A study of the discovery and redundancy of link keys between two RDF datasets based on partition pattern structures

Nacira Abbas, Alexandre Bazin, Jérôme David, Amedeo Napoli

To cite this version:
Nacira Abbas, Alexandre Bazin, Jérôme David, Amedeo Napoli. A study of the discovery and redundancy of link keys between two RDF datasets based on partition pattern structures. CLA 2022 - 16th International Conference on Concept Lattices and Their Applications, Jun 2022, Tallinn, Estonia. pp.175-189. hal-03933868

HAL Id: hal-03933868
https://hal.science/hal-03933868
Submitted on 10 Jan 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
A Study of the Discovery and Redundancy of Link Keys Between Two RDF Datasets Based on Partition Pattern Structures

Nacira Abbas¹, Alexandre Bazin², Jérôme David³ and Amedeo Napoli¹,*

¹Université de Lorraine, CNRS, Inria, Loria, F-54000 Nancy, France
²Université de Montpellier, CNRS, LIRMM, F-34095 Montpellier, France
³Université Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France

Abstract
A link key between two RDF datasets \( D_1 \) and \( D_2 \) is a set of pairs of properties allowing to identify pairs of individuals \( x_1 \) and \( x_2 \) through an identity link such as \( x_1 \text{ owl:}\text{sameAs} x_2 \). In this paper, relying on and extending previous work, we introduce an original formalization of link key discovery based on the framework of Partition Pattern Structures (PPS). Our objective is to study and evaluate the redundancy of link keys based on the fact that \( \text{owl:sameAs} \) is an equivalence relation. In the PPS concept lattice, every concept has an extent representing a link key candidate and an intent representing a partition of instances into sets of equivalent instances. Experiments show three main results. Firstly, redundancy of link keys is not so significant in real-world datasets. Nevertheless, the link key discovery approach based on PPS returns a reduced number of non-redundant link key candidates when compared to a standard approach. Moreover, the PPS-based approach is efficient and returns link keys of high quality.

1. Introduction

In this paper, we are interested in data interlinking whose objective is to discover identity links across two RDF datasets over the web of data [1, 2, 3]. The same real-world entity can be represented in two RDF datasets by different subjects in RDF triples having the form \( (\text{subject, property, object}) \). Then, for cleaning data and providing data of better quality, it is meaningful to detect such identities. There are both numerical and logical approaches for discovering these identities. For example, interlinking methods have been implemented in systems such as LIMES [4] and SILK [5]. These systems use link specifications, i.e. rules that declare whether two IRIs should be linked. Link specifications can also be specified by users or learned from data [6, 7, 8].

In particular, link keys which are under investigation in this paper are special kinds of rules allowing to infer identity links between two RDF datasets. More formally, a link key is composed of two sets of pairs of properties \( \{(p_i, q_i)\}_i, \{(p'_j, q'_j)\}_j \) associated with a pair of...
classes \((c_1, c_2)\). Then, whenever an instance \(a_1\) of class \(c_1\) has the same (non empty) set of values as an instance \(b_1\) of class \(c_2\), i.e. \(p(a_1) = q(b_1)\) for all pairs of properties in the first set (universal quantification), and shares at least one value for all pairs of properties in the second set (existential quantification), i.e. \(p'(a_1) \cap q'(b_1) \neq \emptyset\), then \(a_1\) and \(b_1\) denote the same entity, i.e., an \(\text{owl:sameAs}\) relation can be established between \(a_1\) and \(b_1\). More concretely, the expression \(k=\{(\text{designation,title}),(\text{designation,title},\text{creator,author}),\text{(Book,Novel)}\}\) states that whenever an instance \(a\) of class \text{Book} has the same non empty values for the property \text{designation} as an instance \(b\) of the class \text{Novel} for \text{title}, and that \(a\) and \(b\) share at least one value for the properties \text{creator} and \text{author}, then \(a\) and \(b\) denote the same entity an \(\text{owl:sameAs}\) link can be established between \(a\) and \(b\).

Link keys are not provided with the datasets and algorithms are designed for automatically discovering such link keys \cite{9,10,11}. Given two RDF datasets, these algorithms reduce the search space and focus on the discovery of “link key candidates” instead of checking every combination of pairs of properties and pairs of classes. The notion of a link key candidate –made precise below– involves maximality and closure. Following this line, natural links were established in \cite{10,11} between link key discovery and Formal Concept Analysis (FCA \cite{12}). Indeed, FCA appeared as a suitable framework for the discovery of link key candidates, which are then evaluated thanks to appropriate quality measures \cite{9}.

Given two RDF datasets, FCA is applied in \cite{10} to a binary table where rows correspond to pairs of individuals and columns to pairs of properties. The intent of a resulting concept corresponds to a link key candidate, which remains to be validated thanks to suitable quality measures. The extent of the concept includes the potential identity links between individuals. A generalization of the former approach is proposed in \cite{11}, which is based on pattern structures \cite{13} and which takes into account different pairs of classes at the same time in the discovery of link keys.

