Schema Discovery in RDF Data Sources

Kenza Kellou-Menouer(✉) and Zoubida Kedad(✉)

PRISM - University of Versailles Saint-Quentin-en-Yvelines, Versailles, France
{kenza.menouer,zoubida.kedad}@prism.uvsq.fr

Abstract. The Web has become a huge information space consisting of interlinked datasets, enabling the design of new applications. The meaningful usage of these datasets is a challenge, as it requires some knowledge about their content such as their types and properties. In this paper, we present an automatic approach for schema discovery in RDF(S)/OWL datasets.

We consider a schema as a set of type and link definitions. Our contribution is twofold: (i) generating the types describing a dataset, along with a description for each of them called type profile; (ii) generating the semantic links between types as well as the hierarchical links through the analysis of type profiles. Our approach relies on a density-based clustering algorithm and it does not require any schema-related information in the dataset. We have implemented the proposed algorithms and we present some evaluation results showing the effectiveness of our approach.

Keywords: Schema extraction · Clustering · Semantic Web

1 Introduction

The Web has evolved into a huge global information space, consisting of interlinked datasets, expressed in the Semantic Web standard languages such as RDF, RDFS or OWL. An unprecedented amount of data is made available, enabling the design of new applications in many domains. But if access to these datasets is provided, their meaningful usage is still a challenge, because it requires some knowledge about their content. For example, a user willing to query a dataset needs to know some of the properties and types existing in this dataset. This information could be obtained by randomly browsing the data, but this would be a tedious task. Writing the query would be straightforward with a schematic description of the dataset. This schema would be useful for other purposes, such as creating links between datasets; some interlinking tools have been proposed, such as Silk[1], which was used to link Yago [14] to DBpedia [1], but they require type and property information about the datasets to generate the appropriate *owl:sameAs* links.

An RDF(S)/OWL dataset may provide some information about its schema, such as *rdf:type*, specifying the type of a given entity, but this information is

[1] Silk: wifo5-03.informatik.uni-mannheim.de/bizer/silk.

P. Johannesson et al. (Eds.): ER 2015, LNCS 9381, pp. 481–495, 2015.
DOI: 10.1007/978-3-319-25264-3_36

not always complete. Even when datasets are automatically extracted, such as DBpedia which is extracted from Wikipedia, type information can be missing. The experiments presented in [12] show that at most 63.7 % of the data have complete type declarations in DBpedia, and at most 53.3 % in YAGO. We argue that providing a schematic description of a dataset is essential for its meaningful exploitation. In the context of Web data, the notion of schema is understood as a guide, not as a strict representation to which the data must conform. Indeed, languages used to format data on the Web do not impose any constraint on its structure: two entities having the same type may have different properties. In addition, an entity may have several types.

The goal of our work is to extract the schema describing an RDF(S)/OWL dataset and we propose a deterministic and automatic approach for schema discovery. We consider a schema as a set of type and link definitions. Our contribution is twofold: (i) generating the types describing a dataset, along with a description for each of them called type profile, where each property is associated to a probability; (ii) generating the semantic links between types as well as the hierarchical links through the analysis of type profiles. Our approach relies on a density-based clustering algorithm, it does not require any schema related information and it can detect noisy instances. We have implemented the algorithms underlying our approach and we present some evaluation results using different datasets to show the quality of the resulting schema.

The paper is organized as follows. We introduce a formal definition of the problem in Sect. 2 and we give some definitions related to entity description in Sect. 3. In Sect. 4, we present the core algorithms of our schema discovery approach. Section 5 presents the generation of overlapping types and Sect. 6 is devoted to links generation. Our evaluation results are presented in Sect. 7. We discuss the related works in Sect. 8 and finally, Sect. 9 concludes the paper.

2 Problem Statement

Consider the sets R, B, P and L representing resources, blank nodes (anonymous resources), properties and literals respectively. A dataset described in RDF(S)/OWL is defined as a set of triples $D \subseteq (R \cup B) \times P \times (R \cup B \cup L)$. Graphically, a dataset is represented by a labeled directed graph G, where each node is a resource, a blank node or a literal and where each edge from a node e to another node e' labeled with the property p represents the triple (e, p, e') of the dataset D. In such RDF(S)/OWL graph, we define an entity as a node corresponding to either a resource or a blank node, that is, any node apart from the ones corresponding to literals.

Figure 1(a) shows an example of such dataset, related to conferences. We can see that some entities are described by the property *rdf:type*, defining the classes to which they belong, as it is the case for "UVSQ", defined as a university. For other entities, such as "WWW", this information is missing. Two entities having the same type are not necessarily described by the same properties, as we can see for "UVSQ" and "MIT" in our example, which are both associated to the "University" type, but unlike "UVSQ", "MIT" has a "website" property.

