Datalift

Un ascenseur pour les données

ANR Contint – ANR-10-CORD-009

D4.1.2 Keys and Pseudo-keys Detection for Web Datasets Cleansing and Interlinking

Coordinator: François Scharffe (LIRMM, Université Montpellier 2)
With contributions from: Jérôme David (Université Pierre Mendès-France) and Manuel Atencia (Université Joseph Fourier)

Quality reviewer:	Serena Villata (INRIA)
Reference:	Datalift/2011/D4.1.2/v0.2
Project:	Datalift ANR Contint ANR-10-CORD-009
Date:	December 19, 2012
Version:	0.2
State:	preliminary
Destination:	public

Executive Summary

This report introduces a novel method for analysing web datasets based on key dependencies. This particular kind of functional dependencies, widely studied in the field of database theory, allows to evaluate if a set of properties constitutes a key for the set of data considered. When this is the case, there won't be any two instances having identical values for these properties. After giving necessary definitions, we propose an algorithm for detecting minimal keys and pseudo-keys in a RDF dataset. We then use this algorithm to detect keys in datasets published as web data and we apply this approach in two applications: (i) reducing the number of properties to compare in order to discover equivalent instances between two datasets, (ii) detecting errors inside a dataset.

Document Information

ANR Project Number	ANR Contint – ANR-10-CORD-009	**Acronym**	Datalift
Full Title	Un ascenseur pour les donnes		
Project URL	`http://www.datalift.org/`		
Document URL			

Deliverable	**Number**	4.1.2	**Title**	Keys and Pseudo-keys Detection for Web Datasets Cleansing and Interlinking
Work Package	**Number**	4	**Title**	Data interlinking

Date of Delivery	**Contractual**	M24	**Actual**	30-10-2012
Status	preliminary		final ☐	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination level	public ☒ consortium ☐			

Authors (Partner)	François Scharffe (LIRMM, Université Montpellier 2), Jérôme David (Université Pierre Mendès-France) and Manuel Atencia (Université Joseph Fourier)			
Resp. Author	**Name**	François Scharffe (LIRMM, Université Montpellier 2)	**E-mail**	Francois.Scharffe@lirmm.fr
	Partner	LIRMM		

Abstract (for dissemination)	**This report introduces a novel method for analysing web datasets based on key dependencies.**
Keywords	data linking, instance matching, record linkage, co-reference resolution, ontology alignment, ontology matching

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
05/09/2012	1	J. Euzenat	Set up file and outline
07/11/2012	2	J. David	Inserted the content
12/11/2012	3	F. Scharffe	A few corrections

Table of Contents

1. Introduction

The notion of key is fundamental for relational databases.[1] Keys allow to uniquely identify each tuple in a relation. The unicity property of keys is also exploited in order to optimize data access through the construction of indexes. Usually, keys are identified and chosen by the relational schema engineer, as part of the schema normalization process. However, there exist algorithms allowing to detect functional dependencies[2] inside a given database [5, 4].

In the semantic web framework, it is only since version 2 of the web ontology language OWL[3] that modelling keys is possible. A key in OWL2 for a given class is a set of properties allowing to uniquely identify an instance of this class. According to OWL2 semantics[4], two instance having same values for the properties of a key are considered identical. Using keys thus requires to know in advance the data that will be represented according to this ontology. This is not compatible with the decentralized publication of datasets on the web.

When considering a particular dataset, it will be possible to find out if one or more keys exist for a given class by analysing properties of this dataset. Given the variable quality of web data, it will be necessary to tolerate a few instances having same values for the properties of the key. In that case, we will use the term *pseudo-key*. Detected keys can then be associated to the dataset as metadata, for instance by extending the VoID vocabulary [1].

In this paper we propose an algorithm for discovering minimal keys and pseudo-keys in RDF datasets and we demonstrate the usefulness of this algorithm through two applications important for the web of data: the discovery of links between two datasets and the detection of errors in a dataset.

We first formally define the notion of key dependency for an RDF dataset. We then propose an algorithm for detecting keys (Section 2). Discovering keys is useful for many interesting applications (Section 5). We propose an application for reducing the number of properties to compare in order to find equivalence between instances from two datasets. We then see how pseudo-keys allow the detection of errors by showing redundancies in a dataset.

[1]This deliverable is an extended version of our EKAW 2012 paper, [2].

