VICKEY: Mining Conditional Keys on Knowledge Bases

Danai Symeonidou[1], Luis Galárraga[2], Nathalie Pernelle[3],
Fatiha Saïs[3], Fabian Suchanek[4]
[1]INRA, France [2]Aalborg University, Denmark
[3]LRI, France [4]Télécom ParisTech, France

Abstract. A conditional key is a key constraint that is valid in only a part of the data. In this paper, we show how such keys can be mined automatically on large knowledge bases (KBs). For this, we combine techniques from key mining with techniques from rule mining. We show that our method can scale to KBs of millions of facts. We also show that the conditional keys we mine can improve the quality of entity linking by up to 47 percentage points.

1 Introduction

Recent years have seen the rise of large knowledge bases (KBs), such as YAGO [26], Wikidata [29], and DBpedia [18] on the academic side, and the Google Knowledge Graph [7] or Microsoft's Satori graph on the commercial side. These KBs contain millions of entities (such as people, places, or organizations), and millions of facts about them. This knowledge is typically expressed in RDF [19], i.e., as triples of the form $\langle Einstein, won, NobelPrize\rangle$. A key constraint on such data specifies that no two distinct entities can share a certain set of properties (e.g., no two people share given name, family name, and birthdate). Key constraints are used for applications such as knowledge base fusion [8], knowledge base enrichment [22] and data linking [1, 9, 23].

It is impractical to specify keys manually for large KBs (with millions of triples and hundreds or thousands of properties). Therefore, several approaches [4, 21, 25, 27] have been developed to automatically discover keys from RDF data. However, these works have also shown that for several datasets, there are no or only few keys that are valid in the entire dataset. This is the reason why we aim to mine *conditional keys* in this paper, i.e., keys that are valid in only a part of the data.

A conditional key is an axiom saying that under particular conditions, no two distinct entities can have the same values on a particular set of properties. For example, we can say that at a German university, no two professors can advise the same doctoral student. The situation might be different at a French or American university – hence the key is "conditional" to German universities. In this paper, we distinguish conditional keys from classical keys, which hold for an atomic class (or for every tuple of a table in a relational database). Conditional keys can express constraints on entities and are strictly more expressive than classical keys. Therefore, they can be more productive in tasks such as entity

linking – as we show in our experiments. Apart from this, conditional keys carry knowledge in themselves. For example, it is interesting to know that France allows several advisors, while Germany does not.

Mining conditional keys automatically from the data is a challenging endeavor, for several reasons. First, the KBs we consider here contain only binary relations, which means that keys usually do not live in a single table, but can be a join of up to a dozen relations. Second, KBs are usually incomplete [12]. If a student at a German university has only one advisor in the KB, she could still have several in real life. Thus, any approach that automatically mines conditional keys risks being misled. Finally, the challenge is to scale: Today's KBs contain millions of statements. This means that there are billions of possible conditions and property combinations that could define a conditional key. In the example, professors could be distinguished by their doctoral students, but also by their given and family name or by their discipline and birthdate. These keys could hold only for German professors, only for Danish ones, only for professors at a certain university in Mexico, or only for professors born in a certain city in Iowa. This huge search space is one of the main reasons why there is today no approach that could mine conditional keys on KBs.

Our proposal is to combine key mining techniques [27] with techniques from rule mining [13]. More precisely, VICKEY discovers first the set of maximal non-keys from which the conditional keys can be computed. Thus, the search space can be significantly reduced while avoiding to scan all the data. Secondly, VICKEY applies a breadth-first strategy to discover frequent candidate conditional keys and efficiently check their validity. More precisely, our contributions are as follows:

- We develop an algorithm that can mine conditional keys efficiently.
- We show that our method scales gracefully to KBs of millions of facts.
- We show that the use of our conditional keys improves the F1 measure of KB linking by up to 47 percentage points over the use of classical keys.

The rest of this paper is structured as follows. We discuss related work in Section 2, and introduce preliminaries in Section 3. In Section 4 we present our approach. Section 5 showcases our experiments, before Section 6 concludes.

2 Related Work

Relational Databases. Two types of key discovery approaches have been proposed for relational databases [16,24]: data-driven [24], where keys are discovered from the tuples in a table, and schema-based [16], where property combinations of a certain size are generated and then checked on the tuples. Such approaches cannot be applied directly to KBs, because they are geared towards relations that contain one single value for each subject. In KBs, in contrast, a relation can contain several objects for the same subject.

Knowledge Bases. Approaches for KBs [3, 4, 21, 25, 27] can be roughly classified into two groups, depending on how they deal with multivalued properties [2]: The *forall-key approaches* [4,25] discover keys that fire when two entities

share *all* values for each property. In [4], the authors have developed a level-wise schema-based approach based on TANE [17], to discover pseudo-keys (keys with exceptions). Rocker [25] is a refinement-operator-based approach that efficiently discovers pseudo-keys using a top-down strategy guided by a discriminability score function designed for forall-keys. The *some-key approaches* [3, 21, 27], on the other hand, discover keys that fire as soon as two entities share *at least one* value for each property. Some-keys can be particularly useful under the Open World Assumption (OWA), where the KB may not contain all relevant facts. Thus, it is for example sufficient that two researchers share their last name, their first name, and one of their publications in order for them to be linked – even if the KB does not know all of their publications. [3] discovers discriminative combinations of corresponding properties that can be used to link two datasets with different schemas. KD2R [21] extends the relational data-driven approach of [24] in order to exploit ontology axioms (such as the subsumption relation) and considers multivalued properties. SAKey [27] introduces additional filtering and pruning techniques to discover efficiently some-keys with exceptions. To avoid scanning the entire dataset, both KD2R and SAKey discover first the maximal non-keys and then derive the keys from this set. Yet, none of these approaches is able to mine the *conditional keys* that we aim at in this paper.