Actually, “good” link key candidates over two RDF datasets have to generate different and maximal link sets, i.e., a mapping between individuals which is “close to a bijection”. However, it appears that two different link key candidates may generate the same link set when the link set is considered as a partition w.r.t. the \(\text{owl:sameAs}\) equivalence relation. This means that these two link key candidates present a certain redundancy, and then they can be considered as equivalent and merged in a way to be defined. This redundancy can be detected thanks to the properties of \(\text{owl:sameAs}\) as an equivalence relation, i.e. reflexivity, symmetry, and transitivity. Then, the \(\text{owl:sameAs}\) relation generates partitions among pairs of individuals that can be used for reducing the number of potential link key candidates. Indeed, two candidates relying on the same partition are considered as redundant and can be merged. However, we do not have a concrete idea of the importance of such redundancy and we should find a way to measure it. This is one objective of this paper to try to materialize this redundancy and to measure its importance.

For doing so, taking inspiration from the work carried out in \cite{14} on the discovery of functional dependencies, we provide a formalization of link key discovery based on “partition pattern structures” (pps), which allow us to take into account sets of equivalent individuals w.r.t. \(\text{owl:sameAs}\) as partitions. Then, a pattern concept represents a link key candidate and the related partition induced by the candidate. This approach is able to retrieve all link key candidates as a set of “non redundant” link keys. Moreover, this link key discovery process
based on pps is operational and original. Actually, this is the first time that the characteristics of owl:sameAs as an equivalence relation are considered, and that the related partitions of pairs of individuals are directly used for defining link key candidates. Thanks to pps, we are able to define redundancy of link key candidates and we also introduce a new measure based on the size of partitions for evaluating the quality of the discovered candidates. Finally, the experiments proposed in the last part of this paper provide three main results. Firstly, the redundancy of link keys in real-world datasets appears to be not so significant. Nevertheless, the current link key discovery approach based on pps is efficient and returns a reduced number of non-redundant link key candidates. Moreover, this pps-based approach returns link keys of very high quality when compared to competitors.

The summary of the paper is as follows. Section 2 presents some basics about link keys. Then the discovery of non-redundant link keys based on pattern structures and partition pattern structures is made precise in Section 3. A running example illustrates all these constructions. New quality measures related to non-redundant link keys are defined in Section 4. Finally, experiments in Section 5 show the capability and efficiency of the pps-based approach in link key discovery.

2. Preliminaries

This section introduces the basic definitions relative to data interlinking with link keys. We recall what an RDF dataset is and then we introduce two forms of link keys, namely link key expressions and link key candidates.

2.1. RDF Data

Definition 1 (RDF Dataset). Let $U$ denote a set of IRIs, i.e., “Internationalized Resource Identifiers”, $B$ a set of blank nodes, i.e., “anonymous resources”, and $L$ a set of literals, i.e., “string values”.

An RDF dataset is a set of triples $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$.

Let $D$ be an RDF dataset, $S(D) = \{s \mid \exists p, o (s, p, o) \in D\}$ denotes the set of individual identifiers, $P(D) = \{p \mid \exists s, o (s, p, o) \in D\}$ the set of property identifiers, and $C(D) = \{c \mid \exists s (s, \text{rdf:type}, c) \in D\}$ the set of class identifiers.

Moreover, $I(c) = \{s \mid \exists s (s, \text{rdf:type}, c) \in D\}$ denotes the set of instances of $c \in C(D)$ while $p(s) = \{o \mid (s, p, o) \in D\}$ denotes the set of objects –or values– associated with $s$ through property $p$.

An identity link is an RDF triple of the form $(a, \text{owl:sameAs}, b)$ stating that the IRIs $a$ and $b$ are referring to the same entity, alternatively $a$ and $b$ are denoting the same individual.

Example 1. In Figure 1, the RDF datasets $D_1$ and $D_2$ include $P(D_1) = \{p_1, p_2, p_3, p_4\}$ and $P(D_2) = \{q_1, q_2, q_3, q_4\}$ as sets of property identifiers, and $C(D_1) = \{c_1\}$ and $C(D_2) = \{c_2\}$ as class identifiers. In addition, $I(c_1) = \{a_1, a_2, a_3, a_4, a_5\}$ and $I(c_2) = \{b_1, b_2, b_3, b_4, b_5\}$ denote respectively the sets of instances of class $c_1$ and class $c_2$. Finally, considering subject $b_3$ and property $q_2$, the related set of objects or the “value” of $b_3$ for property $q_2$ is $q_2(b_3) = \{o_8, o_9\}$.

The present paper extends a short preliminary version published in [15].
2.2. Link Keys

Link keys are logical constructions allowing to infer identity links between instances of two classes lying in two RDF datasets. In the following we introduce two syntactic definitions of link keys, namely "link key expressions" and "link key candidates". A definition of a link key and its semantics using Description Logics interpretation is proposed in [16]. However, in this paper we will stick to these two syntactic definitions for the sake of simplicity.

