BECKEY: Understanding, comparing and discovering keys of different semantics in knowledge bases

Danai Symeonidou, Vincent Armant, Nathalie Pernelle

BECKEY: Understanding, Comparing and Discovering Keys of Different Semantics in Knowledge Bases

Danai Symeonidou[a], Vincent Armant[a,b], Nathalie Pernelle[c]

[a]*INRAE, SupAgro, UMR MISTEA, Université de Montpellier, France*
[b]*Insight Centre for Data Analytics, University College Cork, Ireland*
[c]*Université Paris Sud, Université Paris Saclay, France*

Abstract

Integrating data coming from different knowledge bases has been one of the most important tasks in the Semantic Web the last years. Keys have been considered to be very useful in the data linking task. A set of properties is considered a key if it uniquely identifies every resource in the data. To cope with the incompleteness of the data, three different key semantics have been proposed so far. We propose BECKEY, a semantic agnostic approach that discovers keys for all three semantics, succeeding to scale on large datasets. Our approach is able to discover keys under the presence of erroneous data or duplicates (i.e., *almost keys*). A formalisation of the three semantics along with the relations among them is provided. An extended experimental comparison of the three key semantics has taken place. The results allow a better understanding of the three semantics, providing insights on when each semantic is more appropriate for the task of data linking.

Keywords: Semantic Web, Key discovery, Data linking, Key semantics, RDF, Semantic agnostic approach

1. Introduction

Over the last years, the Web of data has received a tremendous increase, containing a huge number of RDF triples. Integrating data described in different RDF datasets and creating semantic links among them, has become one of the most important goals of RDF applications. These links express semantic correspondences between ontology entities, or semantic links between data such as *owl:sameAs* links. By comparing the number of resources published on the Web with the number of *owl:sameAs* links, the observation is that the goal of building a Web of data is not accomplished yet.

Many approaches that aim to automatically discover *owl:sameAs* links (see [9, 14] for a survey) have been already proposed. Some of them are knowledge-based and guided by ontology axioms (i.e., disjunctions, (inverse) functional properties, composite keys) or declared linkage rules.

A key expresses a set of properties whose values uniquely identify every resource of a dataset. Keys can be used as logical rules to clean or link data when a high Precision is needed [12, 2, 1] or to construct more complex similarity functions that can be used in linking platforms that support manually specified rules such as [25, 15]. Moreover, keys can be also exploited by blocking methods that are defined to partition the data and reduce the search space for the linking step [6, 21]. Nevertheless, in most of the datasets published on the Web, the keys are not available and it can be difficult, even for an expert, to determine them. Furthermore, when data are heterogeneous and incomplete, a large set of keys composed of different properties is needed to obtain a high Recall.

Key discovery approaches have been proposed recently in the setting of the Semantic Web [23, 5, 17, 22] demonstrating the importance of keys in the data linking process. These works differ not only in the method proposed to discover keys from a dataset but also in the way a key is defined. More precisely, three different key semantics have been proposed so far in the context of the Semantic Web, the *S-keys* [23, 17], the *F-keys* [22] and the *SF-keys* [5]. The interpretation of the multivalued nature of the properties in the Semantic Web along with the incompleteness of the data has led to these three semantics.

Data published on the Web are usually created automatically, thus may contain erroneous information or duplicates. When these data are exploited to discover keys, relevant keys can be lost. For example, let us con-

sider a "dirty" dataset where two different people share the same social security number (SSN). In this case, SSN will not be considered as a key, since there exist two people sharing the same SSN. Allowing some exceptions can prevent the system from losing keys. Furthermore, there exist datasets where there exist only few or no sets of properties that uniquely identify every instance in a dataset. However, even if a set of properties is not a key, it can be very discriminative and provide many correct links. We call these keys with exceptions *almost keys*. For example, usually a telephone number corresponds to one restaurant. Nevertheless, there can be cases where two different restaurants located in the same place share phone numbers. In this scenario, the telephone number corresponds to an almost key. While [17] is able to discover keys only when no errors or duplicates exist, [23, 22, 5] propose strategies for the discovery of almost keys. [23] discovers almost keys following the *S-key* semantic, [22] following the *F-key* semantic and [5] following the *SF-key* semantic.

The main goal of this work is to provide a deep understanding of the different key semantics and their data linking capability. From a theoretical viewpoint, we provide a general framework for understanding and computing the three key semantics highlighting the common and distinct characteristics of each semantic. From an application viewpoint, our extended experimental evaluation provides a deeper understanding of the semantics and their capability to link efficiently data. More precisely, this evaluation allows us to identify in which cases each semantic is more appropriate to use. We are the first to introduce such a wide evaluation allowing to provide significant examples where each semantic is more adapted to be used. While a first attempt to present and compare the three existing semantics has been done in [3], this work takes this comparison to a further level presenting a deep analysis of different aspects of these semantics. First, we represent these key semantics as logical linking rules and demonstrate the existing inclusion relations between the inferred links. Then, to provide a better understanding of the relations among the discovered keys themselves, we also exhibit the inclusion relations among the discovered keys. Afterwards, we focus on the automatic key discovery and present BECKEY, a uniform key discovery approach that is able to discover almost keys for the three key semantics proposed in the Semantic Web. The core component of BECKEY is an efficient semantic agnostic approach based on different filtering and pruning techniques that are able to discover all three types of key semantics for large RDF datasets.

In order to demonstrate the data linking capability of each semantic, an extended experimental comparison of the different key semantics takes place in this work. We illustrate through different examples under which conditions each semantic will be more likely to bring the best linking results. Therefore, this work provides not only a guide for understanding the difference among the semantics of keys but also yields a map for choosing a given semantic depending on the nature of the data to be linked.

More precisely, our contributions are as follows:

1. a theoretical comparison of the three semantics of keys, the notion of exception in these semantics and the conditions under which a set of properties can be considered as a key
2. a methodology for rewriting datasets into a common and compact structure that eases the automatic discovery for the three existing key semantics
3. a semantic agnostic algorithm that ensures the discovery of all three types of key semantics
4. an extended experimental comparison of the proposed semantics allowing to demonstrate in which cases each semantic is more appropriate to use.

The paper is organised as follows. Section 2 discusses the related works on key discovery. Section 3 introduces first the data and ontology model and then it presents the notions of keys. Section 4 presents the semantic agnostic algorithm for the discovery of keys under all three semantics. Section 5 showcases our experiments before Section 6 concludes.

2. Related Work

The data linking problem in data graphs has been the main focus of numerous studies (see [9, 14] for survey), and applied in different research fields such as knowledge extraction [23, 24], geospatial analysis [27], sentiment analysis [19, 10], etc.

Some of the existing approaches are based on expressive linking rules that can be learned from a set of existing reference links [18, 16]. These rules consist of attribute-specific comparisons, aggregation functions along with different weights and thresholds.

Some other approaches are based on keys since such rules can be used either to infer logically *owl:sameAs* links or to construct more complex data linking rules. [11, 12] exploit a forward-reasoner that infers all the *owl:sameAs* facts (and possibly *owl:different-from*) that can be logically entailed from the input rules and facts. The import-by-query algorithm developed in [2] uses

the rules to build SPARQL queries for importing from external sources the necessary data for resolving a link query while [1] allows to model and reason on uncertain RDF facts and rules, based on the semantics of probabilistic Datalog. In [7], the authors propose two algorithms that exploit more complex keys that can be expressed as graph patterns to link data. Since such keys are rarely available, some approaches focus on learning keys from the data.

The problem of discovering Functional Dependencies (FD) and keys has been intensively studied in the relational databases field. However, the approaches that have been developed in the relational setting cannot be applied directly to knowledge graphs. Indeed, these approaches are geared towards relations that contain one single value for each subject while in knowledge graphs a property can contain several objects for the same subject. Additionally, most of the existing approaches do not take into account the data incompleteness which is very present in the data found on the Web. Finally, key discovery approaches in the Semantic Web often benefit from the knowledge found in the ontology, e.g., keys can be discovered for a given class and are also valid for every subclass in the ontology. OWL2 has introduced the *owl2:key* construct allowing to declare a set of properties as a key for a given class of an ontology.

Recent approaches have been defined that aim to discover keys from knowledge bases. These approaches can be distinguished regarding a) the semantics of the discovered keys and b) the strategies that are used to explore the search space.

Key semantics. In knowledge graphs, properties can be multivalued (e.g., a publication is described by several authors, a person can be described by a list of email addresses). Additionally, data may be incomplete and depending on how the absence of some property values is interpreted, it has led to consider different key semantics. [17, 23, 24, 4] consider the OWL2 semantics [26]: in order to infer an identity link between two instances, it suffices that these instances share at least one value for each property involved in the key (e.g., if the property email is considered as a key for the class person, a shared email address is sufficient to decide that two people are the same). Thus, a set of properties is considered to be a key under this semantic, if there exist no instances in the dataset sharing at least one value per property in the key. Keys discovered under this semantic are called *S-keys*. Unlike [17, 23, 24, 4], in [5, 22] two instances have to share all the values for each property involved in the key to be considered identical (e.g., if the property email is considered as a key, the sets of emails must be equal to decide that two people are the same). Thus, a set of properties is considered to be a key under this semantic, if there exist no instances in the dataset sharing the same set of values for all properties in the key. While [17, 23, 24, 4, 5] consider that two instances can be compared using only the values appearing in the data, [22] introduces a new interpretation for the absence of values. More precisely, considering a set of properties as a key, two instances are identical for [22] if for every property in the key they either have the exact same set of values or they both have no value provided in the data. Keys under the semantic proposed in [22] are called *F-keys* while under the semantic proposed in [5] are called *SF-keys*. A theoretical study of *S-keys* and *F-keys* along with a preliminary experimental comparison of *S-keys*, *F-keys* and *SF-keys* has been proposed in [3]. This first comparative study has shown that when the discovered keys are used to link RDF datasets, the results can vary depending on the dataset and the studied class. However, the evaluation of the approaches has been conducted on a very limited number of classes and has not provided insights on when each semantic is more appropriate to use.

Key discovery strategies in RDF data. The number of candidate keys for a given class is exponential w.r.t. the number of properties used to describe the data of that class. Therefore, different strategies and heuristics have been proposed to optimize both the time and the space complexity of key discovery. Approaches can be roughly classified in two categories, the ones discovering keys directly and the ones discovering first *non keys* (i.e., sets of properties that are not keys) and then using them to derive keys. In the first category, all approaches exploit the monotonic characteristic of keys to prune the search space. In [21], discriminating data type properties (i.e., approximate keys) are discovered from a dataset to develop a data linking blocking method. Keys of a specific size are explored only if there is no smaller key with a high discriminative power. These approximate keys are then exploited to construct blocks of instances. In this work, the aim is not to learn the complete set of minimal keys, but to discover a set of properties that can be used to partition the data efficiently. Therefore, no additional optimisation strategies are addressed. In [5], the authors introduce a bottom-up approach based on TANE [13], to discover *pseudo-keys* (keys with exceptions). In order to improve the scalability of the approach, the authors discover explore combinations of properties that are highly instantiated. In ROCKER [22], the authors propose a more sophisticated bottom-up approach based on a refinement oper-

ator allowing to efficiently discover pseudo-keys. As [5], [22] also filter the search space using highly instantiated sets of properties. However, the key discovery tool provided in this work, used in the experimental evaluation, in some cases fails to compute the complete set of minimal keys as expected (i.e., some of the discovered keys are not minimal and some valid keys are not discovered) (see `https://github.com/danaiS/BECKEY/tree/master/Annexe/OtherTools` for an example). To avoid scanning all the data, some approaches [17, 23] extend [20] proposed in the context of relation databases. This approach discovers the complete set of maximal non keys first and uses them to derive minimal keys. Such approaches exploit the anti-monotonic characteristic of non keys to prune the search space. KD2R [17] is based on a prefix tree structure to store the data but can be overwhelmed by large datasets. Indeed, deriving keys from non keys is a time consuming step for such approaches. Furthermore, this approach requires datasets that fulfill the Unique Name Assumption to discover keys that are valid in all the dataset (i.e., no exceptions allowed). SAKey [23] discovers almost keys in large datasets that may contain errors or duplicates. SAKey introduces different additional prunings based on the detection of irrelevant sets of properties for the non key discovery and a very efficient key derivation approach allowing to scale to millions of triples. VICKEY [24] discovers *conditional keys*, i.e., keys that are valid only for some instances of class that satisfy a given condition. To scale, VICKEY introduces a hybrid strategy discovering first maximal non keys and then searching minimal conditional keys in the search space defined by these non keys, using a bottom-up approach.

In this paper, we propose a extended theoretical and experimental comparison of the three different semantics and we introduce BECKEY, a semantic agnostic approach that can discover all three types in an efficient way.

