Structure Inference for Linked Data Sources Using Clustering

Klitos Christodoulou(✉), Norman W. Paton, and Alvaro A.A. Fernandes

School of Computer Science, University of Manchester,
Oxford Road, Manchester M13 9PL, UK
{christodoulou,norm,alvaro}@cs.man.ac.uk

Abstract. Linked Data (LD) overlays the World Wide Web of documents with a Web of Data. This is becoming significant as shown in the growth of LD repositories available as part of the Linked Open Data (LOD) cloud. At the instance-level, LD sources use a combination of terms from various vocabularies, expressed as RDFS/OWL, to describe data and publish it to the Web. However, LD sources do not organise data to conform to a specific structure analogous to a relational schema; instead data can adhere to multiple vocabularies. Expressing SPARQL queries over LD sources – usually over a SPARQL endpoint that is presented to the user – requires knowledge of the predicates used so as to allow queries to express user requirements as graph patterns. Although LD provides low barriers to data publication using a single language (i.e., RDF), sources organise data with different structures and terminologies. This paper describes an approach to automatically derive structural summaries over instance-level data expressed as RDF triples. The technique builds on a hierarchical clustering algorithm that organises RDF instance-level data into groups that are then utilised to infer a structural summary over a LD source. The resulting structural summaries are expressed in the form of classes, properties and, relationships. Our experimental evaluation shows good results when applied to different types of LD sources.

Keywords: Schema · Linked Data · Clustering · Query formulation

1 Introduction

In recent years there has been a significant growth in the amount of publicly available structured data on the Web using a graph-based representation model and a set of simple principles, the so-called *Linked Data Principles* [3]. A motivation for the adoption of these principles is the fact that they are based upon established web infrastructures (like URIs and HTTP) and semantic web standards (like RDF and RDFS), thus providing low barriers to data publication. The adoption of these principles is apparent in the number of Linked Data (LD) repositories that form the Linked Open Data (LOD) cloud[1]. An interesting aspect of this kind of Web is that datasets are not only published in isolation

[1] http://lod-cloud.net.

but also interconnected with other datasets by the use of links. As the size of the LOD graph is constantly growing, with billion of triples publicly available from different domains, so does the need to consume data in this distributed environment.

Accessing the LOD cloud follows paradigms previously used for the web of documents. For example, various techniques from information retrieval are used by LD search engines (e.g., SWSE [10]) to support keyword queries. However, the web of data seems to provide the opportunity to move beyond keyword search queries into more precise query answering using structural queries. Often knowledge for answering a query is distributed across different datasets, so multiple queries need to be formulated and sent to more than one dataset for the user to get the desired answer. For the web of data this support is provided by SPARQL [18], the query language for RDF data sources. However, even though SPARQL supports querying over RDF sources it may still be difficult to formulate such queries. The basic building blocks for SPARQL queries require an understanding of how concepts are represented that may not be readily available. The fact that the RDF model [12] does not impose any constraints on the structure of a source makes it difficult to know what graph patterns can be formulated over a given source.

Evaluating structured queries over a LD repository that exposes its data as either a SPARQL endpoint or an RDF dump often requires the user to browse the source or to issue exploratory queries (e.g., Listing 1.1) in order to understand how the data are organised and what predicates are used to describe the entities. This, however, is time consuming and requires queries to be formed and asked manually. Although, it might be possible to browse a small RDF source, usually the knowledge as to how the data are organised in a source requires careful observation of the triples at the instance-level, which presents scalability challenges for browsing.

```
SELECT DISTINCT ?concept
WHERE {
      [ ] rdf:type ?concept .
    }

SELECT DISTINCT ?concept ?prop
WHERE {
     ?s rdf:type ?concept .
     ?s ?prop ?v .
   }
```

Listing 1.1. Exploratory SPARQL queries.

Generally, forming a meaningful query over a source is challenging without a structural summary of the underlying RDF source, therefore, it is important to have such an understanding. In relational databases, for instance, such an organisation of data is achieved using a logical schema to which data must adhere.

Such an organisation of data is not only useful for browsing the structure of the database but also for formulating queries or even capturing statistics that could enable query optimisation [6]. In contrast to the logical organisation imposed in relational databases, an RDF source does not conform to any analogous structure. In the context of LD, a schema of an RDF source is a combination of terms from various vocabularies that are used to represent the data, where their semantics are defined in various RDFS/OWL vocabularies [9].

Problem and Approach. It is often good practice for RDF datasets to provide a VoID[2] description that captures various metadata about a source. By retrieving such descriptions, users can get an idea about size statistics, which vocabularies are used in the source, which classes or predicates are used to describe the data, or how to access the source. Such descriptions however, lack sufficient structural metadata to provide a detailed description as to how the data is organised in a source. Typically, VoID descriptions are handcrafted by data publishers, and are not always available. As of August 2011, only 32.2 % of the LOD cloud data sources provided such descriptions[3]. In this paper we propose a technique based on cluster analysis that, by looking at instance-level data from an RDF source, can infer a structural summary (i.e., a schema) that captures information about *classes* that the data instantiate, their *properties* and how they relate to each other. Having such a structural summary over a source can be helpful in many application scenarios, such as for discovering sources, understanding the structure of a source, and supporting query formulation. Exploring such challenges is outside the scope of this paper.

Contributions. We note that the vision of distributed query processing over RDF sources can benefit from having structural summaries available for each source. At the same time we recognise that the low barriers to data publication introduced by the linked data principles must be preserved. We have proposed in our previous work [16] that pay-as-you-go data integration [5] can be used to enable distributed query processing over structurally heterogeneous and distributed LD sources. In such a context, there is a need to explore automatic techniques that support structure inference from RDF sources. With the work presented in this paper we take a step forward in this direction; our contributions are as follows:

– We have designed and implemented a technique that allows structural summaries to be inferred over RDF sources that contain explicitly stated instance-level data.
– We describe an experimental evaluation of the approach using different use-case scenarios that demonstrate its effectiveness.

In the remainder of this paper we begin by introducing a more formal definition of the problem in Sect. 2, followed by a detailed description of our technique in Sect. 3. We elaborate on the methodology used for evaluating the approach in Sect. 4, and we conclude in Sect. 6.

[2] See Vocabulary of Interlinked Datasets: http://www.w3.org/TR/void/.
[3] For more statistics, see http://www4.wiwiss.fu-berlin.de/lodcloud/state/.

2 Problem Description

By observing instance-level triples that are explicitly stated in a source, our aim is to have a synopsis of the sources. Following [1], we define an *RDF triple* as $(subject, predicate, object) \in (R \cup B) \times P \times (R \cup B \cup L)$, given a set of Resources R identified with URIs, a set of Blank Nodes B, a set of Predicates P and a set of Literals L. A set of RDF triples T forms an *RDF Graph*. Given T we are interested to derive a schema description as follows:

Definition 1. *A schema S is composed of a set of classes $\{C_1, ..., C_\mu\}$, where each $C_i (i = 1, ..., \mu)$ contains a set of predicates $\{C_i.P_1, ..., C_i.P_p\}$.*

As pointed out earlier, at the instance-level an RDF source does not conform to any specific structure that forces an organisation of the triples in the source and thus can be considered as schema-less. The data do not adhere to any explicit schema definition. The RDF model can describe resources with a mixture of terms from different vocabularies, where there is no restriction on the number of vocabularies or terms (i.e., predicates) used to describe a resource. Assume that the RDF graph T describes resources with the set of predicates P, where $|P| = J$. Then we define a description of a particular resource as a *Candidate Description*:

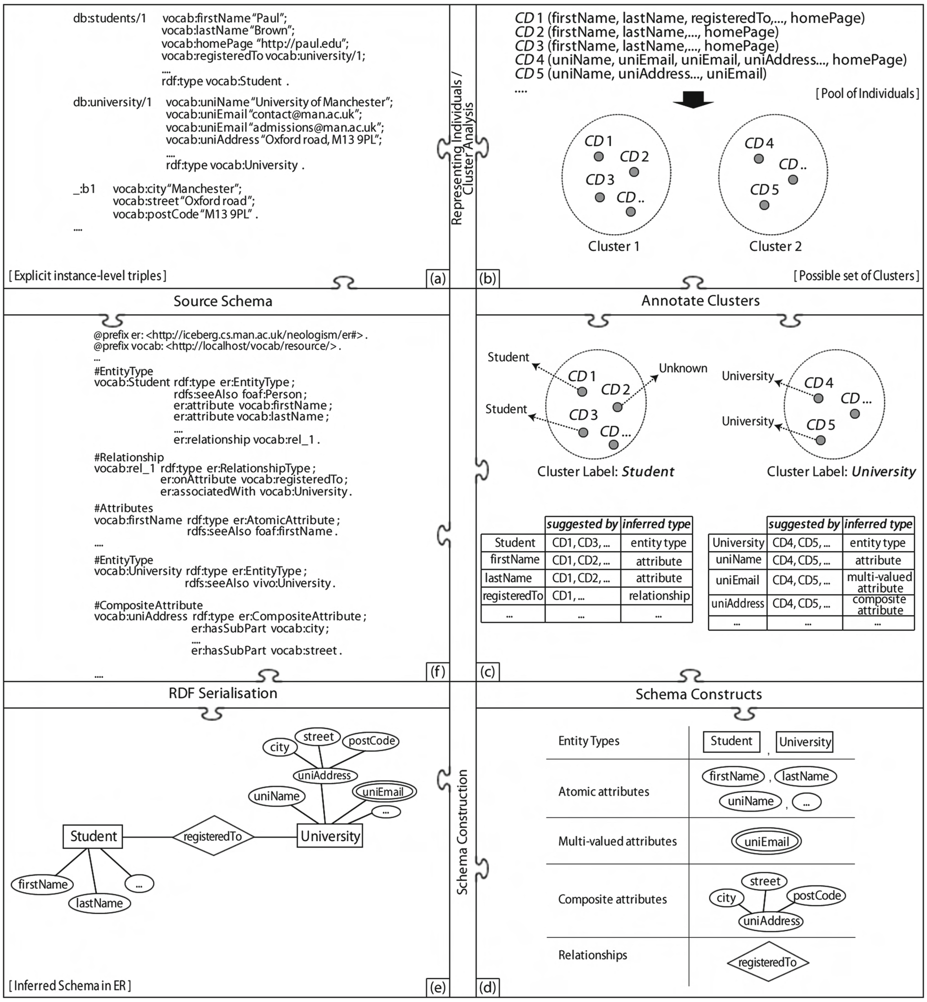
Definition 2. *A candidate description CD is a subset $P' \subseteq P$; the subset composed by a set of $j \leq J$ predicates of P.*

Given $\mathcal{CD}$, the set of all CDs, our target is to group (i.e., cluster) $\mathcal{CD}$ into k clusters, where k is computed through the algorithm described in Sect. 3.2. Our methodology will create a set of clusters $U = \{U_1, ..., U_k\}$, where $U_i \subseteq \mathcal{CD} \quad \forall \quad i = 1, ..., k$. Optimally all CD's in each U_i belong to the same *class*.

In brief, given an RDF graph, the technique looks for resource descriptions that are potential instances of the same *class* by looking at the predicates used to describe a resource and any RDF typing information available.

3 Method Description

In the following, we describe our technique for inferring a structural summary over an RDF source. Our approach is based on the assumption that we can identify recurring structural patterns of a possible concept by observing instance-level resource descriptions that are RDF triples. To detect such patterns we organise resource descriptions that are possible instantiations of a class together in groups using *cluster analysis*. Simply put, the idea of cluster analysis is to discover groups from a set of objects. These groups are known as *clusters* and a set of clusters is known as a *clustering*. The goal is to assign similar objects into the same cluster and separate them from clusters that contain objects which are not similar. Similarity between objects is measured by a distance function; more details on the clustering algorithm can be found on Sect. 3.2. From this point onwards we refer to objects as individuals and we represent them as candidate descriptions. Our technique uses a hierarchical clustering algorithm to detect groups among a pool of individuals that are *candidate descriptions* (as in Definition 2); each group identified as a result of the clustering algorithm can inform the identification of *classes* that instantiate data, their *properties* and their relationships to other classes. We use a toy example in Fig. 1 to describe our technique. The final inferred schema is represented by a simple Entity-Relationship (ER)


```
db:students/1     vocab:firstName "Paul";
                  vocab:lastName "Brown";
                  vocab:homePage "http://paul.edu";
                  vocab:registeredTo vocab:university/1;
                  ....
                  rdf:type vocab:Student .
db:university/1   vocab:uniName "University of Manchester";
                  vocab:uniEmail "contact@man.ac.uk";
                  vocab:uniEmail "admissions@man.ac.uk";
                  vocab:uniAddress "Oxford road, M13 9PL";
                  ....
                  rdf:type vocab:University .
_:b1     vocab:city "Manchester";
         vocab:street "Oxford road";
         vocab:postCode "M13 9PL" .
....
```

```
CD1 (firstName, lastName, registeredTo,..., homePage)
CD2 (firstName, lastName,..., homePage)
CD3 (firstName, lastName,..., homePage)
CD4 (uniName, uniEmail, uniEmail, uniAddress..., homePage)
CD5 (uniName, uniAddress..., uniEmail)
....
```

	suggested by	inferred type
Student	CD1, CD3, ...	entity type
firstName	CD1, CD2, ...	attribute
lastName	CD1, CD2, ...	attribute
registeredTo	CD1, ...	relationship
...	...	...

	suggested by	inferred type
University	CD4, CD5, ...	entity type
uniName	CD4, CD5, ...	attribute
uniEmail	CD4, CD5, ...	multi-valued attribute
uniAddress	CD4, CD5, ...	composite attribute
...	...	...

```
@prefix er: <http://iceberg.cs.man.ac.uk/neologism/er#> .
@prefix vocab: <http://localhost/vocab/resource/> .
...
#EntityType
vocab:Student rdf:type er:EntityType ;
              rdfs:seeAlso foaf:Person ;
              er:attribute vocab:firstName ;
              er:attribute vocab:lastName ;
              ....
              er:relationship vocab:rel_1 .
#Relationship
vocab:rel_1 rdf:type er:RelationshipType ;
            er:onAttribute vocab:registeredTo ;
            er:associatedWith vocab:University .
#Attributes
vocab:firstName rdf:type er:AtomicAttribute ;
                rdfs:seeAlso foaf:firstName .
....
#EntityType
vocab:University rdf:type er:EntityType ;
                 rdfs:seeAlso vivo:University .

#CompositeAttribute
vocab:uniAddress rdf:type er:CompositeAttribute ;
                 er:hasSubPart vocab:city ;
                 ....
                 er:hasSubPart vocab:street .
....
```

Fig. 1. (a) RDF triples represented with turtle (b) representation of individuals & clustering (c) annotation of clusters (d) explanation of inferred ER constructs (e) inferred schema represented as an ER diagram (f) inferred schema serialised as RDF-triples.

diagram where *classes* are represented as *entity types*, *properties* as *attributes* and *relationships* as *entity type relationships.*

3.1 Pre-processing

Initialisation. To gather instance-level information, the algorithm is presented with a SPARQL endpoint or an RDF dump. Figure 1(a) shows a snapshot of an RDF-graph represented in *turtle* notation. In the case of a SPARQL endpoint, such information is obtained by posing SPARQL queries such as the ones shown in Listing 1.1. This is done to determine resources that are stated to be instantiations of some class by looking for RDF typing information (i.e., *rdf:type* statements). The resources that are determined as results of the queries are stored locally in a triple store. The flexibility of the RDF model does not impose any restriction on the use of *rdf:type* statements that determine whether a certain resource is an instantiation of some class. Thus we do not expect our simple heuristic to work for every RDF source. In case of an RDF dump we do not restrict the approach to resources that are instantiations of some class since the algorithm can look at all resources identified as being similar by the distance function and grouped in the same cluster, as we shall discuss later in Sect. 3.2. The RDF dump is imported into a local triple store that is used by the algorithm to access the resources.

