Mining Cardinalities from Knowledge Bases

Emir Muñoz[1,2(✉)] and Matthias Nickles[2]

[1] Fujitsu Ireland Limited
[2] Insight Centre for Data Analytics, National University of Ireland, Galway
Emir.Munoz@ie.fujitsu.com

Abstract. Cardinality is an important structural aspect of data that has not received enough attention in the context of RDF knowledge bases (KBs). Information about cardinalities can be useful for data users and knowledge engineers when writing queries, reusing or engineering KBs. Such cardinalities can be declared using OWL and RDF constraint languages as constraints on the usage of properties over instance data. However, their declaration is optional and consistency with the instance data is not ensured. In this paper, we address the problem of mining cardinality bounds for properties to discover structural characteristics of KBs, and use these bounds to assess completeness. Because KBs are incomplete and error-prone, we apply statistical methods for filtering property usage and for finding accurate and robust patterns. Accuracy of the cardinality patterns is ensured by properly handling equality axioms (owl:sameAs); and robustness by filtering outliers. We report an implementation of our algorithm with two variants using SPARQL 1.1 and Apache Spark, and their evaluation on real-world and synthetic data.

1 Introduction

The Resource Description Framework (RDF) is a widely used framework for representing knowledge (bases) on the Web. RDF is schema-less, which means that it gives freedom to data publishers in describing entities and their relationships using facts without the need to observe some specified unique global data schema. Most RDF knowledge bases (KBs) avoid to include domain, range or cardinality restrictions because of the contradictions that they can generate [6,21]. For instance, having two different properties (e.g., from different ontologies) to represent the population of a country, or using two or more labels to refer to the same entity. However, the lack of a central schema causes a series of difficulties in the consumption of such data (e.g., [9,11,1,14]), e.g., having two different population numbers in the same KB. For instance, data users and knowledge engineers need an understanding of what information is available in order to write queries, and to reuse or engineer KBs [15,26]. In data management, *cardinality* is an important aspect of the structure of data. We show that the aforementioned problem can be overcome partially by mining cardinalities from instance data, and complement other methods (e.g., [30]) towards the generation of a central schema. In RDF, the cardinality of a property limits the number of values that may have for a given entity, and can be declared using the Web Ontology Language (OWL) or

RDF constraint languages; however, such declarations are hand-crafted and application dependent. Therefore, ontologies rarely include cardinality declarations in practice, highlighting the need for cardinality mining methods.

In this work, our main goal is to discover such structural patterns using a bottom-up or extensional approach for mining cardinality bounds from instance data. Cardinality bounds are hidden data patterns that unveil the structure of data, and in some cases they might not even be intuitive for data creators [14]. Our approach produces *accurate* cardinalities by taking into consideration the semantics of `owl:sameAs`[1] equality axioms in KBs; and *robust* ones by not assuming completely correct data, i.e., instance data can have errors. By doing so, the output of this work can serve users to analyse completeness and consistency of data, and thus contribute towards higher levels of quality in KBs [7,26,16].

Recently, RDF constraint languages (e.g., Shape Expressions [19], Shapes Constraint Language (SHACL)[2]) have been defined to satisfy the latent requirements for constraints definition and validation of RDF in real-world use cases. They build upon SPARQL or regular expressions to define so-called *shapes* that perform for RDF the same function as XML Schema, DTD and JSON Schema perform for XML or JSON: delimit the boundaries of instance data. Here, we consider that properties in KBs are constrained by multiplicities, which cover a critical aspect of relational data modelling, referring to the number of values that a property can have. Specifically, we consider a set of cardinalities or multiplicities as part of the internal structure of an entity type in a KB. In databases, internal structure-based methods are also referred to as constraint-based approaches, and have been used for schema matching [20]. Unlike databases, RDF and OWL assume the open-world semantics, and absence of the unique name assumption (nUNA). This makes the problem of extracting cardinalities more complex than a simple application of SPARQL queries using the `COUNT` operator. Take as example the constraint "a person must have two parents": if the data contain an entity of type `person` with only one parent, *this does not cause a logical inconsistency, it just means it is incomplete, and in RDF/OWL incomplete is different from inconsistent.* To deal with these specificities, we propose a method which tackles two important challenges: (1) *KB Normalisation*, where we must deal with `owl:sameAs` (or alike) axioms representing equality between entities, and (2) *Outliers Filtering*, where we account for the probability of noise in the data, in order to extract robust cardinality patterns.

This paper is organised as follows: In Section 2 we review the related work about cardinality, consistency, and schema discovery in KBs. Section 3 introduces some preliminaries about the RDF model. Section 4 provides a definition and semantics for cardinality bounds in KBs considering existing languages. In Section 5 we define an algorithm for mining cardinality bounds in an accurate and robust manner, and propose one implementation with two normalisation variants. Finally, we evaluate our algorithm over different datasets in Section 6, and present our conclusions and outlook in Section 7.

