KD2R : a Key Discovery method for semantic Reference Reconcilation

Danai Symeonidou[1], Nathalie Pernelle[1], and Fatiha Saïs[1]

LRI (CNRS & Paris-Sud XI University)/INRIA Saclay,
LRI, Bât 650 UniversitéParis-Sud 11, 91405 Orsay Cedex France
Danai.Symeonidou@inria.fr,
{Nathalie.Pernelle, Fatiha.Sais}@lri.fr

Abstract. The reference reconciliation problem consists of deciding whether different identifiers refer to the same world entity. Some existing reference reconciliation approaches use key constraints to infer reconciliation decisions. In the context of the Linked Open Data, this knowledge is not available. We propose KD2R, a method which allows automatic discovery of key constraints associated to OWL2 classes. These keys are discovered from RDF data which can be incomplete. The proposed algorithm allows this discovery without having to scan all the data. KD2R has been tested on data sets of the international contest OAEI and obtains promising results.

1 Introduction

The reference reconciliation problem tries to find whether different references refer to the same entity (e.g. the same restaurant, the same gene, etc.). There are a lot of approaches (see [3] or [10] for a survey) that aim to reconcile data. Recent global approaches exploit the existing dependencies between reference reconciliation decisions [7, 2, 1]. In such approaches, the reconciliation of one reference pair may entail the reconciliation of another reference pair. A knowledge based approach is an approach in which an expert is required to declare knowledge that will be used by the reference reconciliation system [5, 2]. Some approaches such as [5] use reconciliation rules that are given by an expert, while other approaches such as [7] use the (inverse) functional properties (or the keys) that are declared in the ontology. Nevertheless, when the ontology represents many concepts and when data are numerous, such keys are not easy to model for the ontology expert.

The problems of key discovery in OWL ontologies and key discovery or Functional Dependency discovery in relational databases are very similar. In the relational context, key discovery is a sub-problem of extracting functional dependencies (FDs) from the data. [9] proposes a way of retrieving probabilistic FDs from a set of data sources. Two strategies have been proposed: the first one merges the data before discovering FDs while the second one merges the FDs obtained from each data source. This paper focuses on the problem of finding probabilistic FDs with only a single attribute in each side.In order to find the FDs, TANE [4] partitions the tuples into groups based on their attribute values. The goal is to find approximate functional dependencies: functional dependencies that almost hold. In the context of Open Likned Data, [6] have proposed a supervised approach to learn functional dependencies on a set of reconciled data.

There are a lot of works that deal with the discovery of FDs in relational databases, however only a few of them focus on the specific problem of retrieving keys. The Gordian method [8] allows discovering composite keys in relational databases. In order to avoid to checking all the possible combinations of candidate keys, the method proposes the discovery of the non-keys in a dataset and then using them to find the keys. In this method a prefix tree is built and explored (using a merge step) in order to find the maximal non keys. To optimize the tree exploration, they exploit the anti-monotone property of a non key. Nevertheless, it is assumed that the data are completely described (without null values). Furthermore, multivalued attributes are not taken into account.

In this paper, we present KD2R which is an automatic approach for key discovery in RDF data sources which conform to the same (or aligned) OWL2 ontology (the ontology can be represented in RDFS or in OWL). These keys are discovered from data sources where the UNA (Unique Name Assumption) is fulfilled and the information can be incomplete and multi-valued. To avoid scanning the whole data source, KD2R discovers first maximal non keys before inferring the keys. KD2R exploits key inheritance between classes in order to prune the non key search space. KD2R approach has been implemented and evaluated on two different data sources.

The paper is organized as follows: in section 2, we describe the data and the ontology model. In section 3, we present the KD2R and then we present first experiment results in section 4. Finally, in section 5 we conclude and give some future work plans.

