Keys and Pseudo-Keys Detection for Web Datasets Cleansing and Interlinking

Manuel Atencia[1,2], Jérôme David[1,3], and François Scharffe[4]

[1] INRIA & LIG, France
{manuel.atencia,jerome.david}@inria.fr
[2] Université de Grenoble 1, France
[3] Université de Grenoble 2, France
[4] Université de Montpellier 2 & LIRMM, France
francois.scharffe@lirmm.fr

Abstract. This paper introduces a method for analyzing web datasets based on key dependencies. The classical notion of a key in relational databases is adapted to RDF datasets. In order to better deal with web data of variable quality, the definition of a pseudo-key is presented. An RDF vocabulary for representing keys is also provided. An algorithm to discover keys and pseudo-keys is described. Experimental results show that even for a big dataset such as DBpedia, the runtime of the algorithm is still reasonable. Two applications are further discussed: (i) detection of errors in RDF datasets, and (ii) datasets interlinking.

1 Introduction

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

In the context of the Semantic Web, it is only since the release of OWL2 that modelling keys is possible. A key in OWL2 for a given class consists of a set of properties allowing to uniquely identify an instance of the class. According to the semantics of OWL2, two instances having the 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 the ontology. This is not compatible with the decentralized publication of datasets in the Web. However, the discovery of keys in RDF datasets allows to perform integrity checking such as duplicate detection. More interestingly, keys can be used to select sets of properties with which to compare data issued from different datasets.

In this paper we propose to discover potential keys in RDF datasets. Given the variable quality of web data, our approach allows to tolerate a few instances to have the same values for the properties of a key. In that case, we will use the

A. ten Teije et al. (Eds.): EKAW 2012, LNAI 7603, pp. 144–153, 2012.

term pseudo-key. We also put forward to associate discovered keys to the dataset as metadata by extending the VoID vocabulary [3].

The remainder of the paper is organized as follows. Section 2 includes formal definitions of key and pseudo-key dependencies in RDF datasets, and presents a brief description of an algorithm to discover keys and pseudo-keys. Section 3 shows experimental results. An RDF vocabulary for representing keys is given in Section 4. Two distinct applications are explained in Section 5. Related work is summarized in Section 6 and Section 7 concludes the paper.

2 Key and Pseudo-Key Dependencies in RDF Datasets

In this section we introduce the definitions of a key and a pseudo-key in an RDF dataset (Section 2.1). Then we briefly describe an algorithm for detecting keys and pseudo-keys in RDF datasets (Section 2.2). This algorithm is based on TANE [2], an algorithm for discovering functional approximate dependencies in relational databases.

2.1 Definitions

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*

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

2.2 Algorithm

The algorithm to compute keys and pseudo-keys in RDF datasets uses the same partition representation and breadth-first search strategy as TANE [2]. In order to prune the search space we discard any set of predicates including

- a key or a pseudo-key (for a given discriminability threshold λ_d),
- a subset the support of which is lower than a given threshold λ_s,
- a subset in which a functional dependency holds.

Unlike TANE, we discard properties by looking at their support. Indeed, this is useful because in RDF datasets properties may not be instantiated for each individual. In contrast, we cannot apply the optimization used in TANE based on stripping partitions (discarding singleton sets). Finally, since our goal is to find keys only, there is no need to test exhaustively all the functional dependencies.

3 Experimental Results

The key and pseudo-key discovery algorithm is implemented in Java. In order to improve time performance, datasets are locally stored on disk and indexed using Jena TDB[1]. We ran the algorithm on a quad-core Intel(R) Xeon(R) E5430 @ 2.66GHz computer with 8GB of memory. Table 1 shows the amount of time needed (computation+disk access) for computing all the keys and pseudo-keys for every class in DBpedia, DrugBank, DailyMed and Sider. The algorithm was parametrized with support and discriminability thresholds $\lambda_s = 0.1$, $\lambda_d = 0.99$.

Table 1. Datasets size, number of keys and pseudo-keys found, and runtime

	#triples	#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"

The runtime results of Table 1 agree with the fact that the number of minimal keys can be exponential in the number of properties. However, even for a dataset of the size of DBpedia with 13,8M of triples, the runtime is still reasonable.

4 An RDF Vocabulary for Representation 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 that allows to represent keys and pseudo-keys in RDF. This vocabulary gives an alternative to the `owl:hasKey` property. As mentioned in Section 1, OWL2 keys for a class expressed in an ontology imposes that every dataset using the class must respect the key. When a key is not general enough to be applied to every dataset, it might 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.