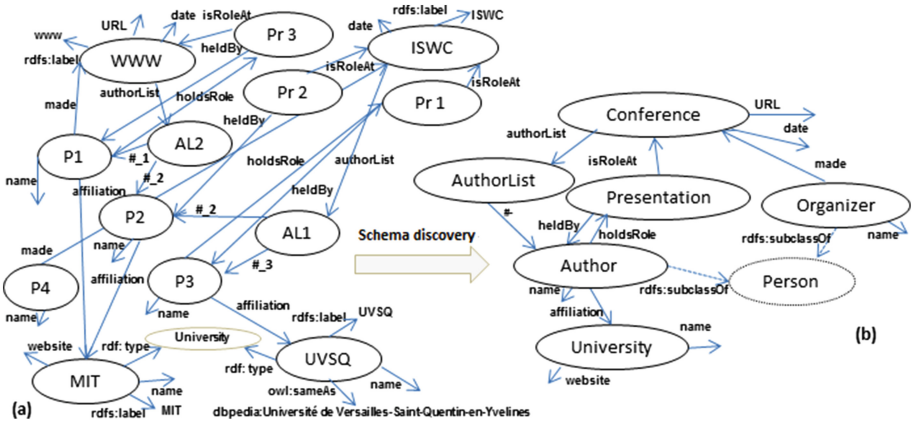

Fig. 1. Example of dataset and generated schema.

Our problem can be stated as follows: given an RDF(S)/OWL dataset with incomplete schema information, how to infer type definitions and links between these types? As a result, we provide an extracted schema defined as follows:

Definition. The extracted schema S of a dataset D is composed of:

- A set of possibly overlapping classes $C = \{C_1, ..., C_n\}$, where each class C_i corresponds to a set of entities in D and defines their type;
- A set of links $\{p_1, ..., p_m\}$, such that each p_i is a property for which both the range and the domain are two types corresponding to two classes in C;
- A set of hierarchical links involving two types corresponding to classes in C, expressing that one is the generic type of the other.

Table 1. Entity types in the conference dataset (Fig. 1(a)).

Types	Entities
"Conference"	"WWW", "ISWC"
"Presentation"	"Pr1", "Pr2"
"AuthorList"	"AL1" and "AL2"
"University"	"UVSQ", "MIT"
"Organizer"	"P1", "P4"
"Author"	"P1", "P2" and "P3"
"Person"	"P1", "P2", "P3" and "P4"

In order to define the schema describing a dataset, we first need to evaluate the similarity between entities and then group the similar ones into types,

knowing that entities having the same type could be described by heterogeneous properties and that a given entity may have several types. In the example given in Fig. 1(a), seven type definitions would be inferred (see Fig. 1(b)), the respective sets of entities corresponding to each type is given in Table 1. As we can see in the results, the classes are not necessarily disjoint: the entity "P1" has three types, namely "Author", "Organizer" and "Person".

Schema discovery also requires the identification of links between types on the basis of existing properties between entities, given the heterogeneity of the corresponding property sets. In our example, as entities "P4" and "ISWC" are related through the "made" property, there is a link labeled "made" between "Organizer" and "Conference" in the resulting schema. Finally, another problem we are faced with is the one of identifying types generalizing other types, which could be represented by *rdfs:subClassOf* properties, such as the ones between the generic type "Person" and its specific types "Author" and "Organizer".

3 Entity Description

We consider that an entity is described by different kinds of properties. Some of them are part of the RDF(S)/OWL vocabularies, and we will refer to them as primitive properties, and others are user-defined. We distinguish between these two kinds because all the properties should not be used for the same purpose during type discovery. Some predefined properties may provide information about the schema of the dataset and could be applied to any entity, therefore they should not be considered when evaluating the similarity between entities. Some of these properties can be used to validate the resulting schema: for example, if two entities e and e' are linked with the *owl:sameAs* primitive property, then we can check that the types inferred for e and e' are the same. We define the description of an entity as follows.

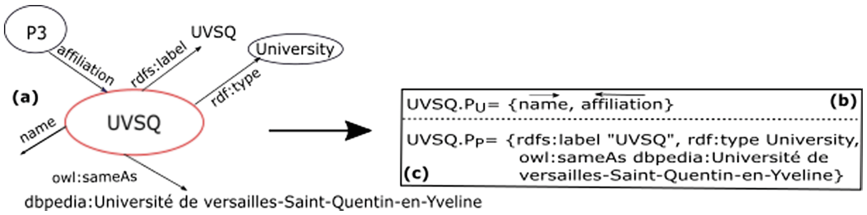

Fig. 2. Example of entity description.