[2]keys are a special case of functional dependencies, see `http://en.wikipedia.org/wiki/Functional_dependency`

[3]`http://www.w3.org/TR/owl-overview/`

[4]`http://www.w3.org/TR/owl2-semantics/#Keys`

2. Key dependencies

Data representation on the semantic web is realized using the RDF language. In this paper, we denote the sets of all URIs, blank nodes and literals by $\mathbf{U}$, $\mathbf{B}$ and $\mathbf{L}$, respectively. An RDF *triple* is a tuple $t = \langle s, p, o\rangle$ where $s \in \mathbf{U} \cup \mathbf{B}$ is the *subject* or instance of t, $p \in \mathbf{U}$ is the *predicate* or property, and $o \in \mathbf{U} \cup \mathbf{B} \cup \mathbf{L}$ is the *object* of t. An RDF *graph* is a set of RDF triples.

Given an RDF graph G, the sets of subjects, predicates and objects appearing in G are denoted by $sub(G)$, $pred(G)$ and $obj(G)$, respectively.

Let G be an RDF graph. A predicate $p \in pred(G)$ can be seen as a relation between the subject and object sets of G, i.e., $p \subseteq sub(G) \times obj(G)$. It can also be seen as a partial function between the subject set and the powerset of the object set, i.e., $p : sub(G) \to 2^{obj(G)}$. This is the formalization that we will follow in this paper. To be more precise,

$$p(s) = \{o \in obj(G) : \langle s, p, o\rangle \in G\}$$

Then, the domain of the predicate p is the set

$$dom(p) = \{s \in sub(G) : \text{there exists } o \in obj(G) \text{ with } \langle s, p, o\rangle \in G\}$$

In the following definition we introduce our notions of key and minimal key in an RDF graph.

Definition 1 *Let G be an* RDF *graph and $P \subseteq pred(G)$. The set of predicates P is a* key *in G if for all $s_1, s_2 \in sub(G)$ we have that, if $p(s_1) = p(s_2)$ for all $p \in P$ then $s_1 = s_2$. The set P is a* minimal key *if it is a key and there exists no set $P' \subseteq pred(G)$ such that P' is a key and $P' \subset P$.*

The above definition is analogous to that one of the relational model in databases. The first main difference is that, unlike attributes, predicates can take multiple values: if p is a predicate and $s \in dom(p)$, then $p(s)$ is, in general, a non-singleton value set. The second one is that properties are not necessary defined on the whole set of individuals.

In line with TANE algorithm, given an RDF graph G, our approach lies in building the partition of $sub(G)$ induced by a set of predicates P. If this partition is made up of singletons, then P is a key.

Definition 2 *Let G be an* RDF *graph and $p \in pred(G)$. The* partition induced by the predicate *p is defined by*

$$\pi_p = \{p^{-1}(p(s))\}_{s \in dom(p)}$$

Let $P = \{p_1, \ldots, p_n\} \subseteq pred(G)$. The partition induced by the predicate set *P is defined by*

$$\pi_P = \{S_1 \cap \ldots \cap S_n\}_{(S_1,\ldots,S_n) \in \pi_{p_1} \times \ldots \times \pi_{p_n}}$$

Lemma 1 *Let G be an* RDF *graph and $P \subseteq pred(G)$. The predicate set P is a key in G if and only if π_P is made up of singletons, i.e., $|S| = 1$ for all $S \in \pi_P$.*

The complexity of finding keys in an RDF graph is polynomial in the number of subjects, but exponential in the number of predicates. For this, we introduce two criteria to reduce the search space. First, we discard sets of predicates which share few subjects compared to the total number of subjects in the graph, as they are not interesting for the applications we have in mind (Section 5). Second, we restrict ourselves to compute "approximate" keys, what we call pseudo-keys.

