Scaling Up Schema Discovery for RDF Datasets

Redouane Bouhamoum, Kenza Kellou-Menouer, Zoubida Kedad and Stéphane Lopes
DAVID - University of Versailles Saint-Quentin-en-Yvelines
Versailles, France
firstName.lastName@uvsq.fr

***Abstract*—An increasing number of data sources is published on the Web, expressed using the languages proposed by the W3C such as RDF. In these sources, data is not constrained by a schema, which results in some sources having an incomplete schema, or even no schema at all; this makes the use of the data sources difficult. Some works have addressed the problem of automatic schema discovery but their scalability and their use in a big data context remains a challenge.**

In this work, we address this scalability issue, which is mainly related to the clustering algorithms at the core of schema discovery. In order to process large amounts of data, we propose to built a condensed representation of the initial dataset by extracting patterns representing all the existing combinations of properties. The clustering is then performed on the patterns instead of the initial dataset. In this paper, we describe our approach, and present its implementation using a big data technology. We also present some experimental evaluations performed on real datasets.

I. Introduction

Large amounts of data are made available on the Web, as more and more interlinked datasets are published. These datasets are described in languages such as RDF, which does not impose any constraint on the structure of the data: resources of the same type may have various property sets, and an entity may have several types. Moreover, the schema may be incomplete or even missing in the datasets. The lack of schema can limit the use of these data sources: for example, writing a query without the knowledge of the existing types and their properties is not obvious.

Some approaches have addressed the problem of the automatic extraction of a schema for RDF datasets to facilitate their exploitation. However, the scalability of these approaches and their use in a big data context remains a challenge.

In this paper, we address the problem of schema discovery for a massive RDF data source. Our goal is, given a large RDF dataset where the schema is missing, to ensure the scalability of a schema discovery approach. We rely on previous works on schema discovery which describe an approach using a density-based clustering algorithm (DBSCAN) to extract a schema from an RDF dataset [1], [2]; this approach provides a schema with a good quality, but the underlying algorithms are costly and not suitable for massive datasets.

In order to overcome this limitation, we propose to generate a condensed representation of a dataset, and to apply the clustering algorithm on this representation instead of the initial dataset. This condensed representation is composed of property patterns, each one corresponding to a combination of properties which exists in the initial dataset. Applying the clustering algorithm on this representation of a dataset is less costly than applying it on the whole dataset and gives exactly the same result.

Our proposal is a contribution towards the scalability of schema discovery. We have performed some experiments on several real datasets which have shown that using the condensed representation significantly improved the performances of schema discovery; we also propose a parallel execution of our approach, implemented using Spark, in order to further improve the performances.

In the remainder of this article, we present some related works in Section 2 and an overview of the schema discovery approach in section 3. We then present our approach for computing a condensed representation of an RDF datasets for schema discovery and its implementation in section 4. Experimental evaluations are presented in section 5, and we conclude the paper in section 6.

II. Related Work

Several works have focused on schema discovery for different purposes, such as summarizing datasets, indexing them, or querying the data.

Some approaches propose to discover the implicit schema of a dataset using clustering algorithms. These works do not require any schema-related information in the considered dataset and they generate the types from the entities of the dataset and their properties. The approach proposed in [1], [2] uses a density-based clustering algorithm (DBSCAN) for discovering the implicit types. Another proposal uses an ascending hierarchical clustering algorithm for type discovery [3]. Once discovered, types are annotated by the most frequent value of the *rdf:type* property. These approaches are unsuitable for massive datasets as they are costly. Furthermore, the implementation of the underlying algorithms using a big data technology is not straightforward.

Some works have provided approaches for summarizing an RDF dataset to make it easily readable and to support the formulation of queries. In [4], the proposed approach summarizes an RDF dataset by grouping the properties of the entities according to the similarity of their subjects or/and their objects; the summary is then refined by grouping resources of the same type. Some works have proposed a recommendation system to help users formulate complex Sparql queries based on a summary of the initial RDF graph [5]. A summary is provided by aggregating resources having the same type or the same set of properties. The summaries proposed in [4], [5] are based on *rdf:type* declarations but this property is not always specified in RDF datasets. If the type is not provided, the approach groups the entities that are structurally identical; such approach in not well-suited to the context of the semantic Web, as entities with different property sets may belong to the same type.