Intuitively, a link key expression \( k = (E_q, I_n, (c_1, c_2)) \) is composed of two sets of pairs of properties, i.e., \( E_q \) and \( I_n \), where \( E_q \) is based on equality and \( I_n \) on non empty intersection, while links are generated between instances of classes \( c_1 \) and \( c_2 \). More formally we have:

**Definition 2** (Link key expression). Let \( D_1 \) and \( D_2 \) be two RDF datasets, \( k = (E_q, I_n, (c_1, c_2)) \) is a link key expression over \( D_1 \) and \( D_2 \) iff \( I_n \subseteq P(D_1) \times P(D_2), E_q \subseteq I_n, c_1 \in C(D_1) \) and \( c_2 \in C(D_2) \).

The set of links generated by \( k \) is denoted by \( L(k) \) and includes a set of pairs of instances \( (a, b) \in I(c_1) \times I(c_2) \) satisfying:

(i) for all \( (p, q) \in E_q \), \( p(a) = q(b) \) and \( p(a) \neq \emptyset \),

(ii) for all \( (p, q) \in I_n \setminus E_q \), \( p(a) \cap q(b) \neq \emptyset \).

The number of link key expressions may be exponential w.r.t. the number of properties. To reduce the search space, algorithms for link key discovery only consider "link key candidates", i.e., link key expressions which generate at least one link and which are "maximal" among the set of link key expressions.
Table 1
The pattern structure related to datasets $D_1$ and $D_2$ given in Fig. 1.

<table>
<thead>
<tr>
<th>PS objects ($g$)</th>
<th>descriptions ($\delta(g)$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a_1, b_1)$</td>
<td>${p_1, q_1}$</td>
</tr>
<tr>
<td>$(a_1, b_2)$</td>
<td>${p_2, q_1}$</td>
</tr>
<tr>
<td>$(a_2, b_1)$</td>
<td>${p_1, q_1}$</td>
</tr>
<tr>
<td>$(a_2, b_2)$</td>
<td>${p_1, q_1}$</td>
</tr>
<tr>
<td>$(a_3, b_1)$</td>
<td>${\forall(p_1, q_1), \exists(p_2, q_1)}$</td>
</tr>
<tr>
<td>$(a_3, b_3)$</td>
<td>${\forall(p_3, q_1), \exists(p_2, q_1)}$</td>
</tr>
<tr>
<td>$(a_4, b_2)$</td>
<td>${\exists(p_2, q_1), \forall(p_3, q_1)}$</td>
</tr>
<tr>
<td>$(a_5, b_5)$</td>
<td>${\exists(p_2, q_1), \forall(p_3, q_1)}$</td>
</tr>
<tr>
<td>$(a_4, b_4)$</td>
<td>${\forall(p_3, q_1), \exists(p_2, q_1)}$</td>
</tr>
</tbody>
</table>

**Definition 3** (Link key candidate). A link key expression $k_1 = (Eq_1, In_1, (c_1, c_2))$ is a link key candidate if:

(i) $L(k_1) \neq \emptyset$,

(ii) there does not exist another link key expression $k_2 = (Eq_2, In_2, (c_1, c_2))$ over $D_1$ and $D_2$ such that $Eq_1 \subset Eq_2$, $In_1 \subset In_2$, and $L(k_1) = L(k_2)$.

For example, consider the expressions $k_1$ and $k_2$:

(i) $k_1 = \{\{(p_1, q_1)\}, \{(p_1, q_1), (p_2, q_2)\}, (c_1, c_2)\}$,

(ii) $k_2 = \{\{(p_1, q_1)\}, \{(p_1, q_1)\}, (c_1, c_2)\}$.

The related link sets are $L(k_1) = L(k_2) = \{(a_3, b_3)\}$. Then $k_1$ and $k_2$ are both link key expressions but only $k_1$ is a link key candidate as it is maximal on the link set $\{(a_3, b_3)\}$ while $k_2$ is not.

The set of link key expressions is denoted by $\text{lke}$, the set of link key candidates by $\text{lkc}$, and we have that $\text{lkc} \subseteq \text{lke}$. The definition of link key candidates is based on the idea of maximality which involves a certain form of closure. This gave rise to a number of papers studying the potential relations existing between Formal Concept Analysis (FCA [12]) and the discovery of link keys [9, 10]. Following the same line, we make precise in the next section two ways of discovering link key candidates based on two extensions of FCA, pattern structures [13, 17] and partition pattern structures [14].

### 3. From Link Keys to Non Redundant Link Keys

#### 3.1. Discovering Link Keys with Pattern Structures

In this section we shortly recall how pattern structures can be used in the discovery of link keys (details can be read in [11, 18]). For the sake of simplicity and readability, we rely on a motivating example. Then we show how “Partition Pattern Structures” [14, 19] allow to discover the so-called “non-redundant link key candidates” denoted as $\text{nrlkc}$.

**Example 2.** Let us consider the pattern structure $\text{lkps} = (G, (E, \cap), \delta)$ displayed in Table 1. All details for building $\text{lkps}$ and the associated pattern concept lattice are given in [11]. In the rows (“PS objects”), $G$ includes pairs of related instances, i.e., $(a_i, b_j)$ with $a_i \in I(c_1)$ and $b_j \in I(c_2)$, which correspond to the objects of $\text{lkps}$. The set of potential descriptions $(E, \cap)$ includes all possible pairs of properties preceded either by $\forall$ or $\exists$. For the sake of simplicity, the $\cap$ operator corresponds here to set intersection.
is an Equivalence Relation.