2 Reference Reconciliation based on key constraints

2.1 Ontology and Data Model

Data are represented in RDF–Resource Description Framework– (`www.w3.org/RDF`). For example, the RDF source S1 contains the RDF descriptions of four museums in the form of a set of class facts and property facts (relational notation):

Source S1:
$ArchaeologicalMuseum(S1\_m1), museumName(S1\_m1, Archaeological\ Museum),$
$located(S1\_m1, S1\_c1), museumAddress(S1\_m1, 44\ Patission\ Street),$
$inCountry(S1\_m1, Greece), Museum(S1\_m2), museumName(S1\_m2,$
$Centre\ Pompidou), contains(S1\_m2, S1_p4), contains(S1\_m1, S1\_p5),$
$museumAddress(S1\_m2, 19\ rue\ Beaubourg), inCountry(S1\_m2, France),$
$Museum(S1\_m3), museumName(S1\_m3, Musee\ d'orsay),$
$museumAddress(S1\_m3, 62\ rue\ de\ Lille), inCountry(S1\_m3, France)$
$WaxMuseum(S1\_m4), museumName(S1\_m4, Madame\ Tussauds), located(S1\_m4,$
$S1\_c4), museumAddress(S1\_m4, Marylebone\ Road), inCountry(S1\_m4, England)$

The examined RDF data are in conformity with a domain Ontology represented in OWL2 (`http://www.w3.org/TR/owl2-overview`). The OWL 2 Web Ontology Language provides classes, (data or object) properties, individuals and data values. In the Museum ontology (see Figure 1), the class $Museum$ is described by its address (*owl:DataProperty* $museumAddress$), its location (*owl:ObjectProperty* $located$), its name (*owl:DataProperty* $museumName$) and its country (*owl:DataProperty*

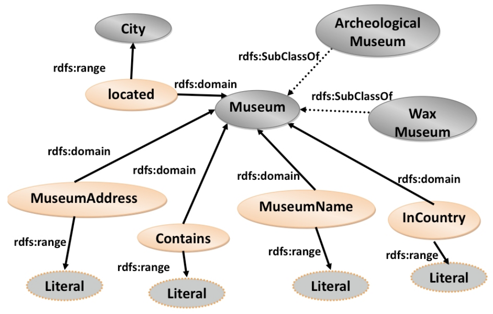

Fig. 1. Museum Ontology

$inCountry$). The classes $ArcheologicalMuseum$ and $WaxMuseum$ are subsumed by the class $Museum$.

In OWL2, it is possible to express key axioms for a given class: a key axiom HasKey(CE (OPE1 ... OPEm) (DPE1 ... DPEn)) states that each (named) instance of the class expression CE is uniquely identified by the object property expressions OPEi and by the data property expressions DPEj. This means that no two distinct (named) instances of CE can coincide on the values of all object property expressions OPEi and all data property expressions DPEj. An ObjectPropertyExpression is either an ObjectProperty or InverseObjectProperty. A data property expression is an *owl:DataProperty*. For example, we can express that the object property $located$ is a key for the class $City$ using $HasKey(kd2r : City(inverse(kd2r : museumAddress))())$

2.2 Constraint integration in Reference Reconciliation

LN2R [7] is a logical (L2R) and a numerical (N2R) method for reference reconciliation. L2R and N2R use the knowledge given in a OWL (or OWL2) ontology to reconcile data. L2R translates keys, disjunctions between classes and the Unique Name Assumption (UNA) [1] into reconciliation rules. These rules infer both (non) reconciliations facts and synonyms for literal values. For example, since $located$ is a key for the $City$ class (one museum is located in only one city) the following rule is generated by L2R:

$$City(L1) \wedge City(L2) \wedge Reconcile(X, Y) \wedge Located(X, L1) \wedge Located(Y, L2) \implies Reconcile(L1, L2)$$

A logical reasoning based on the unit-resolution inference rule is used to infer all the (non) reconciliations.