Some approaches were proposed in a big data context and were implemented with big data technologies such as Hadoop or Spark to process massive data. The approach presented in [6] groups entities that have the same type and describes the primitive type of the grouped entities with a regular expression. The approach presented in [7] proposes a strategy for inferring a versioned schema for NoSQL datasets. This approach considers that the different structures of entities which have the same type are versions of this type. These works rely on type declarations and are not suitable when the schema is incomplete or missing in a dataset, which may be the case in semantic Web data.

The survey of related works shows that there are two categories of approaches: (i) approaches which are able to manage big amounts of data but require some schema-related information, and therefore can not be used when this information is not provided in the dataset; and (ii) approaches which use costly clustering algorithms that can not be applied in a big data context. Our work is an attempt to fill the gap between these approaches by proposing a way to scale up clustering algorithms to massive datasets with incomplete or missing type information.

III. Schema Discovery

In our proposal, we rely on previous works on schema discovery for RDF data sources [1], [2]. This approach discovers the types contained in a data source and the links between them, without requiring any schema related information in the data source or any parameter provided by the user. Moreover, it allows to assign several types to an entity, and achieves a good quality for the discovered schema. In the following, we present an overview of this approach.

A. Description of the Data Source

Before presenting the general principle of the schema discovery approach, we define the notions of *data source* and of *entity*.

Data Source. An RDF data source (dataset) is defined as a set of triples $D \subseteq (R \cup B) \times P \times (R \cup B \cup L)$, where R, B, P and L represent resources, blank nodes (anonymous resources), properties and literals respectively.

Example: Figure 1 presents an RDF dataset storing information about a group of persons such as the name, the profession, the members of family, etc.

It's a graphical representation of the dataset where each node represents an entity which could be one of the followings:

- *Person* such as ":Margo" and ":Paul". The properties that could be associated to an entity representing a person are the name, the occupation, the relation with other people (wife, husband or child), the city where she lives and the car she drives;
- *City* such as ":Paris", which could be described by its name, the country and the region it belongs to;
- *Car* such as ":Golf" or ":Polo", which could be described by its name, its color, the year it has been produced ant its manufacturer;
- *Manufactory* such as ":Volkswagen", which could be described by a name;
- *Primary school* such as ":VersaillesPrimary", which is not described by any property in our example.

The objects represented by rectangles are the literals. The edges are directed and labeled by the name of the property they represent. An edge links either two resources, such as the property "LivesIn" between ":Paul" and ":Paris", or a resource and a literal, such as the property "WorksAs" between ":Paul" and "Doctor".

Entity. An entity e in a data source D is a resource described by a property set P where each property is annotated by an arrow indicating its direction, and such that:

- If $\exists (e, p, n) \in D$ then $\overrightarrow{p} \in P$;
- If $\exists (n, p, e) \in D$ then $\overleftarrow{p} \in P$.

Example: In our context, we consider that the properties describing an entity are the outgoing and incoming arcs linked to the same node. In Figure 1, *:Paul* is an entity which is described by the property set $\{\overrightarrow{Nm}$, $\overrightarrow{Wk}$, $\overrightarrow{DA}$, $\overrightarrow{HsH}$, $\overrightarrow{LIn}$, $\overrightarrow{HsC}$, $\overleftarrow{HsW}\}$.

B. Type Discovery

In order to infer the types from an RDF data source, a density-based clustering algorithm is used [8]. It has been extended to provide for each discovered type a profile composed of properties and their respective probabilities. This profile is used to discover overlapping types.

Density-based clustering. The approach uses a density-based clustering algorithm to discover the types, because it is robust to noise and deterministic. Furthermore, the number of classes is not required and the resulting clusters are of arbitrary shape. In [2], an extension of this algorithm is proposed to determine automatically the similarity threshold for grouping entities, according to the density distribution of the data. A cluster of similar entities corresponds to a type definition. The similarity between two given entities is evaluated using Jaccard similarity, considering the respective sets of both incoming and outgoing properties of the entities.

Type profiles. Each type is described by a profile, which is composed of the set of properties describing the type, and where each property is associated with a probability. The profile corresponding to a type T_i is denoted $TP_i = \{(p_{i1}, \alpha_{i1}), \ldots, (p_{in}, \alpha_{in})\}$, where each p_{ij} represents a property and where each α_{ij} represents the probability for an entity of T_i to have the property p_{ij}. The type profile represents the canonical structure of the type T_i.