– or simply generates an sameAs

Then, for example, in the extent of 𝑝𝑐

\(\{\text{(𝑎, 𝑏)}\}\) ≠ ∅

\(\exists(𝑝, 𝑞)\)

\(\forall(𝑝, 𝑞)\)

\(\{\text{(𝑎, 𝑏)}\}\) includes \(\exists(𝑝, 𝑞)\) which is subsumed by the pattern concept 𝑝𝑐, i.e., the intent \(\{\exists(𝑝, 𝑞)\}\) of 𝑝𝑐 is included in the extent of 𝑝𝑐 which is \(\{\text{(𝑎, 𝑏)}\}, \{\text{(𝑎, 𝑏)}\}, (\text{“(𝑎, 𝑏)”}), (\text{“(𝑎, 𝑏)”})\}

In this way, the set of all pattern concepts is organized within the pattern concept lattice in Fig. 2. Moreover, all link key candidates are lying in the intents of the pattern concepts.

3.2. Discovering Non Redundant Link Keys with Partition Pattern Structures

3.2.1. Motivation: sameAs is an Equivalence Relation.

Having a careful look at Fig. 2, one can verify that more compact link keys could be designed. For example, in the extent of 𝑝𝑐, \(\{\text{(𝑎, 𝑏)}\}, (\text{“(𝑎, 𝑏)”}), (\text{“(𝑎, 𝑏)”})\), the three first pairs have a non empty intersection when considered in a particular order, i.e., \(a_1\) is shared by \(\{\text{(𝑎, 𝑏)}\}, (\text{“(𝑎, 𝑏)”})\) and \(b_2\) is shared by \(\{\text{(𝑎, 𝑏)}\}, (\text{“(𝑎, 𝑏)”})\). Actually there exists a potential owl:sameAs – or simply sameAs – relation between the elements in such pairs. Being an equivalence relation, sameAs generates an

Figure 2: The pattern concept lattice related to Table 1. Every intent in a pattern concept corresponds to a link key candidate.
equivalence class. Hence, we can “merge” some of these pairs in such an equivalence class. For example \((a_1, b_1)\) and \((a_1, b_2)\) can be merged, i.e., \(a_1 \texttt{sameAs} b_1 \) and \(a_1 \texttt{sameAs} b_2\) yield \(b_1 \texttt{sameAs} b_2\) (symmetry and transitivity of \texttt{sameAs}). Moreover \((a_1, b_2)\) and \((a_2, b_2)\) can be merged because \(a_1 \texttt{sameAs} b_2\) and \(a_2 \texttt{sameAs} b_2\) yield \(a_1 \texttt{sameAs} a_2\). Finally, from \(a_1 \texttt{sameAs} a_2\) and \(a_1 \texttt{sameAs} b_1\) it comes \(a_2 \texttt{sameAs} b_1\). Then \(a_1, a_2, b_1, b_2,\) are in the same equivalence class w.r.t. \texttt{sameAs}.

Two important facts should be noticed: (i) the link \(a_2 \texttt{sameAs} b_1\), absent in \(pc_3\) but present in \(pc_4\), is inferred from the extent of \(pc_3\) thanks to the properties of \texttt{sameAs} while in the same way \(a_1 \texttt{sameAs} b_2\) could be inferred in \(pc_4\), (ii) the equivalence class \(\{a_1, b_1, a_2, b_2\}\) which can be built in \(pc_3\) and as well in \(pc_4\), allows us to merge the link keys in \(pc_3\) and \(pc_4\) because they have now the “same extent”. Relying on this observation, we define an equivalence relation over the extents of pattern concepts as follows.

Let us consider two RDF datasets \(D_1\) and \(D_2\), a link key \(k\), two classes \(c_1\) and \(c_2\), and the set of instances \(I = I(c_1) \cup I(c_2)\). Given \(x_0\) and \(x_n \in I\), there exists a chain of \texttt{sameAs} relations between \(x_0\) and \(x_n\) iff there exists a sequence of elements \(x_1, x_2, …, x_{n-1}\) such that \(x_0 \texttt{sameAs} x_1, x_1 \texttt{sameAs} x_2, …, x_{n-1} \texttt{sameAs} x_n\) holds, where the symmetry of \texttt{sameAs} may be used. Then, we define a relation between \(x_0\) and \(x_n\) in \(I\) w.r.t. the link key \(k\), denoted as \(x_0 \equiv_k x_n\), iff there exists a chain of \texttt{sameAs} relations between \(x_0\) and \(x_n\). It can be checked that \(\equiv_k\) is an equivalence relation as \texttt{sameAs} itself is an equivalence relation. For example, \(\text{Ext}(pc_3)/\equiv_{\texttt{sameAs}(p_3,q_3)} = \{(a_1, b_1, a_2, b_2), (a_1, b_3)\}\) where \(\text{Ext}(pc_3)\) denotes the extent of \(pc_3\). In the same way, \(\text{Ext}(pc_4)/\equiv_{\texttt{sameAs}(p_4,q_4)} = \{(a_1, b_1, a_2, b_2), (a_1, b_3)\}\). Then the two link keys \(\exists(p_1, q_1)\) and \(\exists(p_2, q_2)\) can be “identified” or “merged” because they have the same equivalence classes.