3. Keys for data linking

The goal of this section is to present and illustrate the different semantics of keys existing in the Semantic Web. To do so, we introduce the notations that will assist with the formalisation of a uniform framework for key discovery in RDF datasets. We first present the data model. Then we provide three definitions of keys that have been envisaged for data linking before introducing the inclusion relationships between these definitions. At the end of this section, we introduce the common structures and definitions that will be re-used for discovering these keys.

3.1. Data Model

In this work we consider a knowledge base as a set of instances $\mathcal{I}$ (e.g., a specific person referred as *i1*), a set of literals $\mathcal{L}$, a set of properties $\mathcal{P}$ (e.g., *FirstName*), and a set of classes $\mathcal{C}$ (e.g., *University*). Statements of instances in a knowledge base are usually represented as triples $< s, p, o >$ where subject $s \in \mathcal{C} \cup \mathcal{I}$, property $p \in \mathcal{P}$, and object $o \in \mathcal{C} \cup \mathcal{I} \cup \mathcal{L}$, which we write as $p(s, o)$. Every instance is typically associated to one or more classes by the *type* property, and these classes can be arranged in a hierarchy by the *subclassOf* property. A set of such statements constitutes a knowledge base (KB)[1]. Given a KB $\mathcal{K}$, an RDF dataset D for a class c of $\mathcal{K}$ is the set of all statements that have as subject an instance of c or of a subclass of c. Table 1 shows an example dataset about people $i1, ..., i7$, each described by the properties *FirstName*, *LastName*, *SSN*, *DateOfBirth*, *StudiedIn* and *HasSibling*, with one or more objects for each property. Note that given an RDF dataset D, we write $p(x, y)$ to mean $p(x, y) \in D$.

3.2. Understanding and comparing key semantics

General concept of a key. A key represents a set of properties that uniquely identifies resources stored in a dataset. The three semantics of keys proposed in the context of Semantic Web i.e., *S-key* [23], *F-key* [22] and *SF-key* [5] also converge to define a key as a set of properties that uniquely identifies an instance. However, these semantics diverge on when to consider that the set of values, associated to an instance and a given set of properties, is unique. The *S-key* semantic assumes data incompleteness and only considers a set of properties as a key if there exist no instances sharing a value for the given set of properties. The *SF-key* semantic assumes data completeness and considers a set of properties as a key if there exist no instances having the exact same set of values for the given set of properties. Finally, the *F-key* semantic as *SF-key* semantic also assumes data completeness but in addition also considers the absence of values for pairs of instance-property when deciding if a set of properties is a key. In the following, we recall the definitions and the characteristics of the three key semantics and exhibit the theoretical relationship in terms of the inferred links, multivaluation and data completeness assumptions.

[1] No blank nodes are considered in this work.

Comment citer ce document :
Symeonidou, D., Armant, V., Pernelle, N. (2020). BECKEY: Understanding, comparing and discovering keys of different semantics in knowledge bases. Knowledge-Based Systems, 105708. , DOI : 10.1016/j.knosys.2020.105708

	FirstName	LastName	SSN	DateOfBirth	StudiedIn	HasSibling
i1	Helen	Dond	121558745	10/08/79	UCC, Stanford	i2, i45
i2	George	Dond	232351234	05/03/85	UCC, UCD	i1, i45
i3	Cathrine	Roger	767960154	–	–	–
i4	Mike	Jones	–	28/02/75	Oxford, MIT	i67
i5	George	Dupont	–	10/08/79	UCC, UCD, UCL	–
i6	Helen	James	325318695	–	Stanford	i75
i7	Helen	Dond	–	10/08/79	Stanford, UCC	i2
i8	Cathrine	Roger	767960154	–	–	–

Table 1: Example dataset D1 for instances of the class Person

Intuitively, an *S-key* is a rule stating that any pair of instances sharing at least one common value for each property of the *S-key*, refers to the same object. Keys under the semantic of *S-key* are discovered in [23, 17].

Definition 1 (*S-key*). *Let $P = \{p_1, \ldots, p_n\}$ be a set of properties ($P \subseteq \mathcal{P}$), the S-key$(p_1, \ldots, p_n)$ is the rule defined as follows:*

$$\forall x \forall y \forall z_1 ... z_n (\bigwedge_{i=1}^{n} (p_i(x, z_i) \wedge p_i(y, z_i)) \rightarrow \ x = y)$$

According to the definition of *S-key*, two instances x and y are considered to be identical if they share at least one common value for all key properties p_i.
In the instances of class *Person* shown in Table 1, if the *S-key*$(FirstName, LastName, HasSibling)$ is declared, the instances $i1$ and $i7$ will refer to the same person, i.e., $i1 = i7$.

Note that, while $i3$ and $i8$ share the same first name and last name, they have no declared sibling. Therefore, the *S-key*$(FirstName, LastName, HasSibling)$ does not allow to decide whether $i3$ and $i8$ refer or not to the same instance. Implicitly here, it cannot be inferred that an instance is identical to another, if the instance has at least one absent value for a property expressed in the *S-key*.

In OWL2[2], it is possible to declare that a set of properties is an *S-key* for a given class c. More precisely, *owl:hasKey*$(c(ope_1, \ldots, ope_m)\ (dpe_1, \ldots, dpe_n))$ states that each instance of the class c is uniquely identified by the object property expressions ope_i and the data property expressions dpe_j. An *object property expression* is either an *object property* or an inverse *object property*. The semantic of the construct *owl:hasKey* is defined in [26] [3].

[2]http://www.w3.org/TR/owl2-overview

[3]Note that the definition of the `owl:hasKey` axiom in OWL 2 additionally enforces the considered instances to be named

F-key and *SF-key* have different semantics. Both definitions consider that property values are completely known for all instances of a given dataset. Indeed, it can be also meaningful to consider that the instances should coincide for all property values (e.g., a list of authors of a given paper, or the list of universities a person has studied in). Keys following this semantic are referred as *F-keys* or *SF-keys* depending on how empty values are considered. According to the definition of *F-key*, two instances x and y are considered to be the identical if they share the exact same set of values for all properties p_i expressed in the *F-key*. Keys under the semantic of *F-key* are discovered in [22].

Definition 2 (*F-key*). *Let $P = \{p_1, \ldots, p_n\}$ be a set of properties ($P \subseteq \mathcal{P}$), the F-key$(p_1, \ldots, p_n)$ is the rule defined as follows:*

$$\forall x \forall y (\bigwedge_{i=1}^{n} (\forall z_i (p_i(y, z_i) \rightarrow p_i(x, z_i)) \wedge$$

$$(\forall w_i (p_i(x, w_i) \rightarrow p_i(y, w_i)))) \ \rightarrow \ x = y)$$

In the instances of class *Person* shown in Table 1, if the *F-key*$(FirstName, LastName, HasSibling)$ is declared, the instances $i3$ and $i8$ will infer to the same person while there is not enough information to decide for $i1$ and $i7$. Compared to an *S-key*, an *F-key* uniquely identifies each instance by the the values associated to each property present in the *F-key*. Moreover, this definition allows two instances to refer to the same real world object even the set of values associated to some or all of the properties expressed in *F-key* is empty. Implicitly in this definition the absence of values for a property is treated as an "known no value" rather than "possibly missing values" for a property. This hypothesis may appear to be too permissive to identify identical instances of real world datasets. The next definition of *SF-key* overcomes this issue.

An *SF-key* can be viewed as a specific type of *F-key* that requires two instances to be considered as identical when they share similar non empty set values for each properties declared in the *SF-key*. Keys under the *SF-key* semantic are discovered in [5].
More formally we have:

Definition 3 (*SF-key*). *Let $P = \{p_1, \ldots, p_n\}$ be a set of properties ($P \subseteq \mathcal{P}$), the SF-key$(p_1, \ldots, p_n)$ is the rule defined as follows:*

$$\forall x \forall y \Big(\bigwedge_{i=1}^{n} (\exists t_i (p_i(x, t_i)) \wedge (\exists u_i (p_i(y, u_i)) \rightarrow (\forall z_i (p_i(y, z_i) \rightarrow p_i(x, z_i)) \wedge (\forall w_i (p_i(x, w_i) \rightarrow p_i(y, w_i))) \rightarrow (x = y))$$

According to the definition of *SF-key*, two instances x and y are considered to be the same when they share the exact non empty set of values for all the properties p_i expressed in the *F-key*.

In the example of class *Person* shown in Table 1, if the *SF-key*$(FirstName, LastName, HasSibling)$ is declared, we cannot infer that $i3$ and $i8$ or $i1$ and $i7$ refer to the same person. An example of identical instances can be inferred when the *SF-key*$(LastName, StudiedIn)$ is declared. In this case $i1$ and $i7$ refer to the same person under the *SF-key* semantic, but not $i3$ and $i8$.

3.3. Inclusion relation between inferred links

As previously mentioned, the declaration of a set of properties as a key leads to the inference of *owl:sameAs* links among instances. Since different definitions of keys have been proposed to declare that two instances refer to the same real world object in the literature, we provide a comparison among the links that can be inferred. More precisely, given a set of properties declared as a key, we compare the sets of *owl:sameAs* links that can be inferred from each definition and deduce relationships between them. Characterising the relationships between the links inferred by different semantic of keys is crucial for better understanding the consequences of using one definition instead of another. It is also the first step for designing a uniform approach that will be able to compute different definitions of keys. In this study, the characterisation is led by two singularities of the data: multivaluation and empty values. First, we introduce links inferred by a set of properties P.

Definition 4 (*Inferred S-Link*, *F-Link*, *SF-Link*). *Given the S-key(P) (resp. F-key(P), SF-key(P)) where $P = \{p_1, \ldots, p_n\}$, an S-Link (resp. F-Link, SF-Link) is an owl:sameAs link inferred from an S-key(P) (resp. F-key(P), SF-key(P)). S-Links (resp., F-Links, and SF-Links) denotes the set of all owl:sameAs links inferred from S-key(P), (resp. F-key(P), SF-key(P)).*

Given a set of properties P, Table 2 shows the variation of the inclusion/equality relationships between inferred links depending on the characteristics of the properties involved. If each instance has only one value per property (i.e., no multivaluation and no empty values allowed), then the sets of *S-Links*, *F-Links* and *SF-Links* inferred from different key semantics are identical. When the properties P are multivalued and map at least one value to each instance (i.e., multivaluation allowed, empty values not allowed), all links inferred from *SF-key*(P) and *F-key*(P) will also be inferred from *S-key*(P). In this case the links inferred from *SF-key*(P) and *F-key*(P) are identical. Inversely, when for each property all instances map to at most one value (i.e., no multivaluation allowed, empty values allowed), all inferred links from *SF-key*(P) and *S-key*(P) will also be inferred from *F-key*(P). In this case the links inferred from *SF-key*(P) and *S-key*(P) are identical. In the general case, when the properties P are single-valued, i.e., for each property all instances map to zero or multiple values, all inferred *SF-Links* will also be inferred from *F-key*(P) and *S-key*(P).

3.4. Key discovery under different semantics

Data on the Web are describing instances of different classes using numerous properties. In this context, keys that can be exploited for data linking are hidden and cannot easily be specified even by a human expert. Therefore, automatic methods are needed to discover them from the data. When the Unique Name Assumption (UNA) is fulfilled, a set of properties can be considered as a valid key in a dataset if every instance can be uniquely identified using the values of this set of properties.

Thus, keys can first be discovered in each data source and then merged according to a merging method that computes keys that are valid in all the datasets [17] or that rank and select keys based on quality measures [8]. In this paper, we focus on the key discovery processes that can be defined for one dataset.

Keys with exceptions. RDF datasets may contain erroneous data and duplicates. Thus, discovering keys in RDF datasets without taking into account these data characteristics may lead to lose keys. Furthermore, there exist sets of properties that even if they are not keys, due to a small number of shared values, they can

Multivaluation	no	yes	no	yes
Empty values	no	no	yes	yes
Relations	*S-Links = F-Links = SF-Links*	*F-Links = SF-Links* *SF-Links ⊆ S-Links* *F-Links ⊆ S-Links*	*SF-Links = S-Links* *SF-Links ⊆ F-Links* *S-Links ⊆ F-Links*	*SF-Links ⊆ S-Links* *SF-Links ⊆ F-Links*

Table 2: Relations between inferred sets of links

be useful for data linking or data cleaning. These sets of properties are particularly necessary when a class has no keys.

In this work we will refer to keys with exceptions as *n-almost keys*, first used in [23]. A set of properties is a *n-almost key* if there exist at most n instances that share values for this set of properties. However, the different key semantics lead to different definitions of the notion of exception.

For an *S-key* defined for a class $c \in C$, the exception set $E_{S,P}$ corresponds to the set of instances of this class that share at least one value with at least another instance, for each property of the set of key properties P.