Conditional Functional Dependency Mining. A conditional functional dependency (CFD) expresses a functional dependency between two sets of attributes that holds on a subset of tuples [6]. For example, a CFD could state that when two customers are based in the UK, the zipcode uniquely determines the city. A conditional key is a particular type of CFD, where the second set of attributes is a unique identifier for a record in the database. CFD discovery has been addressed in [6, 10, 15]. The work of [6] uses a breadth-first strategy inspired by the schema-based approach TANE [17]. FastCFD [10] finds a canonical cover of all minimal CFDs that satisfy a given support using a depth-first strategy. Compared to [6] (which works well when the number of tuples is large), FastCFD [10] is efficient when the number of attributes is large. Nevertheless, none of these approaches is able to discover conditional keys in KBs, because they cannot efficiently deal with multivalued properties and would require a post-processing to mine conditional keys from the obtained CFDs.

Rule Mining. Finally, one possibility to find conditional keys would be to use rule mining approaches. AMIE [13], e.g., can learn logical rules with up to 4 atoms on KBs that contain millions of facts. However, even relatively simple conditional keys can easily contain five or more atoms. This leads to an exponential increase of the search space that such rule mining approaches cannot handle – as we show in our experiments. In [5], the authors propose a more efficient rule mining approach that implements a series of parallelization and pruning techniques. However, it focuses only on Horn rules with ungrounded atoms. Thus, it cannot be applied for conditional key discovery. In [11], the authors propose to apply, first, a rule mining tool like [5, 13] and then refine the obtained rules by adding negated atoms. However, the result are rules with negations, not con-

Table 1. Example dataset

	FirstName	LastName	Gender	Lab	Nationality
r1	Claude	Dupont	Female	Paris-Sud	France
r2	Claude	Dupont	Male	Paris-Sud	Belgium
r3	Juan	Rodríguez	Male	INRA	Spain, Italy
r4	Juan	Salvez	Male	INRA	Spain
r5	Anna	Georgiou	Female	INRA	Greece, France
r6	Pavlos	Markou	Male	Paris-Sud	Greece
r7	Marie	Legendre	Female	INRA	France

ditional keys. In this paper, we propose a method called VICKEY, which we believe is the first approach to mine conditional keys efficiently on large KBs.

3 Preliminaries

Knowledge Bases. The knowledge bases that we consider here [18, 26, 29] use a set $\mathcal{I}$ of instances (such as a researcher identified as *r1*), a set $\mathcal{L}$ of literals, a set $\mathcal{P}$ of properties (such as *nationality*), and a set $\mathcal{C}$ of class names (such as *country*). A fact is a triple of a subject $s \in \mathcal{C} \cup \mathcal{I}$, a property $p \in \mathcal{P}$, and an 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 facts constitutes a knowledge base (KB)[1]. Given a KB $\mathcal{K}$, a *dataset* $\mathcal{D}$ for a class c of $\mathcal{K}$ is the set of all facts that have as subject an instance of c or of a subclass of c. Table 1 shows an example dataset[2] about researchers $r1, ..., r7$, each having the properties *firstName*, *lastName*, *gender*, *lab*, and *nationality* – with one or more objects for each property. When $\mathcal{D}$ is given, we write $p(x, y)$ to mean $p(x, y) \in \mathcal{D}$.
Keys. In our setting, a key is defined as follows [21, 27].

Definition 1. ***(Key)*** *A key in a dataset $\mathcal{D}$ is a set of properties $p_1, \ldots, p_n$ of $\mathcal{D}$ such that:*

$$\forall x, y,\ u_1, \ldots, u_n \left(\bigwedge_{i=1 \ldots n} p_i(x, u_i) \wedge p_i(y, u_i) \Rightarrow x = y \right)$$

In our example, the property set {*lastName*, *gender*} is a key while {*lab*, *nationality*} is not a key, because $r3$ and $r4$ are both *Spanish*. Note that {*lab*, *nationality*} is a forall-key since no two people share the lab and the entire set of nationalities.

Definition 2. ***(Maximal non-key)*** *A maximal non-key for a dataset $\mathcal{D}$ is a set of properties P of $\mathcal{D}$ such that P is not a key, and the addition of any other property makes P a key.*

In our example, {*firstName, lastName, lab*} is a maximal non-key, because adding any other property makes the set a key.

[1] The KBs considered in this work do not contain blank nodes.
[2] For readability, the table does not distinguish literals and instances.

Key Discovery. To discover the keys of a dataset automatically, a naive algorithm would have to compare all subjects of the dataset to all other subjects – which is prohibitively expensive. To avoid this complexity, the SAKey algorithm [27] first finds maximal non-keys. This is more efficient, because to verify that a set of properties is a non-key, it suffices to find two subjects that share values for these properties. SAKey starts with property combinations that contain only a single property, and incrementally adds more and more properties until it arrives at maximal non-keys. When it has found all non-keys, all other property combinations must be keys – which is what SAKey outputs.

Conditional Keys. Our example in Table 1 shows two researchers with the last name Dupont. Therefore the property *lastName* is not a key. The combination {*firstName, lastName*} is not a key either, because there are two researchers with the same first and last names. However, when we restrict our set of researchers to those working at INRA, the property *lastName* identifies researchers uniquely. In contrast, this is not true for the researchers in Paris-Sud. Thus, {*lastName, lab*} is not a key in general. We say that *lastName* is a *conditional key* for people working at INRA. In this work, we chose to focus on conditions that can be expressed using constraints on property values. More formally, a condition is a pair composed of a property p and an object o, written $p = o$ (e.g., $lab = INRA$). A condition cd with property p and object o holds for a subject x, written $cd(x)$, if $p(x, o)$. In the example, the condition $lab = INRA$ holds for $r3$, $r4$, $r5$ and $r7$.