As a consequence, one may obtain more comparable sets of linked elements and thus minimize the number of possible link key candidates. Moreover, it can be noticed that an equivalence class determines a partition within the set of instances under study (singletons if any are omitted here).

### 3.2.2. The design of a PPS for “Non Redundant Link Keys”.

Hereafter, one main objective is to build a specific pattern structure where concepts are related to unique equivalence classes and yield “non redundant link key candidates” denoted as \texttt{nrlkc}. Here “non redundant” means any two elements in \texttt{nrlkc} are associated with different equivalence classes. In addition, an equivalence class corresponds to a partition of the set of instances. Accordingly, we define a “partition pattern structure” (pps) based on descriptions which are partitions and used to discover non redundant link key candidates. Indeed, in such a pps, objects in the rows correspond to pairs of properties and descriptions correspond to partitions. An example of pps is proposed in [14] where pps are introduced for mining functional dependencies.

Accordingly, we define a partition pattern structure over classes \(c_1\) and \(c_2\) as a triple \((\texttt{lkc}, \texttt{Part}(I), \sqcap_{\texttt{part}}, \delta)\) such as:

- \texttt{lkc} is the set of all link key candidates w.r.t. classes \(c_1\) and \(c_2\) which are given by a pattern concept lattice (as displayed in Fig. 2).
As all elements in the extents of the \( \text{Ext}(p_{c1}) \) and \( \text{Ext}(p_{c2}) \), while partitions involving elements are declared in the descriptions and thus will compose pps-concept intents. Moreover, partitions reduced to a singleton are omitted in the descriptions of the pps objects.

Let us examine a concrete example of pps based on the pattern structure given in Table 1. For \( p_{c3} \), we have \( \text{Ext}(p_{c3}) = \{(a_1, b_1), (a_1, b_2), (a_2, b_2), (a_3, b_3)\} \) and \( \text{Int}(p_{c3}) = \{\exists(p_2, q_3)\} \). The related partition, \( \text{Ext}(p_{c3})/\sim_{\exists(p_2,q_3)} = \{(a_1, b_1, a_2, b_2, (a_3, b_3)\} \), where singletons are omitted, is associated with row \( k_3 \) (\( \exists(p_2,q_3) \)) in Table 2. In the same way, considering \( p_{c4} \) and \( \{\exists(p_1, q_1)\} \), \( \text{Ext}(p_{c4})/\sim_{\exists(p_1,q_1)} = \{(a_1, b_1, a_2, b_2, (a_3, b_3)\} \) is associated with row \( k_4 \) (\( \exists(p_1,q_1) \)) in Table 2. Moreover, \( \text{Ext}(p_{c4})/\sim_{\exists(p_1,q_1)} = \{(a_1, b_1, a_2, b_2, (a_3, b_3)\} \) and is equal to \( \text{Ext}(p_{c3})/\sim_{\exists(p_2,q_3)} \). As already observed, \( k_3 \) and \( k_4 \) are inducing the same partition and can be "merged". The other pps objects, namely \( k_1, k_2, k_5, \) and \( k_6 \), are obtained from the corresponding pattern concepts in Fig. 2 in the same way. Finally, the pps over classes \( c_1 \) and \( c_2 \) related to the datasets \( D_1 \) and \( D_2 \) is displayed in Table 2.

The meet –or similarity– operation used to compare the rows in the pps corresponds to the meet of two partitions \( \text{part}_1 \) and \( \text{part}_2 \) over \( I \) and is denoted by \( \text{part}_1 \cap \text{part}_2 \). This meet operation is classically defined as the intersection of the respective equivalence classes, also known as the “coarsest common refinement of the two partitions” (see [14]). For example the meet \( \delta(k_4) \cap \text{part} \delta(k_2) \) is given by \( \{(a_1, b_1, a_2, b_2, (a_3, b_3)\} \cap \text{part} \{(a_1, b_1, a_2, b_2, (a_3, b_3)\} \) which yields the partition \( \{(a_1, b_1, a_2, b_2, (a_3, b_3)\} \). In particular, this means that \( \delta(k_4) \cap \delta(k_2) = \delta(k_5) \) and thus that \( \delta(k_5) \subseteq \delta(k_4) \).

Based on this meet operation, the pps concept lattice is constructed and shown in Fig. 3. The extent of a pps concept includes the \( Eq \) and the \( In \) parts of a link key candidate (\( Eq.In, (c_1,c_2) \)) in LKC. For example, the intent of \( pp_{sc_6} \) is \( \{a_3, b_3\} \) and its extent is \( k_6 \), i.e., \( \{\forall(p_1, q_1), \exists(p_1, q_1), \exists(p_2, q_2)\} \).