Definition 3 *Let G be an* RDF *graph and $P \subseteq pred(G)$. The* support *of P in G is defined by*

$$support_G(P) = \frac{1}{|sub(G)|} \Big| \bigcap_{p \in P} dom(p) \Big|$$

The predicate set P fulfills the minimum support criterion *if $support(P) \geq \lambda_s$ where $\lambda_s \in [0,1]$ is a given* support threshold*.*

Definition 4 *Let G be an* RDF *graph and $P \subseteq pred(G)$. The predicate set P is a* pseudo-key *in the graph G with* discriminability threshold λ_d *if*

$$\frac{|\{S \subseteq sub(G) | S \in \pi_P \text{ and } |S| = 1\}|}{|\pi_P|} \geq \lambda_d$$

3. Algorithm for discovering keys

The minimal key discovery algorithm in a RDF graph uses the partition representation and the same level-based search strategy (breadth first) as the functional dependancies discovery algorithm TANE [4]. There are two advantages in this approach. It allows to prune the search space and to reduce the cost for computing partitions. To prune the search space, we ignore any set of predicates which includes:

- a key, a pseudo-key;
- a subset having a support lower than the threshold;
- a subset in which a functional dependency holds.

Contrary to the TANE algorithm, we introduces a support threshold. This is useful because in rdf datasets, properties are not necessary instanciated for each individual. But, in counterpart, the optimization consisting in stripping partition [4] (removing singleton sets) can not be used. Finally, since the goal of the algorithm is to find keys only, there is not need to test exhaustively all the functional dependencies.

3.1 Details of the algorithm

Algorithm 1 works as follows. For each predicate set c tested at the previous iteration (*candidates*), it tests each new set formed by the union of c and one of the predicates P_G (not in c). If this new set is a key or a pseudo-key it is then added to the key set. If this set is not a key but its support is greater that the threshold value and no functional dependency holds over this set, it is then kept for the next iterations (i.e. added to list *nextCandidates*). The iterations continue until there is no more predicate set in *candidates*.

In order to avoid any non-minimal key generation or considering predicates with a too low support, the algorithm uses a set of blacklists (*skipLists*). The principle is the following: when the union of a candidate c and a predicate p has generated a key or pseudo-key, if there is some functional dependency over this set, or if this set has a support lower than the threshold, then the candidate c is added to the exclusion list of the predicate p. Each time a new set is generated from a candidate c' and predicate p, we verify if c' does contain a subset excluded for p.

Algorithm 1 as described does not detect keys on a given class, but analyses valid keys on the entire graph. In order to obtain keys for class, it is sufficient to execute it on the subgraph containing only instances of the considered class: $G_c = \{\langle s, p, o\rangle \in G :< s, rdf : type, c >\in G\}$.

3.2 Algorithm efficiency and scalability

The key and pseudo-key discovery algorithm was implemented in Java. In order to allow scalability, many intermediary results (the partitions) are stored on disk. Moreover, datasets are locally stored on disk and indexed using Jena TDB[1].

We have run the algorithm on a quad-core Intel(R) Xeon(R) E5430 @ 2.66GHz computer with 8GO memory. Table 3.1 gives the amount of time needed (computation+disk access) for computing all keys and pseudo-keys for every class successively taken in the datasets.

[1]http://incubator.apache.org/jena/documentation/tdb/

Algorithm 1 Key discovery in a graph G

```
$predicates \leftarrow P_G$
$candidates \leftarrow \{\emptyset\}$
$parts \leftarrow \emptyset$
$skipLists \leftarrow \{\emptyset\}$
while $|candidates| > 0$ do
    $nextCandidates \leftarrow \emptyset$
    for all $c \in candidates$ do
        $firstIdx \leftarrow max(predicates.indexOf(c[c.length - 1]), 0)$
        for $i = firstIdx \rightarrow predicates.length$ do
            if $\nexists l$ s.t. $l \in skipLists[predicates[i]] \wedge l \subseteq c$ then
                $P \leftarrow c \cup predicates[i]$
                $parts[P] \leftarrow \pi(parts[c], parts[predicates[i]])$
                if $key(part[P]) \vee pseudo\_key(part[P], \lambda_r)$ then
                    $keys \leftarrow \cup\{P\}$
                    $skipLists[predicates[i]] \leftarrow \cup\{c\}$
                else if $support(P) < \lambda_s$ or $\exists a \in P$ s.t. $\pi(P) = \pi(P - \{a\})$ then
                    $skipLists[predicates[i]] \leftarrow \cup\{c\}$
                else if $i < predicates.length - 1$ then
                    $nextCandidates \leftarrow \cup\{P\}$
                end if
            end if
        end for
        $candidates \leftarrow nextCandidates$
    end for
end while
```

	# triple	# classes	# properties	# instances	# keys	# pseudo-keys	runtime
DBPedia	13,8 M	250	1,100	1,668,503	2,945	6,422	179'48"
DrugBank	0,77 M	8	119	19,693	285	2,755	6'58"
DailyMed	0,16M	6	28	10,015	3	1,168	1'46"
Sider	0,19M	4	11	2,674	11	3	5"

Table 3.1: Datasets size, number of keys and pseudo-keys found, and computation time

The algorithm was parametrized with a support threshold $\lambda_s = 0.1$ and a discriminability threshold $\lambda_d = 0.99$.