Definition 3. ***(Conditional key)*** *A conditional key for a dataset $\mathcal{D}$ is a non-empty set of conditions $\{cd_1, ..., cd_n\}$ and a non-empty set of properties $\{p_1, ..., p_m\}$ of $\mathcal{D}$ (disjoint from the properties in the conditions), such that:*

$$\forall x, y,\ u_1, \ldots, u_m \left(\bigwedge_{i=1..n} (cd_i(x) \wedge cd_i(y)) \ \wedge \bigwedge_{i=1..m} (p_i(x, u_i) \wedge p_i(y, u_i)) \Rightarrow x = y \right)$$

Definition 4. ***(Minimal conditional key)*** *A conditional key with conditions CD and properties P is minimal, if the removal of a condition in CD, the removal of a property from P, or the transfer of a property p from CD to P (with the corresponding removal of the condition), all result in something that is neither a conditional key nor a key.*

In our example, *lastName* is a conditional key with condition *nationality* = *Spanish* $\wedge$ $lab = INRA$, but this conditional key is not minimal, because there exists a simpler version of the key with fewer conditions, namely *nationality* = *Spanish*. In the same vein, {*lastName*} with the condition *gender=male* is not a minimal conditional key, because {*lastName, gender*} is a key.

The *support* of a conditional key with properties $\{p_1, ..., p_m\}$ and conditions $\{cd_1, ..., cd_n\}$ is the number of subjects x such that $\bigwedge_{i=1..m} \exists u_i : p_i(x, u_i)$ and $\bigwedge_{i=1..n} cd_i(x)$. A proportional version of the support, which we call the *coverage*, measures the ratio of subjects in the dataset identified by the conditional key. In our example, the support of the key {*lastName*} under the condition *lab=INRA* is 4, and the coverage is $\frac{4}{7}$, since there are 7 subjects.

Keys in OWL. Conditional keys can be defined in the ontology language OWL2 [20]. OWL2 allows defining keys not just on atomic classes (such as

researcher), but also on more complex class expressions. We can define, e.g., the class "Researchers who work at INRA" as $c = Researcher \sqcap \exists lab.\{INRA\}$. Then, $\{lastName\}$ is a key on the dataset of c according to Definition 1.

4 Mining Conditional Keys

We now present our approach to automatically discover conditional keys on a dataset. To learn conditional keys under the Open World Assumption, we assume that all instances in a dataset refer to distinct real world objects, and that all unknown values are different from the existing ones in the dataset [7,13,14,21,27]. The discovery of simple keys alone already requires checking a large number of property combinations (of which there are $2^{|\mathcal{P}|}$ in total, where $\mathcal{P}$ is the set of properties). Discovering conditional keys is even more complex, since the search space is in the order of $O(|\mathcal{V}|^{|\mathcal{P}|})$, where $\mathcal{V}$ is the set of objects in the dataset. Our algorithm can discover conditional keys efficiently in spite of this large search space. Our method takes as input a dataset $\mathcal{D}$ and a threshold θ for the minimal support of the discovered keys. We proceed in three phases:
1) Discovery of non-keys: Instead of exploring the whole set of combinations of properties, we focus our search on those combinations that are not keys.
2) Generation of Conditional Key Graphs: We use the non-keys to generate candidate keys, which we store in conditional key graphs.
3) Mining of Conditional Key Graphs: The conditional key graphs are then mined for minimal conditional keys.

4.1 Discovery of non-keys

The naive method to mine conditional keys explores all possible combinations of properties and conditions in the input KB and verifies whether they fulfill Definition 3. Such an approach is infeasible on large datasets. Our main idea (the key insight, so to speak) is the following (see [28] for more details):

Observation 1 ***(Conditional Keys and Non-Keys)*** *Given a minimal conditional key for a dataset $\mathcal{D}$ with properties P and conditions $\{p_1 = o_1, ..., p_n = o_n\}$, the set of properties $P \cup \{p_1, ..., p_n\}$ must be a non-key for $\mathcal{D}$.*

This follows from Definition 4. In our example from Table 1, {*firstName*} is a minimal conditional key with condition *gender=Female*, and {*gender, firstName*} is a non-key. Thus, if we want to mine the complete set of minimal conditional keys, it suffices to consider only the property combinations given by non-keys. Since maximal non-keys are super-sets of all other non-keys (Definition 2), it is sufficient to explore only property combinations given by maximal non-keys. The maximal non-keys in the input dataset can be mined efficiently with the SAKey algorithm [27] (Section 3). Thus, we concentrate in the following on mining the conditional keys from these maximal non-keys. As a running example, consider again the dataset in Table 1. It contains two maximal non-keys: {*firstName*, *lastName*, *lab*} and {*firstName*, *gender*, *lab*, *nationality*}.

4.2 Generation of Conditional Key Graphs

Our method for discovering conditional keys from non-keys relies on a modifiable data structure that we call a *conditional key graph*. Such a graph is a tuple $\langle P^k, P^c, cond, G\rangle$ with the following components:

- P^k and P^c are disjoint sets of properties, called *key properties* and *condition properties*, respectively.
- *cond* is a set of conditions on P^c.
- G is a directed graph. Each node v is associated to a set $v.p \subseteq P^k$ and to a boolean flag $v.explore$ set by default to true. There is a directed edge from u to v if $u.p \subset v.p$ and $|u.p| = |v.p| - 1$.