Representing Individuals. Each resource identified with a unique URI is represented as a *candidate description* as in Definition 2. Examples of candidate descriptions are shown in Fig. 1(b). Having a pool of CDs, the algorithm proceeds to organise them into clusters using the clustering algorithm described in the following section.

3.2 Clustering Algorithm

To identify groups of similar instances, a hierarchical agglomerative clustering algorithm (as described in Algorithm 1) is used to group candidate descriptions into clusters. Our main criterion for choosing a hierarchical solution was the fact that we do not know in advance the appropriate number of clusters, and hierarchical clustering does not require any prior knowledge of this number. In non-hierarchical techniques, such as the k-means algorithms, the number of clusters needs to be specified in advance; this is in fact similar to deciding at which level to cut the final dendrogram in hierarchical clustering. Our approach uses the *silhouette coefficient* to determine the final number of clusters, as discussed in this section.

Typical hierarchical algorithms use a *similarity matrix* to cache the similarities of each pair of elements to be clustered. In our case the algorithm constructs a $|CD| \times |CD|$ similarity matrix that holds the pairwise similarities between clusters of CDs. To calculate the similarity between (CD_i, CD_j) the algorithm represents each CD as a set of features. For example, given the following candidate description $CD_1 = \{$vocab:firstName, vocab:lastName, vocab:homePage,

Algorithm 1. `Cluster Candidate Descriptions`

```
Require: Set of $\mathcal{CD} = \{CD_1, CD_2, ..., CD_{|CD|}\}$
 1: $m \leftarrow 0$
 2: $U^m \leftarrow \{\{CD_1\}, \{CD_2\}, ..., \{CD_{|CD|}\}\}$
 3: Construct similarity matrix $M = |CD| \times |CD|$
 4: Let $(U_i^m, U_j^m)$ be the most similar pair in $M$:
 5:         $\underset{(U_i^m, U_j^m) \in M}{\text{argmax}} \; cluster\_sim(\{U_i^m\}, \{U_j^m\})$
 6: $max \leftarrow cluster\_sim(\{U_i^m\}, \{U_j^m\})$
 7: while $(max \geq t)$ do
 8:     $m \leftarrow m + 1$
 9:     $U_{ij}^m \leftarrow U_i^{(m-1)} \cup U_j^{(m-1)}$
10:     $U^m \leftarrow (U^{(m-1)} \setminus \{U_i^{(m-1)}, U_j^{(m-1)}\} \cup U_{ij}^m)$
11:     $C \leftarrow C \cup U^m$
12:     Update similarity matrix $M$
13:     Let $(U_i^m, U_j^m)$ be the most similar pair in $M$:
14:         $\underset{(U_i^m, U_j^m) \in M}{\text{argmax}} \; cluster\_sim(\{U_i^m\}, \{U_j^m\})$
15:     $max \leftarrow cluster\_sim(\{U_i^m\}, \{U_j^m\})$
16: end while
17: return $C$
```

vocab: registeredTo} to find the set of features that characterises it we strip the namespace prefix from each predicate $p_j \in P'$. The output set of features will then be {firstName, lastName, homePage, registeredTo}. Let the function *cd_sim* (CD_i, CD_j) be the similarity measure between a pair of candidate descriptions where $i \geq 1, j \leq |CD|$. We then use the *Jaccard similarity coefficient* as the similarity measure, that is,

$$cd\_sim(CD_i, CD_j) = Jaccard(CD_i, CD_j) = \frac{|CD_i \cap CD_j|}{|CD_i \cup CD_j|} \in [0, 1]. \quad (1)$$

Having computed the similarity of each pair of CDs and stored it into the similarity matrix, the clustering algorithm proceeds as follows. Initially each CD is assigned to a singleton cluster. The algorithm keeps track of each iteration m by assigning a sequence number *0,1, ...,(m-1)* (Line 1). The set of clusters (i.e., clustering) produced at each iteration is denoted as U^m, thus by assigning each CD into its own cluster, $U^0 = \{\{CD_1\}, \{CD_2\}, ..., \{CD_{|CD|}\}\}$ (Line 2). In Lines 4 and 5 the algorithm identifies the most similar pair of clusters to be merged (according to our similarity measure). Then agglomerative hierarchical clustering proceeds iteratively (Line 6) by merging the *most similar* pair of clusters to form the next clustering $m = m + 1$ (Line 7) where $U_i^{(m-1)}$ and $U_j^{(m-1)}$ are the clusters. As a result of the merge step at each iteration m, the number of clusters decreases by one $|U^m| = |U^{(m-1)}| - 1$. The merge step produces a new cluster, $U_{ij}^m = U_i^{(m-1)} \cup U_j^{(m-1)}$ and clustering (Lines 8–9). In addition, at each step the algorithm stores each clustering (Line 10), to be used later as input for determining the best silhouette coefficient. Then the algorithm needs to update the similarities between the new (merged) cluster and the other clusters from the similarity matrix (Line 11). Numerous approaches have been developed for computing the similarity between two clusters [23]. For our algorithm we define the similarity between clusters U_i^m and U_j^m as the mean similarity between elements of each cluster (a.k.a average linkage),

$$cluster\_sim(U_i^m, U_j^m) = \frac{\sum\limits_{CD_a \in U_i^m} \sum\limits_{CD_b \in U_j^m} cd\_sim(CD_a, CD_b)}{|U_i^m||U_j^m|} \in [0,1]. \quad (2)$$

To compare the results of average linkage, we observed the results of running our clustering algorithm with other cluster similarity measures, such as:

- **Single Linkage:** The similarity between clusters U_i^m and U_j^m is calculated based upon the *maximal similarity* between elements of each cluster, defined as,

$$cluster\_sim(U_i^m, U_j^m) = \max_{CD_a \in U_i^m, CD_b \in U_j^m} cd\_sim(CD_a, CD_b). \quad (3)$$

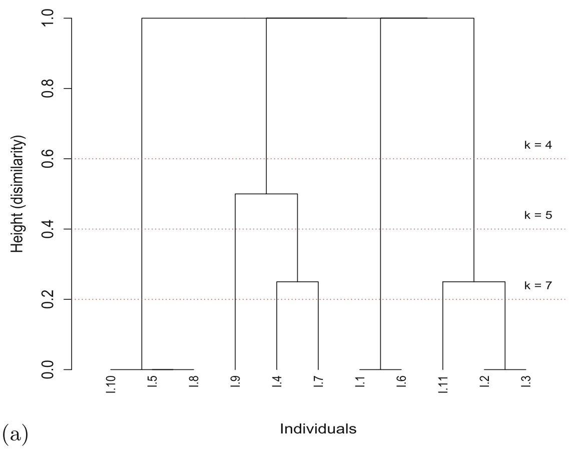
- **Complete Linkage:** The similarity between clusters U_i^m and U_j^m is calculated based upon the *minimum similarity* between elements of each cluster, defined as,

$$cluster\_sim(U_i^m, U_j^m) = \min_{CD_a \in U_i^m, CD_b \in U_j^m} cd\_sim(CD_a, CD_b). \quad (4)$$

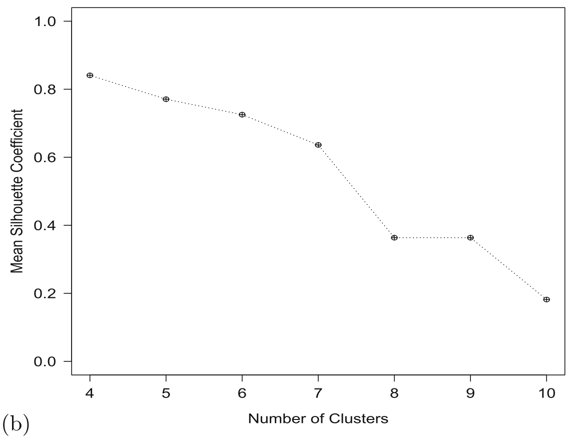
As discussed in Sect. 4.2, we have empirically determined that the above similarity schemes tend to give similar results. As for the termination condition, the algorithm stops when the most similar pair of clusters is below a certain threshold t, that is $cluster\_sim(U_i^m, U_j^m) < t$ (we elaborate on the choice of t in Sect. 4.2). To complete our discussion on the algorithm, by decreasing the number of clusters at each step the output (Line 15) is a sequence of clusterings.