Table 3.1 shows that computation time strongly depends on the number of keys found but is less sensible to datasets size. This is in line with the exponential complexity of the algorithm with regard to the number of properties. Even though computation times prevent an interactive usage, they show the approach can be generalized for very large datasets. In the Sider dataset, the computation time is very low. This can be explained because most of properties in sider are keys.

4. Representing Keys

Once computed, keys and pseudo-keys constitute a new body of knowledge that can be linked to the dataset as part of its metadata. We present in this section a small vocabulary allowing to represent keys and pseudo-keys in RDF. This vocabulary gives an alternative to the `owl:hasKey` property. As seen in Section 1, OWL keys expressed on the ontology imply that every dataset using the class must respect this key. When a key is not general enough to be applied to every datasets, it can be more convenient to attach it at the dataset level instead. Keys can be computed by analysing the dataset using an algorithm like the one introduced in this paper.

A convenient vocabulary to attach metadata to RDF datasets is the Vocabulary of Interlinked Datasets (VoID [1]). In particular, VoiD defines the class `void:Dataset` to represent datasets. Keys can thus be attached to datasets using this class as a hook.

The “Keys vocabulary” contains 1 class: Key, and 8 properties:

isKeyFor link a key to a VoID dataset.

hasKey indicates a key for a given class

property a property belonging to a key

nbProp the number of properties in a key

support support for a key as defined in Section 2

discriminability discriminability threshold for a key as defined in Section 2

instanceCount the number of instances for a class in the dataset

hasException subjects violating the key (in case of a pseudo-key).

This vocabulary is illustrated in the Figure 4.1 below. Keys computed for diverse datasets, including DBPedia, are published according to this vocabulary and available as linked-data on our server.[1]

[1]http://data.lirmm.fr/keys/

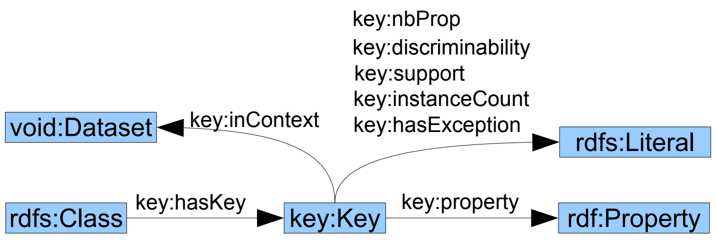

Figure 4.1: Key vocabulary

5. Applications

5.1 Datasets Interlinking

In the context of the Datalift project[1][7] we are building a platform for publishing raw data available in a structured format (CVS, XML, relational database) on the web of data. We have identified the following four necessary steps towards having a dataset published on the web according to the linked-data principles: (1) the selection of a vocabulary for describing the data, (2) the convertion of the raw data into RDF according to the selected vocabulary, (3) the publication of the data on a server providing content negociation, URI de-referencing and a SPARQL endpoint, and finally (4) the interlinking of the published data with other datasets available on the web. In this context, we are focusing on developing a method allowing to link data in an automated fashion.

The data linking problem is the following: given two datasets, how to find out what are the equivalent instances between them, that is what are the instances that represent the same real-world objects ? This problem is described in details in [3].

Our approach is based on the following process [8]: in a first step ontologies of both datasets are aligned. In a second step a set of properties is selected in order to minimize the number of comparisons required to identify similar instances. In a third step, similarity measures most fitting for comparing properties values are assigned to each property pair. These three steps allow to parametrize a tool that will compare instances. Many tools can be used for this task, for example [12, 6, 9]. If the first step can partly be automated, the last two need a large amount of manual input.