As all elements in the extents of the pps concept lattice, \( k_6 \) is an element of \( \text{nrlkc} \), i.e., a non redundant link key candidate. Moreover, \( k_6 \) corresponds to a closed set and thus verifies maximality, and it is non redundant w.r.t. the partition in the associated intent, i.e., no other link key candidate induces the same partition.

---

<table>
<thead>
<tr>
<th>pps objects ((k_i))</th>
<th>Descriptions (partitions, ( \delta(k_i)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(k_1 = \exists(p_2, q_4) \cup (p_2, q_4))</td>
<td>({(a_2, b_2), (a_3, b_3)})</td>
</tr>
<tr>
<td>(k_2 = \exists(p_2, q_3) \cup (p_2, q_3))</td>
<td>({(a_2, b_2), (a_3, b_3)})</td>
</tr>
<tr>
<td>(k_3 = \exists(p_2, q_2))</td>
<td>({(a_2, b_2), (a_3, b_3)})</td>
</tr>
<tr>
<td>(k_4 = \exists(p_1, q_1))</td>
<td>({(a_1, b_1, a_2, b_2, (a_3, b_3)})</td>
</tr>
<tr>
<td>(k_5 = \exists(p_1, q_1) \cup \exists(p_2, q_2))</td>
<td>({(a_2, b_2), (a_3, b_3)})</td>
</tr>
<tr>
<td>(k_6 = \forall(p_1, q_1) \cup \exists(p_2, q_2))</td>
<td>({(a_2, b_2)})</td>
</tr>
</tbody>
</table>

Table 2
The partition pattern structure related to the pattern structure in Table 1 over the classes \( c_1 \) and \( c_2 \) in datasets \( D_1 \) and \( D_2 \).

- \((\text{Part}(I), \cap_{\text{part}})\) is the set of potential descriptions or partitions and \( \cap_{\text{part}} \) is the "meet" of two partitions.
- \(\delta\) maps a link key candidate \( k \) to the partition \( I/\sim_k \), which is the quotient set of \( I \) w.r.t. \( \sim_k \).
3.2.3. Discussion: From LKC to NRLKC.

We now discuss what is gained in relying on pattern structures and then on partition pattern structures. Actually, the discovery of link key candidates (lkc) is based on the construction of the pattern concept lattice. Then the construction of the pps concept lattice yielding the non redundant link key candidates should be considered as a "refinement" where equivalent link key candidates, i.e., link key candidates inducing the same partition, are merged. For example, both concepts $pc_3$ and $pc_4$ in Fig. 2 are merged into a single concept $ppsc_{34}$ in Fig. 3. The resulting pps concept lattice is of smaller size and more concise as link key candidates in the concept extents are non redundant.

The relations existing between lkc and nrlkc can be characterized as follows. By construction, $nrlkc \subseteq lkc$ as the discovery of non redundant link keys is based on lkc, i.e., the link key candidates lying in the intents of concepts in the pattern concept lattice. Then, the candidates which have the same equivalence classes w.r.t. the sameAs relation are merged to produce the non redundant link key candidates in nrlkc. Thus, it comes that $nrlkc \subseteq lkc$ and that $|nrlkc| \leq |lkc|$ (where $|X|$ denotes the cardinality of set $X$). In other words, a non redundant link key candidate is always a link key candidate while the converse is not true.

The definition of nrlkc allows us to introduce a new quality measure for evaluating and validating non redundant link key candidates. In the next section we make precise this new quality measure and we discuss the benefits related to the discovery of non redundant link key candidates.

4. Quality Measures for Non Redundant Link Key Candidates

Not all link key candidates are eligible to be final link keys. Some candidates can be too general or related to noise in the data and thus they may generate owl:sameAs links which are not correct or have a bad quality. For example the link key candidate ({}; {{country,country}}, (Person,Human)) states that two individuals (subjects) sharing the same country represent the same person, and this is obviously not true. The candidate ({}; {{name,title}}, (Person,Book)) may be discovered by chance and will generate non correct links as well.

An ideal link key candidate should be “correct” and “complete”, and specific “quality measures” are defined for assessing these properties. In a supervised setting, when a set of reference links is available, the correctness of a link key candidate is measured thanks to “precision” and the completeness thanks to “recall”. In an unsupervised setting as this is the case here, the measures of “coverage” and “discriminability” are defined over the set of links directly generated by a link key candidate [9]. The global quality of a link key candidate is then estimated thanks to the harmonic mean of these two measures.

The coverage of a set $L$ of links over $c_1$ and $c_2$ is $cov(L, c_1, c_2) = \frac{|\pi_1(L) \cup \pi_2(L)|}{|I(c_1) \cup I(c_2)|}$ where $\pi_1(L) = \{a|(a, b) \in L\}$ and $\pi_2(L) = \{b|(a, b) \in L\}$.

The discriminability of $L$ is $dis(L) = \frac{\min(|\pi_1(L)|, |\pi_2(L)|)}{|L|}$. 


Coverage and discriminability evaluate how close a set of links is to a total, respectively bijective, mapping. Coverage is maximum when all instances of the considered classes are linked to at least another instance, while discriminability is maximum when instances are linked to at most one instance.

