	Signal
P4	Interval(c),Reverse_time,onTimeline	45634	Interval
P5	Record(c),date,image,Reverse_made, available-as,maker, track,title,taggedwith	5786	Record
P6	Tag(c),Reverse_taggedwith,tagName	9235	Tag
P7	Lyrics(c),text,Reverse_factor	8823	Lyrics
P8	MusicArtist(c),name,made,Reverse_maker, based-near, homepage, img	3346	MusicArtist
P9	MusicArtist(c),name,made,Revese-maker	159	MusicArtist
P10	Document(c),Reverse-license	92	Document
P11	recorded-as	45634	...
P12	Factor	8823	....
P13	description	880	....
P14	Torrent(c)	11572	Torrent
P15	Ed2k(c)	11572	Ed2K

In comparison with the original schema of the Jamendo dataset, which was reported in Fig 4, the results contain exactly the same information. In other words, all the classes and all the properties are correctly identified (are the same with the original schema of the dataset) and the corresponding instances are correctly classified.

Table 4 shows the results of applying the PANDA$^+$ with the charm sorting and typed Xor Cost function parameters on the untyped Jamendo dataset. In comparison these results with the results of the Table 3, we find that these results miss 2 patterns, the pattern P14 which represents *Torrent* class and the pattern P15 which represents *ED2k* class. Note here that our experiments provide indication that the algorithm works adequately well even in the absence of all the schema information. One thing that can be noted here and needs to be further investigated is that both those patterns are having only one member, which is the corresponding class information and which is now deleted from the dataset. Thus not finding these two patterns is completely reasonable since this information does not exist anymore in the KB. Nevertheless we need to further look into the matter.

Table 4. PANDA$^+$ with untyped Jamendo dataset

ID	Pattern	Extent	corresponding class
P1	Playlist(c),Reverse_available-as,format	102804	Playlist
P2	Track(c),available-as,title,license,track-number,Reverse-published-as, Reverse_track	45634	Track
P3	Signal(c),published-as,Reverse-recorded-as,time	45634	Signal
P4	Interval(c),Reverse_time,onTimeline	45634	Interval
P5	Record(c),date,image,Reverse_made, available-as,maker,track,title,taggedwith	5786	Record
P6	Tag(c),Reverse_taggedwith ,tagName	9235	Tag
P7	Lyrics(c),text,Reverse_factor	8823	Lyrics
P8	MusicArtist(c),name,made,Reverse_maker, based-near,homepage,img	3346	MusicArtist
P9	MusicArtist(c),name,made,Revese-maker	159	MusicArtist
P10	Document(c),Reverse-license	92	Document
P11	recorded-as	45634	...
P12	Factor	8823	....
P13	description	880	....

5 Related Work

5.1 Graph Summarization

In the literature we find works that deal with the (RDF) graph summarization problem either partially or to its full extent. So we can find relevant works under the more generic concepts of graph compression [1, 18], graph synopsis [2], graph simplification [24] and network abstraction [29]. All refer to the same problem, i.e. how to extract from a graph the most representative nodes and edges, thus minimizing the graph. The most extensive literature exists in the field of graph compression, especially for Web graphs [1, 18]. One of the problems usually encountered in these works is that the result is not RDF graph itself, something not suitable for our case since we need to be able to keep querying the graphs using the same techniques (e.g. SPARQL).

Few efforts have been reported in the literature on summarizing the Data graphs. These efforts fall under two categories based on the type of algorithms used and the goal of the summarization. The first category contains algorithms [21, 22, 17, 28, 11, 23] for summarizing the homogenous directed labeled graph based on an aggregation algorithm. The main goal of algorithms in this category is to produce understandable concise graph representation, which is smaller than the original graph in size, in order to facilitate the visualization and to highlight communities in the input Data graph, which greatly facilitates its interpretation based on an aggregation algorithm. The idea behind that is to group the nodes of data graph G into clusters/groups based on the similarity of attributes's values and neighborhood relationships associated with nodes of G. The most known