An adequate vocabulary to attach metadata to RDF datasets is the so-called Vocabulary of Interlinked Datasets (VoID) [3]. 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 "Key Vocabulary" contains 1 class (**Key**) and the following 8 properties (see also Figure 1):

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

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 2.1
discriminability discriminability threshold for a key as defined in Section 2.1
instanceCount the number of instances for a class in the dataset
hasException subjects violating the key (in case of a pseudo-key)

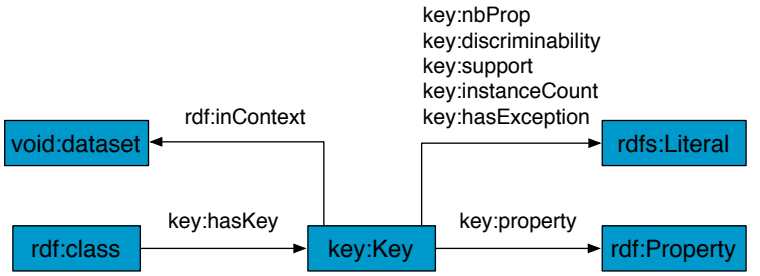

Fig. 1. Key vocabulary

Keys computed for diverse datasets, including DBPedia, are all published according to this vocabulary and available as linked-data on our server.[2].

5 Applications

Below we describe two applications: error detection (Section 5.1) and datasets interlinking (Section 5.2).

5.1 Error Detection

The discovery of pseudo-keys in a dataset may reveal the existence of duplicates or errors in the dataset. In order to find errors, each pseudo-key is transformed into a SPARQL query to retrieve the instances that have the same values for the properties of the pseudo-key.[3] The query results can be used as a basis for error correction. This workflow is illustrated in Figure 2.

We have applied this method over the 250 classes of DBPedia. Table 2 shows the pseudo-keys for the class `dbpedia:Person` computed with a minimal support $\lambda_s = 0.2$ and a discriminability threshold $\lambda_d = 0.999$.

The first row of Table 2 tells us that there exist persons who were born and dead in the same days, which is possible but unlikely to happen. The following query finds which resources of the class `DBPedia:Person` have the same values for `dbpedia:birthDate` and `dbpedia:deathDate`:

[2] `http://data.lirmm.fr/keys/`
[3] The transformation is performed by the program DuplicateFinder available at `https://gforge.inria.fr/projects/melinda/`

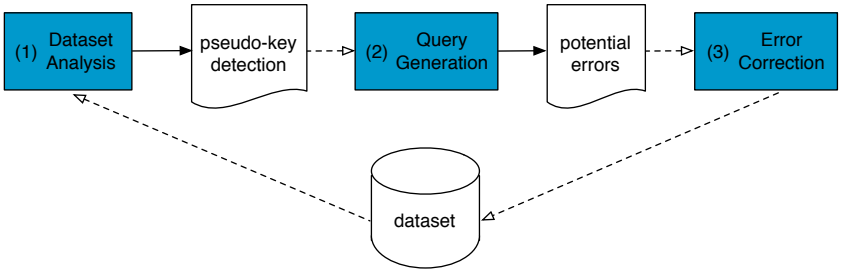

Fig. 2. Workflow for error detection using pseudo-keys

Table 2. Pseudo-key detection for the class `DBPedia:Person`

Pseudo-keys	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

```
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:birthDate ?dpx2.
         ?y dbpedia-owl:deathDate ?dpy1;
            dbpedia-owl:birthDate ?dpy2.
         FILTER (?dpx1!=?dpy1)
         FILTER (?dpx2!=?dpy2)
      }
   FILTER (?x!=?y) }
```

In this particular example, the MINUS query pattern is not required because `dbpedia-owl:birthDate` and `dbpedia-owl:deathDate` are both single valued properties, but it would be necessary in the case of multivalued properties.

The above query returned 122 pairs of instances.[4] Manual analysis confirmed several kinds of errors. A first type of error is due to the existence of resources describing the same object. For example, `dbpedia:Louis_IX_of_France` and `dbpedia:Louis_IX_of_France__Saint_Louis___1`. Now, a second type of error seems to be due to the process of infobox extraction when generating DBpedia. These errors usually lead to resource misclassification problems. For example, `dbpedia:Timeline_of_the_presidency_of_John_F._Kennedy` is classified as a person, even though it is actually a timeline. Finally, a third type of error is caused by Wikipedia inconsistencies between the infobox and the article,[5] or from documents from which these articles were created.[6] Table 3 shows error repartition for the class `dbpedia:Person`.

Table 3. Repartition of errors in the DBPedia class Person

Class	duplicate	misclassification	others
`dbpedia:Person`	31	75	16

An exhaustive analysis of the experiment results on every DBpedia class is out of the scope of this article. These results are available online in RDF.[7] This method can be reproduced on any dataset without any prior knowledge of the data.

5.2 Datasets Interlinking

The data interlinking problem can be formulated as follows: given two distinct datasets, which resources represent the same real-world objects? This problem is fully described in [4].

The discovery of keys (and pseudo-keys) in datasets can be helpful for the task of interlinking when combined with ontology matching techniques [5]. More specifically, we propose the following approach:

1. use the algorithm to detect keys in two datasets,
2. use an ontology matcher to find equivalent properties in the two datasets,
3. find instances that violate keys made up of equivalent properties,
4. relate these instances by means of `owl:sameAs`.