Determining the Best Clustering. As explained in the previous section, several clusterings are produced by the algorithm (as depicted in Fig. 2(a)). Deciding which clustering best fits the data is a well known issue in cluster analysis [7], and approaches to solve this problem build on techniques that assess the quality of the clustering results. Usually such approaches determine the validity of a clustering based on the ideas of (a) *compactness*, i.e., how close the elements of a cluster are, and (b) *separation*, i.e., how distinct a cluster is from other clusters. We would like to keep our technique independent of any external information such as externally supplied class labels, thus we have used *silhouette coefficients* for evaluating the various clusterings (for more on clustering validation see [7]).

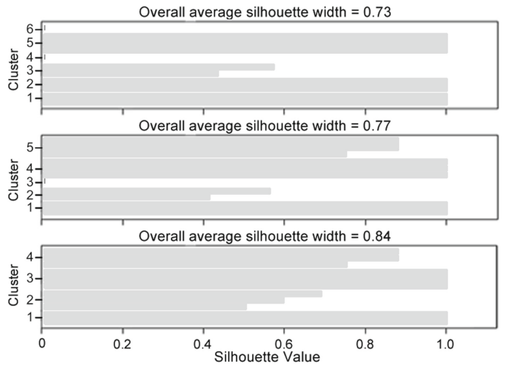
A silhouette coefficient (SC) is calculated for each individual and is a measure of how similar the individual is to other individuals in its own cluster compared to



(a)



(b)



(c)

Fig. 2. (a) Hierarchical clustering result represented as a dendrogram with possible cut points, (b) shows the average silhouette coefficient for different clusterings, and (c) *silhouette-plot* that shows the silhouette coefficient calculated for each individual in each cluster.

other individuals from other clusters. Considering both cohesion and separation, the SC is defined as follows [11],

$$sil(i) = \frac{b(i) - a(i)}{max\{b(i), a(i)\}} \in [0, 1]. \tag{5}$$

Given an individual i in some cluster U, a(i) is the average dissimilarity[4] of i and all other individuals in cluster U and b(i) is the average dissimilarity between i and individuals of the closest cluster to U. We follow the definition by Kaufman and Rousseauw [11] which states that singleton clusters have $sil(i) = 0$. Having the silhouette values for each individual we can calculate the *average silhouette width* (ASW) for each cluster $ASW_{cluster}$ as the mean value of all $sil(i) \quad \forall \quad i \in U$ and the $ASW_{overall}$ for the entire population as the mean of all individual $sil(i)$ silhouettes. This is defined as,

$$ASW_{overall}(k) = \frac{\sum_{i=1}^{n} sil(i)}{n} \in [0, 1], \tag{6}$$

where n denotes the number of all individuals. To determine the best clustering and thus the number of clusters k we choose the clustering with the highest $ASW_{overall}$. Using a dendrogram Fig. 2(a) visualises the outcome of running the clustering algorithm with possible cut points that give rise to different number of clusters, k. For each such clustering Fig. 2(b) shows the value of the silhouette coefficient for different number of clusters whereas in more detail Fig. 2(c) is a *silhouette plot* that shows the silhouette coefficient calculated for each individual in each cluster.

In our example, the clustering algorithm terminates when the dissimilarity between clusters is at the maximum (i.e., 1) and therefore there are no more clusterings produced fewer than 4, which also happens to be the clustering with the maximum silhouette coefficient; thus, $k = 4$ in this case.

3.3 Annotation of Clusters

Class Names. Each cluster contains individuals that are similar according to the distance measure used. In the ideal case all similar individuals are grouped in the same cluster. The resulting clustering suggests that individuals classified in some group are possible instantiations of some class. By observing the individuals, any RDF typing information that is explicitly stated is used to annotate each cluster with a descriptive label, as shown in Fig. 1(c). This label gives rise to the names of the classes we are looking to infer. We have used a greedy algorithm that suggests that the class label that occurs the most in a particular cluster is chosen as the class name. As our evaluation in Sect. 4 shows, this simple approach yields good results. In the case of clusters that contain individuals without *rdf:type* statements a special label, "Unknown" is used as the inferred

[4] Is the opposite of a similarity.

cluster label. One might then use the inference semantics of RDF graphs to infer the RDF typing information in cases where such information is not available. Finally, in cases where the clustering approach gives rise to different clusters that partition individuals of the same type, the technique will have to choose which cluster to use to infer a description for that class. A cluster with more of the individuals with that *rdf:type* stands a better chance of containing sufficient information that can guide the development of a schema, therefore the technique prefers such a cluster. We discuss this case in our evaluation in Sect. 4.3. To sum up, cluster labels represent the names of possible classes that organise data in an RDF source; examples are *Student* and *University* as shown in Fig. 1(c).

Class Properties. The formal definition of an *RDF triple* (see Sect. 2) suggests that an object could be either a literal, a blank node or a resource. The algorithm follows a simple heuristic and takes into consideration predicates that are literals to annotate a discovered class with its attributes. However, often resources have predicates that point to blank nodes. In such cases the algorithm considers blank nodes as evidence for identifying multi-valued or composite attributes for our ER model representation, as depicted in Fig. 1(d). We have also observed that the RDF model does not restrict resources, that are instantiations of the same class to have the same number of attributes. For example, a resource that is an instantiation of a class *Student* could use only predicates {firstName, lastName, registeredTo, homePage} whereas another resource could be described using just {firstName, lastName, homePage}, omitting *registeredTo*. So as not to miss any attributes the algorithm takes the union of the predicate labels in the cluster as the list of identified attributes. For example, the *Student* class could have attributes {firstName, lastName, registeredTo, homePage}. In addition, this phenomenon of missing attributes could be due to cases of specialisation/-generalisation relationships. With this reflected upon the instances the algorithm chooses to union the properties identified in each cluster.

Class Relationships. To infer relationships between our identified class labels we observe predicates that are URI-links to other resources rather than literals. Simply put, we follow the heuristic: *an RDF triple that is a URI-link rather than a literal is a candidate relationship, and it is a relationship within the RDF graph if it refers to another entity within the same RDF graph.* For each identified class, by observing the predicates of its individuals, we extract all predicates that are candidate relationships according to our heuristic. This provides the algorithm with enough information to identify relationships for inferred classes.

3.4 Schema Constructs

The simple Entity-Relationship (ER) conceptual model is expressive enough to model our inferred schema, and we have used a simple set of mapping rules, to express our inferred structure using ER constructs. As shown in Fig. 1(d), inferred Classes are modelled as *entity types*, class properties that consist of single atomic values are modelled as *atomic attributes* whereas ones that occurred multiple times are considered as evidence of *multivalued attributes* and are modelled as such. Any properties that point to anonymous resources (i.e., a BNodes) are modelled as *composite attributes*, and finally any properties that point to other resources are modelled as *relationship types*. To depict the constructs of an inferred schema we have used the classic diagrammatic conventions of ER.

3.5 Schema Construction and Serialisation

Finally, having collected enough information to annotate entity types with attributes and relationships, the algorithm proceeds to infer a structural summary over the imported RDF source. Figure 1(e) shows a simple schema (represented as an ER-diagram) that has been inferred from a simple RDF source. To conform with the context of LD, we have used a simple RDFS vocabulary to serialise the inferred schema as RDF triples. To the best of our knowledge[5] there is no RDFS vocabulary that captures the constructs of the ER model. Thus, we have created one and published it using best practises suggested by the Linked Open Vocabularies (LOV) project, and then used it to serialise our inferred schema as an RDF graph (see Fig. 1(f)). To conclude, we have noticed that there are some attempts in the literature [4] to map ER diagrams to OWL ontologies. The outcome of our structure inference technique can be used as an input to such techniques.

4 Empirical Evaluation

In the following, we present the methodology used for evaluating our technique. To ensure diversity in the LD sources used for the evaluation we distinguish between two different types of RDF sources: those that have been generated by a translation from relational databases using a systematic approach (such as the D2RServer tool [2]) and real-world Linked Data sources from the Web of Data. We have categorised the sources into two groups according to their generation method, as in Table 1.