Definition 5. (Exception set for an *S-key*). *Let P be a set of properties ($P \subseteq \mathcal{P}$). The exception set $E_{S,P}$ is defined as:*

$$E_{S,P} = \{X \mid \exists Y(X \neq Y) \wedge (\bigwedge_{p \in P} \exists U\, p(X,U) \wedge p(Y,U))\}$$

For example, in *D1* of Table 1 the set of exceptions for the *S-key*($LastName, StudiedIn$) is: $\{i1, i2, i7\}$. Indeed, the instances $i1$, $i2$, $i7$ have the same last name and have all studied in *UCC*. Similarly, the set of exceptions for the *S-key*($FirstName, LastName$) is: $\{i1, i3, i7, i8\}$. Indeed, all four instances share a first name and a last name with another instance, i.e., the instances $i1$ and $i7$ are called both *Helen Dond* and the instances $i3$ and $i8$ are called both *Catherine Roger*.

The exception set $E_{F,P}$ for an *F-key* corresponds to the set of instances that has equal sets of values (eventually empty) with at least another instance, for each property of the set of key properties P.

Definition 6. (Exception set for a *F-key*). *Let P be a set of properties ($P \subseteq \mathcal{P}$). The exception set $E_{F,P}$ is defined as:*

$$E_{F,P} = \{X \mid \exists Y(X \neq Y) \wedge (\bigwedge_{p \in P}$$

$$(\forall z_i(p_i(y,z_i) \rightarrow p_i(x,z_i)) \wedge (\forall w_i(p_i(x,w_i) \rightarrow p_i(y,w_i))))\}$$

For example, in *D1* of Table 1 the exception set for the *F-key*($LastName, StudiedIn$) is: $\{i1, i3, i7, i8\}$. Indeed, the instances $i1$ and $i7$ have the same last name and they have studied in the same institutes. Similarly, the instances $i3$ and $i8$ are both called *Roger* and no information is provided for the institutes where they have studied in. Therefore, there exist in total 4 instances, i.e., $\{i1, i3, i7, i8\}$ that are belonging in the exception set of the *F-key*($LastName, StudiedIn$).

The set of exceptions $E_{SF,P}$ for an *SF-key* corresponds to the set of instances that share all values with at least one instance, for each property of a given set of properties P.

Definition 7. (Exception set for a *SF-key*). *Let P be a set of properties ($P \subseteq \mathcal{P}$). The exception set $E_{SF,P}$ is defined as:*

$$E_{SF,P} = \{X \mid \exists Y(X \neq Y) \wedge (\bigwedge_{p \in P} \exists U\, p(X,U) \wedge p(Y,U)$$

$$\wedge(\forall z_i(p_i(y,z_i) \rightarrow p_i(x,z_i)) \wedge (\forall w_i(p_i(x,w_i) \rightarrow p_i(y,w_i))))\}$$

In *D1* of Table 1 the set of exceptions for the *SF-key*($LastName, StudiedIn$) is: $\{i1, i7\}$.
Using the corresponding exception set E_P we give the following definition of a *n-almost key*.

Definition 8. (*n-almost key*). *Let P be a set of properties ($P \subseteq \mathcal{P}$) and n an integer. P is a n-almost S-key(resp. F-key, SF-key) if $|E_{S,P}|$ (resp. $|E_{F,P}|$, $|E_{SF,P}|$) $\leq n$.*

This means that a set of properties is considered as a *n-almost key*, if there exist from 1 to n exceptions in the dataset for this set of properties. For example, in *D1*, for the class *Person*, $\{LastName, StudiedIn\}$ is a *3-almost S-key* $\{i1, i2, i7\}$, a *4-almost F-key* $\{i1, i7, i3, i8\}$, and a *2-almost SF-key* $\{i1, i7\}$.

By definition, if a set of properties P is a *n-almost key*, every superset of P is also a *n-almost key*. We are interested in discovering only minimal *n-almost keys*, i.e., *n-almost keys* that do not contain subsets of properties that are *n-almost keys* for a fixed n.

Support. The discovered keys can be valid for a number of class instances that can be few, compared to the total number of class instances described in the dataset.

For *S-keys* and *SF-keys*, the support of a key is defined as the number of instances that are instantiated at least once for all the key properties. The coverage is the ratio of support to the total number of class instances.

Definition 9. (*S-key* and *SF-key* support). *Let P be a set of properties ($P \subseteq \mathcal{P}$), the support is defined as:*

$$support(P) = |x : \bigwedge_{p \in P} \exists v p(x, v)\ |$$

Definition 10. (*S-key* and *SF-key* coverage). *Let c be a class ($c \in C$), and P be a set of properties ($P \subseteq \mathcal{P}$), the support is defined as:*

$$coverage(P) = \frac{support(P)}{|\ x : c(x)\ |}$$

If the support/coverage is too low, a key can be discarded.

Since an *F-key* can be triggered even if the key property values are empty, the support can be simply defined as the number of all class instances and the coverage is always 100%.

Definition 11. (*F-key* support). *Let c be a class ($c \in C$), and P be a set of properties ($P \subseteq \mathcal{P}$), the support is defined as:*

$$support(P) = |\ x : c(x)\ |$$

3.5. Inclusion relations for key discovery

Relations between *S-key* and *F-key* sets discovered in a given dataset have been initially presented in [3]. These relations vary depending on the characteristics of the dataset used to discover keys. Table 3 summarises these relations and extends them with the case of *SF-keys*. Note that, Table 3 uses *S-keys* (resp. *SF-keys*, *F-keys*) to refer to the complete set of keys discovered in a dataset under the *S-key* (resp. *SF-key*, *F-key*) semantic.

Considering a dataset where no multivaluation and no empty values occur, the sets of keys discovered under the three semantics are the same. When this is not the case, different relations of inclusion appear. In the case where the multivaluation is present and all the properties are instantiated for all instances in the data, the sets of *SF-keys* and *F-keys* are identical while all the *S-keys* are also keys under both *F-key* and *SF-key* semantics. In the case when no multivaluation occurs but not all properties are instantiated, the set of *SF-keys* is identical to the set of *S-keys* while all *F-keys* are also keys under the *SF-key* and the *S-key* semantics. Finally, when both multivaluation and empty values appear in a dataset, all *S-keys* and *F-keys* are also keys under the *SF-key* semantic.

3.6. Discovery of n-almost keys from n-non keys

In order to check if a set of properties is a *n-almost key* for a class *c* in a dataset *D*, a naive approach would scan all the instances of a class *c* to verify if at most *n* instances share values for these properties. Even when a class is described by few properties, the number of candidate *n-almost keys* can be huge. For example, if we consider a class *c* that is described by 60 properties and we aim to discover all the *n-almost keys* that are composed of at most 5 properties, the number of candidate *n-almost keys* that should be checked will be more than 6 million. An efficient way to obtain *n-almost keys*, as already proposed in [20, 17], is to discover first all the sets of properties that are not *n-almost keys* and use them to derive the *n-almost keys*. Indeed, to show that a set of properties is not a *n-almost key*, it is sufficient to find only *(n+1)* instances that share values for this set. We call the sets that are not *n-almost keys*, *n-non keys*.

Definition 12. (*n-non key*). *Let P be a set of properties ($P \subseteq \mathcal{P}$) and n an integer. P is a n-non key if $|E_P| \geq n$.*

Note that, every subset of *P* is also a *n-non key* since the dataset also contains *n* exceptions for this subset. We are interested in discovering only maximal *n-non keys*, i.e., *n-non keys* that are not subsets of other *n-non keys* for a fixed *n*.

4. The BECKEY Approach

In this section, we introduce BECKEY, a general approach for the discovery of *n-non keys* following three different semantics. To compute the set of minimal *n-almost keys* without scanning all the data, the BECKEY approach, as SAKey [23], computes first the set of maximal *n-non keys* in the data and then uses them to derive the set of minimal *n-almost keys*. BECKEY provides a preprocessing step allowing to adapt to different semantics and new strategies that ensure the scalability of the key discovery.

We present the BECKEY approach through three phases: (1) the preprocessing step that aims at representing the data and converging the three semantics into one common structure, called *property-exception map*,

Multivaluation	no	yes	no	yes
Empty values	no	no	yes	yes
Relations	*S-keys = F-keys = SF-keys*	*F-keys = SF-keys* *S-keys ⊆ SF-keys* *S-keys ⊆ F-keys*	*S-keys = SF-keys* *F-keys ⊆ SF-keys* *F-keys ⊆ S-keys*	*S-keys ⊆ SF-keys* *F-keys ⊆ SF-keys*

Table 3: Relations between discovered sets of keys

(2) the discovery of maximal *(n+1)-non keys* (see Algorithm 1) and finally, (3) the derivation of *n-almost keys* from the set of*(n+1)-non keys* (see Algorithm 2). The last two phases are semantic agnostic, i.e., they do not depend on the semantic of the keys to be discovered. It is at the level of the construction of the *property-exception map* that a given semantic of keys is captured.

4.1. Preprocessing steps

One of the main contributions of BECKEY is to present a unified approach for the automatic discovery of different key semantics. To this aim, for each key semantic, we present a strategy that encodes keys' exceptions into a structure called the *property-exception map*. More precisely, a *property-exception map* is a compact structure that stores for each property all the sets of instances that violate the targeted key definition. This structure is the starting point of the discovery of *n-non keys*.

We first construct the *property-exception map*. Then, to improve the scalability of our approach, we introduce a new structure called *shared exception graph*. This graph helps to guide and circumscribe efficiently the non key search.

4.1.1. Construction of property-exception map

For each key semantic, a *property-exception map* represents a structure mapping each property to a collection of sets denoted as *property-exception sets*. Each *property-exception set* of a property represents a set of instances sharing the same valuation w.r.t. a given key semantic. More formally, we introduce the valuation function, mapping each instance to a set of values, for a given property.

Definition 13. (instance valuation: h_p). *Let $p \in P$ be a property and $i \in I$ an instance, the valuation $h_p(i)$ is a function mapping i to a set of values s.t. $h_p(i) = \{v | p(i, v) \in D\}$*

Intuitively, the valuation function $h_p(i)$ maps instances to their set of assigned values under the property p. When no value is associated to an instance i for a specific property p, $h_p(i)$ returns an empty set. For example, the valuation function for the property *StudiedIn* and the instance *i1* is: $h_{studiedIn}(i1) = \{UCC, Stanford\}$. Thanks to this valuation function we can now define the *property-exception set*.

Definition 14. (*property-exception set*: X_p). *Given a semantic of keys sem ∈{S-key, F-key, SF-key}, and a property $p \in P$, the set of instances $X_p = \{i_1, .., i_n\}$ is a property-exception set w.r.t. sem if:*

- *$h_p(i_1) \cap ... \cap h_p(i_n) \neq \emptyset$ and $|X_p| > 1$, when sem = S-key,*
- *$h_p(i_1) = ... = h_p(i_n) \neq \emptyset$ and $|X_p| > 1$, when sem = SF-key,*
- *$h_p(i_1) = ... = h_p(i_n)$ and $|X_p| > 1$, when sem = F-key.*

Intuitively, a *property-exception set* can be viewed as a set containing at least 2 instances sharing a common valuation, responsible for the violation of a given semantic of keys.

If $\{i_1, .., i_n\}$ is a *property-exception set* w.r.t. the *S-key* semantic then $i_1, .., i_n$ share at least one common value under the property p. For example, in Table 1, $\{i1, i2, i5, i7\}$ is a *property-exception set* for the property *StudiedIn* since the individuals $i1$, $i2$, $i5$ and $i7$ have all studied in a common place (i.e., *UCC*).

If $\{i_1, .., i_n\}$ is a *property-exception set* w.r.t. the *SF-key* semantic then $i_1, .., i_n$ are described exactly by the same set of values under the property p. For example, in Table 1, $\{i1, i7\}$ is a *property-exception set* w.r.t. the *SF-key* semantic since the individuals $i1$ and $i7$ have studied in the same universities (i.e., $\{UCC, Stanford\}$).

If $\{i_1, .., i_n\}$ is a *property-exception set* w.r.t. the *F-key* semantic then $i_1, .., i_n$ are described exactly by the same set of values under the property p and, unlike the *S-key* and *SF-key* semantics, this set can be empty (i.e., corresponding to a missing value). For example, $\{i3, i8\}$ is a *property-exception set* w.r.t. the *F-key* semantic since

for both individuals $i3$ and $i8$ there is no information on whether they have pursued any studies.

In order to obtain the complete set of *n-non keys* from the data, we store all the *property-exception sets* per property in a map called *property-exception map*. Note that, depending on the selected key semantic, a different *property-exception map* is created.