algorithm in this category is the K-SNAP [23, 22] algorithm which produces a summary graph with size K (contains K groups) by grouping nodes based on set of user-selected node attributes and relationships. It begins with a grouping based on attributes of the nodes, and then tries to divide the existing groups according to their neighbors groups. Two super-nodes are connected by a super-edge if there is a pair of nodes, one from each group, connected in the original graph. They require nodes in each group having the same attribute information, so the total number of possible attribute values cannot be too many. Otherwise, the size of summaries will be too large for users to explore. K-SNAP allows summaries with different resolutions, but users may have to go through a large number of summaries until some interesting summaries are found. The second limitation of the K-SNAP that it is only applicable for homogeneous graphs. In other words, it is only applicable for the graphs which represent single community of entities (e.g., student community, readers community), where all these entities have to be characterized by the same set of attributes. Something not suitable for the semantic web graphs since the RDF graphs are usually heterogeneous and it also may be without knowledge (nodes are not attributed). The third limitation is that it handles only the categorical node attributes but in the real world, many node attributes are not categorical, such as the age of a user or the salary.

The second category contains algorithms [4, 8, 5, 27, 6, 7, 3, 9, 10, 20] for summarizing the hetero- or homo-geneous RDF graphs, based on an equivalence relation. The main goal of this type of summarization is to extract some kind of schema in order to understand the data and the interlinks that are used both within and across the input linked datasets. A summary graph G_ss is obtained based on an equivalence relation on the RDF data graph G, where a node represents an equivalence class on nodes of G. Khatchadourian, Shahan, and Consens [6, 7] propose a software called ExpLOD, which produces summary graphs for one or more specific aspects of an RDF dataset, e.g., class or predicate usage. Their summary can be generated even if the RDF input graph does not use the full schema or it uses multiple schemas. Summary is computed over the RDF graph using each nodes bisimulation label: two nodes v and u are bisimilar if they have the same set of types and properties. Some statistics, like the number of instances per class or the number times a property is used to describe all instances, are aggregated with the structural information. The advantage of ExpLOD approach is that its generated summaries show a datasets structure as homo- or heterogeneous as it may be. The level of detail (i.e., the granularity of the summary graph) can be controlled by changing the labels that are created for nodes. The big disadvantage is represented by the need for preprocessing the whole RDF graph to the labeled graph, a process that requires the materialization of the whole dataset for many of the investigated aspects. The second limitation is that the created summaries are not RDF graphs themselves. These approaches are similar in principle with our approach in that they try to extract some kind of schema. The main difference between us and them is that very few of these summarization approaches are concentrating on RDF KBs and only one

of them [4] is capable of producing a guaranteed RDF schema as the result. Producing valid RDF schema as a summary allows us to use standard RDF tools (e.g. SPARQL) to query the summary. Our approach provides comparable or better results in most cases.

5.2 Approximate frequent pattern mining

The classical definition of frequent item set requires that all the items of each mined set actually occur in the supporting transactions. In order to deal with noisy and large databases, the common approach is to relax the notion of *support* of an item set by allowing missing items in the supporting transactions. Different approaches proposed different cost functions which are tackled with specific greedy strategies. Asso [15] is a greedy algorithm aimed at finding the pattern set Π_k that minimizes the amount of noise in describing the input data matrix $\mathcal{D}$. This is measured as the L^1-norm $\|\mathcal{N}\|$ (or Hamming norm), which simply counts the number of 1 bits in matrix $\mathcal{N}$. The Hyper+ [25] algorithm also tries to minimize the patterns cost $\|P_I\| + \|P_T\|$ in order to find a compact pattern set. Finally, in [16] an information theoretical approach is adopted, where the cost of the pattern set and of the noise is measured by their encoding cost in bits.

PaNDa$^+$ was shown to be more computationally efficient, able to extract high quality patterns both from binary and from graph data [13], and that such patterns can be successfully exploited for other data mining tasks, e.g., classification [14]. Differently from other algorithms, PaNDa$^+$ allows to tune the maximum allowed row-wise and column-wise missing items (noise) accepted in each pattern. For these reasons, we adopted PaNDa$^+$ a general approximate pattern mining tool.