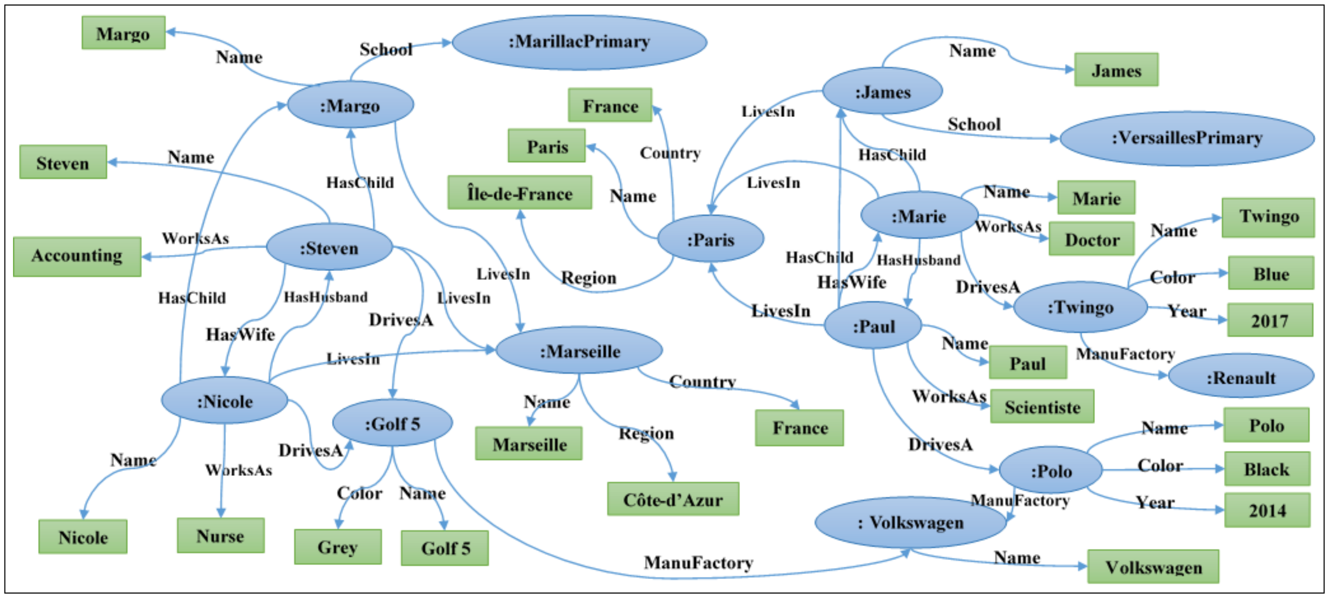

Fig. 1. An RDF Dataset describing people

Overlapping types. An important aspect of RDF data sources is that an entity may have several types. Density-based clustering provides disjoints clusters. Overlapping types are then discovered by analyzing the type profiles: considering that a *strong property* of a type is a property with a probability of 1 in the type profile, if an entity e is described by all the *strong properties* of a type, then this type could be assigned to e.

C. Link Discovery

Beside type definitions, links between these types are also important to understand the content of the data source. The approach presented in [1], [2] generates the following two types of links: *semantic links*, corresponding to user-defined properties and *hierarchical links*, corresponding to the *rdfs:subClassOf* property.

Semantic links. They represent properties that are not primitive properties. A primitive property is a property from the standard RDF or RDFS vocabularies. A link is generated by analyzing the type profiles. Two types T_i and T_j are linked by a property p if $\overrightarrow{p}$ belongs to the type profile TP_i and $\overleftarrow{p}$ belongs to the type profile TP_j. A generated link is checked in a data source D by finding two entities $e \in T_i$ and $e' \in T_j$ such that $(e, p, e') \in D$.

Hierarchical links. They represent the *rdfs:subClassOf* properties linking two types. A hierarchical clustering algorithm is used on the type profiles to define these links. It is adapted to process the type profiles by building a generic type profile at each step of the hierarchy. A similarity measure between two type profiles inspired from the Jaccard similarity is defined to consider the probabilities of properties.

D. Limitations

The approach described in this section allows to discover a schema with a good quality, however, it does not allow to process a massive data source because it uses a density-based clustering algorithm, which has a complexity of $o(n^2)$.