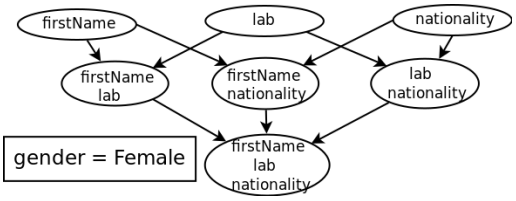


Fig. 1. Example of a conditional key graph with $P^k = \{$*firstName, lab, nationality*$\}$, $P^c = \{$*gender*$\}$, $cond = \{gender = Female\}$.

We construct the initial conditional key graphs with Algorithm 1. This algorithm takes as input the dataset, the support threshold θ, and the non-keys discovered in Section 4.1. We first construct all possible conditions $p = a$ that combine a property p from the non-keys with an instance or literal a from the dataset (Lines 2-3). Conditions with support less than θ are not considered (Line 4). We then look at all non-keys N in which p appears (Line 5). The conditional key graph for the condition $p = a$ will contain as nodes all subsets of $N \setminus \{p\}$ (Line 6) except the empty set (Line 7). As an example, let us consider again the dataset of Table 1 and its two maximal non-keys {*firstName*, *lastName*, *lab*} and {*firstName*, *gender*, *lab*, *nationality*}. Figure 1 depicts the conditional key graph associated to the condition *gender* = *Female* constructed by Algorithm 1.

Lemma 1. ***(Graph Construction)*** *If Algorithm 1 is given a dataset $\mathcal{D}$, a complete set of maximal non-keys $\mathcal{N}$ for $\mathcal{D}$ and a support threshold θ, then for each conditional key of $\mathcal{D}$ with a single condition and with at least support θ, there is a graph in the output that contains the key condition and a node with the key properties.*

Lemma 1 follows from the fact that Algorithm 1 (a) iterates over all conditions $p = o$ with a least support θ and (b) considers *all* the possible subsets of properties $2^{N\setminus\{p\}}$ with $N \in \mathcal{N}$ (except for $\emptyset$). From Observation 1 we recall that for any conditional key with properties P and condition $p = o$, the set of properties $P \cup \{p\}$ must be a non-key. Thus, from the completeness of our set of maximal non-keys, it follows that our graph contains in its nodes all possible keys with support higher than θ for a given condition $p = o$.

Algorithm 1: ConstructGraphs

Input: dataset $\mathcal{D}$, min. support θ, set of non-keys $\mathcal{N}$
Output: set of conditional key graphs $\mathcal{G}$

1 $\mathcal{G} \leftarrow \emptyset$
2 **for** $p \in \bigcup_{N \in \mathcal{N}} N$ **do**
3 **for** $a \in \mathcal{I} \cup \mathcal{L}$ *such that* $\exists x : p(x, a) \in \mathcal{D}$ **do**
4 **if** *number of* x *with* $p(x, a)$ *is at least* θ **then**
5 $V \leftarrow \emptyset$
6 **for** $N \in \mathcal{N}$ where $p \in N$ **do** $V \leftarrow V \cup 2^{N \setminus \{p\}}$
7 $V \leftarrow V \setminus \emptyset$
8 **for** $v \in V$ **do** $v.explore = true$
9 $E \leftarrow \{u, v \in V : u.p \subset v.p \wedge |u.p| = |v.p| - 1\}$
10 $P^k = \bigcup_{v \in V} v.p$
11 $P^c = \{p\}$
12 $cond = \{p = a\}$
13 $\mathcal{G} \leftarrow \mathcal{G} \cup \{\langle P^k, P^c, cond, (V, E)\rangle\}$
14 **return** $\mathcal{G}$

4.3 Mining of Conditional Key Graphs

Mining conditional keys. We mine the conditional key graphs for keys with Algorithm 2. It takes as input a dataset, a support threshold, and the set of conditional key graphs constructed in the previous phase. All these conditional key graphs have conditions of size 1 (see Algorithm 1). The algorithm proceeds in batches, looking first at the graphs with condition size 1, then size 2, etc. (Lines 2-3). For each batch, it mines the conditional keys (Line 6). The graphs in one batch are then post-processed (Line 7) to give rise to new graphs with conditions of larger size. The algorithm iterates until all sizes are processed (Line 4).

Algorithm 2: ConditionalKeyDiscovery

Input: dataset $\mathcal{D}$, minimum support θ, set of conditional key graphs $\mathcal{G}$
Output: set of minimal conditional keys CKs

1 $CKs \leftarrow \emptyset$
2 **for** $size = 1$ *to* ∞ **do**
3 $\mathcal{G}' \leftarrow \{g \in \mathcal{G} : |g.cond| = size\}$
4 **if** $\mathcal{G}' = \emptyset$ **then return** CKs
5 **for** $\langle P^k, P^c, cond, G\rangle \in \mathcal{G}'$ **do**
6 $CKs \leftarrow CKs \cup MineGraph(\mathcal{D}, \theta, \langle P^k, P^c, cond, G\rangle, CKs)$
7 $\mathcal{G} \leftarrow newConditions(size, \mathcal{G}, \theta, \mathcal{D})$

Mining a conditional key graph. Let us now discuss how one conditional key graph can be mined for keys (Line 7 in Algorithm 2). This task is done by Algorithm 3. This algorithm takes as input a dataset, the support threshold, a conditional key graph, and the set of conditional keys found so far. The algorithm proceeds in levels, looking first at the nodes that contain one property, then two properties, etc. For each level, we consider every node *cand*. If the node is still

marked for exploration (Line 3), we construct a candidate conditional key, with the input conditions as condition part, and the properties in $cand.p$ as the key part (Line 4). We then verify if the candidate key (a) meets the definition of a conditional key and (b) is minimal with respect to the other keys that have already been mined (Lines 5-6). If that is the case, the conditional key is added to the ouput (Line 7). If the key is a minimal key, then any extension of the key with more properties in the key part must be non-minimal and can safely be abandoned. Likewise, if the support of the candidate key is below the given threshold, so are its refinements. In both cases, we can prune the node and all descendants (Lines 8-11).