N2R exploits keys in order to generate a similarity function that computes similarity scores for pairs of references. This numerical approach is based on equations that model the influence between similarities. In the equations, each variable represents the (unknown) similarity between two references while the similarities between values of data properties are constants The functions modelling the influence between similarities are a combination of the maximum and the average functions in order to take into

[1] UNA declares that all the references that appear in a source cannot be reconciled.

account the keys declared in the OWL ontology in an appropriate way (see [7] for more details).

2.3 Key discovery Problem Statement

When RDF data are numerous, heterogeneous and published in clouds of data, the keys that are needed for the reconciliation step are not often available and cannot be easily specified by a human expert. Therefore, we need methods to discover them automatically from data. The key discovery has to face several kinds of problems, due to data heterogeneity: absence of UNA, syntactic variations in data, erroneous values and incompleteness of information. When UNA is not fulfilled, we cannot distinguish between the two cases: (i) two equal property values describing two references which refer to the same real world entity and (ii) two equal property values describing two references which refer to two distinct real world entities. This ambiguity leads to missing keys that can be discovered. In RDF data each instance of a class can be described by a subset of properties that are declared in the ontology. The incompleteness of data entails the discovery of keys that may be incorrect.

In this paper we focus on the problem of key discovery in RDF data when UNA assumption is declared for each data source and where there are no erroneous values.

3 KD2R: Key Discovery method for Reference Reconciliation

KD2R method aims to discover keys as exact as possible, with respect to a given dataset in order to enrich a possible existing key set.

The most naive automatic way to discover the keys is to check all the possible combinations of properties that refer to a class. The keys should uniquely identify each instance of a class. Let us assume that we have a class which is described by 15 properties in order to estimate the cost of this naive way. In this case the number of candidate keys is $2^{15} - 1$. In order to minimize the number of computations as much as possible we have proposed a method inspired from [8] which first retrieves the set of maximal non keys and then computes the set of minimal keys, using this set of non keys. Indeed, to make sure that a set of properties is a key we have to scan the whole set of instances of a given class. On the contrary, finding two instances that share the same values for the considered set of properties would suffice to be sure that this set of properties is a non-key.

We present, first, how we have defined non keys, keys and undetermined keys for a class in a given RDF data source and for a given set of RDF data sources. Then we will present the KD2R-algorithm that is used to find keys for the ontology classes.

3.1 Keys, Non Keys and Undetermined Keys

Let S be a data source for which the UNA is declared, and P_c be the set of RDF properties defined for a class C of the ontology O.

Definition 1 (Non keys). A set of property expressions $nk_{CSi} = \{pe_1, \ldots, pe_n\}$ is a non key for the class C in S if:

$\exists X \in S, \exists Y \in S$ s.t. $(C(X) \wedge C(Y) \wedge pe_1(X,a_1)) \wedge pe_1(Y,a_1) \wedge \ldots \wedge pe_n(X,a_n)) \wedge pe_n(Y,a_n)) \wedge X \neq Y)$

We denote NK_{CS} the set of non keys $\{nk_{CS1},...,nk_{CSm}\}$ of the class C w.r.t the data source S.

Definition 2 (Keys). A set of property expressions $k_{CSi} = \{pe_1,\ldots,pe_n\}$ is a key for the class C in S if:

$\forall X \in S, \forall Y \in S\ (C(X) \wedge C(Y)) \rightarrow (\exists pe_j \in k_{CSi}$ s.t. $pe_j(X,a)) \wedge pe_j(Y,b)) \wedge a \neq b)$

We denote K_{CS} the key set $\{k_{CS1},...,k_{CSm2}\}$ of the class C w.r.t the data source S.

Definition 3 (Undetermined Keys) A set of property expressions $uk_{CSi} = \{pe_1,\ldots,pe_n\}$ is an undetermined key for the class C in S if: (i) $uk_{CSi} \notin NK_{CS}$ and (ii) $\exists X \in S, \exists Y \in S$ s.t. $((C(X) \wedge C(Y) \wedge \forall pe_j \in uk_{CSi}(pe_j(X,a) \wedge pe_j(Y,b) \implies a = b) \wedge \exists pe_w \in uk_{CSi}$ s.t.$(\nexists pe_w(X,Z) \vee \nexists pe_w(Y,V))$

For example, $\{InCountry, Located\}$ is an undetermined key, since there are two museums in the same country but one of the cities is unknown. Hence, we cannot decide if it represents a key or a non-key.
We denote UK_{CS} the set of keys $\{uk_{CS1},...,uk_{CSm3}\}$ of the class C w.r.t the data source S.