In the present setting, we are interested in the “redundancy” of link key candidates, and we would like to evaluate the quality of “non redundant” candidates. A link key candidate $k$ is said to be “redundant” in $H \subseteq lkc$ if there exists another link key candidate $h \in H$ such that $I/\equiv_k = I/\equiv_h$, i.e., $k$ generates the same partition as $h$. This is the case in the running example for the link key candidates $k_3$ and $k_4$. One main consequence of detecting redundancy is to reduce the number of candidates by merging candidates inducing the same partition. Then, since $\equiv_{k_3}$ and $\equiv_{k_4}$ induce the same partition, $k_3$ and $k_4$ are merged into a non redundant link key candidate $k_{34} = \{k_3, k_4\}$ interpreted as “$k_3$ or $k_4$”.

Accordingly, we introduce the new quality measure $p\text{Size}(k)$ that is compliant with the semantics of $\text{sameAs}$ and defined w.r.t. the partition associated with the equivalence relation $\equiv_k$:

$$p\text{Size}(k) = |\{x \in I/\equiv_k | |x| > 1\}|$$

where $I = I(c_1) \cup I(c_2)$ and $I/\equiv_k$ is the quotient set of $I$ w.r.t. $\equiv_k$.

For example, $p\text{Size}(k_3) = p\text{Size}(k_4) = |\{(a_1, b_1, a_2, b_2), (a_3, b_3)\}| = 2$. The lower bound of $p\text{Size}(k)$ is 1 since $k$ generates at least one link. The upper bound is $\min(|I(c_1)|, |I(c_2)|)$ and this is achieved when $k$ determines a 1-1 mapping. Moreover, $p\text{Size}$ can be normalized into $np\text{Size}(k) = p\text{Size}(k)/|I/\equiv_k|$. Then $np\text{Size}(k_3) = 2/6$ as $I/\equiv_{k_3} = \{(a_1, b_1, a_2, b_2), (a_3, b_3), a_4, b_4, a_5, b_5\}$, where this time singletons are counted.
The normalized measure \( npSize(k) \) evaluates how close are \( c_1 \) and \( c_2 \) w.r.t. \( \approx_k \), and is maximal when all instances are linked. However, this is not always satisfactory, especially when the cardinalities of the two classes significantly differ (i.e., classes are not balanced). Then, it is more accurate to maximize the measure as soon as all instances of one class are linked as follows: \( sspc(k) = pSize(k) / \min(|I(c_1)|/\approx_k, |I(c_2)|/\approx_k) \) (also known as the “Szymkiewicz–Simpson partition coefficient”). For example, \( sspc(k_3) = 2/4 \) with:
\[
sspc(k_3) = \left| \{(a_1, b_1, a_2, b_2), (a_3, b_3)\} \right| / \min\left( \left| \{(a_1, a_2, a_3, a_4, a_5)\} \right|, \left| \{(b_1, b_2, b_3, b_4, b_5)\} \right| \right).
\]

5. Experiments

5.1. Datasets and experimental settings

We run experiments on DB-Yago datasets provided in [20]. These datasets are used by the approaches to which we compare our results in the last series of experiments. They have been rewritten into Terse RDF Triple Language (Turtle) which is a syntax for expressing RDF data [21]. They are derived from DBpedia and Yago and organized into nine tasks. One particular task consists in finding links between instances of a class in DBpedia and instances of a class in Yago, e.g., db:Actor and yago:Actor. A set of reference links (i.e., \( \text{owl:sameAs} \) links) denoted by \( L_{\text{ref}} \) is provided for each task. The statistics of DB-Yago datasets are given in Table 3.

Experiments were run on a MacBook Pro 2018 with Intel Core i7-8850H@2.6 GHz, 16GB of RAM. The link key candidates are provided by a tool called Linkex\(^2\) in which we have implemented the \( \text{lkps} \) algorithm i.e. the link key discovery algorithm based on pattern structures proposed in [11]. A basic text normalization consisting in removing diacritics, tokenizing and sorting the resulting bag of tokens is performed.

5.2. Redundancy of Link Key Candidates is not so Significant

In Table 3, column \( |\text{lkc}| \) represents the number of link key candidates discovered by \( \text{lkps} \) algorithm while column \( |\text{nrlkc}| \) represents the number of non redundant link key candidates. In most of the tasks, we can observe that \( |\text{nrlkc}| \) is equal to \( |\text{lkc}| \), except for the tasks \( \text{Actor} \) and \( \text{Film} \) where \( |\text{nrlkc}| \) is lower than \( |\text{lkc}| \) by 1% and by 5% respectively. This means that, in general, link key candidates produce different partitions and only a few are redundant. By contrast, if redundancy does not significantly reduce the number of link key candidates, it gives a good idea of the compactness of partitions related to link key candidates.

Nevertheless, even if redundancy is not so significant, we decided to compare the quality of the resulting sets of candidates measured with \( pSize \) and the results of competitors, namely keys and conditional keys as they are studied in [20].

