Discovering a Representative Set of Link Keys in RDF Datasets

Nacira Abbas[1], Alexandre Bazin[2], Jérôme David[1], and Amedeo Napoli[3]

[1] Université Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France nacira.abbas@univ-grenoble-alpes.fr,jerome.david@inria.fr
[2] Université de Montpellier, CNRS, LIRMM, F-34095 Montpellier, France alexandre.bazin@lirmm.fr
[3] Université de Lorraine, CNRS, Loria, F-54000 Nancy, France amedeo.napoli@loria.fr

Abstract. A link key is based on a set of property pairs and can be used to identify pairs of individuals representing the same real-world entity in two different RDF datasets. Various algorithms are aimed at discovering link keys which usually output a large number of candidates, making link key selection and validation a challenging task. In this paper, we propose an approach combining Formal Concept Analysis (FCA) for discovering link key candidates and building a link key lattice, and then hierarchical clustering over a given set of candidates for building a representative set of link keys. Such a link key set should minimize the number of candidates to be validated while preserving a maximal number of links between individuals. The paper also provides a series of experiments which are performed over different RDF datasets, showing the effectiveness of the approach and the ability of hierarchical clustering to return a concise and meaningful set of candidates while preserving the ordinal structure of the link key lattice.

Keywords: Link Key Discovery · Formal Concept Analysis · Hierarchical Clustering · RDF Dataset · Representative Set.

1 Introduction

The present research work relies on the discovery of *link keys*, which are expressions composed of sets of property pairs and a class pair, allowing to identify two individuals lying in different RDF datasets. For example, let us consider the link key $(\{(dsg, tit)\}, \{(dsg, tit), (cre, aut)\}, (book, novel))$. Whenever an instance a of class *book* in dataset D_1 has the same values for dsg (designation) as an instance b of class *novel* in dataset D_2 for tit (title), and in addition a and b share at least one value for cre (creator) and aut (author), then an identity link $(a, \texttt{owl:sameAs}, b)$ can be generated between a and b, and it can be inferred that a and b denote the same entity. More formally, a link key is composed of two sets of pairs of properties and a pair of classes, i.e., $(\{(p_i, q_j)_{i\in I_1, j\in J_1}\}, \{(p_i, q_j)_{i\in I_2, j\in J_2}\}, (c_1, c_2))$. The first set of properties corresponds to a *universal quantification*, i.e., the sets of attached values should be equal, and the second set to an *existential quantification*, i.e., the sets of attached values should have a non-empty intersection. The application of link keys across two RDF datasets can be viewed as a data cleaning task, allowing duplicate identification and thus improving data quality.

Given two RDF datasets D_1 and D_2, every combination of property pairs and class pairs can be potentially considered as a link key expression. Then, link key discovery can be considered as a knowledge discovery problem, and efficient data mining algorithms should be designed for reducing the search space and mining interesting and useful link keys. Several dimensions should be taken into account to guide the mining process, among which *maximality*, *discriminability*, and *coverage*. Accordingly, a link key should generate a maximal link set, while the mapping between the pairs of instances in D_1 and D_2 should be close to a one-to-one mapping. In this paper, link key discovery relies on a specific algorithm based on Formal Concept Analysis (FCA) [10]. The algorithm takes as input two RDF datasets and returns a concept lattice, called the LK-lattice, where each concept encapsulates a so-called *link key candidate* generating a maximal set of potential identity links [2,7]. Then, discriminability and coverage are controlled thanks to adapted quality measures that enable the ranking and the validation of link keys.

Meanwhile, the number of discovered link key candidates can still be (very) large and some candidates may be preferred, raising a representation problem: is it possible to design a *compact* and *representative set* of candidates which preserves a maximal number of identity links and which can be navigated by a domain analyst? Finding such a set amounts to designing an algorithm capable of selecting a set of representative candidates preserving the largest part of the identity links. The size of the subset of candidates and the number of preserved links are suitable characteristics to be considered for computing such a representative subset of candidates. While optimizing at the same time the size of the set of candidates and the number of preserved links is not an easy task, an acceptable compromise can be achieved.

Accordingly, our objective is to propose the LKCLUST algorithm that builds a representative set of link keys, abbreviated as REPLKSET, including a minimal number of link keys and preserving a maximal number of identity links. Such a REPLKSET can be presented to domain analysts for validation in data interlinking [13], e.g., for detecting duplicates in library management or cleaning hand-made datasets. The algorithm LKCLUST is based on agglomerative hierarchical clustering (AHC [11]). As input, it takes an LK-lattice, a dissimilarity measure defined w.r.t. the links generated by the candidates, a cutting level, and a linkage criterion. As output it returns a REPLKSET. LKCLUST combines both FCA and hierarchical clustering for reducing a large search space of candidates in an original way, as the clustering process respects the LK-lattice ordering and every cluster corresponds to a concept interval in the LK-lattice. The present work proposes a follow-up to [1] where the reduction of the link key candidate sets is based on crisp set equality in the framework of FCA and partition pattern structures. Here we propose an alternative and extend the preceding purpose by considering set similarity and clustering.

There are several close approaches in data interlinking which are based on keys [14,15,3,16] and on link keys [8,7,6]. In both cases, keys and link keys can be seen as rules allowing to infer links between individuals, and can be used for checking data consistency as this is performed with keys and functional dependencies in database management systems. However, while keys are attached to only one dataset[4], link keys are involving two different datasets [6].

The summary of the paper is as follows. In Section 2 we recall useful basics about link keys, FCA, and hierarchical clustering. Then, in Sections 3 and 4, we introduce the characteristics of a RepLkSet and then the lkclust algorithm for building such a RepLkSet. In Section 5, we report experiments performed on different RDF datasets that demonstrate the effectiveness of the present approach, before concluding.

2 Background

2.1 Link Key Candidates and Link Sets

In the following, an RDF dataset D is composed of a set of triples $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$, where U is a set of IRIs[5], B is a set of blank nodes, and L is a set of literals. In (s, p, o), s denotes the subject, p the property or predicate, and o the object or value. Moreover, $C(D) = \{c \,|\, \exists\, s \ (s, \texttt{rdf:type}, c) \in D\}$ denotes the set of class identifiers in D, $I(c) = \{s \,|\, \exists\, s \ (s, \texttt{rdf:type}, c) \in D\}$ the set of instances of class $c \in C(D)$, $P(D) = \{p \,|\, \exists\, s, o \ (s, p, o) \in D\}$ the set of property identifiers in D, and $p(s) = \{o \,|\, (s, p, o) \in D\}$ the set of objects –or values– related to s through property p.