Definition. Given the set of primitive properties P_P and the set of user-defined properties P_U in the dataset D, an entity e is described by:

1. A user-defined property set $e.P_U$ composed of properties p_u from P_U, each one annotated by an arrow indicating its direction, and such that:

– If $\exists(e, p_u, e') \in D$ then $\overrightarrow{p_u} \in e.P_U$;
– If $\exists(e', p_u, e) \in D$ then $\overleftarrow{p_u} \in e.P_U$.

2. A primitive property set $e.P_P$, composed of properties p_p from P_P with their values, such that:

– If $\exists(e, p_p, e') \in D$ then $p_p \in e.P_P$.

Figure 2 shows an example of entity description. In order to infer type definitions, entities are compared according to their structure. Our type discovery algorithm takes as input the set $D_U = \{e_i.P_U : i = 1, ...n\}$, where $e_i.P_U$ represents the set of user-defined properties describing the entity e_i. The set $D_P = \{e_i.P_P : i = 1, ...n\}$, where $e_i.P_P$ represents the set of primitive properties describing the entity e_i, can be used to validate the results of the algorithm and to specify type checking rules. This validation process is out of the scope of this paper.

4 Generating Types

Our requirements for type discovery are as follows: (i) the number of types in the dataset is not known, (ii) an entity can have several types, and (iii) the datasets may contain noise. The most suitable grouping approach is density-based clustering [3] because it is robust to noise, deterministic and it finds classes of arbitrary shape, which is useful for datasets where entities are described with heterogeneous property sets. In addition, unlike the algorithms based on k-means and k-medoid, the number of classes is not required.

Algorithm 1. Density-based Clustering with Type Profile

Require: D_U, ε, $MinPts$
  $setOfTypeProfile \leftarrow \emptyset$
  $class \leftarrow 0$
  while $\exists$ "no marked" $e.P_U \in D_U$ **do**
    mark $e.P_U$
    $setOfNeighbor \leftarrow$ FindNeighbor(D_U, $e.P_U$, ε)
    if $|setOfNeighbor| \geq MinPts$ **then**
      $class++$;
      $setOfTypeProfile \leftarrow setOfTypeProfile \cup$ ExpandCluster(D_U, $e.P_U$, $MinPts$, ε, $class$, $setOfNeighbor$)
    end if
  end while
  return $setOfTypeProfile$, typed D_U

Our density-based algorithm has two parameters: ε, representing the minimum similarity value for two entities to be considered as neighbors, and $MinPts$, representing the minimum number of entities in the neighborhood required for

an entity to be a core [3] and to generate a type; $MinPts$ is used to exclude the outliers and the noise. We use Jaccard similarity to compute the closeness between two property sets $e.P_U$ and $e'.P_U$ describing respectively two entities e and e'.

Beside the set of types, we provide a description of each of them called a type profile. The profiles will be used to find links between types and to generate overlapping types, i.e. multiple types for each entity. A profile consists in a property vector where each property is associated to a probability. The profile corresponding to a type T_i is denoted $TP_i = ((p_{i1}, \alpha_{i1}), ..., (p_{in}, \alpha_{in}))$, 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}. It is evaluated as the number of entities in T_i having the property p_{ij} over the total number of entities in T_i. In Fig. 1, the profile of the "University" type is: $((\overrightarrow{name}, 1), (\overrightarrow{website}, 0.5), (\overleftarrow{affiliation}, 1))$. The probability of "website" is 0.5 because this property is defined for "MIT" but not for "UVSQ".

In order to group similar entities and build the profile of each class, we have adapted a density-based clustering algorithm (Algorithm 1); it uses the **FindNeighbors** function which returns for a given entity all the entities having a distance smaller than ε. For each new entity e such that the number of its neighbors is lower than $MinPts$, we expand the class of e using the **ExpandCluster** function, defined in Algorithm 2, which finds all entities belonging to the same class as the current entity e. Each time a new entity e belonging to the current class is found, the type profile of this class is updated using the **UpdateTypeProfile** function which adds the properties of e if they are not already in the profile and recomputes the probabilities.

5 Generating Overlapping Types

An important aspect of RDF(S)/OWL datasets is that an entity may have several types [12]. A fuzzy clustering algorithm such as FCM or EM could be used to assign several types to an entity. However, they require the number of classes, as their grouping criterion is the similarity between an entity and the center of each class. This parameter can not be defined in our context, we therefore propose to derive the set of disjoint classes first, and then to form overlapping or fuzzy classes by analyzing the type profiles.