Keys for a given set of data sources. Let $\mathcal{S} = \{S1, S2, \ldots, Sm\}$ be a set of m data sources for which the UNA is declared. Let $K_{CS1},\ldots,K_{CSm}$ be the respective set of keys of $S1, S2, \ldots, Sm$, the set of keys $Kc_{\mathcal{S}}$ that is satisfied in all the sources is the set of minimal keys that belong to the Cartesian product of $K_{CS1},\ldots,K_{CSm}$.

3.2 KD2R Algorithm

Given a set of datasets and a domain ontology, KD2R-algorithm allows to find keys for each instantiated class. It follows a top-down computation in the sense that the keys that are discovered for a class are inherited by its sub-classes. KD2R uses a compact representation of RDF data expressed in a prefix-tree in order to compute the complete set of maximal undermined keys and maximal non keys and then the complete set of minimal keys.

Prefix-Tree creation. In this section we will present the creation of the prefix-tree which represents the RDF descriptions of a given class.

As it is illustrated in Figure 2, each level of the tree corresponds to an instantiated property expression. Each node contains a variable number of cells. Each cell contains the distinct data property values or the distinct URIs of the object property expression of the considered level. Each cell contains the list of URIs referring to the corresponding class instances. Each non-leaf cell has a pointer to a single child node. Each unique prefix path represents the set of instances that share one value or one URI for all the properties involved in the path.

For the sake of simplicity we will use the term $value$ to either refer to basic values of data properties or to URIs of object properties.

In order to represent the cases where property values are not given (i.e. null values in relational databases) we create first an intermediate prefix-tree. In this intermediate prefix-tree, an artificial null value is created for those properties. Then, the final prefix-tree is generated by assigning the set of all the possible values to each artificial null value, i.e. those existing in the dataset.

Intermediate Prefix-Tree creation. In order to create the intermediate prefix-tree we use the set of all properties that appear at least in one instance of the considered class. For each instance, for each property and for each value if there is no cell which already contains the property value a new cell is created. Otherwise, the cell is updated by adding the instance URI to its associated list of URIs. When a property does not appear in the source, we create or update, in the same way, a cell with an artificial null value. Let it be noted that the intermediate prefix-tree creation is done by scanning the data only once.

Final Prefix-Tree creation. The final prefix-tree is generated from the intermediate prefix-tree by assigning the set of the possible values contained in the cells of this node to each artificial null value of a given node, if it exists. More precisely, the null cell is deleted and its URI list is added to all the other cells of the node. Then, for the descendants of this node, we recursively apply the processing of artificial null values and the node merge operation which is described in the following.

Node merge operation. This operation takes the list of nodes that need to be merged as input and provides a merged node which contains one cell per distinct value that exists in the input list of nodes as output. The new URI list of each cell contains all the URI lists of the merged cells. This merge operation is performed recursively for all the descendants of the considered nodes.

In figure 2, we give the final prefix-tree of the RDF data described in section 2.