Given two RDF datasets D_1 and D_2, $k = (Eq, In, (c_1, c_2))$ is a link key expression composed of two sets of property pairs Eq and $In \subseteq P(D_1) \times P(D_2)$, with $Eq \subseteq In$, $c_1 \in C(D_1)$, and $c_2 \in C(D_2)$. For all for all $(p, q) \in Eq$, $p(a) = q(b)$ and $p(a) \neq \emptyset$, where $a \in c_1$ and $b \in c_2$, i.e., Eq is based on equality and corresponds to a $\forall$ quantifier. Moreover, for all $(p, q) \in In, p(a) \cap q(b) \neq \emptyset$, where $a \in c_1$ and $b \in c_2$, i.e., In is based on non-empty intersection and corresponds to an $\exists$ quantifier.

When such a link key expression is verified for an instance $a \in c_1$ and an instance $b \in c_2$, an identity link of the form $(a, \texttt{owl:sameAs}, b)$, actually an RDF triple, can be generated. For example, in Figure 1, subjects a_3 and b_3 are sharing v_4 through (p_1, q_1) and have the same value v_5 for (p_2, q_2). Then the link key $(\{(p_2, q_2)\}, \{(p_1, q_1), (p_2, q_2)\}, (c_1, d_1))$ generates the potential identity link (a_3, b_3), with $Eq = \{(p_2, q_2)\}$ and $In = \{(p_1, q_1), (p_2, q_2)\}$.

[4] The OWL2 construction `HasKey` allows keys to be defined for a given class, stating that each named instance of a class is uniquely identified by a property or a set of properties, as keys in a database system (see `https://www.w3.org/TR/owl2-syntax/`).

[5] Internationalized Resource Identifier.

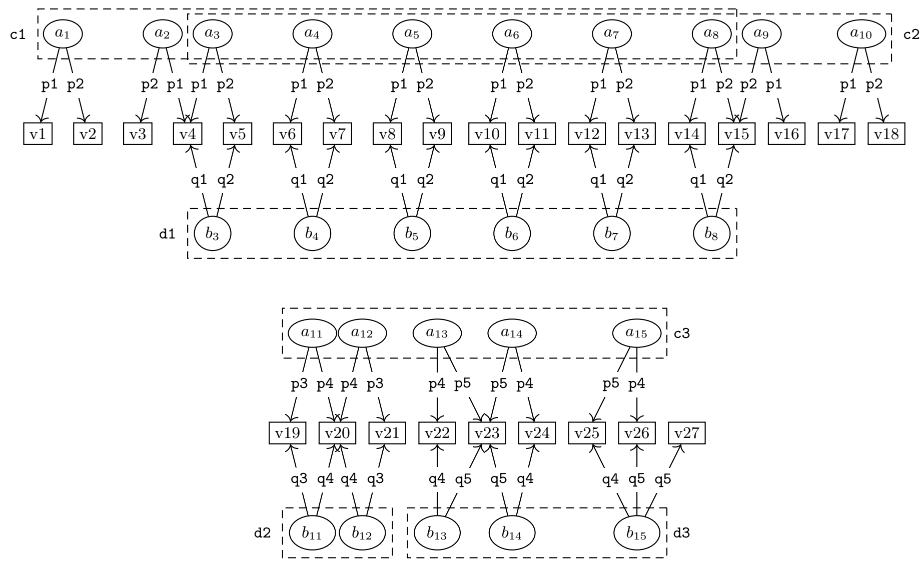

Fig. 1: Two examples of RDF datasets. Dataset D_1 includes instances prefixed by a and dataset D_2 those prefixed by b. D_1 includes classes $c1$, $c2$, $c3$, and D_2 contains classes $d1$, $d2$, and $d3$.

The expression $k_1 = (Eq_1, In_1, (c_1, c_2))$ over D_1 and D_2 denotes a *link key candidate* if the set of potential links generated by k_1 is not empty, i.e., $L(k_1) \neq \emptyset$, and k_1 is *maximal*. The latter means that there should not exist an expression $k_2 = (Eq_2, In_2, (c_1, c_2))$ such that $In_1 \subset In_2$, $Eq_1 \subset Eq_2$, and $L(k_1) = L(k_2)$. In the following, we will simply write "candidate" if there is no ambiguity. In addition and for the sake of simplicity, we will only consider the In part in a link key expression as $Eq \subseteq In$, i.e., $k = (In, (c_1, c_2))$.

The objective of link key discovery is to mine candidates in the possibly very large power set $P(D_1) \times P(D_2)$. The notion of candidate involves maximality, meaning that a candidate should be maximal and unique among a set of link key expressions generating the same link set. Maximal sets attached to a given relation, here inclusion, are usually related to a closure operator. This was one main reason for mining link key candidates thanks to Formal Concept Analysis (FCA [10]), as introduced in [5] and then revisited in [7]. The next section makes this approach more precise.

2.2 Link Key Discovery based on FCA

We recall hereafter basics of FCA for allowing a good understanding of the paper. FCA [10,9] is a mathematical framework based on lattice theory and aimed at data analysis and classification. The basic data structure in FCA is a context $K = (G, M, I)$ where G denotes a set of objects, M a set of attributes, and $I \subseteq G \times M$ a binary relation indicating that object g has attribute m.