Recall that the importance of a property for a given type is captured by the associated probability. Figure 3(a) shows three classes generated by our algorithm: $C_1 = \{P1\}$, $C_2 = \{P4\}$ and $C_3 = \{P2, P3\}$, described respectively by the following type profiles:

- $TP_1 = ((\overrightarrow{name}, 1), (\overrightarrow{made}, 1), (\overrightarrow{affiliation}, 1), (\overleftarrow{heldBy}, 1), (\overrightarrow{holdsRole}, 1), (\overleftarrow{\#\_}, 1))$
- $TP_2 = ((\overrightarrow{name}, 1), (\overrightarrow{made}, 1))$
- $TP_3 = ((\overrightarrow{affiliation}, 1), (\overleftarrow{heldBy}, 1), (\overrightarrow{holdsRole}, 1), (\overleftarrow{\#\_}, 1))$.

The entity "P1" has all the properties of the two types TP_2 and TP_3 for which the probability is 1, we can therefore assign the corresponding types to "P1".

Algorithm 2. Expand Cluster

Require: D_U, $e.P_U$, $MinPts$, ε, $class$, $setOfNeighbor$
  $nbEntitiesInClass \leftarrow 0$
  Type profile of the class $TP_{class} \leftarrow \emptyset$
  add $class$ to $e.P_U$ types
  UpdateTypeProfile(TP_{class}, $class$, $e.P_U$, $nbEntitiesInClass$)
  $nbEntitiesInClass{+}{+}$
  while $\exists\ e'.P_U \in setOfNeighbor$ **do**
    if "not marked"$e'.P_U$ **then**
      mark $e'.P_U$
      $setOfNeighbor' \leftarrow$ FindNeighbor(D_U, $e'.P_U$, ε)
      if $|setOfNeighbor'| \geq MinPts$ **then**
        $setOfNeighbor \leftarrow setOfNeighbor \cup setOfNeighbor'$
      end if
    end if
    add $class$ to e' types
    UpdateTypeProfile(TP_{class}, $class$, $e'.P_U$, $nbEntitiesInClass$)
    $nbEntitiesInClass{+}{+}$
  end while
  return TP_{class}

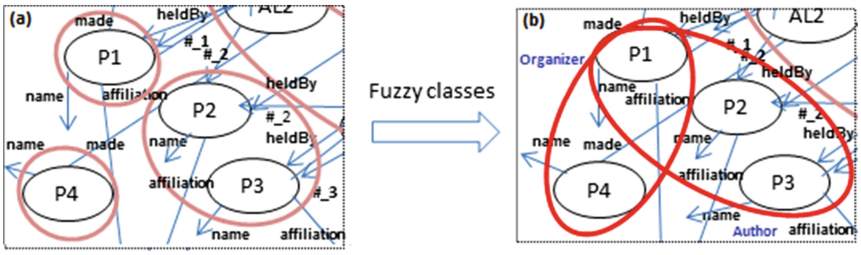

Fig. 3. Generating fuzzy classes

Fuzzy classes are generated using type profiles as follows; consider the type T_i described by the profile TP_i; if all the properties p of TP_i having a probability $\alpha = 1$ belong to another type profile TP_k, then the type T_i falls within the types of entities of the group k. This is expressed by the following rule:

– If $(\forall(p, \alpha)$ in TP_i, $\alpha = 1$: (p, α) in $TP_k)$ then (Add T_i to the types of the entities of class k).

In our example (Fig. 3), the cluster containing "P1" (class k here) is the intersection of the classes C_2 and C_3. Note that instead of comparing only the properties having a probability of 1, we could extend the comparison to the ones having a probability greater that a given threshold.

6 Generating Links

Beside type definitions, links between these types are also important to understand the content of the dataset at hand, as it is shown in Fig. 1(b). We are interested in two types of links: semantic links, corresponding to user-defined properties and hierarchical links, corresponding to the *rdfs:subClassOf* property.

Semantic links. Two types T_i, T_j are linked by a property p if $\overrightarrow{p}$ belongs to the properties of type profile TP_i and $\overleftarrow{p}$ belongs to the properties of type profile TP_j, as follow:

– If ($\exists$ p: $(\overrightarrow{p}, \alpha_i)$ in $TP_i \wedge (\overleftarrow{p}, \alpha_j)$ in TP_j) then (Add $\overrightarrow{p}$ from T_i to T_j).

These generated links are checked by finding two entities $e \in T_i$ and $e' \in T_j$ such that $(e, p, e') \in D$.