Definition 15. (*property-exception map*: μ). *Given a semantic of keys sem ∈ {S-key, F-key, SF-key}, the structure μ is a property-exception map w.r.t. sem if $\forall p \in P$, $\mu(p)$ maps the property p to the collection of property-exception sets $\{X_p^1,..,X_p^n\}$, s.t. $\forall X_p^k \in \mu(p)$, X_p^k is a maximal property-exception set w.r.t. sem.*

Intuitively, for each property, a *property-exception map* groups instances violating the selected key semantic using *property-exception sets*. To reduce the size of the map and speed-up the non key discovery process, only maximal *property-exception sets* are kept. A property for which the union of *property-exception sets* is equal or greater than n is a *n-non key* for the class w.r.t. the input semantic. Similarly, a property for which the union of *property-exception sets* is smaller than n, i.e., less than n instances share valuation for this property, is a *(n-1)-almost key* for the class w.r.t. the selected semantic and will not appear in the *property-exception map*. Overall, the *property-exception map* contains properties corresponding to single *n-non keys* that can be prolongated to maximal *n-non keys*.

We illustrate in Tables 4, 5 and 6 the *property-exception maps* corresponding to the dataset *D1* (see Table 1). These *property-exception maps* are going to be used for discovering respectively *S*, *SF*, and *F n-non keys*.

FirstName	{{i1, i6, i7}, {i2, i5}, {i3, i8}}
LastName	{{i1, i2, i7}, {i3, i8}}
SSN	{{i3, i8}}
DateOfBirth	{{i1, i5, i7}}
StudiedIn	{{i1, i2, i5, i7}, {i1, i6, i7}}
HasSibling	{{i1, i2}, {i1, i7}}

Table 4: *property-exception map* w.r.t. *S-key* semantic for dataset D1

Table 4 shows the *property-exception map* computed from the dataset *D1* (see Table 1) for the *S-key* semantic when n=2. In this map, the individuals $p1$, $p2$, $p5$ and $p7$ have all studied in *UCC*. Since they have all studied in at least one common place, this group of individuals violates the definition of *S-keys* and represents a *property-exception set* of the property *StudiedIn*.

FirstName	{{i1, i6, i7}, {i2, i5}, {i3, i8}}
LastName	{{i1, i2, i7}, {i3, i8}}
SSN	{{i3, i8}}
DateOfBirth	{{i1, i5, i7}}
StudiedIn	{{i1, i7}}

Table 5: *property-exception map* w.r.t. *SF-key* semantic for dataset D1

FirstName	{{i1, i6, i7}, {i2, i5}, {i3, i8}}
LastName	{{i1, i2, i7}, {i3, i8}}
SSN	{{i3, i8}, {i4, i5, i7}}
DateOfBirth	{{i1, i5, i7}, {i3, i6, i8}}
StudiedIn	{{i1, i7}, {i3, i8}}
HasSibling	{{i3, i5, i8}}

Table 6: *property-exception map* w.r.t. *F-key* semantic of dataset D1

Similarly, $p1$, $p6$ and $p7$ have all studied in *Stanford*, thus they represent another *property-exception set* w.r.t. the *S-key* semantic. Note that the people $p2$ and $p5$ have both studied in *UCD*. However, they do not appear as a *property-exception set* for the property *StudiedIn*. This *property-exception set* is subsumed by the set $\{p1, p2, p5, p7\}$ containing people that have studied in *UCC*. Indeed, all non keys that can be discovered using the *property-exception set* $\{p2, p5\}$ will be discovered by the *property-exception set* $\{p1, p2, p5, p7\}$.

Table 5 shows the *property-exception map* computed from the dataset *D1* that is used for discovering *SF-keys* when n=2. In this map, the individuals $p1$ and $p7$ have both studied in the exact same universities, i.e., *UCC* and *Stanford*. These two individuals represent a *property-exception set* that prevents *StudiedIn* to be considered as an *SF-key*. Note that the property *HasSibling* does not appear in the map since it corresponds to a minimal *1-almost SF-key*.

Table 6 shows the *property-exception map* computed from the dataset *D1* that is used for discovering *F-keys* when n=2. For all properties, each *property-exception set* w.r.t. *SF-key* semantic also appears as a *property-exception set* w.r.t. *F-key* semantic. In addition, a *property-exception map* w.r.t. *F-key* semantic contains *property-exception sets* for which instances map to missing values. This is the case for the individuals $p3$ and $p8$ that are grouped within a *property-exception set* since there is no information on whether they have pursued or not studies according to the example in Table 1.

Note that for *SF-key* (Table 5) and *F-key* (Table 6) semantics, each person appears in at most one *property-exception set* for each property. This is not the case for

for *S-key* semantic (Table 4). More generally, due to the different interpretation of the data incompleteness, in *SF-key* and *F-key* semantics, each instance can only appear in at most one of the *property-exception sets* associated to a property. This is not the case for the *S-key* semantic. This leads to the following remark:

Remark 1. *The collection of property-exception sets associated to each property p of a property-exception map w.r.t. SF-key or F-key semantics represents a partition.*

The partitioning condition is particularly helpful to distinguish in which case to apply some filtering methods without having the knowledge of the key semantic in our agnostic non key discovery method.

In the next section we introduce the *shared exception graph*, another structure that will assist with the efficient discovery of *n-non keys*.

4.1.2. Construction of shared exception graph

The goal of *n-non key* search is to extend *property-exception sets* valid for one property to a set of properties in order to discover maximal *n-non keys*. In some datasets the properties are numerous, and the number of candidate *n-non keys*, which is exponential in the number of properties, may be intractable. This is why we need a *n-non key* search mechanism that will avoid exploring irrelevant combinations of properties. For example, in the DBpedia dataset, the properties *depth* and *mountainRange* are never used to describe the same instances of the class *NaturalPlace*. Indeed, *depth* is used to describe natural places that are lakes while *mountainRange* natural places that are mountains. Therefore, *depth* and *mountainRange* cannot participate together in a *n-non key*. In general, if two properties have less than *n* instances in common, these two properties will never participate together to a *n-non key*. Therefore, such couples of properties should not be explored since they will never lead to *n-non keys*. In order to take this information into account, we introduce a graph that will be used to both drive and limit the *n-non key* search in an efficient way.

Definition 16. ***(shared exception graph:*** *$G(P,A,<)$). The graph $G(P, A, <)$ is the shared exception graph inferred from property-exception map μ if:*

- *P: the set of vertices represents the properties in μ,*
- *A: the set of edges is s.t.:*
  $A = \{(p, p') | (\bigcup_{X^k \in \mu(p)} X^k) \cap (\bigcup_{X^{k'} \in \mu(p')} X^{k'})| \geq n\}$,
- *<: defines a total order among the properties in P.*

Since in G two properties are linked only when the unions of their *property-exception sets* contain at least n common instances, a maximal *n-non key* is a subset of a clique in G. Implicitly, using G, we are able to limit the search of maximal *n-non keys* to the sets of properties belonging to at least one clique in G. In addition, to efficiently lead the search for *n-non keys*, the relation $<$ establishes a total order and prioritises the properties that will be explored first.

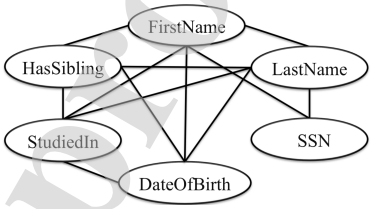

Figure 1: Shared exception graph w.r.t. the *S-key* semantic when n=2

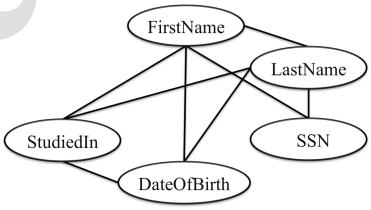

Figure 2: Shared exception graph w.r.t. the *SF-key* semantic when n=2

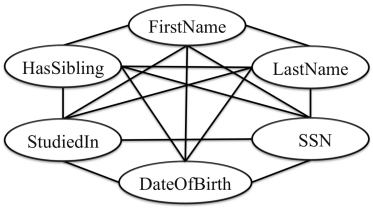

Figure 3: Shared exception graph w.r.t. the *F-key* semantic when n=2

Figure 1 shows the *shared exception graph* corresponding to the dataset *D1* w.r.t. *S-key* semantic when the number of exceptions is set to 2, i.e., n=2. Within the graph, the set of properties {*FirstName, LastName, SSN*} forms a clique. Therefore, the search space is pruned since no *n-non keys* including the set {*FirstName, LastName, SSN*} should be explored. Figures 2 and 3 show the *shared exception graph* for the *SF-key* and *F-key* semantics when n=2. Note that in this approach, unlike SAKey [23], it is not necessary to pre-

compute the set of all cliques associated to a *property-exception map* since it can be very time consuming on datasets with numerous properties. Contrariwise, before exploring a new property, this approach checks within G on the fly whether the new property is connected to all properties appearing in the current *n-non key* (see function *nextProperty* in Algorithm 1 below). Thanks to this checking, we only explore cliques of G.

Given the *property-exception map* and the *shared exception graph* that correspond to the data, the computation of *n-non keys* is agnostic from the selected key semantic. In the next section we introduce the *shared exception graph*, another structure that will assist with the efficient discovery of *n-non keys*.

4.2. n-non key mining

In order to compute the set of maximal *n-non keys*, we introduce the *maxNonKeyMining* algorithm (see Algorithm 1). This algorithm is a depth-first algorithm trying to discover the fastest possible the complete set of maximal non keys. More precisely, the algorithm incrementally searches for maximal non keys by checking if the current *n-non key* and a new property also form a *n-non key*. Before introducing *maxNonKeyMining* algorithm, we introduce the structures and subroutines used by the approach.

Definition 17. ***(intersect operator:*** $\bigotimes$***)*** *Given two collections of property-exception sets* $\mathcal{X}$ *and* $\mathcal{X}'$*, the intersect operator returns a new collection of property-exception sets defined as follows:*
$\mathcal{X} \bigotimes \mathcal{X}' = \{X^k \cap X^{k'} \mid X^k \in \mathcal{X}, X^{k'} \in \mathcal{X}',\ |X^k \cap X^{k'}| > 1\}$

In *maxNonKeyMining* algorithm, this operator is used for intersecting the current *property-exception set* X, valid for the current *n-non key* Q, with all *property-exception sets* of the next property to be explored. The output of this intersection operation is a collection of *property-exception sets*.

Since only maximal sets can lead to new *n-non keys*, the function ***maxSets*** takes as input a collection of *property-exception sets* $\mathcal{X}$ and returns the maximal sets of $\mathcal{X}$ s.t.

$X \in maxSets(\mathcal{X})$ iff $X \in \mathcal{X}$ and $\nexists X' \in \mathcal{X}$ s.t. $X \subseteq X'$.
The function *maxSets* is used both for filtering *property-exception sets* for each property p in $\mu(p)$, and for pruning non maximal *n-non keys* from the output.

The ***non key exception map*** ξ is a structure that stores for each explored non key Q the union of visited *property-exception sets* associated to Q, i.e., the set of instances that share values for the given non key.

The function ***nextProperty*** takes as input a property p, a non key Q, the *shared exception graph* $G(P, A, <)$ and returns the next property p' s.t. $p' > p$ and $\forall p'' \in Q, (p'', p') \in A$. In other words, p' corresponds to the next property following p in the ordered list of properties and is also connected to all the properties present in the non key Q. Implicitly $\{p'\} \cup Q$ forms a clique in G.

The algorithm *maxNonKeyMining* takes as input the number of authorised exceptions n, the current property to explore p, the current *property-exception set* X, the current *n-non key* Q, the *property-exception map* μ, the *shared exception graph* G, an initially empty *non key exception map* ξ and an initially empty set of maximal *n-non keys* M.

The algorithm returns the complete set of maximal *n-non keys* w.r.t. the key semantic encoded by the *property-exception map* μ. Initially *maxNonKeyMining* is called with the following parameters $maxNonKeyMining(n, p_1, I, \emptyset, \mu, G, \xi, M)$ with p_1 representing the first property to be explored and I the complete set of instances appearing in the data. At line 4, $\mathcal{X}$ represents the intersection of the current *property-exception set* X with all the *property-exception sets* associated to the property p in the *property-exception map* μ. At this line, several filterings are performed. First, by definition, the operator $\bigotimes$ prunes both *property-exception sets* with size smaller than 2 and duplicate *property-exception sets*. Then, the function ***maxSets*** ensures that only maximal *property-exception sets*, resulting from the intersection, are kept. As a result, $\mathcal{X}$, the collection of *property-exception sets* to be explored contains no subsumed *property-exception sets*. This function is called only when $\mathcal{X}$ does not represent a partition, i.e., it contains subsumed *property-exception sets* (Remark 1). Otherwise, in the case of *SF-key* and *F-key* semantics, the function $maxSets$ is not used.

Then, for each *property-exception set* in $\mathcal{X}$ (line 7), the algorithm computes the new non key Q' associated to the *property-exception set* X' (line 9). The set of violating instances appearing in the explored *property-exception set* X' is updated for each corresponding non key Q' (line 11) by using the *non key exception map* ξ. When the set of violating instances exceeds the number of authorised exceptions, the corresponding candidate *n-non key* is added to the non keys. The *non key exception map* allows only maximal non keys. Afterwards, the algorithm tries to extend recursively Q' with p' (line 12). p' is the next property in the order forming a clique with the properties appearing in Q' (line 13).