Algorithm 3: MineGraph

Input: dataset $\mathcal{D}$, minimum support θ,
conditional key graph $\langle P^k, P^c, cond, G = (V, E)\rangle$,
set of conditional keys found so far CKs
Output: modified CKs

```
1  for level = 1 to max_{v∈V}|v.p| do
2      for cand ∈ V where |cand.p| = level do
3          if cand.explore then
4              ck ← ⟨cond, cand.p⟩
5              isMinimal ← ck is a minimal key w.r.t CKs
6              if isMinimal ∧ support(ck, D) ≥ θ then
7                  CKs = CKs ∪ {ck}
8              if isMinimal ∨ support(ck, D) < θ then
9                  cand.explore ← false
10                 for child ∈ descendants(cand, G) do
11                     child.explore ← false
12 return CKs
```

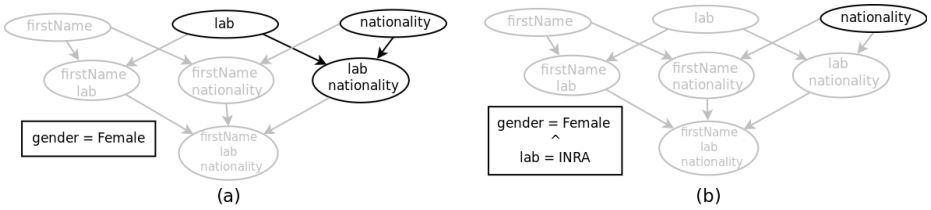


Fig. 2. (a) Keys of size 1 explored for the condition $gender = Female$. (b) Example of a merged graph with condition $\{gender = Female, lab = INRA\}$

As an example, let us consider again the data from Table 1, with the condition *gender=Female* and the maximal non-key {*firstName*, *gender*, *lab*, *nationality*}. The corresponding conditional key graph after scanning the first level is shown in Figure 2(a). Nodes with the *explore* flag set to false are greyed out. At the end of this step, only the property *firstName* is discovered as a key, since first

names are unique among female researchers. It follows that nodes containing this property in the next levels of the graph define non-minimal keys. They are therefore discarded for further exploration (the *explore* flag is set to false). The search for conditional keys is then applied to the nodes on levels 2 and 3, for which the *explore* flag is still true.

Lemma 2. ***(Graph Mining)*** *Given a conditional key graph $\langle P^k, P^c, cond, G\rangle$ for a dataset $\mathcal{D}$ and a threshold θ, Algorithm 3 will ensure that the result set CKs contains all minimal conditional keys for the condition set cond whose key properties are given by one of the nodes in G, and whose support is at least θ.*

This lemma holds because Algorithm 3 traverses all nodes in the conditional key graph, and checks each of them for being a conditional key. It excludes only (a) those nodes whose ancestors already had a support smaller than θ, in which case the node itself must also have a support smaller than θ, and (b) the nodes that lead to non-minimal keys.

Merging conditions. Let us now look at the process of generating more complex conditions (Line 8 in Algorithm 2). This work is done by Algorithm 4. It takes as input a set of conditional key graphs, a support threshold, a dataset, and a size parameter. It looks at all conditional key graphs that have a condition set of the given size (Lines 2-3). For each of them, it constructs a clone (Line 4). It then adds one more condition to the condition set of the clone. This new condition consists of a property and a constant (Section 3). The property is taken from a node of size 1 having its *explore* flag still set to true (Lines 5-6). The constant is taken from the constants that appear with that property in the conditional key graphs of size 1 (Line 9). If the new combined condition has a support that is large enough (Line 10), the conditional key graph of the singleton condition is merged with the clone and added to the output set.

Algorithm 4: newConditions

Input: size of condition set *size*
set of conditional key graphs $\mathcal{G}$
support threshold θ, dataset $\mathcal{D}$
Output: modified set of conditional key graphs $\mathcal{G}$

```
1  G_1 ← {g ∈ G : |g.cond| = 1}
2  G_size ← {g ∈ G : |g.cond| = size}
3  for g ∈ G_size do
4      g ← clone(g)
5      for v ∈ g.V where |g.V.p| = 1 do
6          if v.explore then
7              v.explore ← false
8              for v' ∈ g.descendants(v) do v'.explore ← false
9              for g_1 ∈ G_1 where g_1.P^c = v.p do
10                 if support(g.cond ∧ g_1.cond, D) ≥ θ then G ← G ∪ merge(g, g_1)
11 return G
```

The *merge* operation between two conditional key graphs $\langle P_1^k,\ P_1^c,\ cond_1,\ (V_1, E_1)\rangle$ and $\langle P_2^k, P_2^c, cond_2, (V_2, E_2)\rangle$ with $P_1^c \cap P_2^c = \emptyset$, produces a new conditional graph $\langle P^k, P^c, cond, (V, E)\rangle$ with:

- $P^k = P_1^k \cap P_2^k$ and $P^c = P_1^c \cup P_2^c$.
- $cond = cond_1 \cup cond_2$
- $V = \{\langle v.p, v.explore\rangle : \exists v_1 \in V_1, v_2 \in V_2 : v_1.p = v_2.p = v.p \quad \wedge \quad v.explore = (v_1.explore \wedge v_2.explore)\}$
- $E = \{u, v \in V : u.p \subset v.p \wedge |u.p| = |v.p| - 1\}$

As an example, Figure 2(b) shows the conditional graph with the set of conditions $\{gender = Female, lab = INRA\}$ produced by Algorithm 4 from the conditional graphs with conditions $gender = Female$ and $lab = INRA$. This graph is a clone of the graph with the condition $gender = Female$. A node is marked to be explored only if it was marked to be explored in both of the original graphs.