Algorithm 3. Hierarchical Links Generation

Require: $setOfTypeProfile$
  $setOfHierarchicalLinks \leftarrow \emptyset$
  while $|setOfTypeProfile| > 1$ **do**
    Find the most similar type profile and its *bestSimilarity*
    if $bestSimilarity = 0$ **then**
      Group all of the rest of type profile into the Generic type *Thing* and **STOP**
    else
      Construct the type profile of the Generic type that groups the most similar type profile
      $setOfHierarchicalLinks \leftarrow setOfHierarchicalLinks \cup$ {"rdfs:subClassOf" links between these most similar types profile and the Generic type}
      $setOfTypeProfile \leftarrow setOfTypeProfile \cup$ {type profile of the Generic type}

      Remove these most similar types profile from the $setOfTypeProfile$
    end if
  end while
  return $setOfHierarchicalLinks$

Hierarchical links. Two types T_i, T_j can be linked by a hierarchical property (*rdfs:subClassOf*), such as "Organizer" and "Person" in Fig. 1(b). We can use a hierarchical clustering algorithm on the entire dataset to generate these links but this would be very costly; to find the best partition, all the generated hierarchy has to be explored. In addition, the result would consist in many hierarchical links, not all of them being meaningful. We generate instead the hierarchy from the type profiles, which is less expensive because the number of profiles is small compared to the size of the entire dataset. Our procedure is given in Algorithm 3. We have adapted an ascending hierarchical clustering algorithm; the profile of the generic type is built at each step of the hierarchy. We define a similarity measure between

two type profiles TP_i, TP_j, inspired from the Jaccard similarity and based on the probability of a property p_k for two classes Ci and Cj. It is defined as follows:

$$ProfileSim(TP_i, TP_j) = \frac{\sum_{\forall p_k \in \{TP_i \cap TP_j\}} Prob_{i,j}(p_k)}{\sum_{\forall p_k \in \{TP_i \cup TP_j\}} Prob_{i,j}(p_k)} \quad (1)$$

where:

$$Prob_{i,j}(p_k) = \frac{\alpha_{ik} \times |C_i| + \alpha_{jk} \times |C_j|}{|C_i| + |C_j|} \quad (2)$$

A generic type is defined from the two most similar type profiles at each level of the hierarchy. The corresponding profile is composed of all the properties of the two types, the probability of a property is calculated as in (2).

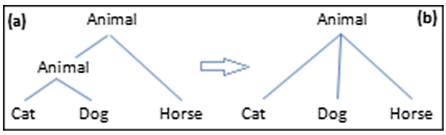

Fig. 4. Eliminating generic type redundancy in a hierarchy.

This approach generates a hierarchy by grouping types in pairs to find their generic type. However, some generic types can be composed of more than two sub-types as in Fig. 4(a). In this case, the user can detect easily intermediate redundant generic types and eliminate them (as in Fig. 4(b)), since the number of types in a dataset is generally not very high.

7 Evaluation

This section presents some experimentation results using our approach. We have evaluated the quality of the generated schema using well-known information retrieval metrics in different real datasets, described in the following section.

7.1 Datasets

For our experiments, we have used three datasets: the Conference[2] dataset, which exposes data for several Semantic Web conferences and workshops with 1430 triples; the BNF[3] dataset which contains data about the French National Library (Bibliothèque Nationale de France) with 381 triples and a dataset extracted from DBpedia[4] with 19696 triples considering the following types: Politician, SoccerPlayer, Museum, Movie, Book and Country.

[2] Conference: data.semanticweb.org/dumps/conferences/dc-2010-complete.rdf.
[3] BNF: datahub.io/fr/dataset/data-bnf-fr.
[4] DBpedia: dbpedia.org.

7.2 Metrics and Experimental Methodology

In order to evaluate the quality of the results provided by our algorithms, we have extracted the existing type definitions from our datasets and considered them as a gold standard. We have then run our algorithm on the datasets without the type definitions and evaluated the precision and recall for the inferred types. We have annotated each inferred class C_i with the most frequent type label associated to its entities. For each type label L_i corresponding to type T_i in the dataset and each class C_i inferred by our algorithm, such that L_i is the label of C_i, we have evaluated the precision $P_i(T_i, C_i) = |T_i \cap C_i|/|C_i|$ and the recall $R_i(T_i, C_i) = |T_i \cap C_i|/|T_i|$. We have set $MinPts = 1$ so that an entity is considered as noise if it has no neighbors. For the Conference dataset, we have empirically set $\varepsilon = 0.75$, which leads to very homogeneous classes. We have empirically stated that $\varepsilon = 0.72$ provides a number of classes equal to the number of types initially declared in the BNF dataset; the value is $\varepsilon = 0.5$ for the DBpedia dataset. Note that the determination of the similarity threshold ε is an open issue for clustering algorithms [3,13].