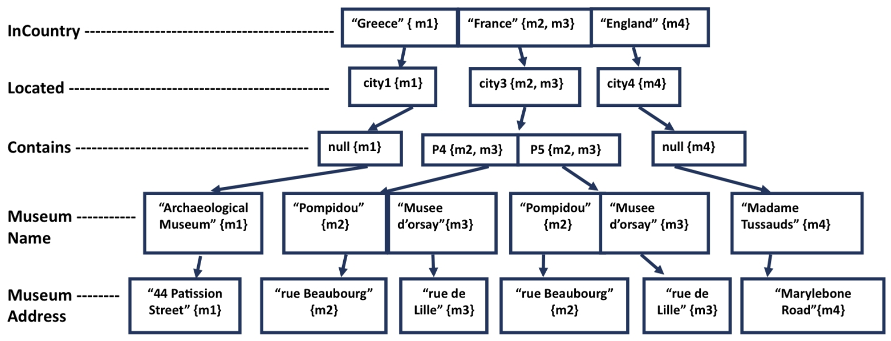

Fig. 2. Final prefix-tree for the museum class instances

Subsumption-driven Key Retrieval. For each set of RDF sources, the method $ClassKeyRetrieval$ applies a depth-first retrieval of the keys by exploiting the subsumption relation between classes declared in the ontology. $ClassKeyRetrieval$ method takes an instantiated class and a possible set of already known keys as input and calculates its complete set of keys. After creating the final prefix-tree of the considered class instances, the $UnKeyFinder$ method is called for retrieving the undetermined and the non keys. Then, the method is recursively called for the set of subclasses using the updated known key set.

```
ClassKeyRetrieval
Input: $C$: class; $KnownKeys$ :=set of known keys
Output: $CKeys$: the complete set of keys of the class $C$.
if $class$ has declared properties then
    $tripleList$.add(all triples of C)
    if $tripleList$ is not empty then
        $rootNode$ := Create-intermediate-prefix-tree(tripleList, $C$)
        $newRootNode$ :=Create-final-prefix-tree($rootNode$)
        $propNo := 0$
        $UNKeySet$ := UnKeyFinder($newRootNode$,$propNo$, $KnownKeys$)
        $keys := ExtractKeysFromUNKeySet(UNKeySet, C)$
        $CKeys := refine(KnownKeys.add(keys))$
    end if
end if
for all subClass $C_i$ of $C$ do
    ClassKeyRetrieval($C_i$, $CKeys$)
end for
return $CKeys$
```

UNK-Finder: Undermined Key and Non Key Finder. The UNKey is the set of undetermined keys and non-keys. The process of the algorithm begins from the root of the prefix tree and makes a depth-first traversal of it. The input of the algorithm is the current root of the tree, its attribute number and the known keys. This method searches the longest path p from the root to a node having a URI list containing more than one URI. p represents the maximal set of properties expressing either a non-key or an undermined key.

To ensure the scalability of the key discovery, KD2R performs two kinds of pruning: (i) the subsumption relation between classes is exploited to prune the key discovery thanks to the set of inherited keys, (ii) the anti-monotonic characteristic of the non-keys and undermined keys is also used to avoid computing the redundant non keys and undermined keys, i.e. if {ABC} is a non key (resp. an undermined key) then all the subsets of {ABC} are also non keys (resp. undetermined keys) and (iii) the monotonic characteristic of keys is used to avoid exploring the descendants of a node representing only one instance.

```
UNK-Finder
Input: $root$: node of the prefix tree; $propNo$: attribute Number; $knownKeys$: given keys.
Output: $UNKeySet$: the set of discovered undermined keys and non-keys.
add $propNo$ to the $curUNKey$
if $root$ is a leaf then
    for all $cells$ in the $root$ do
        if $cell.PropertyList > 1$ then
            add $curUNKey$ to the $UNKeySet$
            break
        end if
    end for
```

```
        remove $propNo$ from $curUNKey$
        **if** $root$ has more that one cell AND at least one of the cells has $PropertyList > 1$ **then**
            add $curUNKey$ to the $UNKeySet$
        **end if**
    **else**
        **if** there is only one URI **then**
            return
        **end if**
        **for all** $cells$ in the $root$ **do**
            **if** $curUNKey$ is not contained in knownKeys Set **then**
                UNK-Finder($cell.getChild$,$propNo$+1)
            **end if**
        **end for**
        remove $propNo$ from $curUNKey$
        **if** there is more that one $cell$ in the $root$ **then**
            **if** $curUNKey$ is already contained in the $UNKeySet$ **then**
                return
            **end if**
            $childNodeList :=$ all the children of the cells in the $root$ node
            $mergedTree$ := $mergeNodes(childNodeList)$
            **if** $curUNKey$ is not contained in $knownKeys$ Set **then**
                UNK-Finder($mergedTree$,$propNo$+1)
            **end if**
        **end if**
    **end if**
    **return** $UNKeySet$
```