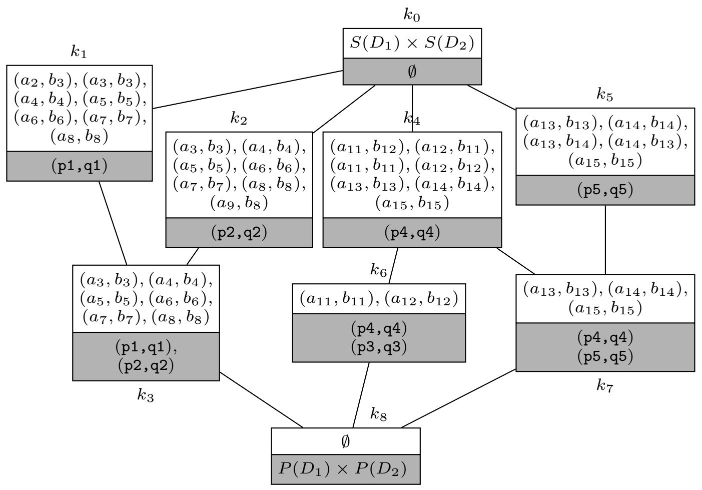
k_0
$S(D_1) \times S(D_2)$
$\emptyset$
k_1
$(a_2,b_3),(a_3,b_3),(a_4,b_4),(a_5,b_5),(a_6,b_6),(a_7,b_7),(a_8,b_8)$
(p1,q1)
k_2
$(a_3,b_3),(a_4,b_4),(a_5,b_5),(a_6,b_6),(a_7,b_7),(a_8,b_8),(a_9,b_8)$
(p2,q2)
k_4
$(a_{11},b_{12}),(a_{12},b_{11}),(a_{11},b_{11}),(a_{12},b_{12}),(a_{13},b_{13}),(a_{14},b_{14}),(a_{15},b_{15})$
(p4,q4)
k_5
$(a_{13},b_{13}),(a_{14},b_{14}),(a_{13},b_{14}),(a_{14},b_{13}),(a_{15},b_{15})$
(p5,q5)
$(a_3,b_3),(a_4,b_4),(a_5,b_5),(a_6,b_6),(a_7,b_7),(a_8,b_8)$
(p1,q1), (p2,q2)
k_3
k_6
$(a_{11},b_{11}),(a_{12},b_{12})$
(p4,q4) (p3,q3)
$(a_{13},b_{13}),(a_{14},b_{14}),(a_{15},b_{15})$
(p4,q4) (p5,q5)
k_7
k_8
$\emptyset$
$P(D_1) \times P(D_2)$

Fig. 2: The LK-lattice based on the datasets given in Figure 1.

Two derivation operators both denoted by $\cdot'$ are working in a dual way, (i) $A' = \{m \in M \mid \forall g \in A, (g,m) \in I\}$ with $\cdot' : 2^G \mapsto 2^M$, and (ii) $B' = \{g \in G \mid \forall m \in B, (g,m) \in I\}$ with $\cdot' : 2^M \mapsto 2^G$. Intuitively, A' is the set of all attributes common to objects in A while, dually, B' is the set of all objects having all attributes in B. The composition of the two operators $\cdot'$ is denoted by $\cdot''$ and corresponds to a *closure operator* (i.e., extensive, increasing, and idempotent).

A pair (A,B) is a concept in K iff $A = B'$ and $B = A'$. Then A is called the "extent" and B the "intent" of concept (A,B). In particular, A and B are closed sets, i.e., $A'' = A$ and $B'' = B$, where A is the largest subset of objects such that $A' = B$ and B the largest subset of attributes such that $B' = A$. The extent A of a concept (A,B) can be considered as a class of objects or instances, while the intent B corresponds to the description of the class. The set of concepts can be ordered by inclusion w.r.t. extents or intents. A concept (A_1,B_1) is subsumed by a concept (A_2,B_2) whenever $A_1 \subseteq A_2$ or dually $B_2 \subseteq B_1$. The set of all concepts is partially ordered thanks to this subsumption relation within a complete lattice named the *concept lattice* and including a to $(\top)$ and a bottom $(\bot)$ element.

For example, given the two RDF datasets D_1 and D_2 in Figure 1, a context $K = (G,M,I)$ can be built, where the set G of objects contains pairs of individuals, say (a_i,b_j), and the set of attributes M includes pairs of properties (p_i,q_j) quantified by $\exists$. When a pair of individuals (a_i,b_j) verifies a such property pair (p_i,q_j), a cross fills the corresponding cell in the context. Then the concepts of K are calculated and a concept lattice called LK-lattice is built thanks to FCA algorithms, as shown in Figure 2. Here it is not possible to show all the details of the construction of the LK-lattice, but the reader may check [7,2]. In the LK-lattice, the intent of k_3, i.e., $\{(p_1,q_1),(p_2,q_2)\}$, represents a link key candidate whose associated pair of classes is (c_1,d_1). The extent of k_3, namely $\{(a_3,b_3),(a_4,b_4),(a_5,b_5),(a_6,b_6),(a_7,b_7),(a_8,b_8)\}$, corresponds to the set of generated links $L(k_3)$. Then, for all $(a_i, b_j) \in L(k_3)$, $p_1(a_i)$ and $q_1(b_j)$, and as well $p_2(a_i)$ and $q_2(b_j)$, are sharing at least one value.

2.3 Link Key Validation based on Discriminability and Coverage

The validation of link key candidates is usually performed within an unsupervised setting and is based on two quality measures, namely *discriminability* and *coverage*, which are defined w.r.t the set of links generated by a candidate [5]. Let us introduce L the link set related to the candidate k, and the two sets of individuals, $\pi_1(L) = \{a|(a,b) \in L\}$ and $\pi_2(L) = \{b|(a,b) \in L\}$. The *coverage* and the *discriminability* of a set of links L over classes c_1 and c_2 are defined as follows:

$$cov(L, c_1, c_2) = \frac{|\pi_1(L) \cup \pi_2(L)|}{|I(c_1) \cup I(c_2)|},\ dis(L, c_1, c_2) = \frac{min(|\pi_1(L)|, |\pi_2(L)|)}{|L|}.$$

Coverage and discriminability evaluate how close a set of links is to a one-to-one mapping between individuals of both datasets. Coverage is maximum when every instance of class c_1 is linked to at least one instance of class c_2, while discriminability is maximum when every instance of c_1 is linked to at most one instance of c_2. The *harmonic mean* of coverage and discriminability may be used to estimate the global quality of a candidate:

$$hmean(L, c_1, c_2) = \frac{2\, cov(L, c_1, c_2).dis(L, c_1, c_2)}{cov(L, c_1, c_2) + dis(L, c_1, c_2)}.$$

3 Characteristics of a Representative Set of Link Keys

The number of link key candidates in an LK-lattice can be very large and it can be very convenient, e.g., for an interactive analysis, to build a *representative set* of candidates denoted as REPLKSET, i.e., a subset of candidates which can be proposed as a summary capturing the significant elements of the original set of candidates. Three desirable characteristics should be verified by a REPLKSET, namely the *compression rate*, the *proportion of preserved identity links*, and the *preservation of the LK-lattice ordering*. Accordingly, the number of candidates which are retained in a REPLKSET should be low while the compression rate should be high. For example, the size of a REPLKSET associated with the LK-lattice given in Figure 2 could be equal to the number of branches (4) in the LK-lattice, while the preserved identity links could be those lying in the extents of the candidates k_3, k_6, and k_7.