5.3. Non Redundant Candidates, Classical Keys and Conditional Keys

The \( pSize \) measure is used to evaluate and select the best link key candidates, –in this experiment the link key candidate ranked first– in every task listed in Table 3. Then, we compare recall, precision, and F-measure of the links generated by the best link key candidates selected by

\(^2\)Linkex is available online at https://gitlab.inria.fr/moex/linkex.
In Figure 4 we can observe that: (i) recall of link keys is significantly better than recall of classical and conditional keys, (ii) precision of classical and conditional keys is slightly better than precision of link keys in most of the tasks, (iii) F-measure is much higher for selected link key candidates than for classical and conditional keys. Link keys have a better recall because they are more flexible than classical keys, i.e., a link key is not necessarily a pair of keys and an instance of class \( c_1 \) may be linked to many instances of class \( c_2 \).

For summarizing, it can be concluded that considering the best link key candidates selected by \( pSize \) will ensure a higher interlinking quality compared to classical and conditional keys. We also observe that the best link key according to discriminability and coverage is the same that the best non redundant link key selected thanks to \( pSize \). Actually, the Actor dataset makes an exception: the link key candidate selected with \( pSize \) obtains an F-measure of 0.95 contrasting the score of 0.34 obtained with coverage and discriminability. In addition, contrasting key-based approaches, link key discovery does not require any prior knowledge such as property or class alignments.

| Interlinking task | datasets         | |triples| |subjects| |properties| |LKC| |NRLKC| |time|
|-------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Actor             | db:Actor        | 94 606          | 1 029 580       | 5 807           | 108 415         | 16              | 16              | 2 198           | 2 177           | 56s             |
|                   | yago:Actor      |                 |                 |                 |                 |                 |                 |                 |                 |                 |
| Album             | db:Album        | 594 144         | 762 238         | 85 002          | 136 848         | 5               | 5               | 44              | 44              | 28s             |
|                   | yago:Album      |                 |                 |                 |                 |                 |                 |                 |                 |                 |
| Book              | db:Book         | 247 372         | 185 032         | 29 846          | 41 849          | 7               | 7               | 82              | 82              | 39s             |
|                   | yago:Book       |                 |                 |                 |                 |                 |                 |                 |                 |                 |
| Film              | db:Film         | 1 369 600       | 1 067 084       | 82 099          | 123 822         | 9               | 9               | 18 718          | 17 643          | 34m             |
|                   | yago:Film       |                 |                 |                 |                 |                 |                 |                 |                 |                 |
| Mountain          | db:Mountain     | 135 442         | 233 562         | 16 397          | 32 874          | 5               | 5               | 39              | 39              | 5m              |
|                   | yago:Mountain   |                 |                 |                 |                 |                 |                 |                 |                 |                 |
| Museum            | db:Organization | 4 487 205       | 4 410 854       | 183 665         | 430 071         | 17              | 17              | 1 425           | 1 425           | 57m             |
|                   | yago:Organization |             |                 |                 |                 |                 |                 |                 |                 |                 |
| Organization      | db:Scientist    | 128 360         | 671 266         | 18 409          | 92 828          | 10              | 18              | 862             | 862             | 2m              |
|                   | yago:Scientist  |                 |                 |                 |                 |                 |                 |                 |                 |                 |
| University        | db:University   | 241 838         | 263 624         | 10 352          | 23 334          | 9               | 9               | 213             | 213             | 56s             |
|                   | yago:University |                 |                 |                 |                 |                 |                 |                 |                 |                 |

**Table 3**

DB-Yago datasets statistics, where |triples| denotes the number of triples in each dataset, |subjects| the number of instances, and |properties| the number of properties.
Figure 4: Interlinking performances of classical keys, conditional keys as given in [20] and link keys. Actually a recall of 0 stands for a very small recall close to 0.

6. Synthesis and Conclusion

This paper introduces a formalization of link key discovery based on partition pattern structures (pps). This approach allows to discover the set \( \text{nrlkc} \) of link key candidates which are not redundant w.r.t. the \( \text{owl:sameAs} \) equivalence relation, while still being correct and complete. In addition, new appropriate quality measures are proposed.

Practically, we observe in experiments that the redundancy of link key candidates in public datasets is rather rare. Nevertheless, the experiments and the associated results show that link key based on \( \text{pps} \) obtain better F-measure values than two other key-based approaches to data-interlinking.

Among the perspectives, a first one is to consolidate the theory and practice of link key discovery based on \( \text{pps} \) introduced and detailed in this paper. A second and very important direction of investigation is related to the discovery of “fuzzy link keys”. We make the hypothesis that the redundancy of link key candidates is rather rare because we are using the crisp equality operator when we are building the link key candidates. Instead, in considering the discovery of link key candidates as depending on a similarity relation, i.e., reflexive and symmetric –also called a tolerance relation– rather than on an equality, then we could propose a formalization of fuzzy link key candidates in the spirit of approximate-matching dependencies as they are studied in [19]. We could also expect much more differences between crisp and fuzzy link key candidates. In addition, a reduction of the size of the concept lattice related to \( \text{nrlkc} \) could also be investigated thanks to similarities between partitions.

References


[18] N. Abbas, A. Bazin, J. David, A. Napoli, Sandwich: An Algorithm for Discovering Relevant