Example of UNK-Finder. We illustrate the UNK-Finder algorithm on the final prefix-tree shown in figure 2. The method begins with the first node and more specifically with the cell containing the value " *Greece*". The property number of the cell, 0, is added on to $curUNKey$. Since the $URIList$ of this cell has size one -thanks to the pruning step- the algorithm will not examine its children. Now the property number is removed. The algorithm moves to the next cell of the root node. The property number 0 is added in the $curUNKey$. This cell contains a $URIList$ with two elements in it. So recursively, we go to node with cell "*city3*". Now the $curUNKey$ is $(0, 1)$. We call the UNK-Finder for the child node of the "*city3*". Now the root node is the node with paintings $P4$ and $P5$ and the property number of the node is added to the $curUNKey$ $(0, 1, 2)$. The process continues with the child node of cell P4. In the $curUNKey$, attribute number 3 is added. Since the cell "Pompidou" has $URIList$ of size one the UNK-Finder will not continue with the child node of "*Pompidou*". The method will continue with the second cell of the root node which is "Musee d'Orsay". Like "*Pompidou*", and since "*Musee d'Orsay*" has $URIList$ size one, the UNK-Finder will not be called for its child node. The UNK-Finder has been called for each cell of the node. The property number of the node is removed from the $curUNKey$ which now is $(0, 1, 2)$. In this step the child nodes of this node are merged and the UNK-Finder is applied to the mergedTree. The UNK-Finder is executed for the new merged node which consists of two cells, "*rue Beaubourg*" and "*rue de Lille*". The attribute number of the merged node is added in the $curUNKey$ which is $(0, 1, 2, 4)$. Since we are in the leaf and there is a cell in the node with $URIList$ bigger than one, the $curUNKey$ (0,1,2) is added in the $UNKeySet$. This means that there are more than two instances that have exactly the same values for the properties in the $curUNKey$. The process of discovering the $UNKeySet$ continues in the same way. In this specific example the final $UNKey$ set consists of one composite $UNKey$, which is {contains, located, inCountry}.

Extraction of keys from UNKeys. In order to compute the set of minimal keys from the $UNKey$ set, we first compute for each UNKey the complement set. Then we apply the Cartesian product on the obtained complement sets. Finally, we remove the non-minimal keys from the obtained multi-set of keys.

In the museum example we have only one $UNKey$ which is {contains, located, inCountry}. The complement set of this $UNKey$ is {{MuseumAddress}, {MuseumName}}. The process finishes by adding the two keys to the $KeySet$. The keys in the $KeySet$ are $MuseumAddress$ and $MuseumName$.

4 First experiments

We have implemented and tested our method on two datasets that have been used in the Ontology Alignment Evaluation Intitiative (OAEI, `http://oaei.ontologymatching.org/2010/`). UNA is declared for each RDF file of the two datasets. Since the two ontologies have been enriched by expert keys, we have compared our results to the set of these existing keys.

Restaurant dataset. The first dataset $D1$ describes 1729 instances (classes $Restaurant$ and $Address$). In the provided Ontology, Restaurants are described using the following properties: $name$, $phoneNumber$, $hasCategory$, $hasAddress$. Addresses are described using: $street$, $city$, $Inverse(hasAddress)$. The first RDF file f1 describes 113 $Address$ instances and 113 $Restaurant$ instances. The second RDF file f2 describes 641 $Restaurant$ instances and 752 $Address$ instances.