The use of big data technologies can enable massive data sources processing. However, density-based clustering is not suited for big data technologies. Indeed, the algorithm browses the entire data source as it requires a pairwise comparison of the entities. Therefore, partitioning the source over cloud nodes and merging the results are problems in their own rights.

IV. Building a condensed representation of an RDF dataset

Schema discovery for RDF datasets raises two important issues: (i) firstly, how to group the entities of the dataset to form classes, taking into account the high heterogeneity of the entities in the dataset in terms of properties and the lack of schema describing the data? (ii) Secondly, how to ensure that the schema discovery approach is capable of processing large datasets?

Our goal in this paper is to address the scalability issue of schema discovery approaches. We rely on the approach presented in section III which gives good quality results, but can not deal with massive datasets. We propose to transform a RDF dataset into a condensed representation and to apply the clustering algorithm on this representation instead of the initial dataset. Moreover, to improve the execution time of our approach, we propose a parallel version which we implemented with a big data technology.

A. Solution Overview

Considering an RDF dataset, we first built a concise representation consisting in all the patterns that represent the existing property combinations in the dataset. Then, the resulting schema is obtained by executing the clustering algorithm on the concise representation which reduces the number of inputs of the clustering step and therefore improves its speed.

Schema discovery approaches which use clustering algorithms are based on the structure of the entities as they evaluate the similarity according to the properties describing them. We propose to extract patterns from the entities described in a dataset. A pattern is defined as follows:

Pattern. A pattern is a set of distinct properties that describes one or more entities. A pattern therefore represents all the entities described by the same properties.

Extracting patterns from a dataset produces as output all the structures (or set of properties) that describe its entities. The clustering algorithm is then applied on the patterns instead of the entities to allow a faster execution while keeping the same quality for the resulting schema.

Example: The condensed representation of the dataset described in figure 1 is presented in table I, which shows the extracted patterns and the corresponding entities.

TABLE I
THE EXTRACTED PATTERNS

Entities	**Pattern**
":Marie", "Nicole"	P1 = ($\{\overrightarrow{Nm}$, $\overrightarrow{Wk}$, $\overrightarrow{DA}$, $\overrightarrow{HsH}$, $\overrightarrow{LIn}$, $\overrightarrow{HsC}$, $\overleftarrow{HsW}\}$, 2)
":Paul", ":Steven"	$P2$ = ($\{\overrightarrow{Nm}$, $\overrightarrow{WAs}$, $\overrightarrow{DrA}$, $\overrightarrow{HasW}$, $\overrightarrow{LIn}$, $\overrightarrow{HasC}$, $\overleftarrow{HasH}\}$, 2)
":James", ":Margo"	$P3 = (\{\overrightarrow{Nm}, \overrightarrow{LIn}, \overrightarrow{Scl}, \overleftarrow{HsC}\}, 2)$
":Polo", ":Twingo", ":Golf"	$P4 = (\{\overrightarrow{Nm}, \overrightarrow{Clr}, \overrightarrow{Yr}, \overrightarrow{Mnf}, \overleftarrow{DA}\}, 3)$
":Renault"	$P5 = (\{\overleftarrow{Mnf}\}, 1)$
":Volkswagen"	$P6 = (\{\overrightarrow{Nm}, \overleftarrow{Mnf}\}, 1)$
":Paris", ":Marseille"	$P7 = (\{\overrightarrow{Nm}, \overrightarrow{Ctry}, \overrightarrow{Rg}, \overleftarrow{LIn}\}, 2)$
":VersaillesPrimary", ":MarillacPrimary"	$P8 = (\{\overleftarrow{Scl}\}, 2)$

We can notice that entities described by the same property set correspond to a pattern; for example the entities ':Marie' and ':Nicole' are represented by the pattern $P1$, the entities ':Twingo', ':Polo' and ':Golf' by the pattern $P4$, etc.

In order to build a schema describing the dataset from the generated patterns, entities have to be grouped according to the similarity of their property sets. After generating the patterns from the initial dataset, we use a density-based clustering algorithm (DBscan) on the set of patterns to group them according to their similarity. This algorithm requires two parameters, *minPts*, representing the minimum number of neighbors for an entity to generate a cluster, and *esp*, the similarity threshold. We have set $minPts = 1$ and $eps = 0.5$.