The candidates are partially ordered within the LK-lattice, where some candidates are too general while some others are too specific. Candidates lying in the upper levels of the LK-lattice have larger extents, i.e., large sets of potential identity links, than candidates lying in lower levels of the lattice. Then a particular form of *redundancy* can be observed where the same set of identity links can be generated by several candidates, some of which being more general and thus less easily interpretable than the others. For example, again the set of candidates $\{k_3, k_6, k_7\}$ is a good potential representative set of the LK-lattice ordering as all these candidates are lower bounds of concept intervals in the LK-lattice branches, i.e., $[k_1, k_3]$ or $[k_2, k_3]$, $[k_4, k_6]$, and $[k_5, k_7]$. Below, we formally define the three main features characterizing a RepLkSet.

- *The compression rate of a RepLkSet w.r.t. the LK-lattice.* A RepLkSet should be minimal in size, i.e., the smaller the number of candidates in RepLkSet the better is the compression rate. Then the compression rate CR of a RepLkSet w.r.t. an LK-lattice can be calculated as follows:
  CR (RepLkSet,LK-lattice) $= 1 - \frac{|\text{RepLkSet}|}{|\text{LK}-lattice|-2}$
  where $\top$ and $\bot$ are excluded in the LK-lattice. The compression rate is ranging from from 0 to 1, where the best values are close to 1. It is equal to 0 when |RepLkSet | = |LK-lattice |-2, i.e., there is no compression at all. The compression rate cannot be equal to 1 as the RepLkSet cannot be empty, i.e., it is assumed that there exists at least one candidate whose link set cannot be empty.
  For example, if RepLkSet $= \{k_3, k_6, k_7\}$ for the LK-lattice in Figure 2, the compression rate is CR $= 1 - 3/7 \simeq 0.57$.
- *The proportion of preserved links (PPL).* A RepLkSet should preserve a maximal number of identity links, i.e., the higher the number of identity links preserved in RepLkSet the better the representativeness of RepLkSet. The proportion of preserved links PPL is evaluated thanks to the formula:
  PPL (RepLkSet,LK-lattice) $= \frac{|\bigcup_{k_i \in \text{RepLkSet}} L(k_i)|}{|\bigcup_{k_j \in \text{LK}-lattice} L(k_j)|}$.
  It should be noticed that LK-lattice and as well $L(k_j)$ cannot be empty sets (thanks to the definition of a link key candidate). The proportion of preserved links PPL ranges in $]0, 1]$, cannot be equal to zero, and is equal to 1 when all identity links are preserved, i.e., RepLkSet = LK-lattice.
- *The preservation of the LK-lattice ordering.* One requirement in building RepLkSet is to preserve in the clustering the ordering of the LK-lattice. Then a cluster should include a set of candidates such that the lower bound $x \wedge y$ and the upper bound $x \vee y$ of any pair of candidates x and y are also included in the cluster (recall that $x \wedge y$ and $x \vee y$ always exist in a lattice and are unique). There can be two main options: (i) a cluster is based either on a concept interval and includes candidates which are forming a chain, (ii) a cluster is based on a sublattice of the LK-lattice.

These three characteristics cannot be simultaneously optimized but a good compromise can be achieved, i.e., discovering a small set of candidates forming the RepLkSet, which maximizes the compression rate and the number of preserved identity links, and respects the LK-lattice ordering. It should be noticed that, given an LK-lattice, the RepLkSet is not unique. In the following, we make precise the construction of a RepLkSet, starting from an LK-lattice and using agglomerative hierarchical clustering.

4 The LKCLUST Algorithm for building a RepLkSet

The LKCLUST algorithm constructs a representative set of link keys, namely RepLkSet, thanks to *agglomerative hierarchical clustering* (AHC) [11]. In LKCLUST (see Function 1), AHC takes as input the set of candidates in LK-lattice and builds a hierarchical partition of clusters. One main requirement is that the cluster hierarchy preserves –as much as possible– the partial ordering of concepts in the LK-lattice. In LKCLUST, AHC is based on a bottom-up strategy, starting with each candidate as a singleton cluster and then successively merging pairs of clusters based on their distance or dissimilarity. AHC stops when all clusters are merged into a top cluster.

At initialization, the dissimilarity between candidates c_1 and c_2 is defined as $\delta(c_1, c_2) = 1 - \frac{val(c_1 \wedge c_2)}{val(c_1 \vee c_2)}$, where $c_1 \vee c_2 \neq \bot$, i.e., c_1 and c_2 cannot be $\bot$ at the same time, and $val(c)$ is a function returning the size of the extent of c. The dissimilarity δ ranges from 0 to 1 and has the following properties:

(i) $\delta(c_1, c_2) = 0$ iff $c_1 = c_2$,

(ii) $\delta(c_1, c_2) = \delta(c_2, c_1)$ (symmetry),

(iii) $\delta(c_1, c_2) = 1$ iff $val(c_1 \wedge c_2) = 0$, then the extents of c_1 and c_2 are disjoint and c_1 and c_2 are not comparable, i.e., $c_1 \not\leq c_2$ and $c_2 \not\leq c_1$, c_1, and c_2 are lying in two different chains in the LK-lattice.