Table 1. Linked Data sources used for evaluation.

	Name	# triples	# classes	BNodes	generated_by
1	cdShop	303	3	Y	D2RServer
2	Conference	300	8	Y	D2RServer
3	BIRT_db[a]	28.5 k	8	N	D2RServer
4	Jamendo	1.1 M	11	N	DBTune.org
5	Magnatune	322 k	7	N	DBTune.org

[a] http://www.eclipse.org/birt/phoenix/db.

[5] Observing the vocabularies listed by the Linked Open Vocabularies (LOV) project: http://lov.okfn.org/dataset/lov/.

The RDF graphs that were used during the evaluation have been manually downloaded and imported into a local triple store since some of the datasets were not always accessible and some others were only published as RDF dumps. For the evaluation of our technique we are looking to investigate whether the individuals have been assigned to the actual classes according to some ground truth. In addition, we would like to establish the extent to which our technique can infer a structural summary of the LD sources by identifying the correct classes, their properties and their relationships.

4.1 Experimental Methodology and Metrics

As previously mentioned, for each of the following experiments we are looking to measure (a) how good the clustering solution is at grouping individuals, and (b) how well the approach has identified the correct entity types, attributes and relationships.

Quality of Clusterings. For measuring the quality of the clustering, we have used the *FScore measure* [14]. This measure required us to manually assign class labels to each individual to form the gold standard. Having the gold standard we proceed to compute the FScore measure as follows. For each particular class label L_r of size n_r and cluster C_i of size n_i, with n_{ri} being the number of individuals in cluster C_i that belong to L_r, we measure the FScore of this class and cluster, using,

$$FScore(L_r, C_i) = \frac{2 \times P(L_r, C_i) \times R(L_r, C_i)}{P(L_r, C_i) + R(L_r, C_i)} \in [0, 1], \tag{7}$$

where, $P(L_r, C_i)$ is the precision value defined as n_{ri}/n_i, and $R(L_r, C_i)$ is the recall value defined as n_{ri}/n_r for the class L_r and cluster C_i. The FScore of the class L_r is the maximum FScore. To understand how good the choice of the determined clustering solution is we also calculate the overall FScore, using the following formula, where $|L|$ is the number of classes and n the number of individuals, that is,

$$Overall\_FScore = \sum_{r=1}^{|L|} \frac{n_r}{n} max(FScore(L_r, C_i)) \in [0, 1]. \tag{8}$$

An ideal clustering solution is the one in which every class from the gold standard has a corresponding cluster where all the individuals of that class made it to the correct cluster, the higher the FScore the better the clustering.

Quality of Inferred Schemas. To measure how well our technique inferred schemas for LD sources we measure Precision/Recall and FScore for each of the ER constructs we are expecting (i.e., entity types, attributes and relationships). We have manually designed the schemas that we are expecting and compared them with the derived result. To design a gold standard, in cases that we had access to, we have observed the SQL schemas that populate a relational version of the data, otherwise we have just observed the resulting RDF-graphs.

For *entity-types* we determine true positives, i.e. entity types needed and inferred, false positives, i.e. entity types not needed but inferred, false negatives, i.e. entity types needed but not inferred. For *attributes* and special types of attributes (e.g., composite) we determine true positives, i.e. attributes needed and inferred as attributes to the correct entity type, false positives, i.e. attributes not needed on an entity type, but inferred, false negatives, i.e. missed attributes. Finally, for *binary relationships* we determine true positives, i.e. relationships inferred between the correct entity types, false positives, i.e. relationships incorrectly inferred between entity types, and false negatives, i.e. any relationships missed.

Before discussing the results of our experiments, note that in the description of the clustering algorithm in Sect. 3.2 the termination condition of the algorithm depends on what the algorithm considers as the minimum similarity value for which a pair of clusters is considered as a candidate for merging. Thus before discussing any results on measuring the effect of our schema inference technique we elaborate on the choice of the minimum similarity value for merging and then we elaborate on the choice of the cluster-to-cluster similarity scheme (i.e., average, single and complete linkage). For the experiments Sects. 4.3 and 4.4, we have used as in Sect. 4.2, values for the above parameters that we have empirically determined.

4.2 Determining Parameters

Min. Threshold for Merging. As explained in Sect. 3.2, the clustering algorithm clusters together individuals with similar features. At each iteration the algorithm selects the most similar pair of clusters to be merged. To characterise that a pair of clusters is similar enough to be merged we have used a threshold t, which is the minimum similarity value a pair of clusters should have, to be considered as a candidate for merging. In this section we elaborate on the choice of t, by observing its effect on the maximum average silhouette width, and thus on the quality of the clustering. In doing so we have run the algorithm with different values for t, iteratively increasing its value by 0.1 until $t = 1.0$. This is shown in Fig. 3 where we have run our experiment using sources 1, 2 and 3 from Table 1. As t varies closer to 1.0 the algorithm becomes stricter in the choice of clusters to be merged. This means that individuals that are not similar enough remain unclustered, thus causing the average silhouette width to decrease, for the reason that most of the individuals remain in their singleton clusters. This is reasonable since, from the definition of the silhouette coefficient [11], singleton clusters have a silhouette value of zero. Such drops are shown in Fig. 3 for sources 2 and 3 when t is around 0.4 and 0.5 respectively. The values of the max. average silhouette width seem to remain constant for choices of t closer to 0, for the reason that the algorithm is more flexible in merging clusters that are not so similar. Varying t closer to 0 allows more merge steps, thus more iterations until termination. For each merge step the algorithm calculates the maximum average silhouette width overall for the clustering, and therefore changing t does not always have an effect on choosing the maximum $ASW_{overall}$.

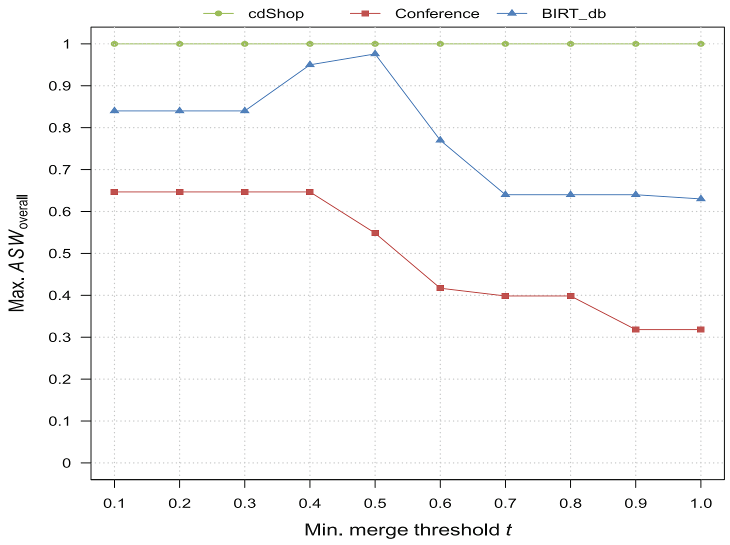


Fig. 3. Choice of t and its effect on the maximum average silhouette width.

Furthermore, in cases where most or all of the individuals from each class are using the same or almost the same set of predicates the choice of t does not seem to have any real effect, as in source 1. Because of the diversity of the data in the sources we cannot be very strict on the choice of t, however from this simple experiment it seems reasonable to choose a value for t to be in the range of [0.4–0.5], therefore for the experiments to follow we choose $t = 0.5$.

Linkage Schemes. To complement our experiments on how good the clustering solution is, we have empirically observed the behaviour of the clustering algorithm using different linkage schemes for measuring the similarity between clusters (as mentioned in Sect. 3.2). For each of the RDF sources in Table 1, we have run the clustering algorithm and observed the effect of using a different scheme on the value of the silhouette coefficient. As shown in Fig. 4, in most cases the different schemes co-occur in identifying the number of clusters with the exception of *complete linkage*. As depicted in Fig. 4(b), using complete linkage the maximum average silhouette coefficient occurs when the number of clusters is 14 where the real number of clusters in the dataset is 9. On the other hand the alternative approaches are closer to identifying the real number of clusters. For our purposes it seems reasonable to choose single or average linkage because of their similar results, and in practise we have used the *average linkage* as the default linkage scheme.