6 Conclusions and Future Work

In this work we apply a top-k approximate graph pattern mining algorithm in order to extract a summary of an RDF KB. The summary is not necessarily the complete schema of the KB but it is the used/active schema of the KB, usually a subset of the original full schema, and always remains a valid RDF/S graph. Comparing it with the original RDF schema that was used while creating the KB, shows us that the summary presented by our system is very close to it, which in the specific examples we run means that the algorithm performs exceptionally well without relying on the existing schema information.

The work shows a lot of potential, so in the near future we plan to:

- perform experiments with bigger datasets, in order to explore the limits of the algorithms and design new more scalable solutions for the problem
- perform experiments with different parameters for the algorithms based on additional experiments or also parameters that will be guided by the data

- add the ability to capture user preferences and provide personalized summaries of the large RDF graphs based not only on size (how big or small a user requires the summary to be) but also based on intended use or based on the application
- provide theoretical proofs on the ability to always create summaries that are valid RDF schemas and can be queried by standard RDF machinery (e.g. SPARQL)
- investigate how we can update the RDF summaries based on the updates in the RDF KB.

Additionally we envision to apply the algorithm in a set of interlinked KBs where we can measure the actual benefits on the overall query performance improvement for a set of queries run over all the KBs. This would allow us to validate the original motivation of this work to its full extent.

References

1. Micah Adler and Michael Mitzenmacher. Towards compressing web graphs. In *Data Compression Conference, 2001. Proceedings. DCC 2001.*, pages 203–212. IEEE, 2001.
2. Charu C Aggarwal and Haixun Wang. *Managing and mining graph data*, volume 40. Springer, 2010.
3. Anas Alzogbi and Georg Lausen. Similar structures inside rdf-graphs. In *LDOW*, 2013.
4. 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.
5. François Goasdoué and Ioana Manolescu. Query-oriented summarization of rdf graphs. *Proceedings of the VLDB Endowment*, 8(12), 2015.
6. 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.
7. Shahan Khatchadourian and Mariano P Consens. Exploring rdf usage and interlinking in the linked open data cloud using explod. In *LDOW*, 2010.
8. Shahan Khatchadourian and Mariano P Consens. Understanding billions of triples with usage summaries. *Semantic Web Challenge*, 2011.
9. 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.
10. 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.
11. Amine Louati, Marie-Aude Aufaure, Yves Lechevallier, and France Chatenay-Malabry. Graph aggregation: Application to social networks. In *HDSDA*, pages 157–177, 2011.
12. Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Mining top-k patterns from binary datasets in presence of noise. In *SDM*, pages 165–176. SIAM, 2010.

13. Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. A unifying framework for mining approximate top-k binary patterns. *IEEE TKDE*, 26:2900–2913, 2014.
14. Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Supervised evaluation of top-k itemset mining algorithms. In *Big Data Analytics and Knowledge Discovery*, pages 82–94. Springer, 2015.
15. P. Miettinen, T. Mielikainen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. *IEEE TKDE*, 20(10):1348–1362, Oct. 2008.
16. Pauli Miettinen and Jilles Vreeken. Model order selection for boolean matrix factorization. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 51–59, 2011.
17. Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 419–432. ACM, 2008.
18. Sriram Raghavan and Hector Garcia-Molina. Representing web graphs. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 405–416. IEEE, 2003.
19. Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.
20. 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.
21. 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.
22. 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.
23. Yuanyuan Tian and Jignesh M Patel. Interactive graph summarization. In *Link Mining: Models, Algorithms, and Applications*, pages 389–409. Springer, 2010.
24. Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 965–973. ACM, 2011.
25. Yang Xiang, Ruoming Jin, David Fuhry, and Feodor F. Dragan. Summarizing transactional databases with overlapped hyperrectangles. *Data Min. Knowl. Discov.*, 23(2):215–251, September 2011.
26. Mohammed J. Zaki and Ching-Jui Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE TKDE*, 17(4):462–478, April 2005.
27. Haiwei Zhang, Yuanyuan Duan, Xiaojie Yuan, and Ying Zhang. Assg: adaptive structural summary for rdf graph data. ISWC, 2014.
28. 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.
29. Fang Zhou and Hannu Toivonen. *Methods for network abstraction.* PhD thesis, The Department of Computer Science at the University of Helsinki, 2012.