By contrast, when c_1 and c_2 are comparable, say $c_1 \leq c_2$, then $\delta(c_1, c_2) = 1 - val(c_1)/val(c_2)$ as $c_1 \wedge c_2 = c_1$ and $c_1 \vee c_2 = c_2$. The more c_1 is close to c_2 the less is $\delta(c_1, c_2)$. Accordingly the less dissimilar concept c_2 from a given concept c_1 is lying among the immediate lower or upper neighbor concepts in the LK-lattice, i.e., respectively the *lower cover* or the *upper cover* (more about this subject in [12]). More precisely:

Proposition 1. *If $c_1 \leq c_2 \leq c_3$ in the LK-lattice then $\delta(c_1, c_2) \leq \delta(c_1, c_3)$.*

Proof. Since $c_1 \leq c_2$, $\delta(c_1, c_2) = 1 - val(c_1 \wedge c_2)/val(c_1 \vee c_2) = 1 - val(c_1)/val(c_2)$, with $val(c_1) \leq val(c_2)$. In the same way, $c_1 \leq c_3$, $\delta(c_1, c_3) = 1 - val(c_1)/val(c_3)$, with $val(c_1) \leq val(c_3)$. Since $c_2 \leq c_3$, $val(c_2) \leq val(c_3)$, and it comes that $\delta(c_1, c_2) \leq \delta(c_1, c_3)$. In particular, if $c_1 \leq c_2$, then $c_1 \wedge c_2 \leq c_1 \leq c_2$, and it comes that $\delta(c_1, c_1 \wedge c_2) \leq \delta(c_1, c_2)$. □

Another parameter of AHC is the distance $dclust(X, Y)$ between two clusters X and Y, actually between a cluster and a singleton cluster. LKCLUST relies on the so-called *complete linkage*, i.e., $dclust(X, Y) = \max_{x \in X, y \in Y} \delta(x, y)$. Single linkage based on "min" and average linkage based on "mean distance" are other alternatives. However, if all pairs of points from two clusters X and Y are connected with complete linkage, this yields a "complete linkage" where all possible pairs are connected. Then, this ensures that dissimilarity between any two concepts in a cluster is bound by the dissimilarity of the infimum and supremum of the two concepts in the LK-lattice.

Let us now explain how the LKCLUST algorithm works. At initialization, in AHC, all $\delta(x, y)$ between singleton clusters x and y should be computed and

	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
k_0									
k_1	**0.97**								
k_2	**0.97**	0.97							
k_3	0.97	**0.14**	**0.14**						
k_4	**0.97**	1	1	1					
k_5	**0.98**	1	1	1	0.99				
k_6	0.99	1	1	1	**0.71**	1			
k_7	0.99	1	1	1	**0.57**	**0.4**	1		
k_8	1	1	1	**1**	1	1	**1**	**1**	

Table 1: Dissimilarity table between concepts lying in the LK-lattice in Figure 2. Only values in bold are needed by the LKCLUST algorithm.

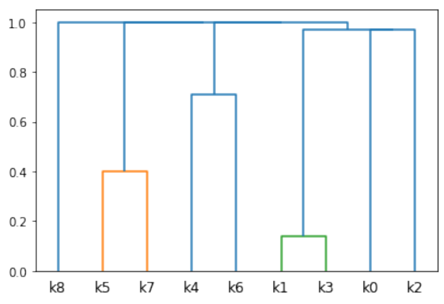

Fig. 3: Dendrogram built by AHC set up based on δ and complete linkage.

recorded in a dissimilarity table, as shown in Table 1. However, in LKCLUST, thanks to proposition 1, only dissimilarities between neighboring candidates are computed, marked in bold in Table 1. Instead of computing $n.(n-1)/2$ dissimilarity values where n is the number of clusters as in AHC, a much smaller number should be computed in LKCLUST, depending on the number of neighbors for a given concept.

Moreover, at each step of AHC, the dissimilarity table should be updated, requiring the computing of $n-2$ dissimilarity values, i.e., the number of clusters minus the two clusters that are merged. Again, in LKCLUST, this number is much smaller as it only involves neighbor concepts. This is illustrated in the experiments, when comparing the running times of AHC and LKCLUST are compared.

For example, the LK-lattice in Figure 2 contains 9 concepts and thus there are 36 dissimilarity values to compute at initialization. However, as shown in the dissimilarity table in Table 1, the computation is restricted to pairs of neighbor concepts, and only 12 values are calculated in fact. At first iteration, either concepts k_1 and k_3 or concepts k_2 and k_3 can be merged as their dissimilarity value is .14, the merging involving the minimal dissimilarity. In both cases, only 3 values need to be updated among the 7 possibilities at the next step.

The LKCLUST stops when all clusters are merged into one final cluster corresponding to the whole set of candidates. The output can be visualized as a *dendrogram*, i.e., a tree-based representation where the clusters are determined thanks to a *cutting level* (see Figure 3). Setting a cutting level mainly depends on the objectives and on the characteristics of the application, and this is discussed in the next section about experiments.

5 Experiments

The discovery of representative link keys returned by the LKCLUST algorithm is evaluated thanks to two series of experiments. Firstly, we demonstrate that

Algorithm 1 The function LKCLUST iteratively builds the hierarchy of clusters, minimizing the number of dissimilarity values to be computed.

```
function LKCLUST(L : a lattice)
    $values \leftarrow createheap()$
    ▷ Computing the dissimilarity values between a concept and its upper cover ◁
    for $x \in L$ do
        for $y \in cover(x, L)$ do
            $d \leftarrow \delta(x, y)$
            $add(values, \langle x, y \rangle, d)$
    while $size(values) > 1$ do
        ▷ Select and merge the clusters with the smallest dissimilarity ◁
        $\langle x, y \rangle, d \leftarrow extractmin(values)$
        $clust \leftarrow createclust(x, y, d)$
        ▷ Update the dissimilarity values based on complete linkage ◁
        for all $(\langle x', y' \rangle, d') \in values$ do
            if ( $x = x'$ or $y = x'$ ) then
                $add(values, \langle clust, y' \rangle, dclust(clust, y'))$
                $remove(values, \langle x', y' \rangle)$
            else if ( $x = y'$ or $y = y'$ ) then
                $add(values, \langle clust, x' \rangle, dclust(clust, x')$
                $remove(values, \langle x', y' \rangle)$
    return $extractmin(values)$
```

LKCLUST builds an REPLKSET with good characteristics and with a shorter runtime than a classical AHC algorithm. We also discuss the effects of selecting a cutting level in the dendrogram. Secondly, the experiments evaluates the characteristics of the representatives in a cluster, and as well the behavior of LKCLUST w.r.t. precision and recall in a supervised setting. The results confirm the high level capabilities of the LKCLUST algorithm in link key discovery.