4.3 Experiment 1: Reverse Engineering

Relational databases played a key factor in expanding the LOD cloud as a source for a large number of RDF triples. Tools like the D2R-Server [2] have been

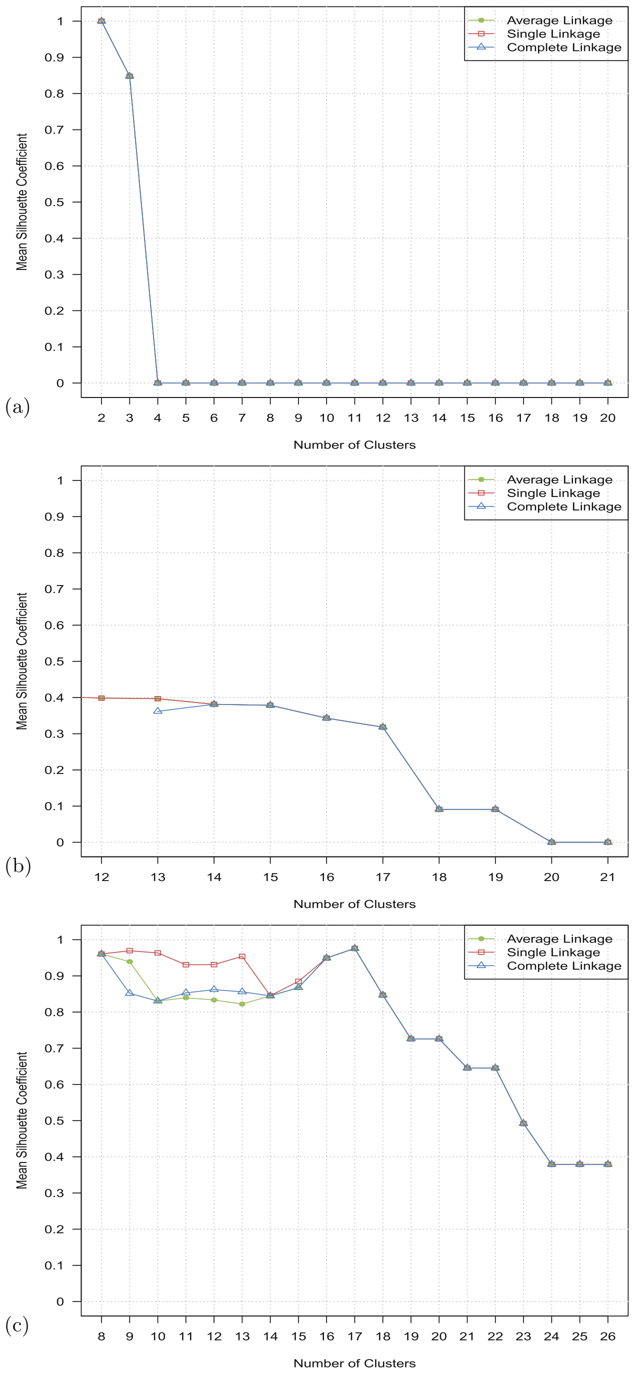


Fig. 4. Shows for each cluster similarity scheme, the top 10–20 occurrences of mean silhouette coefficient. x-axis shows the number of clusters in each clustering and y-axis the corresponding silhouette coefficient.

essential in providing a standardised way of exposing relational databases as LD sources on the Web. However, such datasets lack the organisation of a schema as previously existed in the relational database. With this experiment we are looking to demonstrate the effectiveness of our technique by reverse engineering the schema of some RDF sources that have been generated from a relational database (see Table 1). Before determining how well our technique inferred a schema, we would like to gain some insights regarding the quality of the clustering determined by the algorithm. This is important since our technique aims at looking for recurring patterns from the clusters formed, in order to infer the structure of the source. Figure 5(a) shows the result for a small dataset with only a few class labels. The algorithm has successfully assigned individuals to the correct clusters except those from the *Category* label. This was done intentionally by the algorithm since all instances of the Category class are represented as blank nodes in the RDF source. This evidence has been treated by the algorithm as a composite attribute, in our ER-model representation. This does not mean that the algorithm has missed the individuals of the Category class, instead, knowing the existence of a composite attribute we can easily formulate a SPARQL query to populate data from the Category class (e.g., Listing 1.2). The same also happened in Fig. 5(b) with a source that has more labels, however in this source some instances have not been classified in the correct classes. This is because the particular source has instances of different classes that use the same predicates. For example, *Researcher* and *PhDStudent* share {firstName, lastName, address, homePage}. Figure 5(c) shows the results for an RDF source with no blank nodes. The silhouette coefficient determined 17 clusters instead of 8 and therefore some individuals have not been assigned to any cluster, or individuals of the same type have been partitioned into several clusters. The choice of 17 clusters has been determined by the highest $ASW_{overall}$. For 17 clusters the average silhouette width is 0.976 and the second best, suggesting just 8 clusters is 0.960. However, the effect of the clustering produced by the SC does not influence the post-processing tasks downstream and as such the algorithm manages to get good results for the inferred structural summary (see Table 2).

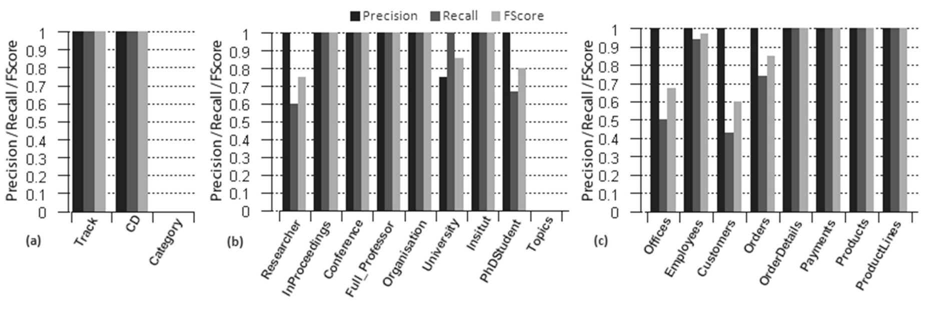


Fig. 5. Quality of the clustering solution.

To conclude this experiment, we observe how the quality of clustering influences the final inferred schema and the judgement of the algorithm in determining the schema of the source. Using the metrics described in Sect. 4.1 to evaluate the effectiveness of our technique, the results are presented in Table 2. For each of the sources we have compared the inferred schema results with the gold standard. The important observation is that the technique managed to infer a structure as expected with minor fallouts that influenced the performance.

Table 2. Evaluation of schema inference technique: (ET): Entity Types, (AT): Attributes, (R): Relationships

	cdShop			Conference			BIRT_db		
	ET	AT	R	ET	AT	R	ET	AT	R
Precision	1	1	1	1	1	1	1	1	1
Recall	1	1	1	1	0.90	0.86	1	0.92	0.86
FScore	1	1	1	1	0.95	0.92	1	0.96	0.92

In more detail, we have noticed that in all cases the algorithm manages to infer all the entity types (i.e., classes) that where expected. For classes that have resources as BNodes the algorithm creates a special type of attribute instead, thus we do not classify them as *false negatives*. Some instances translated from relational tables have NULL values in some of their attributes. Thus, there are cases where the algorithm misses some attributes. Regarding the identification of relationships between classes, the algorithm performs well. We have only noted some *false positives* in cases where classes participate in a class hierarchy. The algorithm is not aware of this and, therefore sometimes misplaces some of the relationships between different classes that participate in `is-a` relations. However, still the results are promising. We understand the diversity of LD sources in terms of representing data with different structures and terminologies, however, our prototype technique performed well in terms of inferring the structure of RDF sources that previously represented data as relational tables.

```
SELECT DISTINCT ?o ?category
WHERE {
   eShop:CdNo9 eShop:category ?o .
   ?o eShop:name ?category .
 }
```

Listing 1.2. Explore *Category* class triples.