Lemma 3. ***(New Conditions)*** *Given a dataset $\mathcal{D}$, a set of conditional key graphs $\mathcal{G}$, a size parameter* size*, and a threshold θ, Algorithm 4 produces all conditional key graphs that contain condition sets of size* size+1*. Each of those graphs contains all the conditional keys for the given condition.*

We can prove Lemma 3 by induction. For $size = 0$, Lemma 1 guarantees that Algorithm 4 starts with all conditional key graphs for conditions of size 1. For $size > 0$, we need to show that (a) Algorithm 4 generates all conditional key graphs of size $size + 1$ and (b) each of these graphs contains all minimal conditional keys for their condition.

We start by showing (b), that is, the merge operation between two conditional key graphs $G_1 = \langle P_1^k, P_1^c, cond_1, (V_1, E_1)\rangle$ and $G_2 = \langle P_2^k, P_2^c, cond_2, (V_2, E_2)\rangle$ does not skip any minimal conditional key for the new condition. There are only two ways a node can be excluded from exploration in the merge operation: (1) the node is explicitly marked for non-exploration and (2) the node does not occur in one of the conditional key graphs. Case (1) occurs when the corresponding nodes are below the support threshold θ or they define non-minimal keys. In Case (2), the claim follows from the fact that if a node v is not contained in one of the graphs (e.g., $v \notin V_1$), then $v.p \cup P_1^c$ must be a key, i.e., it is not contained in any maximal non-key. This rationale applies analogously if $v \notin V_2$.

To show (a) we need to prove that our conditions are complete and correct. To show completeness we observe that Algorithm 4 builds conditions with $|cond| = size + 1$ based on the complete set of conditions with $|cond| = size$ and $|cond| = 1$. From the monotonicity of support, it follows that all conditions with $|cond| = size + 1$ with support greater than θ can be computed from these sets. To show correctness we note that for each graph with condition $cond = \{p_1 = o_1, \ldots, p_{size} = o_{size}\}$ and key properties P^k, Algorithm 4 will merge the graph with all conditions of the form $p_{size} = o_{size}$ where $p_{size} \in P^k$ (conditions of size 1, Line 5). This will produce graphs with conditions of the form $\{p_1 = o_1, \ldots, p_{size-1} = o_{size-1}, p_{size} = o_{size}\}$ with key part $P^k \setminus \{p_{size}\}$ with support greater than θ. (a) follows from Observation 1, since we have just transferred a property from the key part to the condition part.

Theorem 1. ***(Conditional Key Discovery)*** *Given a dataset $\mathcal{D}$, a set of conditional key graphs $\mathcal{G}$, and a threshold θ, Algorithm 2 produces all conditional keys whose properties are a subset of the properties of any node in any graph in $\mathcal{G}$, whose conditions are built from conditions or properties in $\mathcal{G}$, and whose support is at least θ.*

This theorem follows from the fact that Algorithm 2 calls Algorithm 3 for all sizes between 1 and the maximal number of property combinations. Lemma 2 makes sure that all possible graphs are generated. Lemma 3 ensures that all possible combinations of conditions are treated.

Corollary 1. ***(Conditional Key Mining)*** *Our method for conditional key mining is complete and correct.*

The correctness follows from the fact that Algorithm 3 adds a new key if and only if it is a key (Line 5). The completeness follows from Observation 1 and Theorem 1.

4.4 Implementation

Our method, VICKEY, is implemented in Java 7. The conditional key graphs have large condition sets and large associated graphs. Therefore, we do not store the graphs in memory, but rather generate them on the fly when they are accessed [28]. Furthermore, we parallelize the algorithm: the set of input non-keys is split into batches containing up to 50 (potentially non-distinct) properties. The batches are then scheduled to threads in the system, each one running Algorithms 1 and 2. This may lead to mining the same non-key multiple times, and therefore we perform a de-duplication before reporting the final results.

5 Experiments

We evaluate VICKEY in two series of experiments. First, we show the ability of VICKEY to discover conditional keys in large datasets with millions of triples. We compare the runtime of VICKEY to a generic rule mining approach, AMIE [13]. Then, we evaluate the utility of conditional keys for the task of data linking. We compare the conditional keys mined by VICKEY to the classical keys mined by SAKey [27].

5.1 Runtime Experiments

Setting. To evaluate the performance of VICKEY and AMIE [13], we adapt AMIE to mine rules of the form: $P_c \wedge P_k \Rightarrow x = y$. Here, $P_c = \bigwedge_{1..n} pc_i(x, A_i) \wedge pc_i(y, A_i)$ corresponds to the condition part of a key expression, and $P_k = \bigwedge_{1..m} pk_i(x, u_i) \wedge pk_i(y, u_i)$ represents the key part. Both AMIE and VICKEY take as input a set of maximal non-keys. These non-keys are obtained from the input dataset using SAKey [27]. Like VICKEY, our adapted variant of

AMIE uses the non-keys to restrict the search space by pruning the combinations of properties that do not occur in the non-keys. Unlike VICKEY, AMIE searches exhaustively for all rules that define conditional keys in the input dataset, regardless of their minimality. AMIE therefore requires a post-processing phase where all non-minimal conditional keys are removed. Both AMIE and VICKEY are run with a coverage threshold of 1%. We set the confidence threshold of AMIE to 100%, so that VICKEY and the modified AMIE mine exactly the same set of conditional keys. As datasets, we have used nine classes from DBpedia [18], covering different domains such as people, organizations, and locations. All experiments are run on a server with an AMD Opteron 6376 Processor (2.40GHz), 8 cores, and 128GB of RAM under Ubuntu Server 16.04.