5.1 Datasets and Protocol

The experiments are performed over ten different *tasks*, where a task consists in considering two RDF datasets with a set of reference links `(a,owl:sameAs,b)`, and then to discover the candidates. Seven of these tasks are based on synthetic datasets proposed by the "Ontology Alignment Initiative" (OAEI)[6]: (1) Restaurants, Person1, and Person2 tasks are taken from OAEI 2010; (2) Doremus tasks (1-3) about cultural institutions are taken from OAEI 2016; and (3) the SPIM-Bench task is taken from OAEI 2018. Any pair of the OAEI datasets is based on the same ontology/schema.

The three remaining tasks represent real-world cases of data interlinking, where the datasets are based on different ontologies. The "Libraries" task relies on a sample of datasets provided by two French libraries, namely the "Bibliothèque nationale de France" (BnF)[7] and the "Agence bibliographique de l'enseignement

[6] `http://oaei.ontologymatching.org/`
[7] `https://data.bnf.fr/`

Task	datasets	#inst.	#prop.	#cl.	#LKC
Restaurants	Restaurant1	339	7	1	13
	Restaurant2	2,256	7	1	
Person1	Person11	2,000	14	1	537
	Person12	1,000	13	1	
Person2	Person21	2,400	14	1	471
	Person22	800	13	1	
Doremus1	PP-1	797	52	1	22
	BnF-1	692	48	1	
Doremus2	PP-2	4,053	52	1	74
	BnF-2	3,384	54	1	
Doremus3	PP-3	940	52	1	26
	BnF-3	822	53	1	
SPIMBench	Abox1	1,126	47	3	1,398
	Abox2	1,130	67	3	
Libraries	BnF	78,076	414	1	1,594
	Abes	290,247	128	1	
wiki-random	Wikidata	1,195	1,531	382	691
	DBPedia	1,184	413	175	
wiki-persons	Wikidata	8,314	4,092	65	3,788
	DBPedia	7,297	332	81	

Table 2: Statistics about the datasets considered in the experiments.

supérieur" (Abes)[8]. They consist in a selection of the most frequent homonyms. The two last tasks are based on DBPedia and Wikidata samples. The wiki-random task contains randomly selected data while the wiki-person task consists in instances of persons sharing a name and a place of birth. Statistics about all datasets are provided in Table 2.

The LK-lattices are generated thanks to an FCA-based tool (not detailed here) which performs a basic normalization of data values and deals with property composition when possible (the datasets about "wiki" tasks only contain direct data property values). Moreover, as the size of "Libraries" datasets is very large, only properties instantiating 15% of subjects are considered. Finally, the column #LKC in Table 2 represents the number of link key candidates discovered in each task.

All experiments have been conducted on a laptop equipped with an Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz and 8GB of memory dedicated to the JVM. Clustering procedures have been applied to the sets of candidates excepting the top and bottom concepts.

5.2 Evaluation of the LKCLUST algorithm

Below, the two main aspects which are evaluated are the performance of the LKCLUST algorithm compared to a classical AHC algorithm and the quality of the

[8] `https://www.idref.fr/`

partitioning. The experiments were carried out with four configurations based on different variations of the two following parameters: (i) the clustering method, i.e., LKCLUST or AHC algorithms, (ii) whether the convexity of a cluster is forced or not. A convex cluster corresponds either to a concept interval (i.e., a chain) or a sublattice. To ensure the convexity of a cluster, the LKCLUST algorithm is modified and skip a candidate group which is not convex.

For every task and every configuration, the running time spent for clustering and the number of required dissimilarities are recorded. The results are presented in Figure 4 and in Figure 5. These two figures only show the four tasks in which the clustering procedure takes at least 2 seconds.

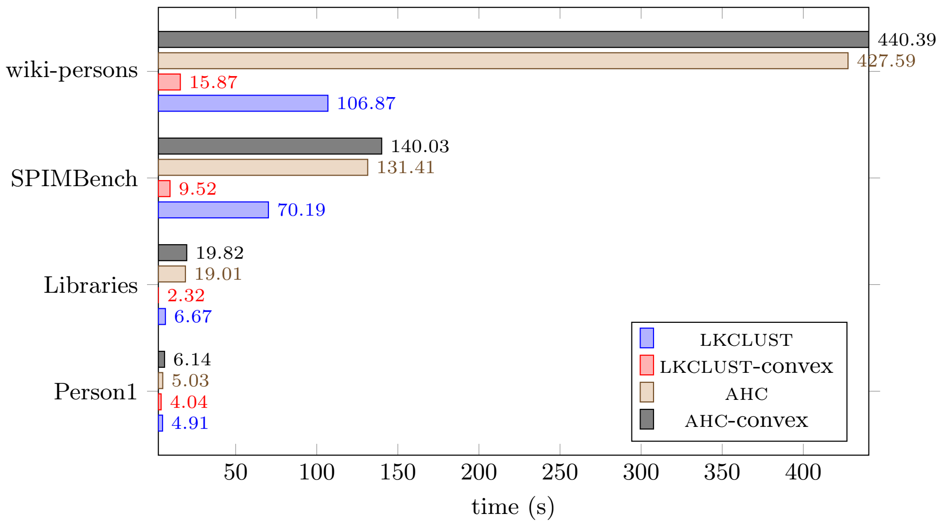

Fig. 4: Four clustering running times in different configurations and interlinking tasks.

It was expected that configurations based on LKCLUST algorithm are faster than those based on a classical AHC algorithm, and this is indeed the case as shown in Figure 4. AHC works faster when we constrain the algorithm to only form convex sublattices. Actually, the convexity requirement reduces the number of possible groupings and therefore the number of dissimilarities to be computed as shown in Figure 5. In line with these observed runtimes, Figure 5 shows that the LKCLUST algorithm significantly reduces the number of calculated dissimilarities. This reduction is even more radical when the convexity constraint is enforced.