4.4 Experiment 2: On Sources from the Web of Data

We repeated the previous experiment on real sources from the LOD cloud that were not generated from existing schemas. For this experiment we have chosen

LD sources from *DBTune.org* namely *Jamendo* and *Magnatune*. Figure 6(a) shows the result from inferring the schema for *Jamendo*. In this source we have observed that some URI resources do not have any *rdf:type* statements therefore they have been classified as elements of our specialised "Unknown" class. Without any RDF typing information the label of the class cannot be determined. In such cases our technique creates several clusters labelled with the "Unknown" label. As already mentioned, the inference semantics of RDF graphs could provide an insight as to what could be a possible class label for our "Unknown" classes. Nevertheless, despite the lack of class label information the algorithm managed to infer the relationships and attributes for the "Unknown" class. Although this class has been inferred by the algorithm during our evaluation, we have considered it as a *false positive*, hence the decrease in the measures for Entity Types. Similarly, the relationships identified for the "Unknown" class are considered as *false positives*. This does not mean that the technique failed to determine the relationships as it should have, but it was unaware of the actual classes that participate in the relationship. Similarly, Fig. 6(b) depicts the results of running the algorithm over *Magnatune*. According to the gold standard designed for this source, the algorithm correctly inferred a structural description of the source as expected. We have noticed that, overall, the schemas of LD sources normally use a few classes to describe their resources and that the algorithm can perform well in inferring structural summaries over published LD sources.

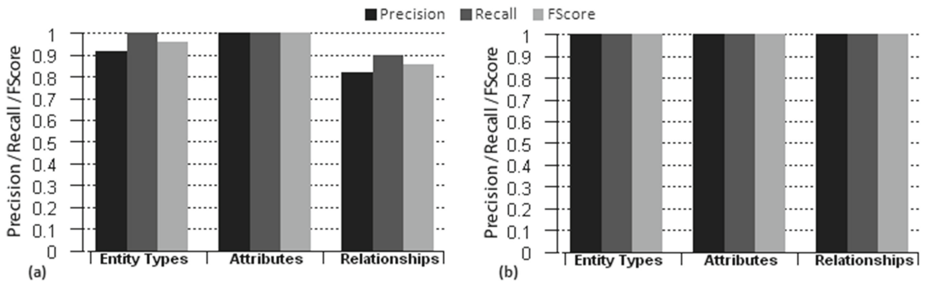


Fig. 6. Evaluation of schema inference technique.

5 Related Work

As already mentioned, the widespread adoption of the RDF model has led to an emerging need to access data from various heterogeneous and distributed data sources. Since the data are distributed, and due to the schema-less nature of the model, efficient retrieval of the data is a challenging task. In fact, several challenges contribute to this, some of which are: (i) the challenge of locating which datasets could possibly contribute answers to a given query, (ii) the lack of a comprehensive instance-level summary of the data, and (iii) scalability issues.

One approach to this challenge is based on *materialisation*, where a complete replica of the RDF graphs is stored in a central triple-store which is then used for query answering. However, such a central store assumes that the data remain static or evolve slowly, and that the most current version of the data is not required. Moreover, with the current size of the LD cloud, with more than 62 billion RDF triples[6], maintaining a replica is resource intensive, and the lack of an intensional description of the data makes query formulation a challenge. In this section we summarise ongoing research on proposals for solving some of the challenges related with efficient data management in the LOD ecosystem. We begin by discussing related work on locating datasets that can possibly contribute answers to a query, discovering RDF-specific schema knowledge from datasets and finally proposals on distributed query processing. We then position our work in relation to other proposals for discovering knowledge that can inform the formulation of SPARQL queries over a given RDF source.

On Source Discovery. There has been some recent research on source selection that provides summaries or descriptions of the RDF triples that can be found in LD sources using index structures [13,21]. We refer to these as *triple-level summaries*. An example of such work is *SchemEX* [13], which uses a stream-based approach for extracting schema information from RDF triples that are traversed from an RDF graph using a fixed-window. The extracted schema is then used to guide the construction of an index structure by linking schematic information to relevant datasets. Given a SPARQL query, SchemEX performs a lookup in the index structure to find which datasets contain instances of a specific RDF schema concept that can contribute to answering the query. As such, SchemEX aims to deal with the challenge of providing a summary of the kind of triples that can be found in a data source. By contrast, our work focuses on inferring a summary of how individuals are organised in a single source that can be used for query formulation, rather than on using the extracted schema for constructing an index that is used for relevant source discovery. In addition, both our approach and SchemEX utilise RDF typing information at the instance-level, aim to support query execution, and model the outcome of the extraction processes as RDF triples.

Harth, *et al.* [8] propose the use of an approximate multidimensional indexing structure (i.e., QTrees) as a data summary for determining which sources can potentially contribute answers to a query. The construction of an index structure is made possible by applying hash functions over the individual components of RDF triples (*subject, predicate, object*) contained in the datasets, to obtain data points that correspond to a three-dimensional QTree. A certain set of similar triples is then approximated by minimal bounding boxes (MBBs). At query time a set of MBBs is returned for each triple pattern in the query that suggest relevant sources that can contribute to the query result. Our work differs from the above approach since we would like to have an understanding of how concepts are represented in the sources and not a summary of what triples exist in which source. Thus the emphasis in [8], as exploited in [17], is on providing an

[6] http://stats.lod2.eu/.

instance-level summary that can inform efficient query evaluation, whereas the emphasis in our work is on providing a schema-level summary that can inform data integration.

On Knowledge Discovery and Ontology Mining. In a complementary approach, Zong, *et al.* [24] explored a method to dynamically generate a concept hierarchy using LD sources from the bio-medical domain. In doing so, their method utilises RDF typing information at the instance-level and builds upon hierarchical clustering. This is similar to our technique where a pre-processing step is necessary for determining the similarities between pairs of individuals using a distance function. The pair-wise similarities are then used as an input to the clustering step (as in Sect. 3). A similar work-flow is followed by their technique, where the similarity between a pair of individuals is measured over the predicate values that are *URIs*, whereas our technique computes the similarity by considering the local names of all predicates that are used in either *RDF-links* or *literal* triples. Despite the similarities, their approach builds on sources that use a single ontology to organise data, and is restricted in terms of dynamically identifying relationships between the inferred concepts, whereas in our proposal the discovery of domain/range axioms is made possible; captured as entity type relationships.

Another example of relevant work is from Völker, *et al.* [22], on mining ontologies from RDF data, an approach referred to as *Statistical Schema Induction*. In contrast to our technique that builds on clustering, their approach mines association rules from RDF data sources to acquire schema-level knowledge. Association rules that satisfy a user-provided confidence threshold contribute to the construction of the ontology. In our technique, the silhouette coefficient is used (as described in Sect. 3.2) to determine the clusters to be considered when inferring structural summaries. Finally, resources need to explicitly provide some RDF typing information for their approach to work. Although this is also useful for our technique, our approach is not as restricted since it organises resources into groups despite the existence of *rdf:type* statements and looks for recurring patterns that can guide the development of a schema.

Distributed Query Processing. Having an understanding of a schema can also support Distributed Query Processing (DQP) over RDF sources. DQP requires an understanding of how concepts are represented, but such information is typically not available for LD sources. Quilitz and Leser [19] propose *DARQ*, an engine for federated SPARQL queries. Transparent access to multiple SPARQL endpoints is provided by making use of hand-crafted source descriptions that summarise the URIs of RDF properties that are used by the source to describe the data. Our technique can provide similar structural summaries automatically. Rather than using indices of the content of each RDF source or statistical information (e.g., VoiD) FedX [20] does not require any metadata upfront; instead, it uses SPARQL ASK queries for source selection at query time to annotate triple patterns in the query with relevant sources, and relies on join order heuristics for efficient SPARQL query processing over several LD sources. Our approach suggests that structural summaries that can be used to inform query formulation can inferred automatically.