Table 2. VICKEY vs AMIE on DBpedia

Class	Triples	Inst.	#Pro	#NKs	VICKEY	AMIE	#CKs
Actor	57.2k	5.8k	71	137	4.52m	12.58h	311
Album	786.1k	85.3k	39	68	1.53h	3.90h	304
Book	258.4k	30.0k	51	95	11.84h	> 1d	419
Film	832.1k	82.6k	74	132	1.37h	3.64h	185
Mount.	127.8k	16.4k	58	47	2.86m	23.57m	257
Museum	12.9k	1.9k	65	17	1.46s	6.45s	58
Organiz.	1.82M	178.7k	553	3221	26.32h	> 36h	28
Scientist	258.5k	19.7k	73	309	27.67m	> 1d	582
Univ.	85.8k	8.7k	89	140	14.45h	> 1d	941

Table 3. Linked classes stats

Class	#Pro	#Ks	#NKs	#CKs
Actor	16	93	22	748
Album	5	1	2	5864
Book	7	5	2	538
Film	9	14	13	26750
Mount.	5	3	2	775
Museum	7	14	5	80
Organiz.	17	149	3	9737
Scientist	10	22	8	407
Univ.	9	5	5	449

Results. Our results are shown in Table 2. The first three columns show some statistics about the testing datasets, followed by the number of discovered non-keys (NKs), the runtimes of both VICKEY and AMIE and finally the number of obtained conditional keys (CKs). We observe that a generic rule mining solution cannot handle some of the input datasets in less than 1 day. VICKEY, in contrast, runs on the smaller datasets *Actor*, *Mountain*, *Museum* and *Scientist* in less than 1 hour. This is because VICKEY's strategy prunes the search space much more effectively by avoiding candidate CKs that are not minimal. Other classes, such as *University* and *Organization*, are more challenging because they have many long non-keys (up to 15 properties). The longer the non-keys, the larger the number of property combinations in the search space. For example, for the class *Album*, AMIE explores more than 12.3k rules (including intermediate rules), where 6.4k rules correspond to potential conditional keys. In contrast, VICKEY explores only 4.1k candidates. This shows that VICKEY's strategy indeed prunes the search space much more effectively. It can mine conditional keys on hundreds of thousands of facts in a matter of minutes.

5.2 Extrinsic Evaluation

Setting. One of the primary application areas of keys is the discovery of equivalent entities across two KBs: If some combination of properties is a key, and if an entity in one KB shares values of these properties with an entity in the other KB, then the two entities must be the same. In this section we investigate the performance of conditional keys with respect to classical keys for this task. We emphasize that entity linking is not the primary goal of this paper. Instead, we

want to show the potential of conditional keys, introduced in this paper, over classical keys introduced by other approaches such as SAKey [27]. Entity linking is only an example setting to this end.

As KBs, we chose DBpedia [18] and YAGO [26], because there is a gold standard available for the entity links on the YAGO Web page. We have used the same set of classes as for the runtime experiments. As this type of entity linking assumes that the properties have been aligned, we mapped the properties of these classes manually, and rewrote the properties of YAGO using its DBpedia counterparts. We ran SAKey [27] and VICKEY on DBpedia to find standard and conditional keys, respectively. Table 3 shows the number of common properties, the number of keys (Ks), non-keys (NKs) and conditional keys (CKs) in each DBpedia class. Among others, VICKEY finds that *motto* is a key for universities in Italy and some other countries – but not in all countries; and that the name is a key for organizations in certain places – but not all places. To link the datasets, we use a simple algorithm [27]: For each key, we iterate over the entities in DBpedia that have the key properties. If there is an entity in YAGO that shares at least one value for every of these properties, we link the two. For conditional keys, we also check whether the conditions of the key are fulfilled in both datasets.

Table 4. Linking results with classical keys (Ks), conditional keys (CKs), and both.

Class		Recall	Precision	F1	
Actor	Ks [27]	0.27	0.99	**0.43**	
	CKs	0.57	0.99	0.73	× 1.75
	Ks+CKs	0.60	0.99	**0.75**	
Album	Ks [27]	0.00	1	**0.00**	
	CKs	0.15	0.99	0.26	× 869
	Ks+CKs	0.15	0.99	**0.26**	
Book	Ks [27]	0.03	1	**0.06**	
	CKs	0.11	0.99	0.20	× 3.48
	Ks+CKs	0.13	0.99	**0.23**	
Film	Ks [27]	0.04	0.99	**0.08**	
	CKs	0.38	0.96	0.54	× 7.1
	Ks+CKs	0.39	0.98	**0.55**	
Mountain	Ks [27]	0.00	1	**0.00**	
	CKs	0.28	0.99	0.44	× 101
	Ks+CKs	0.29	0.99	**0.45**	
Museum	Ks [27]	0.12	1	**0.21**	
	CKs	0.25	1	0.40	× 2.19
	Ks+CKs	0.31	1	**0.47**	
Organization	Ks [27]	0.01	1	**0.02**	
	CKs	0.14	0.98	0.24	× 11
	Ks+CKs	0.14	0.99	**0.24**	
Scientist	Ks [27]	0.05	0.98	**0.11**	
	CKs	0.16	0.99	0.28	× 2.96
	Ks+CKs	0.19	0.99	**0.32**	
University	Ks [27]	0.09	0.99	**0.16**	
	CKs	0.22	0.99	0.36	× 2.44
	Ks+CKs	0.25	0.99	**0.40**	

Results. Table 4 shows the precision, recall and F1 measure of the entity linking task using a) classical keys mined by SAKey [27], b) conditional keys alone and c) both types of keys (VICKEY). We first observe that the precision is always over 98%. Conversely, the recall is low in some cases. This happens mainly due to our simple linking method, which uses a strict string equality when comparing the values of properties, and also due to the incompleteness of the data in

both YAGO and DBpedia. However, even with this simple method, the use of conditional keys can lead to a significant increase in recall – with a negligible impact on precision. For example, for the class *Film*, recall increases from 4% to 38% when conditional keys are considered. Furthermore, when combining classic keys and conditional keys, the recall improves further. Overall, we observe an average increase of 21 percentage points in recall, and of 29 points in F1 when both standard keys and conditional keys are used to link the data. The average drop in precision is only 0.5 percentage points. This shows that conditional keys can significantly increase the performance of entity linking.