We propose here to use the key discovery algorithm in order to automate the second step selecting the minimal set of properties necessary for comparing instances. With that goal we execute the algorithm on the two datasets to be interlinked in order to compute every existing key in each dataset. We then select the smallest common key for the two datasets. Two keys are equivalent if they contain the same set of properties. We also consider properties linked using an alignment specifying their equivalence. We give below an illustration of this process on the two datasets Drugbank and Sider.

Drugbank[2] and Sider[3] are two databases on drugs. We want to interlink drugs, described by classes `drugbank:drugs` and `sider:drugs`. The datasets contain respectively 4772 and 924 drugs in their RDF versions[4] described by respectively 108 and 10 properties.

In this example, our approach consists in computing the smallest property set necessary to identify drugs shared by the two datasets. Execution of Algorithm 1 returns keys given in Table 5.0(a) and Table 5.0(b), ordered by decreasing support.

Analysis of these keys reveals the following: property `rdfs:label` is key for the two datasets. This property is thus a potential candidate for interlinking the datasets. Also, properties `drugbank:genericName` and `sider:drugName` are also candidates as they are keys in both datasets and they are equivalent properties. Finally, we can remark that `foaf:page` is also key in the two datasets, each drug having its web page in each dataset. But this property cannot be used for interlinking as the URL of these pages are not comparable, and it will be impossible to compute a similarity between drugs by comparing values of this property.

[1]Datalift Project (ANR-10-CORD-009). `http://datalift.org`

[2]`http://www.drugbank.ca/`

[3]`http://sideeffects.embl.de/`

[4]See `http://www4.wiwiss.fu-berlin.de/drugbank` and `http://www4.wiwiss.fu-berlin.de/sider/`

(a) Drugbank:drugs

Properties of the key	Support
foaf:page	1
db:genericName	1
db:primaryAccessionNo	1
db:updateDate	1
rdfs:label	1
db:limsDrugId	1
db:smilesStringCanonical db:drugType db:pubchemCompoundId db:creationDate	0.928
db:pubchemCompoundId db:drugType db:creationDate db:smilesStringIsomeric	0.928
db:pubchemSubstanceId	0.922

(b) Sider:drugs

Properties of the key	Support
si:siderDrugId	1
si:drugName	1
foaf:page	1
rdfs:label	1
si:stitchId	1
si:sideEffect	0.965
rdfs:seeAlso	0.848

Table 5.1: Examples of discovered keys

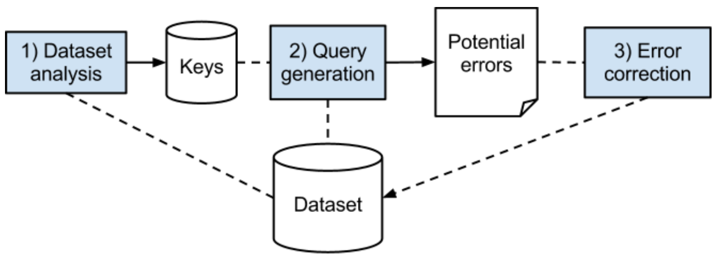

Figure 5.1: Error detection using keys: workflow

Properties of the key	Support
`http://dbpedia.org/ontology/deathDate` `http://dbpedia.org/ontology/birthDate`	0.203
`http://dbpedia.org/ontology/deathDate` `http://dbpedia.org/ontology/deathPlace`	0.216
`http://xmlns.com/foaf/0.1/name` `http://dbpedia.org/ontology/birthPlace`	0.442
`http://xmlns.com/foaf/0.1/surname` `http://purl.org/dc/elements/1.1/description`	0.459
`http://dbpedia.org/ontology/deathPlace` `http://dbpedia.org/ontology/birthDate`	0.480

Table 5.2: Key detection for the class `DBPedia:Person`

We could thus automate the key selection step for interlinking datasets. The above example is simple as the discovered keys only have one property. In practice, it is possible that multiple keys of varying size will be returned by the algorithm. A strategy for selecting the most relevant keys will thus be needed.

5.2 Error detection

Experimenting the algorithm introduced in this paper lead us to consider another application: the detection of errors in a dataset. When slightly relaxing the notion of keys by decreasing the discriminability threshold λ_d in order to detect pseudo-keys, we see appearing keys that are valid for most instances but are not keys for a small number of instances. Observation of these instances reveals the presence of duplicates or errors in the dataset. In order to find errors, we transform pseudo-keys found in that way into SPARQL queries to retrieve only instances having the same values for the properties of the key[5]. We then use the query results as a basis for error correction. This workflow is illustrated Figure 5.1.