From lines 7-13, the algorithm tries to extend the current non key Q with the current property p. To explore

Algorithm 1: maxNonKeyMining

Input: n, number of authorized exceptions
p, current property to explore
X, current *property-exception set* set
Q, current n-non key
μ, *property-exception map*
G, *shared exception graph*
ξ, non key *property-exception map*
M set of maximal n-non keys

Output: M set of maximal n-non keys

```
1  if p = nil or ∃Q_k ∈ maxNonKeys s.t.
2      {p} ∪ Q ∈ Q_k then
3      return
4  X ← {X} ⨂ μ(p)
5  if ¬isAPartition(X) then
6      X ← maxSets(X)
7  for each X' ∈ X do
8      Q' ← {p} ∪ Q
9      ξ(Q') ← X' ∪ ξ(Q')
10     if |ξ(Q')| ≥ n then
11         M ← maxSets(M ∪ Q')
12     p' ← nextProperty(p, Q, G)
13     maxNonKeyMining( n, p', X', Q', μ, G, ξ, M )
14 p' ← nextProperty(p, Q, G)
15 maxNonKeyMining( n, p', X, Q, μ, G, ξ, M )
```

all combinations of properties, in line 15, the algorithm tries to extend the current non key without p. The algorithm stops when the exploration reaches a maximal clique in G or when the next non key to be composed is already included in one of discovered non keys (lines 1-3).

If the algorithm is launched with n=2 (i.e., at least 2 exceptions authorised) in the dataset *D1* presented in the Table 1, then:
2-S non keys = {{*FirstName, LastName, DateOfBirth, StudiedIn, HasSibling*}, {*FirstName, LastName, SSN*}} under the *S-key* semantic,
2-SF non keys ={{*FirstName, LastName, StudiedIn*}, {*FirstName, LastName, SSN*}} under the *SF-key* semantic and
2-F non keys ={{*FirstName, LastName, SSN, DateOfBirth, StudiedIn, HasSibling*}} under the *F-key* semantic.

4.3. Key Derivation

The computation of minimal *n-almost keys* is done using the maximal *(n+1)-non keys*. A set of properties is a *n-almost key*, if it is not equal or included to any maximal *(n+1)-non key*. Indeed, when all the *(n+1)-non keys* are discovered, all the sets not found as *(n+1)-non keys* will have at most n exceptions (i.e., *n-almost keys*). To execute this step, we use *KeyDerivation*, an efficient algorithm introduced in [23] (see Algorithm 2). This algorithm takes as input the *compSets*, i.e., a collection of complement sets for each maximal *n-non key* discovered from the data. To compute efficiently the minimal *n-non keys* from this collection of complement sets, the properties are ordered by their frequencies in the *compSets*. At each iteration, the most frequent property is selected and all the *n-non keys* involving this property are discovered recursively. For each selected property p, p is combined with the properties of the selected complement sets that do not contain p. Indeed, only complement sets that do not contain this property can lead to the construction of minimal *n-almost keys*. When all the *n-almost keys* containing p are discovered, this property is eliminated from every complement set. When at least one complement set is empty, all the *n-non keys* have been discovered. If every property has a different frequency in the complement sets, all the *n-almost keys* found are minimal *n-almost keys*. In the case where two properties have the same frequency, additional heuristics should be taken into account to avoid computations of non minimal *n-almost keys*.

If the algorithm is launched with the *2-non keys* discovered by the *maxNonKeyMining* algorithm then:
1-almost S-keys = {{*SSN, DateOfBirth*}, {*SSN, StudiedIn*} {*SSN, HasSibling*}} under the *S-key* semantic,
1-almost SF-keys ={{*SSN, DateOfBirth*}, {*SSN, StudiedIn*}, {*HasSibling*}} under the *SF-key* semantic and
1-almost F-keys ={} under the *F-key* semantic.

5. Experiments

We conduct two experiments in order to assess the quality of links inferred by each semantic through two real-world scenarios. To this aim, we evaluate the quality of the different types of keys w.r.t. their inferred links by computing the Recall, Precision and FMeasure scores [4]

The resulting scores are based on the comparison between the links obtained by each semantic and the correct links provided in the goldstandard.

In the first real-world scenario, we evaluate the capability of the three key semantics to retrieve correct *owl:SameAs* links among the instances of two versions

[4] For convenience, to compare FMeasure scores (usually expressed between [0,1]) to Precision and Recall scores (expressed in %), we express FMeasure scores between [0, 100], (i.e, ×100).

Algorithm 2: keyDerivation

```
Input: compSets set of complement sets
Output: KeySet set of n-almost key s
KeySet ← ∅
orderedProperties =
getOrderedProperties(compSets)
for each p_i ∈ orderedProperties do
    selectedCompSets ←
    selectSets(p_i, compSets)
    if (selectedCompSets == ∅) then
        KeySet = KeySet ∪ {{p_i}}
    else
        KeySet = KeySet ∪
        {p_i×keyDerivation(selectedCompSets)}
    compSets = remove(compSets, p_i)
    if ( ∃ set ∈ compSet s.t. set == ∅ ) then
        break
return KeySet
```

of the same knowledge base that evolves over time. In the first set of experiments two different versions of DBpedia are compared. This scenario uses BECKEY as a solution for tackling the temporal evolution of a given dataset.

In the second real-world scenario, we evaluate the capability of the three semantics to discover the correct *owl:SameAs* links among the instances of two different knowledge bases built separately, conforming to different ontologies. In the second set of experiments, one version of DBpedia is linked to one version of YAGO [5]. This scenario uses BECKEY as a solution for tackling the heterogeneity of knowledge bases.

For both scenarios the objective of the experiments is twofold: compare the different key semantics through the quality of the links inferred and understand in which case a key semantic may be more suitable for data linking than another. To this aim, for all semantics of keys, the data linking step is computed as follows. Given two datasets containing instances of the same class, the keys discovered by each semantic are applied to provide *owl:sameAs* links among the instances of the two datasets. No similarity measures are applied even if [23] has shown that similarity measure may positively impact the computed scores. This simple data linking step allow us to have a fair comparison of the different key semantics and have a clearer understanding of the cases where a specific key semantic may be more suitable.

[5] https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago

Setting. In all experiments, the data are stored in a dictionary-encoded map, where each distinct string appearing in a triple is represented by an integer. The experiments have been executed on a single machine with 16GB RAM, processor 2.50GHz, 8-core. A time limit of 2 days has been set for each execution.

5.1. Linking data of evolving Knowledge Bases

In this first set of experiments, we compare the quality of the three different key semantics for linking two versions of a same knowledge base that evolved over time (see Section 5.1.1). In Section 5.1.2, we demonstrate the impact of exceptions in the data linking process for all three semantics.

Datasets. The experiment is executed on 2 different versions of DBpedia, the latest version published in October 2016[6] (DBpedia 2016) and a previous version published in 2013[7] (DBpedia 2013). The selected datasets are highly heterogeneous as shown in this section, and allow a comparison of the three key semantics when data evolve through time (i.e., instances, properties and property values are added or suppressed). 266 DBpedia classes are randomly chosen for the experimental evaluation. Since the three key semantics differ on the hypothesis on data completeness, such an evolutive dataset is used to compare the three approaches. To compare the results of different semantics of keys on significant classes, we select the classes for which at least one set of keys discovered for a given semantic and a given number of exceptions (among 0, 10, 20, 30) returns an FMeasure equal or higher than 60. 155 classes correspond to this criteria. Therefore, our experiments focus on these 155 classes. These classes contain from 100 to more than 200.000 instances (see Figure 4), from 419 to 1.611.241 triples (see Figure 5) and finally from 7 to 462 properties (see Figure 6). The classes in all three figures are sorted according to their number of instances, triples and properties respectively.

Data linking challenge. The objective of this experiment is to compare the linking performance of key based approaches on the problem of data linking. To this aim, we compare the links found among instances of a given class described by two datasets, i.e., db2013 and db2016. Table 7 shows the evolution of the data with respect to the two versions of DBpedia. More precisely, as shown in this table, 42,22% of the triples in the initial version (db2013) are removed from the newer

[6] https://wiki.dbpedia.org/downloads-2016-10

[7] https://wiki.dbpedia.org/services-resources/datasets/data-set-39/downloads-39

version (db2016) while more than 50% of the triples of the version db2016 are new. Similarly, 19,57% of the initial instances are not appearing in the newer version while around 40% of the instances in the db2016 are new. Finally, this evolution is not only in terms of triples and instances but also in terms of properties. Therefore, 29,13% of the properties used in the older version are not present in the new version while 24,32% of the properties in the db2016 are new. Even if the task of linking two different versions of DBpedia would seem straightforward the heterogeneity and the significant changes at different levels, as shown in Table 7, make this task a challenging problem.

In order to verify the quality of the discovered keys, a goldstandard containing the common instances of both datasets has been created. Note that instances in all versions of DBpedia are using the same URI prefix, i.e., `https://dbpedia.org/resource`.

	db2013	**db2016**
# CommonTriples	57,78%	46,88%
# OldTriplesGone	42,22%	-
# NewAddedTriples	-	53,12%
# CommonInstances	80,43%	60,35%
# OldInstancesGone	19,57%	-
# NewInstancesAdded	-	39,65%
# CommonProperties	70,87%	75,68%
# OldPropertiesGone	29,13%	-
# NewPropertiesAdded	-	24,32%

Table 7: Dataset evolution

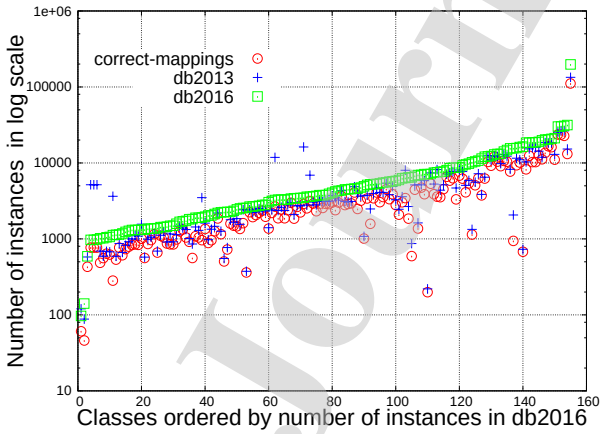

Figure 4: Number of Instances

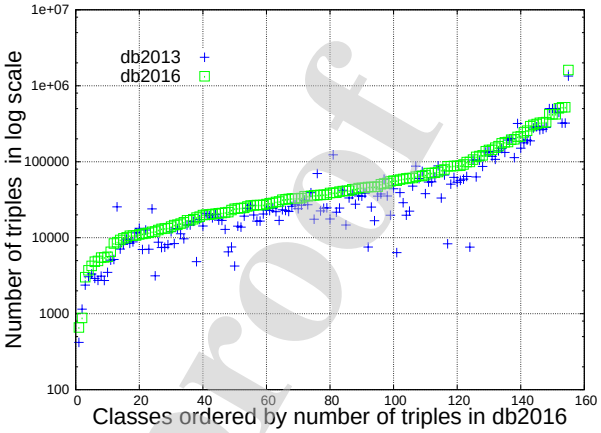

Figure 5: Number of Triples

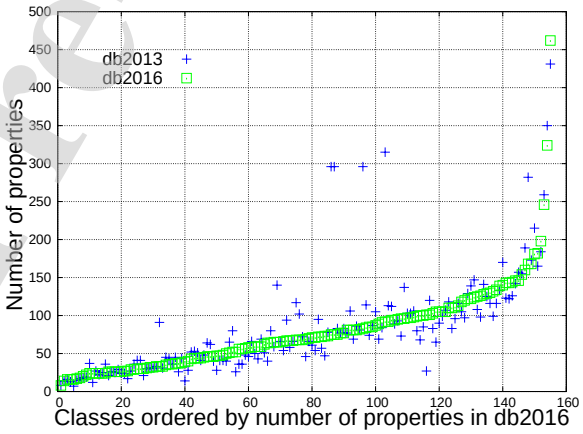

Figure 6: Number of Properties

5.1.1. Data linking using 0-almost keys

In this first set of experiments we focus on keys with 0 exceptions. More precisely, we provide the linking results of all three key semantics in the selected classes and an analysis on which semantic is more appropriate to use depending on the nature of the data. For each class, and each key semantic, the keys discovered in the most recent version of DBpedia (db2016) are used to identify the common instances between the two datasets (db2013 and db2016). We consider the newest dataset as a knowledge base that is more complete and contains less errors than the previous versions.