To study the quality of partitions built by LKCLUST, the compression, link preservation, and convexity criteria are used. In the four above configurations, the cutting level is changed from 0 to 1 -excluded- in steps of length 0.1. For every cutting level, the compression rate, preservation rate, and convexity rate

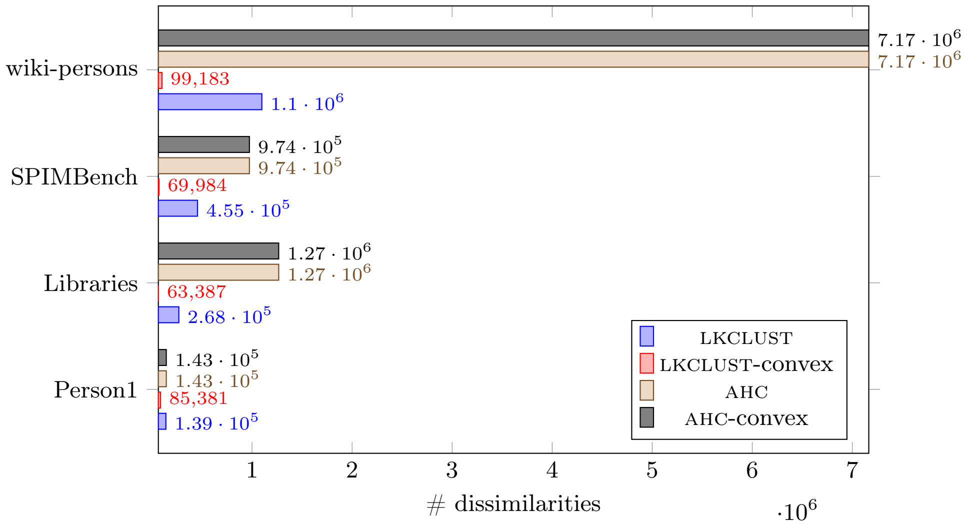

Fig. 5: The numbers of dissimilarities required in the different configurations and interlinking tasks.

are monitored. Figure 6 shows the results in the four configurations for the "SPIMBench" task. The observations that can be made on this task also apply to the other tasks.

AHC and LKCLUST show similar results in term of compression and convexity. When convexity is required, all configurations behave in the same way and almost all links are preserved, independently of the cutting level. In term of preservation, AHC preserves more links than LKCLUST on this task. However, this observation does not apply necessarily to the other tasks where the preservation is roughly the same. This difference in preservation is mainly due to the presence of ties in the dissimilarities. In the event of a tie, a pairing is arbitrarily chosen, which leads to different partitioning. In this case, AHC tends to favor clusters including general concepts which are not necessarily neighbors, because the top concept is omitted.

To sum up, these results show that LKCLUST is much faster than a classical version of AHC. In addition, combining LKCLUST with the convexity constraint provides the best computation times, preserves almost all the links, while the compression ratio is slightly lower .

5.3 The Evaluation of the Candidates Lying in a RepLkSet

This second series of experimental results aims at analyzing the impact of LKCLUST on the selection of the best link key candidates, materialized by the *medoids*. The medoid in a cluster is the link key candidate whose sum of dissimilarities to all other candidates in the cluster is minimal. Then, while the cutting level is varied by step of 0.1, the medoids having a discriminability greater than

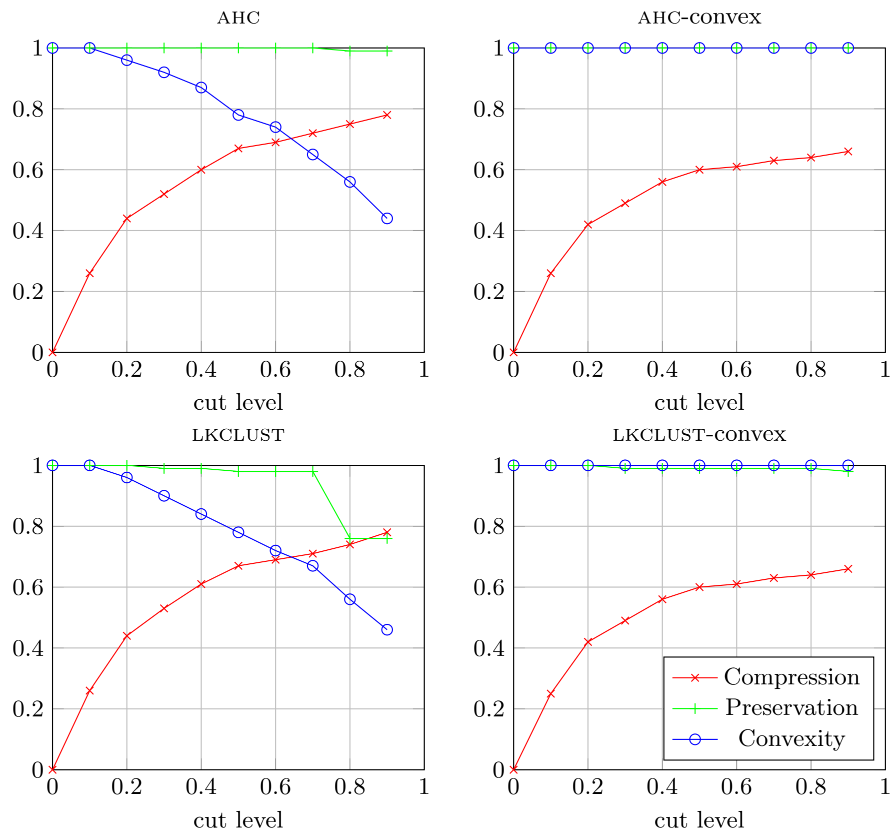

Fig. 6: The variations of compression, link preservation, and convexity w.r.t the cutting level in the task SPIMBench.

or equal to 0.9 are selected at each step. The union of the links generated by the selected medoids is evaluated against a gold standard to assess precision and recall.

Figure 7 shows the precision, recall, and compression ratio that are measured for the tasks “wiki-random” and “Doremus1”. These two tasks are illustrative of the observed trends. The compression ratio is computed over the selection of medoids whose discriminability is higher than 0.9.