6 Conclusions and Future Work

This paper described a technique that uses a hierarchical agglomerative clustering approach and a set of simple heuristics to determine a structural summary over RDF sources, with the aim of informing query formulation and supporting query processing over LD sources. We have shown that having a schema for an RDF source that can be inferred automatically does not contradict the schema-free nature of RDF sources. The flexibility of the RDF model is preserved since we are not forcing the data to adhere to any specific structure; the data are just used to guide the creation of such structural summaries over the sources. In addition, having a structural summary over LD sources aligns with recent trends on publishing datasets that are annotated with metadata, such as VoID descriptions. We propose to organise individuals into clusters which can then used to search for recurring patterns, with the aim of inferring structural summaries over LD sources. Our empirical evaluation over sources that have been constructed from a direct translation from relational databases as well as on real sources from the Web of Data validated that our technique generates good results.

While our results are promising, there remain several challenges to be further explored. In the following we attempt to highlight some of these challenges along with possible solutions on how to improve our structural inference technique:

Identification of class hierarchies. Our evaluation revealed two cases where it might be possible for the approach to determine subsumption relations. Discovering class hierarchies can be an important feature of the approach, we discuss it here with an abstract example however we leave this feature as a potential future work. Let us assume the existence of a single cluster c_1 that does not overlap with any other cluster and that the set of individuals of c_1 suggest more that one class label for the cluster. This might be possible when individuals, that potentially belong to different concepts, end up in the same cluster c_1 for the reason that they are using the same set of predicates to describe their data but have different RDF typing information. Another possible case for observing potential subsumption relations is when clusters overlap. Assume the existence of two overlapping clusters c_1 and c_2, where, some individuals belong to both clusters, this is the set given by their intersection $c_1 \cap c_2$. Potentially a simple heuristic based on counts can determine `is-a` relations, we leave this feature and its evaluation as a possible future direction for improving the presented technique.

Lack of *RDF typing* information. It is often the case that RDF sources do not explicitly state *rdf:type* information for every resource that appears in the source, in fact, there might exist cases of RDF sources which entirely neglect such information. As previously discussed, RDF typing information is useful for our technique, however, the lack of such knowledge causes the technique to assign the specialised "Unknown" label to discovered classes. Ideas to discover additional knowledge that can assist our approach into discovering a suitable class label in such cases are, among others, *(i)* to utilise the inference semantics of RDF sources with the use of a reasoner [15] that is used over the explicitly stated RDF data to reveal more knowledge, including RDF typing information, that could be utilised by our technique, and *(ii)* to take advantage of the dereference capabilities of predicate URIs to obtain access to their semantics as specified in semantic web ontologies described in RDFS/OWL. For instance, the semantics of *rdfs:domain* as appeared in the definition of properties can suggest that a particular instance is a member of some class. Further investigation of these proposals is left as a future work.

Distance function. In Sect. 3.2 we discuss a simple distance function based on the use of *Jaccard similarity* over localnames of predicates to determine the pairwise similarities between individuals. There might be cases where localnames alone provide insufficient knowledge for suggesting a similarity between a pair of candidate descriptions. In the simplest case of introducing typos in localnames the current distance function will not be able to determine any similarity. For example, the Jaccard similarity between $\{firstName, lastName, homePage\}$ and, $\{frsName, lstName, hmPage\}$ produces zero, which is unacceptable. It is also quite frequent in complex LD sources that predicates from different vocabularies are using identical localnames. Such cases will cause our distance function to derive a misleading conclusion on judging that they are identical. The design of a distance function that overcomes such weaknesses is desirable. This is important for our approach since the pairwise similarities are the foundations on which the clustering algorithm is making its decisions; into forming the right clusters that will then give rise to possible classes, properties and relationships. An improved distance function can perhaps consider several sources of evidence for judging the similarity of a pair of individuals. A possible suggestion is to take into account the predicate values of triples that are RDF-links (i.e., triples that their object's values are URIs). Again, in cases where is possible, dereferencing predicate URIs may reveal additional semantic evidence that could be used as additional knowledge for judging their similarity. Finally, to deal with typos a syntactic distance metric such as edit-distance could be used. We leave the design of an improved distance function that considers the above suggestions and its evaluation as a future direction.

Acknowledgement. Klitos Christodoulou has been supported by funding from the UK Engineering and Physical Sciences Research council, whose support we are pleased to acknowledge.

References

1. Arenas, M., Gutierrez, C., Pérez, J.: Foundations of RDF databases. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.-C., Schmidt, R.A. (eds.) Reasoning Web. LNCS, vol. 5689, pp. 158–204. Springer, Heidelberg (2009)
2. Bizer, C., Cyganiak, R.: D2r server - publishing relational databases on the semantic web. In: 5th International Semantic Web Conference, p. 26 (2006)
3. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semant. Web Inf. Syst. **5**(3), 1–22 (2009)
4. Fahad, M.: Er2owl: generating owl ontology from er diagram. In: Shi, Z., Mercier-Laurent, E., Leake, D. (eds.) Intelligent Information Processing IV. IFIP, vol. 288, pp. 28–37. Springer, Heidelberg (2008)
5. Franklin, M.J., Halevy, A.Y., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Rec. **34**(4), 27–33 (2005)
6. Goldman, R., Widom, J.: Dataguides: enabling query formulation and optimization in semistructured databases. In: Proceedings of the 23rd International Conference on Very Large Data Bases, pp. 436–445. Morgan Kaufmann Publishers Inc. (1997)
7. Halkidi, M., Batistakis, Y., Vazirgiannis, M.: On clustering validation techniques. J. Intell. Inf. Syst. **17**(2–3), 107–145 (2001)
8. Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.-U., Umbrich, J.: Data summaries for on-demand queries over linked data. In: WWW, pp. 411–420 (2010)
9. Heath, T., Bizer, C.: Linked Data: evolving the web into a global data space. In: Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers (2011)
10. Hogan, A., Harth, A., Umbrich, J., Kinsella, S., Polleres, A., Decker, S.: Searching and browsing linked data with swse: the semantic web search engine. J. Web Sem. **9**(4), 365–401 (2011)
11. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. Wiley-Interscience, New York (1990)
12. Klyne, G., Carroll, J.J.: Resource description framework (RDF): concepts and abstract syntax. Technical report, W3C (2004)
13. Konrath, M., Gottron, T., Staab, S., Scherp, A.: Schemex - efficient construction of a data catalogue by stream-based indexing of linked data. J. Web Sem. **16**, 52–58 (2012)
14. Larsen, B., Aone, C.: Fast and effective text mining using linear-time document clustering. In: KDD, pp. 16–22 (1999)
15. Ravi Bhushan Mishra and Sandeep Kumar: Semantic web reasoners and languages. Artif. Intell. Rev. **35**(4), 339–368 (2011)
16. Paton, N.W., Christodoulou, K., Fernandes, A.A.A., Parsia, B., Hedeler, C.: Pay-as-you-go data integration for linked data: opportunities, challenges and architectures. In: Proceedings of the 4th International Workshop on Semantic Web Information Management, SWIM 2012, pp. 3:1–3:8. ACM (2012)
17. Prasser, F., Kemper, A., Kuhn, K.A.: Efficient distributed query processing for autonomous RDF databases. In: Proceedings of the 15th International Conference on Extending Database Technology, EDBT 2012, pp. 372–383. ACM (2012)
18. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation **4**, 1–106 (2008)
19. Quilitz, B., Leser, U.: Querying distributed RDF data sources with SPARQL. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 524–538. Springer, Heidelberg (2008)

20. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: optimization techniques for federated query processing on linked data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 601–616. Springer, Heidelberg (2011)
21. Umbrich, J., Hose, K., Karnstedt, M., Harth, A., Polleres, A.: Comparing data summaries for processing live queries over linked data. World Wide Web **14**(5–6), 495–544 (2011)
22. 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)
23. Zhao, Y., Karypis, G.: Evaluation of hierarchical clustering algorithms for document datasets. In: CIKM, pp. 515–524 (2002)
24. Zong, N., Im, D.-H., Yang, S.-K., Namgoong, H., Kim, H.-G.: Dynamic generation of concepts hierarchies for knowledge discovering in bio-medical linked data sets. In: Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, ICUIMC 2012, pp. 12:1–12:5. ACM (2012)