We have applied this method on the 244 classes of DBPedia dataset using the algorithm introduced in this paper. We give below an example of pseudo-keys obtained for the class `dbpedia:Person` computed with a minimal support $\lambda_s = 0.2$ and a discriminability threshold $\lambda_d = 0.999$. Table 5.2 shows computed keys and their support.

First row of this table indicates there exist persons born on the same day who also died on the same day, which is not impossible but statistically rare. A verification can be performed by transforming pseudo-keys into SPARQL queries and executing them on the dataset.

[5]The transformation is performed by the program DuplicateFinder available at `https://gforge.inria.fr/projects/melinda/`

Class	duplicate	misclassification	other
dbpedia:Person	31	75	16

Table 5.3: Repartition of errors in the DBPedia class Person

The query must check what resources have same values for properties in the key for the given class.

We obtain the following query for the key (`dbpedia:birthPlace` , `dbpedia:deathPlace`):

```
SELECT DISTINCT ?x ?y
WHERE {
  ?x dbpedia-owl:deathDate ?dp1;
      dbpedia-owl:birthDate ?dp2;
      rdf:type dbpedia-owl:Person.
  ?y dbpedia-owl:deathDate ?dp1;
      dbpedia-owl:birthDate ?dp2;
      rdf:type dbpedia-owl:Person.
      MINUS {
         ?x dbpedia-owl:deathDate ?dpx1;
             dbpedia-owl:birthPlace ?dpy1 .
         ?x dbpedia-owl:deathDate ?dpx2 ;
             dbpedia-owl:birthPlace ?dpy2 .
         FILTER (?dpx1=?dpy1)
         FILTER (?dpx2=?dpy1)
      }
   FILTER (?x!=?y) }
```

In this example, the MINUS query pattern is not required because dbpedia-owl:birthDate and dbpedia-owl:deathDate are single valued properties. But in case of multivalued properties this operator is needed.

Manual analysis of the query results[6] shows the 124 instances pairs returned by the query in fact correspond to diverse types of errors in the dataset. The first kind of errors arises when two resources exist for describing a same object, for example
`dbpedia:Louis_IX_of_France__Saint_Louis___1` and
`dbpedia:Louis_IX_of_France`
A second kind of errors seems to be caused by the infoboxes extraction process when generating DBPedia. These errors most of the time lead to resource misclassification problems. For example:
`dbpedia:Timeline_of_the_presidency_of_John_F._Kennedy` is classified as a person although it is in fact a timeline.

Finally, a third kind of errors come from Wikipedia inconsistencies between the infobox and the article[7] or from documents from which these articles were informed.[8]
Table 5.3 below show error repartition for the class `dbpedia:Person`.

A systematic analysis of this experiment results on every DBPedia class goes out of this article scope. These results are available online in RDF[9]. This method can be reproduced on any dataset without any prior knowledge of the data.

[6]Query executed on the DBPedia SPARQL endpoint `http://dbpedia.org/sparql`

[7]See for example `http://dbpedia.org/resource/Phromyothi_Mangkorn` and `http://dbpedia.org/resource/Kraichingrith_Phudvinichaikul`

[8]See for example `http://dbpedia.org/resource/Merton_B._Myers` and `http://dbpedia.org/resource/William_J._Pattison` and the footnote at the end of these articles.

[9]`http://data.lirmm.fr/keys`

6. Related Works

The use of keys and functional dependencies for quality analysis and reference reconciliation of RDF data on the Web is attracting a lot of attention in the Semantic Web community.

The extraction of key constraints for reference reconciliation has been addressed by Symeonidou et al. in [11]. In this work the authors introduce KD2R as a method for automatic discovery of keys in RDF datasets. KD2R is based on the Gordian technique which allows to discover composite keys in relational databases with a depth-first search strategy. However, no experimental results concerning the run-time efficiency and scalability of the proposed algorithm are provided. The biggest dataset tested with KD2R contains only 3200 instances, whereas our algorithm has been tested with DBPedia with more than 1.5 million of instances.