To provide the overall quality of types, semantic and hierarchical links, we have used the precision and recall metrics. The number of generated classes is denoted k, and the number of entities in the dataset, n. To assess the overall type quality, each type is weighted according to its number of entities as follows:

$$P = \sum_{i=1}^{k} \frac{|C_i|}{n} \times P_i(T_i, C_i) \qquad R = \sum_{i=1}^{k} \frac{|C_i|}{n} \times R_i(T_i, C_i)$$

Precision and recall of the generated links are evaluated considering the true/false positives and the false negatives. We have compared the inferred schema to the one of the dataset when it was provided, as for the BNF dataset. If no schema was provided, we have manually designed it based on the information provided in the dataset; to this end, we have built the set D_P as defined in Sect. 3.

7.3 Results

The quality of each discovered type in the Conference dataset is shown in Fig. 5(a). Our approach gives good precision and recall and detects types which were not declared in the dataset: classes 6, 10, 11 and 12 are manually labeled "AuthorList", "PublicationPage", "HomePage" and "City" respectively.

In some cases, types have been inferred relying on incoming properties only. Indeed, for containers, such as "AuthorList", it is necessary to consider these properties as they do not have any outgoing property. Classes 1 and 7 do not have a good precision because they contain entities with different types in the dataset. However, these types have the same structure, it is therefore impossible to distinguish between them. The recall for "Person" is not good because it is split into three classes: class 8 represents persons who have both published and played a role in the conference; class 2 represents persons who have only played a role (e.g. Chair, Committee Member); class 5 represents persons who have only published. Overlapping types are generated based on the analysis of type profiles. This has led to the results shown in Fig. 5(b). Class 8 is associated

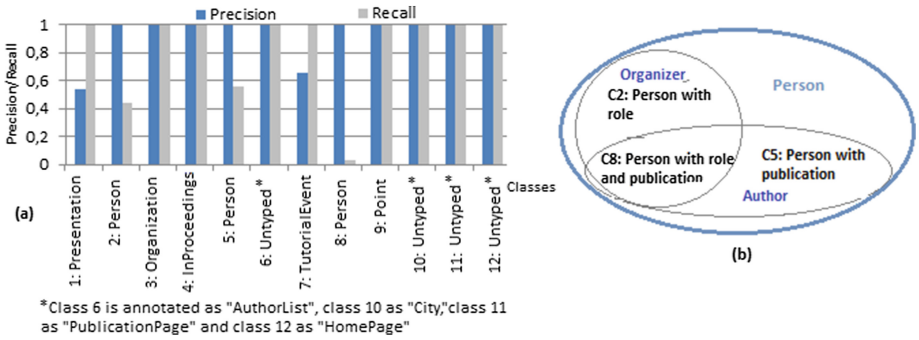

Fig. 5. Quality of the generated types (a) and overlapping types (b) in the conference dataset.

to two types: the one of class 2 (manually labeled "Organizer") and the one of class 5 (manually labeled "Author"), which indeed conforms to the entities of the dataset. Note that finding the labels of classes is an open problem that we will address in future works.

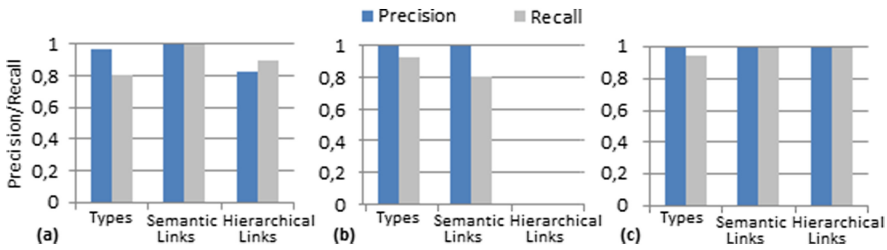

Fig. 6. Evaluation of schema discovery in conference (a) BNF (b) and DBpedia (c).

We can see in Fig. 6 that the approach gives good precision and recall for the generated schema, composed of types, semantic and hierarchical links. For the Conference dataset (see Fig. 6(a)) the precision is not maximum because of classes 1 and 7 as discussed before. The recall is not maximum because the "Person" type is split in three as discussed above. The results for the BNF (see Fig. 6(b)) and DBpedia datasets (see Fig. 6(c)) show that the assignment of types to entities has achieved good precision and recall. The recall is not maximum because noisy instances were detected. For the BNF dataset, some of the semantic links were not declared in the provided schema, which is why the recall is not maximum. However, after checking the entities of the dataset, we found out that these semantic links were valid. For the DBpedia dataset, our algorithm was able to differentiate between entities of the two types "Politician" and "SoccerPlayer" even if they have similar property sets, as it is shown by the corresponding type profiles generated by our algorithm and presented below.

The generated type profiles reflect the heterogeneity of the dataset, e.g. 6 % of the entities of "SoccerPlayer" have a "*deathDate*" outgoing property, and yet the generated grouping was good. The results achieved by our approach are good even when the dataset is heterogeneous.