Person dataset. The second dataset D2 consists of 3200 instances of $Person$ and $Address$. In the ontology, a person is described by the following properties: $givenName$, $state$, $surname$, $dateOfBirth$, $socSecurityId$, $phoneNumber$, age and finally $hasAddress$. An Address is described by the properties: $street$, $houseNumber$, $postcode$, $isInSuburb$ and finally $inverse(hasAddress)$. The first and the second RDF files contain each of them 500 instances of the class $Person$ and 500 instances of $Address$. The third file contains 600 $Person$ instances and 600 $Address$ instances.

To examine the results of our method we compared the KD2R keys with the keys given by an expert. 10% of found keys are equal to the expert keys and 10% are bigger (i.e., contain more properties). The first case is the best we can come up with since our results agree with the expert ones. The second case arises when an expert makes a mistake and declares as keys properties that are not in fact real keys. This means that we detect erroneous keys given by an expert. For instance, the expert has declared that phoneNumber is a key. We are sure that the expert has made a mistake since in our data we can find two different restaurants with the same phone number (managed by the same organization). These two cases (20% of our found keys) represent the definite minimal keys that we extract using the given datasets. Another 20% of KD2R keys are keys that are smaller compared to the expert keys. It is possible to face this case when the given data are not sufficient to find more specific keys. Finally the 60% of the found keys are keys that are not declared by the expert. For example we find that $Inverse(hasAddress)$ can be a key for the address, a property that the expert did not take into account and seems to be relevant (a museum has only one address).

Thus, KD2R may find keys that are not specific enough (the more the data are numerous the more the discovered keys are accurate). However, this method can also find keys that are equal to the expert ones or keys which are missed by the expert.

5 Conclusions and Future work

In this paper, we have described the method KD2R which aims to discover keys in RDF data in order to use them in a reconciliation method. These data conform to the same ontology and are described in RDF files for which the UNA is fulfilled. KD2R takes into account the properties that the RDF files may have : incompleteness and multi-valuation. Since the data may be numerous, the method discovers maximal undetermined/non keys that are used to compute keys and merge them if keys are discovered using different datasets. Furthermore, the approach exploits key inheritance due to subsumption relations between classes to prune the key search for a given class. The first experiments have been conducted on two datasets exploited in the OAEI evaluation initiative. We have compared the retrieved keys with keys given by an expert. Some of the found keys are less specific than the expert ones but errors of the expert can also be detected. We plan to test our approach on more heterogeneous data and extend our method in order to be able to work even when the UNA is not fulfilled.

References

1. Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. ACM Trans. Knowl. Discov. Data 1 (March 2007)
2. Dong, X., Halevy, A., Madhavan, J.: Reference reconciliation in complex information spaces. In: Proceedings of the 2005 ACM SIGMOD. pp. 85–96. SIGMOD '05, NY, USA (2005)
3. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering 19, 1–16 (2007)
4. 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)
5. Low, W.L., Lee, M.L., Ling, T.W.: A knowledge-based approach for duplicate elimination in data cleaning. Information Systemes 26, 585–606 (December 2001)
6. Nikolov, A., Motta, E.: Data linking: Capturing and utilising implicit schema-level relations. In: Proceedings of Linked Data on the Web workshop collocated with WWW'2010 (2010)
7. Saïs, F., Pernelle, N., Rousset, M.C.: Combining a logical and a numerical method for data reconciliation. Journal on Data Semantics, vol 12 pp. 66–94 (2009)
8. Sismanis, Y., Brown, P., Haas, P.J., Reinwald, B.: Gordian: efficient and scalable discovery of composite keys. In: Proceedings of the 32nd International conference VLDB. pp. 691–702. VLDB '06, VLDB Endowment (2006)
9. Wang, D.Z., Dong, X.L., Sarma, A.D., Franklin, M.J., Halevy, A.Y.: Functional dependency generation and applications in pay-as-you-go data integration systems. In: 12th International Workshop on the Web and Databases (2009)
10. Winkler, W.E.: Overview of record linkage and current research directions. Tech. rep., Bureau of the Census (2006)