We evaluate the similarity between two patterns $P1$ and $P2$ described respectively by the property sets $prop(P1)$ and $prop(P2)$ using the Jaccard coefficient, which measures the similarity between finite sets. It is defined as the size of the intersection divided by the size of the union of the considered sets:

$$J(P1, P2) = \frac{|prop(P1) \cap prop(P2)|}{|prop(P1) \cup prop(P2)|}$$

The DBscan algorithm considers that elements with a number of neighbors lower than minPts represent noise. When executed on the set of patterns extracted from a dataset, the number of entities corresponding to each pattern is compared to minPts: if the total number of entities corresponding to a set of patterns is above minPts, a cluster is generated. Every generated cluster represent a class in the schema; the properties of this class are the properties of all the entities that form the cluster. Since a pattern represents one or more entities, the number of inputs for the clustering algorithm will be reduced by a significant factor which reduces its execution time. As the clustering is performed based on the structural similarity of the patterns, the result remains the same as if the algorithm were applied on the entities in the initial dataset.

Considering our running example given in Figure 1, the extracted classes, the patterns that form the classes and the properties of each class are shown in Figure 2.

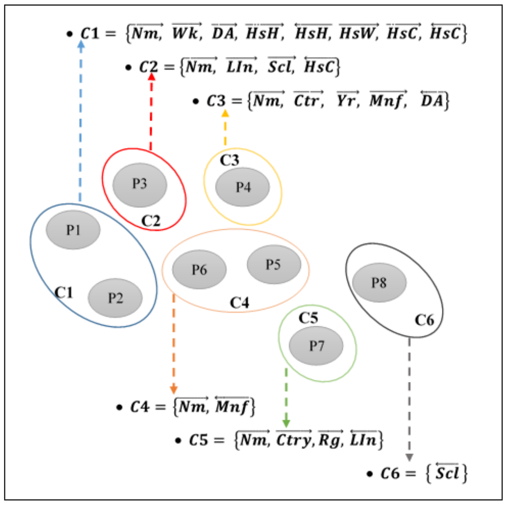

Fig. 2. The resulted clusters

The patterns $P1$ and $P2$ belong to the same cluster $(C1)$ because their similarity according to the Jaccard coefficient is above the similarity threshold eps=0.5, and the total number of entities associated to both patterns is equal to four, which is greater than minPts. The class formed by these two patterns is described by the following property set: $\{\overrightarrow{Nm}$, $\overrightarrow{Wk}$, $\overrightarrow{DA}$, $\overrightarrow{LIn}$, $\overrightarrow{HsC}$, $\overrightarrow{HsH}$, $\overleftarrow{HsH}$, $\overleftarrow{HsW}$, $\overrightarrow{HsW}\}$. The pattern $P3$ forms a cluster $(C2)$ because the number of entities represented by this pattern is superior to minPts.

B. Algorithm

The algorithm for pattern extraction takes as input the triples of the considered dataset and provides as output the set of patterns which represents a concise form of the initial dataset. The initial dataset is a set of triples, each one of the form <Subject Property Object>; a set of triples having the same subject represents a given entity, its properties and their values.

The algorithm extracts from every triple the subject and the properties (line 3-10), these properties are annotated as outgoing (line 5). If the object represents another entity, it will be considered as a subject characterized by the property, which is annotated as incoming (line 7-9). These two kinds of properties are used for link discovery once the classes have been generated. After grouping the properties describing the same entity (line 12), the resulting property set is stored in a distinct list and the algorithm counts the number of duplicates (line 19).

Algorithm 1 Patterns extraction

```
Require: file data
 1: $pattern : (Set(String) : propertySet, int : number)$
 2: //read the files
 3: for t:triplet in data do
 4:     if isPrimitif(getPredicat(t)) then
 5:         $entities \leftarrow entities + Array(getSubject(t), getPredicat(t) +' .out')$
 6:     end if
 7:     if isPrimitif(getPredicat(t)) and isNotLitteral( getObject(t)) then
 8:         $entities = entities + Array(getObject(t), getPredicat(t) +' .in')$
 9:     end if
10: end for
11: //Group all properties that belong to the same entity
12: $entityList \leftarrow entities.groupByEntityId()$
13: //Extract the properties set from the list of entities
14: for e:entity in entityList do
15:     $pattern \leftarrow pattern + (getPropertySet(e), 1)$
16: end for
17: //Count the number of entity for each pattern
18: $pattern \leftarrow pattern.countEntityNumber()$
19: return pattern
```

C. Implementation with Spark

For more efficiency, we have implemented the proposed approach with Spark, an open source distributed computing framework from the Apache Foundation. It was developed at the University of California at Berkeley by AMPLab. Spark overcomes some of the shortcomings of Hadoop MapReduce [9]. To ensure the reliability of the results, Hadoop MapReduce must write all the intermediate calculations in HDFS (Hadoop Distributed File System), which results in a low efficiency, while Spark saves the intermediate calculations in the volatile memory to improve the processing speed.