- Politician: < ($\vec{name}$, 1), ($\vec{party}$, 0.73), ($\vec{children}$, 0.21), ($\vec{birthDate}$, 0.94), ($\vec{nationality}$, 0.15), ($\overleftarrow{successor}$, 0.78), ($\vec{deathDate}$, 0.68), ...>.
- SoccerPlayer: < ($\vec{name}$, 1), ($\vec{height}$, 0.46), ($\vec{surname}$, 0.93), ($\vec{birthDate}$, 1), ($\vec{nationalteam}$, 0.86), ($\overleftarrow{currentMember}$, 0.8), ($\vec{deathDate}$, 0.06), ...>.

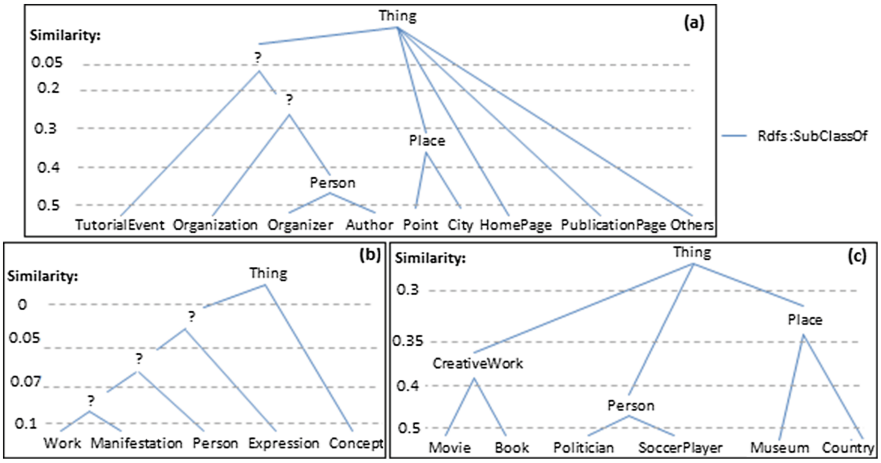

Fig. 7. Hierarchical links generation for conference (a), BNF (b) and DBpedia (c).

The hierarchical links generated for DBpedia are correct as they conform to the existing *rdfs:subClassOf* declarations (see Fig. 7(c)). The BNF dataset has no hierarchical links, the values of both precision and recall are therefore null. When the similarity between two profiles is low, the semantic of the generic type is unclear. This is represented by a question mark in Fig. 7(b). It is the same for some of the hierarchical links generated for the Conference dataset (see Fig. 7(a)): a generic type has been generated for "Person" and "Organization", however, the similarity between their type profiles is low. Our algorithm could not identify a generic type for "HomePage" and "PublicationPage" because their type profiles do not share any property, which explains the recall.

8 Related Works

Schema discovery from semi-structured data has been addressed by some research works. In [16], an approximate DataGuide based on the COBWEB hierarchical

clustering algorithm is proposed. The incoming/outgoing edges are considered in the same way, which could be a problem in RDF datasets as it will not differentiate between the domain and the range of properties. The resulting classes are disjoint, and the approach is not deterministic as it is based on COBWEB. In [8], several types are inferred for an entity, but only when a type is more general than another, such as "Employee" and a "Person". The approach distinguishes between incoming and outgoing edges: incoming edges are considered as roles, and potential labels for the inferred types. This is suitable for the OEM [11] model used in the approach, but not to RDF, where incoming edges do not necessarily reflect the type of an entity. The proposed algorithm requires the threshold of the jump which is not easy to define as it depends on the regularity of the data; this parameter is not known in our context. The approach presented in [9] uses bottom-up grouping providing a set of disjoint classes. Unlike in our approach, the number of classes is required. In [2], standard ascending hierarchical clustering is used to build structural summaries of linked data. Each instance is represented by its outgoing properties and the property set of a class is the union of the properties of its entities, while in our approach, the probability of each property is computed for a type. The algorithm provides disjoint classes; the hierarchical clustering tree is explored to assess the best cutoff level, which can be costly. SchemEX [6] finds the relevant data sources for a query by building an RDF triple index. Unlike in our approach, *rdf:type* declarations are required to find classes. The approach in [5] adds structural information to a database from a set of available databases. It searches for a similar database among the set of existing ones in order to make design decisions.