Figure 7 shows the Recall obtained by each semantic, for each class. In the x-axis, classes are ordered by the overall Recall, i.e., the sum of Recall scores returned for

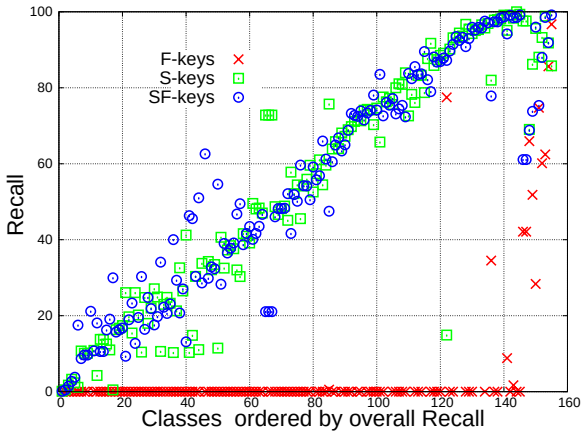

Figure 7: Recall comparison

the three semantics of keys. *S-keys* and *SF-keys* lead to relatively similar Recall scores. As it can be easily observed, Recall scores are uniformly distributed among the classes. *S-keys* and *SF-keys* lead to Recall scores from 0 up to 99% depending on the tested class. In the case of low Recall, this is usually due to the data incompleteness on the properties participating in the keys. For example, in the case of the class *HollywoodCartoon* we obtain a very low Recall using *S-keys* or *SF-keys* (lower than 1%). The *S-key* (resp. *SF-key*) with the highest support is the *IMDB-ID* of the Hollywood cartoon. However, only 139 instances out of 1258 participating in the goldstandard are containing information about their imdb-id in the version db2016 while none of the instances of the version db2013 contains this property. This means that this key leads to zero links. All the other keys discovered of this class have a very low support. For example the key {*Homepage*}, a key that seems to be relevant for the data linking task, is filled for only one Hollywood cartoon in the db2016 version. Nevertheless, Figure 7 shows that both *S-keys* and *SF-keys* bring very good Recall scores for numerous classes (more than 50 classes obtain a Recall higher than 70%).

The Recall obtained for *F-keys* is equal to 0 for 117 of the 155 classes since no *F-keys* are discovered for these classes (see Table 8). Note that, *F-keys* are discovered for only 15 classes while for 23 classes the algorithm reaches the time limit of 2 days. These poor results are mainly due to the "strict" nature of the definition of *F-keys*. For example, the set of properties {*Name, Birthplace*} is an *SF-key* for the class *VolleyballPlayer* and states that a volleyball player can be identified using her/his name and birth place. This *SF-key* obtains 51,09% of Recall, 100,00% of Precision and 67,63 of FMeasure. This set of properties is not discovered as an *F-key* due to the fact that the birthplace of some players is not registered in the data. This means that when no exception is allowed, the *F-key* semantic fails to provide linking solutions for the majority of the classes tested.

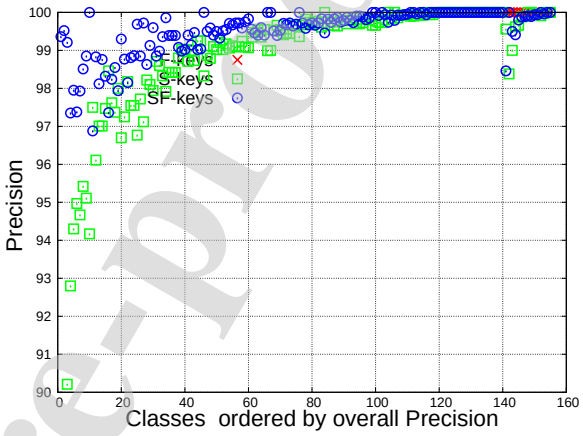

Figure 8: Precision

Figure 9: FMeasure

From Precision perspective, i.e., the capacity to retrieve correct sameAs links among the obtained links, Figure 8 shows that *S-keys* and *SF-keys* are highly accurate and return a Precision score that starts at 90% for the poorest classes and rapidly reaches almost 100% for the remaining classes. The high Precision scores for both *S-keys* and *SF-keys* highlight the relevance and high accuracy of key based approaches for data linking.

Since no *F-keys* are discovered for most of the classes, the Precision in this case cannot be defined (i.e., Precision = UNDEF).

Figure 9 shows both the accuracy and the coverage of the discovered links by computing the FMeasure, when it is possible (i.e., when Precision ≠ UNDEF). As observed before, Precision scores obtained by *S-keys* and *SF-keys* are between 90% and 100% therefore the FMeasure for these semantics is following the Recall pattern while obtaining higher scores. Concerning the FMeasure of *F-keys*, the score can be computed only for 15 out of 155 classes since no *F-keys* have been discovered for the remaining classes.

In Table 9, we provide a comparison of the different key semantics w.r.t. the best FMeasure scores obtained for each class. When 0 exceptions are allowed, the *S-key* semantic obtains better FMeasure scores for more than 50% of the classes (88 out of 155 classes). More than 40% of the classes are obtaining the highest FMeasure using the *SF-key* semantic while no class is obtaining the best FMeasure using the *F-key* semantic.

5.1.2. Data Linking with n-almost Keys

In this section we evaluate the linking quality of discovered keys varying the number of allowed exceptions from 0 to 30. More precisely, we analyse the effect of the number of allowed exceptions on Recall, Precision and FMeasure. As previously stated when no exception is allowed, key based data linking approaches are very accurate and show very high Precision scores. This is not the case for Recall for which scores vary uniformly from very low to high depending on the class. Intuitively, by allowing exceptions we expect an improvement on the overall linking quality of keys (i.e., FMeasure) accepting to trade accuracy (i.e., Precision) for better coverage (i.e., Recall).

Figure 10 shows the evolution of the average Recall, Precision and FMeasure obtained by *S-keys* when exceptions are authorised. When no exception is allowed, the average Precision is 99% and the average Recall is 55%. As shown in this figure, allowing exceptions leads to a relevant increase of the Recall without decreasing significantly the Precision. For example, in the class *Food*, allowing 10 exceptions allows the FMeasure of *S-keys* to increase from 7,36 to 53,58. The best FMeasure of 81 is obtained when 20 exceptions are allowed. In this case, the Recall is reaching 80% while the Precision is at 88%.

Figure 11 shows the evolution of the average Recall, Precision and FMeasure obtained by *SF-keys* when exceptions are authorised. The average Recall, Precision and FMeasure obtained when using the *SF-keys* follow

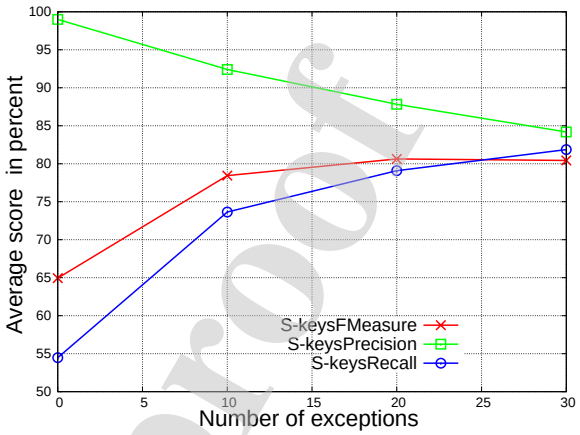

Figure 10: *S-key* RPF when exceptions are allowed

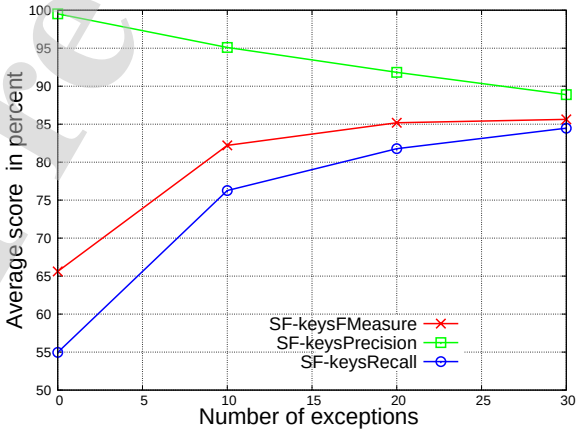

Figure 11: *SF-key* RPF when exceptions are allowed

the same pattern as *S-keys* (Figure 10) while they tend to be slightly better. For example, in the class *Food*, authorising 10 exceptions allows the FMeasure of *S-keys* to increase from 4,57 to 79,61. For more than 20 exceptions, the overall FMeasure remains steady. The best overall quality for the *SF-keys* is obtained when allowing up to 20 exceptions, and reaches an FMeasure of 85.

Figure 12 shows the evolution of average Recall, Precision and FMeasure obtained by *F-keys* when exceptions are authorised. The scores obtained using *F-keys* differ completely from the scores obtained by *S-keys* and *F-keys*. To interpret these results, it is important to highlight that for most of the classes there are either no *n-almost F-keys* discovered or the computation

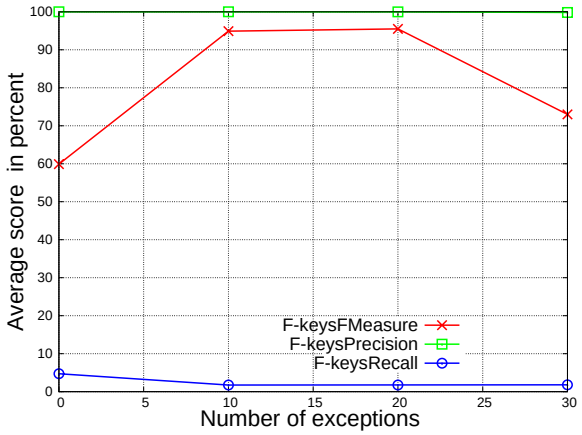

Figure 12: *F-key* RPF when exceptions are allowed

time limit of 2 days has been reached, (see Table 8). As a consequence, the average Recall computed over the classes not reaching the time limit remains extremely poor (less than 5%), even when increasing the number of allowed exceptions. Precision and FMeasure scores can only be computed over the classes for which at least one *owl:sameAs* link has been discovered. For 0, 10, 20, and 30 exceptions there are respectively only 15, 3, 3, and 4 of these classes (see Table 8). As a consequence, the high average Precision and FMeasure shown in Figure 12 are not representative of all the tested classes.

Table 8 reports for each key semantic, and an increasing number of allowed exceptions varying from 0 to 30: the number of classes linked (i.e., at least one *owl:sameAs* link discovered), the number of classes not-linked (i.e., no *owl:sameAs* link discovered), and the number of classes reaching the time limit of 2 days. For *SF-keys* there is only 1 class over 155 reaching the time limit for 20 and 30 exceptions. Similarly, for *S-keys* there are only 2 classes (resp. 1) over 155 reaching the time limit for 0 (resp. 20 and 30) exceptions.

Table 9 compares the three semantics w.r.t. the number of classes for which each semantic returned the best FMeasure. The comparison is shown for exceptions from 0 up to 30. When no exception is authorised, the *S-key* semantic is superior to the other semantics and obtains the highest FMeasure for most of the classes. When increasing the number of allowed exceptions from 10 to 30, the *SF-key* semantic brings better results than the *S-key* semantic. Unlike *S-key* and *SF-key* semantics, *F-key* semantic is never obtaining the best FMeasure score.

Overall, the experiments show that the *S-keys* and *SF-keys* have a high linking capability in terms of FMeasure when exceptions are allowed. The experiments also demonstrate that increasing the number of exceptions also increases significantly the FMeasure by trading "a bit of" Precision for a more important improvement in Recall. When no exception is allowed, the average FMeasure obtained using *S-keys* and *SF-keys* is similar. *SF-keys* lead to a better average FMeasure as the number of allowed exceptions increases up to 20. However, it is shown that *S-keys* can obtain the best FMeasure score for numerous classes whatever the number of exceptions is applied. On the other hand, *F-keys* lead to poor results even when the number of exceptions increases.

5.1.3. Semantic comparison in evolving Knowledge Bases

The goal of this section is to provide a more complete analysis of the experimental evaluation and give more insights on when a semantic is more appropriate to be used, depending on the nature of the data. First, we provide a deeper understanding on why *F-keys* return poor results on data linking. Then, we compare the results of *S-keys* and *SF-keys* based on example classes for which each key semantic provides better linking results.

Understanding why *F-keys* are less suitable for data linking

Overall, the *F-keys* fail to provide good linking results. As shown in Table 9, *F-keys* are never obtaining the best FMeasure among the three semantics, for all numbers of exceptions used in this experimental evaluation. The *F-key* semantic fails to provide any keys for the vast majority of the classes tested. Even when exceptions are allowed, the discovery of *F-keys* does not ensure to find an *F-key*.