First, the dataset is split and saved in HDFS (distributed file system) to allow a parallel processing, as shown in figure 3.a. The first mapper reads the files in parallel, and retrieves the ID (subject) and properties of the entities (line 3-10); these properties will be annotated as outgoing and suffixed by ".Out" (line 5). Properties that are part of the RDF/RDFS or OWL vocabularies and used to provide schema information will be filtered. These properties are prefixed by rdf, rdfs or owl. Their detection is done by a regular expression in the "isPrimitive" operation. Two situations may occur when processing an object. If it represents a literal (a value), it will be ignored. If it represents a resource, the object will be considered as an entity, its ID will be added to the list of entities and the property will be considered as and incoming property for this entity, suffixed by ".In" (line 7-9). This mapper generates pairs of the form (entityID, property). All the properties for the same entity are then grouped together (line 12). To do so, Spark requires a shuffle so that all pairs with the same key (entityID) will be sent to the same node; the reducer then groups the properties of the same entity together and generates couples (e1, {p1,p2,p3, ...}). Figure 3.b shows the pairs generated at this stage.

The next step is the extraction of patterns (line 15-17), where the inputs are the pairs (entity,{properties}) and the outputs are the pairs ({properties}, nbr), where nbr is the number of entities described by the pattern. To this end, a second mapper reads the input of the reducer and produces the pairs (pattern, 1), the pattern being the set of properties for an entity (see figure 3.c); the number 1 indicates that one entity corresponding to this pattern was found. A second reducer then groups the pairs having the same key and counts their number (line 19). At the end of this step, the list of patterns and the number of entities for each one is obtained. Figure 3.d illustrates the result.

V. Experimental Evaluation

This section presents the results of our experiments with our pattern extraction approach with respect to the size reduction rate and the execution time. These tests were performed on a Spark cluster with 6 nodes, 80 core and 213 GB of RAM.

We have performed our experiments on the following real RDF datasets: (i) DBpedia[1] which is extracted from Wikipedia; (ii) DBLP[2] which contains the metadata of more than 1.8 million publications; (iii) Katrina and (iv) Charley[3] which contain descriptions of hurricanes and blizzard sightings in the United States.

Table II shows, for each dataset, the number of triples, the number of entities, the number of extracted patterns and the size reduction rates, computed as the number of extracted patterns divided by the number of entities in the initial dataset.

TABLE II
Size Reduction Rate

Dataset	Triples	Entities	Patterns	Ratio (%)
DBpedia	9 500 000 000	66 195 296	1 918 480	2.89
DBLP	222 375 855	16 086 516	351	0.002
Katrina	203 386 049	3 409	37	1.08
Charley	101 956 760	3 353	52	1.55

The number of output patterns depends on the heterogeneity of entities in the dataset. These entities are not constrained by

[1]https://old.datahub.io/dataset/dbpedia
[2]https://old.datahub.io/dataset/dblp
[3]http://wiki.knoesis.org/index.php/LinkedSensorData