On “wiki-random”, the F-measure is almost stable whatever the cutting level. This shows that the medoids selected thanks to LKCLUST contain good link key candidates. We observe the same trends for the tasks “Person1”, “Restaurants”, “SPIMBench”, and “Libraries”.

Regarding “Doremus1”, there is a drop in recall, while precision remains stable. For all “Doremus” tasks, this drop appears when the cutting level is high (> 0.8). For the tasks “Persons2” and “wiki-person”, the recall break is less noticeable, i.e., -0.12 and -0.13 respectively, but occurs earlier, around 0.3. This drop becomes visible during an acceleration of the compression ratio. However an acceleration of the compression ratio does not necessarily imply a drop in recall in the other tasks. These sudden drops in recall could probably be due to a "discriminability threshold" effect. Indeed, the threshold was arbitrarily set at 0.9 in all the experiments, but should probably be adapted on a case-by-case basis.

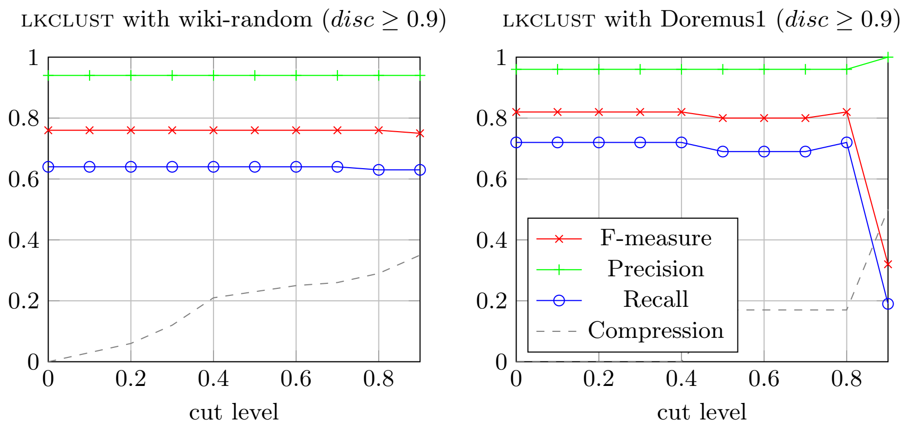

Fig. 7: The variation of quality measures and compression over a selection of link key candidates having a discriminability greater than 0.9 w.r.t the cutting level.

6 Conclusions and Perspectives

In this paper, we are interested in data interlinking based on the discovery of link keys generating identity links between individuals lying in two RDF datasets. Link keys are discovered thanks to FCA algorithms and are organized within a concept lattice called LK-lattice that can be very large in size, making it difficult to visualize and to interpret. Thus we introduce a representative set of link key candidates, i.e., the REPLKSET, which satisfies good properties, namely a high degree of compression, a high preservation of identity link, and the respect of the LK-lattice ordering. The REPLKSET is built thanks to an adapted algorithm named LKCLUST based on agglomerative hierarchical clustering. Experiments show that LKCLUST returns a REPLKSET satisfying the three constraints where the representatives in each cluster show good recall and precision.

As future work, we would like to study other clustering algorithms and distance measures to build a REPLKSET, and to characterize the mapping between an LK-lattice and a REPLKSET. In particular, this approach could also be well-suited to the difficult task of lattice reduction [4].

References

1. Abbas, N., Bazin, A., David, J., Napoli, A.: Discovery of link keys in resource description framework datasets based on pattern structures. International Journal of Approximate Reasoning **161**, 108978 (2023)
2. Abbas, N., David, J., Napoli, A.: Discovery of Link Keys in RDF Data Based on Pattern Structures: Preliminary Steps. In: Proceedings of CLA. pp. 235–246. CEUR Workshop Proceedings 2668 (2020)
3. Al-Bakri, M., Atencia, M., David, J., Lalande, S., Rousset, M.C.: Uncertainty-sensitive reasoning for inferring sameAs facts in linked data. In: Proceedings of ECAI. pp. 698–706 (2016)
4. Aragón, R.G., Medina, J., Ramírez-Poussa, E.: Reducing concept lattices by means of a weaker notion of congruence. Fuzzy Sets and Systems **418**, 153–169 (2021)
5. Atencia, M., David, J., Euzenat, J.: Data interlinking through robust linkkey extraction. In: Proceedings of ECAI. pp. 15–20 (2014)
6. Atencia, M., David, J., Euzenat, J.: On the relation between keys and link keys for data interlinking. Semantic Web Journal **12**(4), 547–567 (2021)
7. Atencia, M., David, J., Euzenat, J., Napoli, A., Vizzini, J.: Link key candidate extraction with relational concept analysis. Discrete Applied Mathematics **273**, 2–20 (2020)
8. Atencia, M., David, J., Scharffe, F.: Keys and pseudo-keys detection for web datasets cleansing and interlinking. In: International Conference on Knowledge Engineering and Knowledge Management (EKAW). pp. 144–153. Springer LNCS 7603 (2012)
9. Ganter, B., Obiedkov, S.A.: Conceptual Exploration. Springer (2016)
10. Ganter, B., Wille, R.: Formal Concept Analysis. Springer (1999)
11. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: an Introduction to Cluster Analysis. Wiley (2009)
12. Monjardet, B.: Metrics on partially ordered sets—a survey. Discrete Mathematics **35**(1), 173–184 (1981)
13. Nentwig, M., Hartung, M., Ngonga Ngomo, A.C., Rahm, E.: A survey of current Link Discovery frameworks. Semantic Web Journal **8**(3), 419–436 (2017)
14. Pernelle, N., Saïs, F., Symeonidou, D.: An automatic key discovery approach for data linking. Journal of Web Semantics **23**, 16–30 (2013)
15. Symeonidou, D., Armant, V., Pernelle, N., Saïs, F.: SAKey: Scalable Almost Key Discovery in RDF Data. In: Proceedings of ISWC. pp. 33–49. Springer LNCS 8796 (2014)
16. Symeonidou, D., Galárraga, L., Pernelle, N., Saïs, F., Suchanek, F.M.: VICKEY: Mining Conditional Keys on Knowledge Bases. In: Proceedings of ISWC. pp. 661–677. Springer LNCS 10587 (2017)