Unlike *S-keys* and *SF-keys*, *F-keys* assume that when a property is not istantiated for all instances, this property cannot be a key. To explain why such a semantic leads to poor linking results let us note that in the data we might encounter instances that are not valued for a given property. This can be either due to the incompleteness of the data (i.e., a missing value) or to the fact that a property is not meaningful for all instances. Let us consider the class *Swimmer*. For example, the set of properties {*Birthplace, Deathdate*} is both an *S-key* and an *SF-key* for a swimmer. It is obvious that this key can be used to link only swimmers that are no longer in life. This means that the property *Deathdate* is applicable to only a part of the instances and the absence of a deathdate is likely not due to the incompleteness of the data. The *F-key* semantic fails to capture keys of this nature. In the case of the set of properties {*Height, Name*} that

	0 Exp.			10 Exp.			20 Exp.			30 Exp.		
	#cL	#cN	#cT	#cL	#cN	#cT	#cL	#cN	#cT	#cL	#cN	#cT
S-keys	153	0	2	155	0	0	154	0	1	154	0	1
SF-keys	155	0	0	155	0	0	154	0	1	154	0	1
F-keys	15	117	23	3	63	89	3	52	100	4	42	109

Table 8: Number of classes: linked (#cL), not-linked (#cN), and reaching time limit (#cT)

	0 Exp.	10 Exp.	20 Exp.	30 Exp.
S-keys	88	62	48	49
SF-keys	69	95	107	107
F-keys	0	0	0	0

Table 9: Number of classes having best FMeasure per number of exceptions for each semantic

is an *S-key* and an *SF-key*, both of the properties can be instantiated for all swimmers since they all have a name and a height. As we can observe in the data, there exist swimmers for which the height information is not provided in the website of wikipedia. Therefore, this key is going to be used to compare only swimmers for which their height exists in the data. *F-key* semantic will not be able to discover keys composed of properties where some values might be missing. Note that these sets of properties when discovered as *S-keys* obtain together a Recall of 26% and have both a Precision of 100%, proving the keys with a coverage lower than 100% can still provide a great accuracy.

Understanding the differences between *SF-keys* and *S-keys*

As highlighted in sections 5.1.1 and 5.1.2, *SF-keys* and *S-keys* obtain similar scores and provide very good linking results. The main question is how to choose the most appropriate semantic according to the nature of the data to be linked. In this part, we try to answer this question by discussing different examples where *SF-keys* or *S-keys* are leading to better linking results.

When are *SF-keys* more suitable? In the case of the class *Language*, when 10 exceptions are allowed, we observe that the FMeasure obtained using the *S-keys* is only 5,65 while the FMeasure for *SF-keys* is 81,22. This big difference is due to the *SF-key*(*Spokenin, Name*) expressing that the set of places where a language is spoken along with its name can be used to uniquely identify a language. This *SF-key* alone leads to an FMeasure of 78,60 and is responsible for the linking of 4409 out of 6799 languages between the two versions of DBpedia. The set of properties {*Spokenin, Name*} is not an *S-key* since there exist numerous languages that are spoken by many countries.

Another example where *SF-keys* are leading to better linking results is the class *Food* describing different meals. In this class, the *SF-keys* allowing 10 exceptions lead to an FMeasure of 79,61 while the *S-keys* lead to an FMeasure of 53,58. Thus, a difference of 26,03 between the two FMeasure scores can be observed. This difference is mainly due to the *SF-key*(*Name, Ingredient*) that manages to link 1168 out of 3554 foods registered in DBpedia, obtaining an FMeasure of 49,47. It is quite obvious that identifying and comparing two meals based on the sets of their ingredients is more relevant than comparing them based on one of the ingredients they are made of.

Both examples help us understand that for certain multivalued properties it is more relevant to consider the complete set of values.

When are *S-keys* more suitable? In the class *MusicGenre*, we observe a significant difference between the FMeasure scores obtained using *S-keys* and *SF-keys*. When 0 exceptions are applied, *S-keys* lead to an FMeasure of 83,71 while *SF-keys* lead to an FMeasure of 24,60. This difference is due to the *S-key(Genre-inv, Name)*. Note that this set of properties is discovered both as an *S-key* and an *SF-key*. When discovered as an *S-key* this key alone is responsible for an FMeasure of 77,09 while when discovered as an *SF-key* it returns an FMeasure of 5,07. The property *Genre-inv* associates a genre to artists, albums, songs etc. It is easily understandable that this property evolves with time since new artists, albums, songs are continuously added to DBpedia and associated to a genre. Therefore, such a property is more relevant to participate in an *S-key* since it can bring no links for the genres that evolve in time.

Another similar example comes from the class *SoccerManager*. In the case where 0 exceptions are authorised, the FMeasure obtained by *S-keys* is 97,75 while the one obtained by *SF-keys* is 75,83. The *S-key* with the highest linking capacity is the set of properties {*Birthdate, Team, Name*}. This *S-key* expresses that two soccer managers are considered identical when they have managed the same team, they have the same birthdate and the same name. This *S-key* manages to link the most of the soccer managers in the goldstandard (5757 out of

	0 Exp.	10 Exp.	20 Exp.	30 Exp.
S-keys	40	69	69	58
SF-keys	11	30	45	45
F-keys	463	1617	1810	1970

Table 10: Average execution time in minutes for each semantic

10277 instances). On the other hand, the corresponding minimal *SF-key(Team, Name)*, is a subset of the *S-key*. This *SF-key* alone obtains an FMeasure score of 22,6 and leads to much fewer links. Intuitively, the *SF-key* considers two managers from different versions of DBpedia to be identical if they have managed the exact same set of teams despite the evolution due to the years. This interpretation could be suitable only for managers that have stopped their activity and no longer manage new teams. Overall, *S-keys* tend to be more suitable for properties that evolve within time.

5.1.4. Time Performance

In Table 10, we provide the average execution time of the 155 classes for each semantic when 0, 10, 20 and 30 exceptions are allowed. In average, BECKEY discovers *S-keys* in 40 minutes when 0 exceptions are allowed and around an hour when the number of exceptions increases. The discovery of *SF-keys* outperforms the discovery of *S-keys*. This is due to the fact that unlike *S-keys*, in *SF-keys* the data are partitioned (see Remark 1) and the structure to explore (i.e., *property-exception map*) is much more compact. Compared to the other semantics, the discovery of *F-keys* is one order of magnitude slower when 0 exceptions are allowed. The difference increases with the number of allowed exceptions. This is due to the fact that every instance is associated to a value (possibly null) for all the properties, therefore the search space to explore is much larger. Additionally, for this semantic the *shared exception graph* contains only one clique representing the complete set of properties, thus the search space cannot be pruned.

5.2. Linking data of heterogeneous Knowledge Bases

In this set of experiments, we compare the linking capability of each of the three semantics for integrating two heterogeneous knowledge bases conforming to different ontologies. As before, we also analyse the impact of allowed exceptions on the data linking performance.

Datasets. (DBpedia 2016 VS. YAGO 2016) Each experiment takes as input the complete et of triples of a given class found in two real-world knowledge bases, DBpedia and YAGO[8], both released in 2016. The data for this experiment have been provided initially in [24] containing data for ten different classes. These classes describe different domains such as organisations, actors, locations etc.

Data linking challenge. While both DBpedia and YAGO contain information extracted from Wikipedia, they remain highly heterogeneous. Since the two knowledge bases conform to different ontologies the vocabulary used to describe identical properties may differ. Note that DBpedia usually contains more properties than YAGO to describe one class. Additionally, a significant part of YAGO is manually verified before being released. As a result, the use of different strategies and methodologies used for the creation of each knowledge base lead to the construction of highly heterogeneous datasets. The authors of [24] have executed a step of property alignment where different properties referring to the same information have been matched. Then, the properties of YAGO have been renamed using its DBpedia counterparts. In the final datasets released, only triples containing properties found in both datasets have been kept. Table 11 shows for each of the provided classes the number of common properties, the number of triples for both YAGO and DBpedia and finally the number of instances for each of them. Note that the goldstandard for each class containing the complete set of correct links among the datasets is available.

5.2.1. Data linking using 0-almost keys

In this experiment, we compare the linking results of the three key semantics when 0 exceptions are authorised. For each class and each semantic, the keys have been discovered in YAGO and then used to provide links towards DBpedia. This choice has been made since YAGO tends to be cleaner and contains less errors. The first three columns of Table 12 contain the FMeasure scores that show the linking capability of each key semantic.

As it can be easily noticed, similarly to the previous experiments, for all classes, *F-keys* obtain an FMeasure score of 0 showing that *F-keys* fail to provide any links when 0 exceptions are allowed. This is due to the strict nature of this semantic. On the contrary, *S-keys* and *SF-keys* provide much better results, reaching an FMeasure score of 56,33 for the class *Book* when using *SF-keys* and an FMeasure score of 79,51 for the class *City* when using *S-keys*.

[8]https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

Class	# CommonProp.	# InstancesDB	# InstancesYAGO	# TriplesDB	# TriplesYAGO
Actor	16	5807	108424	47173	514772
Album	5	85072	137178	297072	381119
Book	7	29849	41855	123586	92499
City	17	19229	83561	634437	1100738
Film	9	82126	123916	673518	533542
Mountain	5	16401	32984	67720	116781
Museum	7	1826	21152	7970	81671
Organisation	17	183716	430331	2243603	2205427
Scientist	10	18409	93169	64178	335633
University	9	10353	23352	119615	131657

Table 11: DBpedia and YAGO datasets' statistics

	0 Exp.			10 Exp.			20 Exp.			30 Exp.		
	S	***SF***	***F***	***S***	***SF***	***F***	***S***	***SF***	***F***	***S***	***SF***	***F***
Actor	68,96	14,37	0	70,92	28,66	0	71,74	29,02	0	72,18	29,08	0
Album	58,32	56,33	0	58,32	56,33	0	58,32	56,33	0	58,32	56,33	0
Book	61,72	58,12	0	62,35	58,78	0	62,32	58,76	0	63,22	59,78	0
City	79,51	41,41	0	87,99	68,88	0	85,30	78,36	0	85,31	78,36	0
Film	46,73	45,94	0	50,08	47,19	0	50,95	47,7	0	51,39	50,09	0
Mountain	3,1	2,95	0	3,1	3	0	3,1	3,06	0	3,19	3,06	0
Museum	24,55	20,49	0	30,48	22,22	0	30,48	22,33	0	31,28	40,3	0
Organisation	15,61	8,89	0	30,46	12,88	0	31,32	13,25	0	31,53	13,38	0
Scientist	50,25	2,47	0	59,16	3,82	0	59,48	6,67	0	59,53	7,04	0
University	22,13	10,68	0	38,72	28,71	0	56,15	28,94	0,99	56,71	29,42	0,99

Table 12: FMeasure scores (×100) for *S-keys*, *SF-keys* and *F-keys* when 0, 10, 20 and 30 exceptions are authorised

When no exceptions are allowed, we observe that *S-keys* lead to significantly better FMeasure scores than *SF-keys*. In the case of the class *Actor*, *S-keys* provide an FMeasure score of 68,96 while *SF-keys* only reach 14,37. We also observe a similar gap in the linking performance the class *Scientist* (i.e., a difference between the FMeasure scores higher than 45 when 0 exceptions are authorised).

5.2.2. Data linking using n-almost keys

In this section, we compare the linking capacity of the three key semantics, when varying the number of allowed exceptions from 0 to 30. In Table 12 we provide, for each authorised threshold of exceptions, the FMeasure score.

When exceptions are authorised, the FMeasure scores increase in most of the cases for both *S-keys* and *SF-keys*. For example, when 30 exceptions are allowed, the FMeasure of *S-keys* for the class *University* is more than twice greater than the FMeasure score when 0 exception is allowed (i.e., from 22,13 to 56,71). Similarly, when 30 exceptions are allowed, the FMeasure of *SF-keys* for the class *City* goes from 41,41 to 78,36. Nevertheless, this increase of number of exceptions does not seem to benefit the *F-keys*. As shown in Table 12, *F-keys* provide 0 links except for the case of the class *University* where they obtain an FMeasure score of 0,99.

Overall, comparing *S-keys* and *SF-keys*, *S-keys* tend to provide significantly better linking results except for the class *Museum* when 30 exceptions are allowed.

5.2.3. Semantic comparison in heterogeneous Knowledge Bases

This section provides insights of when each of the three semantics leads to better linking results in the case of heterogeneous knowledge bases.

Understanding why *F-keys* are less suitable for data linking

As explained extensively in the previous experiment, (see Section 5.1), *F-keys* fail to provide good linking results due to their "strict" definition. More precisely, no *F-keys* are discovered for the 9 out of 10 classes even when exceptions are authorised. Therefore, no links can

be produced. In the case of the class *University*, only one *F-key* is found when 20 and 30 exceptions are authorised. In this case, the *F-key(islocatedin, Preflabel, Wascreatedonyear)* is able to produce 44 links out of the 8885 existing links, leading to an FMeasure score of 0.99.