Song and Heflin also rely on key discovery for data interlinking [10]. Their definition of a key is based on the notions of coverage and discriminability of a property. The coverage of a property is defined as the ratio of the number of instances of a class having that property to the total number of instances of that class. The discriminability of a property is the ratio of the number of distinct values for the property to the total number of instances having that property. Song and Heflin do not consider conjunction of properties, but single properties. A property is a key if it has coverage and discriminability equal to 1.

Instead of key constraints, Yu and Heflin rely on functional dependencies for quality analysis in RDF datasets [13]. In order to adapt the classical notion of functional dependencies in relational databases to the singularities of RDF datasets, the authors introduce the notion of value-clustered graph functional dependency.

Nonetheless, keys are not considered for quality analysis as they are pruned by their algorithm.

7. Conclusion

We have proposed a novel algorithm for computing keys and pseudo-keys in RDF datasets. The algorithm is efficient, even on large datasets thanks to pruning techniques based on minimal support for keys and avoiding generating redundancies.

We have demonstrated the need for such an algorithm in two applications: datasets interlinking and duplicates and error detection in data. Error detection allows to efficiently detect duplicates or correct errors on DBPedia persons. The approach still demands a substantial amount of work to identify the type of error. Also, tuning of the support and discriminability parameters is not trivial. A too high discriminability will lead in missing interesting pseudo-keys, while a too low discriminability will lead in getting properties with many common values, thus not showing any interesting information. Furthermore, parameters pertinence may varies across classes in a same dataset.

In future work we will continue to optimize the algorithm using ontology based optimization like exploiting the class and property hierarchy for pruning. We will also work on methods to optimize value selection for support and discriminability thresholds.

References

[1] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing linked datasets. In *Proceedings of the WWW2009 Workshop on Linked Data on the Web*, volume 538 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[2] Manuel Atencia, Jérôme David, and François Scharffe. Keys and pseudo-keys detection for web datasets cleansing and interlinking. In *EKAW*, pages 144–153, 2012.

[3] Alfio Ferrara, Andriy Nikolov, and François Scharffe. Data linking for the Semantic Web. *Int. J. Semantic Web Inf. Syst.*, 7(3):46–76, 2011.

[4] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[5] Heikki Mannila and Kari-Jouko Raiha. Algorithms for inferring functional dependencies from relations. *Data & Knowledge Engineering*, 12:83–99, 1994.

[6] Andriy Nikolov, Victoria S. Uren, Enrico Motta, and Anne N. De Roeck. Handling instance coreferencing in the KnoFuss architecture. In *Proc. of IRSW'08*, volume 422 of *CEUR Workshop Proceedings*, 2008.

[7] François Scharffe, Laurent Bihanic, Gabriel Képéklian, Ghislain Atemezing, Raphaël Troncy, Franck Cotton, Fabien Gandon, Serena Villata, Jérôme Euzenat, Zhengjie Fan, Bénédicte Bucher, Fayçal Hamdi, Pierre-Yves Vandenbussche, and Bernard Vatant. Enabling linked data publication with the datalift platform. 2012.

[8] François Scharffe and Jérôme Euzenat. MeLinDa: an interlinking framework for the web of data. *CoRR*, abs/1107.4502, 2011.

[9] François Scharffe, Yanbin Liu, and Chunguang Zhou. RDF-AI: an architecture for RDF datasets matching, fusion and interlink. In *Proceedings of IJCAI-09 Workshop on Identity and Reference in web-based Knowledge Representation (IR-KR at IJCAI-09)*, 2009.

[10] Dezhao Song and Jeff Heflin. Automatically generating data linkages using a domain-independent candidate selection approach. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *LNCS*, pages 649–664. Springer, 2011.

[11] Danai Symeonidou, Nathalie Pernelle, and Fatiha Saïs. KD2R: A key discovery method for semantic reference reconciliation. In *Proceedings of the 7th International IFIP Workshop on Semantic Web & Web Semantics (SWWS 2011)*, volume 7046 of *LNCS*, pages 392–401. Springer, 2011.

[12] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk - A link discovery framework for the Web of Data. In *LDOW'09 at WWW'09*, volume 538 of *CEUR Workshop Proceedings*, 2009.

[13] Yang Yu, Yingjie Li, and Jeff Heflin. Detecting abnormal semantic web data using semantic dependency. In *Proceedings of the 5th IEEE International Conference on Semantic Computing (ICSC 2011), Palo Alto, CA, USA, September 18-21, 2011*, pages 154–157. IEEE, 2011.