Some works have addressed the problem of enriching an existing schema by adding more structure through RDF(S)/OWL primitives. SDType [12] enriches an entity by several types using inference rules, provided that these types exist in the dataset, and it computes the confidence of each type for an entity. The focus of the approach is therefore on the evaluation of the relevance of the assigned types. In addition, *rdfs:domain*, *rdfs:range* and *rdfs:subClassOf* properties are required. Works in [4,10] infer types for DBpedia only: [10] uses K-NN and [4] finds the most appropriate type for an entity in DBpedia based on descriptions from Wikipedia and links with WordNet and the Dolce ontology. The statistical schema induction approach [15] enriches an RDF dataset with some RDFS/OWL primitives, however type information must be provided. The approach in [17] uses an ascending hierarchical clustering to form classes by exploiting the existing *rdf:type* declarations. The approach finds hierarchical links, but is specific to the bio-medical ontologies DrugBank and Diseasome. In [7], a reverse engineering method dealing with the derivation of inheritance links embedded in a relational database is presented. Decision rules for detecting existence dependencies and translating them into hierarchies among entities are defined.

9 Conclusion

We have proposed an approach for schema discovery in RDF(S)/OWL datasets. In order to generate several types for an entity, we have adapted a density-based

clustering algorithm. Each generated type is described by a profile, consisting of a property vector where each property is associated to a probability. These type profiles are used to generate overlapping types as well as semantic and hierarchical links. Our experiments show that our approach achieves good quality results regarding both types and links in the generated schema, even when the entities are very heterogeneous, such as in DBpedia. One important problem that we plan to address is the annotation of the inferred types. Indeed, in addition to identifying a cluster of entities having the same type, it is also useful to find the labels which best capture the semantics of this cluster.

Acknowledgements. This work was partially funded by the French National Research Agency through the CAIR ANR-14-CE23-0006 project.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: DBpedia: a nucleus for a Web of open data. In: Aberer, K., et al. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007)
2. Christodoulou, K., Paton, N.W., Fernandes, A.A.: Structure inference for linked data sources using clustering. In: EDBT/ICDT 2013 Workshops (2013)
3. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd (1996)
4. Gangemi, A., Nuzzolese, A.G., Presutti, V., Draicchio, F., Musetti, A., Ciancarini, P.: Automatic typing of DBpedia entities. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 65–81. Springer, Heidelberg (2012)
5. Klettke, M.: Reuse of database design decisions. In: Kouloumdjian, J., Roddick, J., Chen, P.P., Embley, D.W., Liddle, S.W. (eds.) ER Workshops 1999. LNCS, vol. 1727, pp. 213–224. Springer, Heidelberg (1999)
6. Konrath, M., Gottron, T., Staab, S., Scherp, A.: Schemex: efficient construction of a data catalogue by stream-based indexing of linked data. WWW **16**, 52–58 (2012)
7. Lammari, N., Comyn-Wattiau, I., Akoka, J.: Extracting generalization hierarchies from relational databases: a reverse engineering approach. Data Knowl. Eng. **63**(2), 568–589 (2007)
8. Nestorov, S., Abiteboul, S., Motwani, R.: Inferring structure in semistructured data. ACM SIGMOD Rec. **26**(4), 39–43 (1997)
9. Nestorov, S., Abiteboul, S., Motwani, R.: Extracting schema from semistructured data. ACM SIGMOD Rec. **27**, 295–306 (1998). ACM
10. Nuzzolese, A.G., Gangemi, A., Presutti, V., Ciancarini, P.: Type inference through the analysis of Wikipedia links. In: LDOW (2012)
11. Papakonstantinou, Y., Garcia-Molina, H., Widom, J.: Object exchange across heterogeneous information sources. In: Proceedings of the Eleventh International Conference on Data Engineering, pp. 251–260. IEEE (1995)
12. Paulheim, H., Bizer, C.: Type inference on noisy RDF data. In: Alani, H., et al. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 510–525. Springer, Heidelberg (2013)
13. Sánchez-Díaz, G., Martínez-Trinidad, J.F.: Determination of similarity threshold in clustering problems for large data sets. In: Sanfeliu, A., Ruiz-Shulcloper, J. (eds.) CIARP 2003. LNCS, vol. 2905, pp. 611–618. Springer, Heidelberg (2003)

14. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: Proceedings of the 16th International Conference on World Wide Web (2007)
15. Völker, J., Niepert, M.: Statistical schema induction. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part I. LNCS, vol. 6643, pp. 124–138. Springer, Heidelberg (2011)
16. Wang, Q.Y., Yu, J.X., Wong, K.-F.: Approximate graph schema extraction for semi-structured data. In: Zaniolo, C., Grust, T., Scholl, M.H., Lockemann, P.C. (eds.) EDBT 2000. LNCS, vol. 1777, pp. 302–316. Springer, Heidelberg (2000)
17. Zong, N., Im, D.-H., Yang, S., Namgoon, H., Kim, H.-G.: Dynamic generation of concepts hierarchies for knowledge discovering in bio-medical linked data sets. In: ICUIMC. ACM (2012)