Understanding the differences between *SF-keys* and *S-keys*

As shown in Table 12, both *S-keys* and *SF-keys* tend to bring good linking results. Nevertheless, in this experiment we observe the superiority of *S-keys* over *SF-keys* independently of the number of exceptions.

When are *S-keys* more suitable? The superiority of *S-keys* over the *SF-keys* can be explained by the incompleteness of the data which is an expected characteristic of data coming from the Web. For example, the set of properties {*Hasnumberofpeople, Haspopulationdensity, Islocatedin-inv*} is an *S-key* for the class *City*. This key provides an FMeasure of 23,38. On the contrary, the *SF-key*(*Haspopulationdensity, Islocatedin-inv*) leads to an FMeasure of only 0,99. This is due to the fact that in order to consider two cities to be the same, the complete set of institutions, schools, museums, etc., located in the two cities found in each dataset (described by the property *Islocatedin-inv*), have to be the same. As we can easily observe, this is not the case in this example since the two knowledge bases do not contain exactly the same information. Therefore, as suggests the definition of *SF-keys*, comparing the complete set of values can be an obstacle when data come from different sources that do not always contain all the information. Moreover, the different representation of the data in different datasets is another reason why *SF-keys* do not provide good linking results. For example, the property *Preflabel* is used in both knowledge bases to provide a label for each instance in the data. While for YAGO this property contains always only one label per instance, this is not the case for DBpedia. For example, the actor Johan Paulik has two labels in DBpedia, *"paulik, johan"* and *"johan paulik"* while only the second one is found in YAGO. Since the property *Preflabel* appears often in both *S-keys* and *SF-keys*, it is easy to see that *S-keys* containing this property can provide high scores of FMeasure, unlike *SF-keys* that are not able to deal with this heterogeneity. Therefore, depending on the nature of the data and the structure of each knowledge base, *S-keys* or *SF-keys* are more appropriate.

When are *SF-keys* more suitable? In this experiment, the only case where *SF-keys* bring better linking results than *S-keys* is when 30 exceptions are authorised for the class *Museum*. In this example, all keys found by both semantics are the same except for the *SF-key(Islocatedin, Preflabel)* that brings additional links, leading to an increase of the FMeasure. This *SF-key* contains the multivalued property *Islocatedin* that describes the location of a museum in different levels of abstraction, for example, country, city, region etc. Therefore, it leads to a significant number of links when discovered as a part of an *SF-key*.

5.3. Choosing the appropriate semantic depending on the data

In this section we have presented an extended experimental analysis of the three key semantics and their linking capacity over numerous classes. Two scenarios of data linking have been explored. In the first scenario, the data linking task concerns two versions of the same knowledge base that evolves over time while in the second, the data linking task concerns two heterogeneous knowledge bases created independently. This experimental evaluation shows the poor linking capacity of *F-keys*. This is mainly due to the fact that the *F-keys* expect data to be complete, a characteristic rarely observed in data published on the Web. Unlike *F-keys*, *S-keys* and *SF-keys* show their linking capability for different classes in both scenarios presented in this paper. Depending on the nature of the data, one key semantic tends to be more appropriate than the other for the data linking task. The main difference between these two semantics is how they deal with multivalued properties. This difference has a strong impact in both the discovery of keys and the data linking process. For some properties, the complete set of values associated to each instance is more suitable to identify the instance. For example, a recipe will be more easily determined by using the complete list of its ingredients than only one or a subset of its ingredients. Such properties are more appropriate to be used in *SF-keys*. For other properties, only one value will help to differentiate one instance from another. For example, with the years passing, a movie director will tend to have an increasing number of directed movies. As a result, the set of movies associated to the property *DirectorOf* will tend to grow with the years. In this context, a movie director will be more easily identifiable using one movie than the full list of movies. In this case, where the description of an instance for a given property may evolve, *S-keys* are more appropriate then *SF-keys* for the data linking task. Finally, in the case of heterogeneous and incomplete knowledge bases *S-keys* are also more suitable. Therefore, an analysis of the nature of each property describing a class is necessary in order to decide which semantic is more appropriate. Overall, in both *S-keys* and

SF-keys exceptions have been shown to provide better linking results.

6. Conclusion

In this paper we present BECKEY, a generic approach for the discovery of keys in large RDF knowledge bases. BECKEY is able to discover keys under three different semantics introduced in the context of the Semantic Web. The different key semantics are presented and compared theoretically. In order to discover keys of different semantics, a structure able to capture these semantics is introduced. We propose a semantic agnostic algorithm for the efficient discovery of almost keys in large knowledge bases. The extrinsic experimental evaluation has allowed a comparison of the linking capacity of the three semantics. It is demonstrated that *F-keys* are not useful for data linking when data are incomplete, which is often the case for the data on the Web. Overall, *S-keys* and *SF-keys* tend to bring similar results of a significant linking quality, proving their relevance for the data linking task. Both *S-keys* and *SF-keys* obtain better linking results in terms of FMeasure when few exceptions are allowed, while no improvement is stated in the case of *F-keys*. The experimental results show that *S-keys* are more appropriate when no exceptions are allowed. When exceptions are allowed the linking efficiency of *S-keys* and *SF-keys* depends on the level of multivaluation, completeness and heterogeneity of the data to be linked. In the case of evolving knowledge bases, where the descriptions of instances through multivalued properties remain more or less stable, *SF-keys* show a slightly better linking performance than *S-keys*. In the other hand, in the case of heterogeneous or incomplete knowledge bases, *S-keys* clearly dominate *SF-keys*. As a result of the experimental study, depending on the completeness and the nature of the data, we provide different examples where properties are more appropriate to participate in *S-keys* or *SF-keys* and exhibit cases where *S-keys* or *SF-keys* tend to bring better results when exceptions are authorised. As future work we plan to introduce a new approach that is able to select automatically the appropriate semantic depending on different criteria including completeness, multivaluation, evolution and heterogeneity of the data. BECKEY along with the datasets and evaluations are available at `https://github.com/danaiS/BECKEY`.

Acknowledgments

This work has been supported by the Science Foundation Ireland under Grant Nos. 12/RC/2289-P2 and 16/SP/3804, which are co-funded under the European Regional Development Fund. This work has also been supported by the French National Research Agency under the Investments for the Future Program, referred as ANR-16-CONV-0004 (#Digitag).

References

References

[1] Al-Bakri, M., Atencia, M., David, J., Lalande, S., Rousset, M., 2016. Uncertainty-sensitive reasoning for inferring sameas facts in linked data, in: ECAI, pp. 698–706.

[2] Al-Bakri, M., Atencia, M., Lalande, S., Rousset, M., 2015. Inferring same-as facts from linked data: An iterative import-by-query approach, in: AAAI, pp. 9–15.

[3] Atencia, M., Chein, M., Croitoru, M., Jerome David, M.L., Pernelle, N., Saïs, F., Scharffe, F., Symeonidou, D., 2014a. Defining key semantics for the rdf datasets: Experiments and evaluations, in: ICCS.

[4] Atencia, M., David, J., Euzenat, J., 2014b. Data interlinking through robust linkkey extraction, in: ECAI, pp. 15–20.

[5] Atencia, M., David, J., Scharffe, F., 2012. Keys and pseudo-keys detection for web datasets cleansing and interlinking, in: EKAW, pp. 144–153.

[6] Efthymiou, V., Papadakis, G., Papastefanatos, G., Stefanidis, K., Palpanas, T., 2017. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. Inf. Syst. 65, 137–157.

[7] Fan, W., Fan, Z., Tian, C., Dong, X.L., 2015. Keys for graphs. PVLDB 8, 1590–1601.

[8] Farah, H., Symeonidou, D., Todorov, K., 2017. Keyranker: Automatic rdf key ranking for data linking, in: KCAP, ACM. p. 7.

[9] Ferrara, A., Nikolov, A., Scharffe, F., 2011. Data linking for the semantic web. Int. J. Semantic Web Inf. Syst. 7, 46–76.

[10] Hajmohammadi, M.S., Ibrahim, R., Selamat, A., Fujita, H., 2015. Combination of active learning and self-training for cross-lingual sentiment classification with density analysis of unlabelled samples. Information sciences 317, 67–77.

[11] Hogan, A., Zimmermann, A., Umbrich, J., Polleres, A., Decker, S., 2012. Scalable and distributed methods for entity matching, consolidation and disambiguation over linked data corpora. J. Web Semant. 10, 76–110.

[12] Hu, W., Chen, J., Qu, Y., 2011. A self-training approach for resolving object coreference on the semantic web, in: WWW, pp. 87–96.

[13] Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H., 1999. Tane: An efficient algorithm for discovering functional and approximate dependencies. The Computer Journal 42, 100–111.

[14] Nentwig, M., Hartung, M., Ngomo, A.N., Rahm, E., 2017a. A survey of current link discovery frameworks. Semantic Web 8, 419–436.

[15] Nentwig, M., Hartung, M., Ngonga Ngomo, A.C., Rahm, E., 2017b. A survey of current link discovery frameworks. Semantic Web 8, 419–436.

[16] Nikolov, A., d'Aquin, M., Motta, E., 2012. Unsupervised learning of link discovery configuration, in: ESWC, pp. 119–133.

[17] Pernelle, N., Saïs, F., Symeonidou, D., 2013. An automatic key discovery approach for data linking. J. Web Sem. 23, 16–30.

[18] Petrovski, P., Bizer, C., 2019. Learning expressive linkage rules from sparse data. Semantic Web pre-press, 1–19.

[19] Recupero, D.R., Consoli, S., Gangemi, A., Nuzzolese, A.G., Spampinato, D., 2014. A semantic web based core engine to

efficiently perform sentiment analysis, in: European Semantic Web Conference, Springer. pp. 245–248.
[20] Sismanis, Y., Brown, P., Haas, P.J., Reinwald, B., 2006. Gordian: efficient and scalable discovery of composite keys, in: VLDB, pp. 691–702.
[21] Song, D., Heflin, J., 2011. Automatically generating data linkages using a domain-independent candidate selection approach, in: ISWC, pp. 649–664.
[22] Soru, T., Marx, E., Ngonga Ngomo, A.C., 2015. Rocker: A refinement operator for key discovery, in: WWW, pp. 1025–1033.
[23] Symeonidou, D., Armant, V., Pernelle, N., Saïs, F., 2014. Sakey: Scalable almost key discovery in rdf data, in: ISWC, pp. 33–49.
[24] Symeonidou, D., Galárraga, L., Pernelle, N., Saïs, F., Suchanek, F., 2017. Vickey: Mining conditional keys on knowledge bases, in: ISWC, pp. 661–677.
[25] Volz, J., Bizer, C., Gaedke, M., Kobilarov, G., 2009. Discovering and maintaining links on the web of data, in: ISWC, pp. 650–665.
[26] W3C-Recommendation, 2009. Owl2 web ontology language: Direct semantics, in: http://www.w3.org/TR/owl2-direct-semantics.
[27] Zhu, Y., Zhu, A.X., Song, J., Yang, J., Feng, M., Sun, K., Zhang, J., Hou, Z., Zhao, H., 2017. Multidimensional and quantitative interlinking approach for linked geospatial data. International Journal of Digital Earth 10, 923–943.

Declaration of Interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed.

We further confirm that the order of authors listed in the manuscript has been approved by all of us. We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He/she is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs.

We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author and which has been configured to accept email from Danai.symeonidou@inra.fr

Signed by all authors as follows:

Danai Symeonidou
28/06/19

Vincent Armant
28/06/19

Nathalie Pernelle
28/06/19

Comment citer ce document :
Symeonidou, D., Armant, V., Pernelle, N. (2020). BECKEY: Understanding, comparing and discovering keys of different semantics in knowledge bases. Knowledge-Based Systems, 105708. , DOI : 10.1016/j.knosys.2020.105708

Credit Author Statement

Conceptualization	Danai Symeonidou, Vincent Armant, Nathalie Pernelle
Methodology	Danai Symeonidou, Vincent Armant, Nathalie Pernelle
Software	Danai Symeonidou, Vincent Armant
Validation	Danai Symeonidou, Vincent Armant, Nathalie Pernelle
Formal Analysis	Danai Symeonidou, Vincent Armant, Nathalie Pernelle
Investigation	Danai Symeonidou, Vincent Armant, Nathalie Pernelle
Resources	Danai Symeonidou, Vincent Armant
Data Curation	Danai Symeonidou
Writing – Original Draft	Danai Symeonidou, Vincent Armant, Nathalie Pernelle
Writing – Review & Editing	Danai Symeonidou, Vincent Armant
Visualization	Vincent Armant
Supervision	Danai Symeonidou, Vincent Armant
Project Administration	Danai Symeonidou
Funding Acquisition	Vincent Armant

Comment citer ce document :
Symeonidou, D., Armant, V., Pernelle, N. (2020). BECKEY: Understanding, comparing and discovering keys of different semantics in knowledge bases. Knowledge-Based Systems, 105708. , DOI : 10.1016/j.knosys.2020.105708