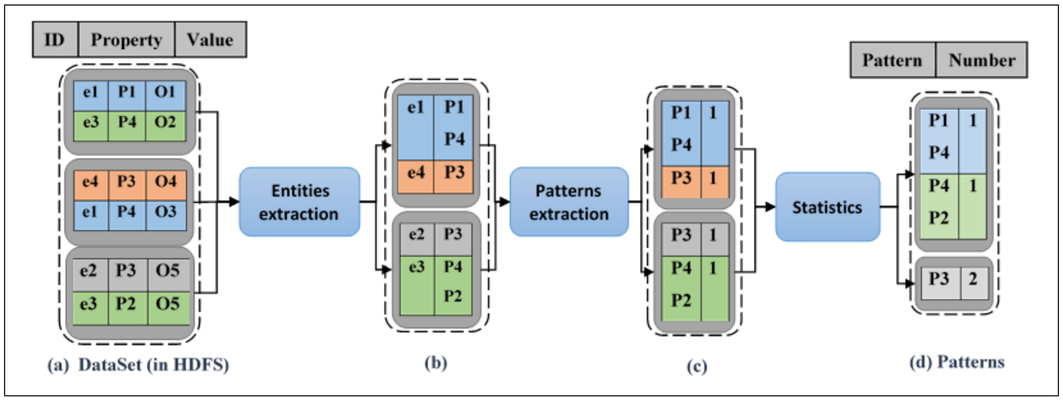

Fig. 3. Size reducing process

a schema. The more heterogeneous the entities, the higher the number of patterns. If we consider DBpedia, a open data source which is very heterogeneous, the number of patterns is high. Data sources such as DBLP, Katrina and Charley are used by scientific communities and and they are less heterogeneous, which results in fewer patterns.

Table III details the execution times of each step of the algorithm, which are entity extraction (Entity), pattern extraction (Pattern) and statistics computation (Stats).

TABLE III
EXECUTION TIMES

Dataset	Total (s)	Entity (s)	Pattern (s)	Stats (s)
DBpedia	750	590	130	30
DBLP	163	120	40	3
Katrina	100	96	3	1
Charley	50	46	3	1

The execution time depends on the number of triples, the number of entities and the number of patterns. The entity extraction phase reads in parallel the triples and writes the resulting entities so as to group pairs with the same ID: the larger the number of triples and entities, the higher the reading and writing time. This operation is a pre-processing step and is the most expensive. Since disk writing is an expensive operation, the processing time is longer for datasets which contains a high number of entities. The same holds for pattern extraction. The final operation reads the patterns and computes the number of entities having the same pattern.

The purpose of the condensed representation of a dataset is to enable the execution of a clustering algorithm and to provide a schema even when the size of the dataset does not allow the use of a clustering algorithm. Our tests show that for some datasets such as DBLP, Katrina and Charley, the number of extracted patterns allows the execution of the clustering algorithm. For DBpedia the clustering remains too costly due to the large number of patterns.

VI. CONCLUSION

We have proposed an approach for building condensed representations for massive RDF data sources in order to discover their schema. Our main contribution is to extract the structural patterns of its entities, then apply a density-based clustering algorithm on the patterns instead of the entities of the initial dataset. Indeed, schema discovery on entities gives exactly the same result as schema discovery on entity patterns. Since patterns are a representation of all existing structures in the dataset, our contribution would optimize any proposed approach for schema discovery based on the structure of the entities and achieve the same result, with the same quality. Our algorithm is implemented with Spark and the experiments performed on real data sources show that it reduces considerably their size in a reasonable time.

This work is a first contribution towards the scalability of schema discovery; we are currently working on a scalable and deterministic version of density-based clustering implemented using big data technologies. This will enable us to process data sources having a high level of heterogeneity which results in a large set of patterns.

ACKNOWLEDGMENT

This work is partially funded by the French National Research Agency (CAIR ANR-14-CE23-0006 project).

REFERENCES

[1] K. Kellou-Menouer and Z. Kedad, “Schema discovery in RDF data sources,” in *Conceptual Modeling - 34th International Conference, ER 2015*. Springer, pp. 481–495.
[2] K. Kellou-Menouer and Z. Kedad, “A self-adaptive and incremental approach for data profiling in the semantic web,” *T. Large-Scale Data- and Knowledge-Centered Systems*, vol. 29, pp. 108–133, 2016.
[3] K. Christodoulou, N. W. Paton, and A. A. Fernandes, “Structure inference for linked data sources using clustering,” *EDBT/ICDT*, 2013.
[4] S. Cebiric, F. Goasdou, and I. Manoles, “Queryoriented summarization of rdf graphs,” *VLDB*, vol. 8, 2015.
[5] S. Campina, T. E. Perry, D. Ceccarelli, R. Delbru, and G. Tummarello, “Introducing rdf graph summary with application to assisted sparql,” *Workshop on Database and Expert Sytems Applications*, 2012.
[6] M.-A. Baazizi, H. B. Lahmar, D. Colazzo, G. Ghelli, and C. Sartiani, “Schema inference for massive json datasets,” *EDBT*, 2017.
[7] D. S. Ruiz, S. F. Morales, and J. G. Molina, “Inferring versioned schemas from nosql databases and its applications,” *ER*, 2015.
[8] M. Ester, H. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, 1996, pp. 226–231.
[9] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *OSDI*, 2004.