6 Conclusion

We have presented VICKEY, an approach to mine conditional keys on knowledge bases. Our approach overcomes the complexity of the search space by restricting it to the non-keys found by SAKey [27], and by pruning it smartly. This allows VICKEY to mine minimal conditional keys in datasets of up to 1.8M triples. In an extrinsic evaluation, we have shown that conditional keys can increase the recall of entity linking by up to 34 percentage points. As future work we plan to extend VICKEY by exploiting ontological classes and axioms, to discover more expressive conditional keys. The VICKEY system, as well as the datasets and evaluations, are available at `https://github.com/lgalarra/vickey`.

Acknowledgments. This research was supported by the grants ANR-11-LABEX-0045-DIGICOSME and ANR-16-CE23-0007-01 ("DICOS"), by the Chair "Machine Learning for Big Data" of Télécom ParisTech, and by the AG-INFRA+ project (Grant Agreement no. 731001).

References

1. Mustafa Al-Bakri, Manuel Atencia, Jérôme David, Steffen Lalande, and Marie-Christine Rousset. Uncertainty-sensitive reasoning for inferring sameas facts in linked data. In *ECAI*, 2016.
2. Manuel Atencia, Michel Chein, Madalina Croitoru, Jérôme David, Michel Leclère, Nathalie Pernelle, Fatiha Saïs, François Scharffe, and Danai Symeonidou. Defining key semantics for the RDF datasets: Experiments and evaluations. In *ICCS*, 2014.
3. Manuel Atencia, Jérôme David, and Jérôme Euzenat. Data interlinking through robust linkkey extraction. In *ECAI*, Czech Republic, 2014.
4. Manuel Atencia, Jérôme David, and François Scharffe. Keys and pseudo-keys detection for web datasets cleansing and interlinking. In *EKAW*, 2012.
5. Yang Chen, Sean Louis Goldberg, Daisy Zhe Wang, and Soumitra Siddharth Johri. Ontological pathfinding. In *SIGMOD*, 2016.
6. Fei Chiang and Renée J. Miller. Discovering data quality rules. In *VLDB*, 2008.
7. X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.
8. Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. In *VLDB*, 2014.

9. Wenfei Fan, Zhe Fan, Chao Tian, and Xin Luna Dong. Keys for graphs. In *VLDB*, 2015.
10. Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. Discovering conditional functional dependencies. *IEEE Trans. on Knowl. and Data Eng.*, 2011.
11. Mohamed H Gad-Elrab, Daria Stepanova, Jacopo Urbani, and Gerhard Weikum. Exception-enriched rule learning from knowledge graphs. In *ISWC*, 2016.
12. Luis Galarraga, Simon Razniewski, Antoine Amarilli, and Fabian M. Suchanek. Predicting Completeness in Knowledge Bases. In *WSDM*, 2017.
13. Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, 2013.
14. Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.*, 24(6), 2015.
15. Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. On generating near-optimal tableaux for conditional functional dependencies. *VLDB*, 2008.
16. A. Heise, Jorge-Arnulfo, Quiane-Ruiz, Z. Abedjan, A. Jentzsch, and F. Naumann. Scalable discovery of unique column combinations. In *VLDB*, 2013.
17. Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *Computer Journal*, 42(2), 1999.
18. Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web J.*, 6(2), 2015.
19. Frank Manola and Eric Miller. RDF primer. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.
20. Peter Patel-Schneider, Bijan Parsia, Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. OWL 2 web ontology language primer. W3C recommendation, W3C, October 2009. http://www.w3.org/TR/2009/REC-owl2-primer-20091027/.
21. Nathalie Pernelle, Fatiha Saïs, and Danai Symeonidou. An automatic key discovery approach for data linking. *J. of Web Semantics*, 23, 2013.
22. Nicoleta Preda, Gjergji Kasneci, Fabian M. Suchanek, Thomas Neumann, Wenjun Yuan, and Gerhard Weikum. Active knowledge: dynamically enriching RDF knowledge bases by web services. In *SIGMOD*, 2010.
23. Fatiha Saïs, Nathalie Pernelle, and Marie-Christine Rousset. Combining a logical and a numerical method for data reconciliation. *J. Data Semantics*, 12, 2009.
24. Yannis Sismanis, Paul Brown, Peter J. Haas, and Berthold Reinwald. Gordian: efficient and scalable discovery of composite keys. In *VLDB*, 2006.
25. Tommaso Soru, Edgard Marx, and Axel-Cyrille Ngonga Ngomo. ROCKER: A refinement operator for key discovery. In *WWW*, 2015.
26. Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.
27. Danai Symeonidou, Vincent Armant, Nathalie Pernelle, and Fatiha Saïs. SAKey: Scalable Almost Key discovery in RDF data. In *ISWC*, 2014.
28. Danai Symeonidou, Luis Galarrága, Nathalie Pernelle, Fatiha Saïs, and Fabian Suchanek. VICKEY: Mining Conditional Keys on RDF datasets. Technical report, `https://doi.org/10.5281/zenodo.835647`, 2017.
29. Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Comm. of the ACM*, 57(10), 2014.