This is in line with the framework presented in [6], in the context of the Datalift project.[8] Below we illustrate the above process on the basis of the two datasets Drugbank and Sider.

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

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

[6] 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.

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

[8] `http://datalift.org`

Drugbank[9] and Sider[10] are two datasets on drugs. We would like to interlink drugs described by the classes `drugbank:drugs` and `sider:drugs`. The datasets contain, respectively, 4772 and 924 drugs in their RDF versions[11] described by 108 and 10 properties, respectively. Execution of our algorithm returned the keys shown in Table 4(a) and Table 4(b) and ordered by decreasing support.

Table 4. Key discovery for `drugbank:drugs` and `sider:drugs`

(a) Keys of `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) Keys of `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

Notice that the properties `drugbank:genericName` and `sider:drugName` are keys for the classes `drugbank:drugs` and `sider:drugs`, respectively. Note also that these properties are equivalent (in practice, ontology matchers can help to automatically discover equivalences between different properties). Our proposal is to identify instances that have the same values for `drugbank:genericName` and `sider:drugName`. This identification can be materialized by means of the property `owl:sameAs`. Furthermore, the property `rdfs:label` is also a key for the two datasets. Thus, `rdfs:label` is a potential candidate for interlinking the datasets. The property `foaf:page` is a key in the two datasets too. This means that each drug has a web page in each dataset. This property, however, is not useful for interlinking as the URL of these pages are hard to compare.

6 Related Work

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.

[9] `http://www.drugbank.ca/`
[10] `http://sideeffects.embl.de/`
[11] See `http://www4.wiwiss.fu-berlin.de/drugbank` and `http://www4.wiwiss.fu-berlin.de/sider/`

The extraction of key constraints for reference reconciliation has been tackled by Symeonidou et al. [7]. 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 [8]. Their definition of a key is different from the one proposed in this paper. It 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 [9]. 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 Conclusions

In this paper we have introduced a method for analyzing web datasets based on key dependencies. We have adapted the definition of a key in relational databases to RDF datasets and introduced the notion of the support of a key. The practical interest of computing "approximate" keys led us to define pseudo-keys on the basis of a discriminability threshold.

We have implemented an algorithm for the discovery of keys and pseudo-keys in RDF datasets. Even for a big dataset such as DBpedia, the runtime of this algorithm is reasonable. This is thanks to pruning techniques based on minimal support criterion and redundancy elimination.

Two applications are described: datasets interlinking, and duplicate and error detection in datasets. We have shown these on computed keys and pseudo-keys of DBpedia. Although a lot of work remains to be done, these applications show the potential of the proposed method.

As future work, we plan to optimize the algorithm by reasoning over class and property hierarchies. The choice of support and discriminability thresholds is not a trivial task, and we would like to look into it. For instance, a too high discriminability may lead to missing interesting pseudo-keys, while, on the other hand, a too low discriminability may lead to discovering very generic and meaningless pseudo-keys. The task of data interlinking is specially interesting

and we also plan to fully develop the approach based on key discovery and property matching described in this paper.

Acknowledgements. This work is supported under the grant TIN2011-28084 from the Ministry of Science and Innovation of Spain, co-funded by the European Regional Development Fund, and under the Datalift (ANR-10-CORD-009) and Qualinca (ANR-2012-CORD-012) projects, sponsored by the French National Research Agency.

The authors would like to thank Daniel Vila-Suero for his helpful comments during the preparation of this paper.

References

1. Mannila, H., Raiha, K.-J.: Algorithms for inferring functional dependencies from relations. Data & Knowledge Engineering 12, 83–99 (1994)
2. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: Tane: An efficient algorithm for discovering functional and approximate dependencies. Comput. J. 42(2), 100–111 (1999)
3. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing linked datasets. In: Proceedings of the WWW 2009 Workshop on Linked Data on the Web. CEUR Workshop Proceedings, vol. 538. CEUR-WS.org (2009)
4. Ferrara, A., Nikolov, A., Scharffe, F.: Data linking for the Semantic Web. Int. J. Semantic Web Inf. Syst. 7(3), 46–76 (2011)
5. Euzenat, J., Shvaiko, P.: Ontology matching. Springer (2007)
6. Scharffe, F., Euzenat, J.: MeLinDa: an interlinking framework for the web of data. CoRR abs/1107.4502 (2011)
7. Symeonidou, D., Pernelle, N., Saïs, F.: KD2R: A Key Discovery Method for Semantic Reference Reconciliation. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2011 Workshop. LNCS, vol. 7046, pp. 392–401. Springer, Heidelberg (2011)
8. Song, D., Heflin, J.: Automatically Generating Data Linkages Using a Domain-Independent Candidate Selection Approach. 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. 649–664. Springer, Heidelberg (2011)
9. Yu, Y., Li, Y., Heflin, J.: 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, pp. 154–157. IEEE (2011)