[1] Henceforth, we use prefixes for namespaces according to `http://prefix.cc/`

[2] `https://www.w3.org/TR/shacl/` (accessed on February 13, 2017)

2 Related Work

Cardinality constraints/bounds. Cardinality constraints in RDF have been defined for data validation in languages such as OWL [13], Shape Expressions (ShEx) [19], OSLC Resource Shapes [24], and Dublin Core Description Set Profiles (DSP)[3]. OSLC integrity constraints include cardinality of relations which are more similar to UML cardinality for associations (i.e., exactly-one, one-or-many, zero-or-many, and zero-or-one). However, the expressivity of OSLC is limited compared to the definitions proposed in OWL[4], DSP, Shapes Constraint Language (SHACL), and Stardog ICV[5]. All of them define flexible boundaries for cardinality constraints: a lower bound in $\mathbb{N}$, and an upper bound in $\mathbb{N} \cup \{\infty\}$. SPIN[6] Modelling Vocabulary is yet another language that based on SPARQL to specify rules and logical constraints, including cardinality. It is worth pointing out that due to the bottom-up approach taken here we do not refer to our cardinalities as constraints but as *bounds*. They can be considered as constraints only after a user assessment and application over a given dataset. Despite this, our work builds upon existing approaches for cardinality constraints in RDF and other data models such as XML [3] and Entity-Relationship [28,10].

Consistency in RDF graphs. Consistency is a relevant dimension of data quality, and many researchers have investigated the checking and handling of inconsistencies in RDF. However, to the best of our knowledge, this is the first work focused on the extraction and study of cardinalities to detect inconsistencies in RDF, through the application of outlier detection techniques. The concept of outliers or anomaly detection is defined as "finding patterns in data that do not conform to the expected normal behaviour" (see [2] for a survey). Under the assumption that KBs are likely to be noisy and incomplete [18], there exist several approaches that aim to enhance or refine their quality and completeness (see Paulheim [16] for a recent survey). Among the most relevant to our work are: [31] which applies unsupervised numerical outlier detection methods to DBpedia for detecting wrong values that are used as literal objects of a property; and [4] that builds upon [31] by identifying sub-populations of instances where the outlier detection works more accurately, and by using external datasets accessible from the `owl:sameAs` links. Our work differs from theirs in that they focus on missing property values, not on their cardinality or multiplicity.

RDF schema discovery. Our goal falls into the broader area of schema discovery. Völker and Niepert in [30] introduce a statistical approach where association rule mining is used to generate OWL ontologies from RDF data, but consider cardinality restrictions (upper bounds) only as future work. Similarly, in [8] the authors extract type definitions described by profiles which consist of a property vector, where each property is associated to a probability. A further analysis of

[3] `http://dublincore.org/documents/dc-dsp/`

[4] OWL allows the expression of cardinalities through the `minCardinality`, `maxCardinality`, and `cardinality` restrictions.

[5] `http://docs.stardog.com/icv/icv-specification.html`

[6] `http://spinrdf.org/`

semantic and hierarchical links between types is performed to extract a global schema. Since most KBs are generated from semi- or un-structured data, [29] analysed the case of DBpedia enrichment with axioms identified during the extraction process from Wikipedia. Such axioms are identified with methods from Inductive Logic Programming (ILP), like in [30]. Despite their bottom-up or extensional approach, similar to ours, such works aim to build or enrich ontologies with missing relations, not considering any notion of cardinality nor their use to analyse completeness and/or consistency. A related approach to detect cardinality in KBs is presented by Rivero et al. in [21], which uses SPARQL 1.1 queries to automatically discover ontological models. Such models include types and properties, subtypes, domain, range, and minimum cardinalities of these properties. However, the approach presented in [21] is not able to deal with the semantics of data: both the existence of `owl:sameAs` axioms and outliers or errors in the data are ignored. For these reasons, our work is orthogonal and complementary to all aforementioned works.

3 Preliminaries

Below we first provide some preliminaries regarding RDF and its semantics.

RDF Model. Let $\mathcal{R}$ be the set of *entities*, $\mathcal{B}$ the set of *blank nodes*, $\mathcal{P}$ the set of *predicates*, and $\mathcal{L}$ the set of *literals*. A finite knowledge base $\mathcal{G}$ is a set of *triples* $t := (s, p, o) \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{P} \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L})$, where s is the *subject*, p is the *predicate*, and o is the *object* of t. We define the functions $\textsc{Pred}_{\mathcal{G}}(\tau) = \{p \mid \exists s, o\ (s, p, o) \in \mathcal{G} \wedge (s,$ `rdf:type` $, \tau) \in \mathcal{G}\}$ that returns a set of predicates appearing with instances of entity type τ; and $\textsc{Triples}_{\mathcal{G}}(p) = \{(s, p, o) \mid \exists s, o\ (s, p, o) \in \mathcal{G}\}$ that returns the triples in $\mathcal{G}$ with property p.

UNA 2.0. The *unique name assumption* (UNA) is a simplifying assumption made in some ontology languages and description logics. It means that two different names always refer to different entities in the world [23]. On the one hand, OWL default semantics does not adopt the UNA, thus two different constants can refer to the same individual—a desirable behaviour in an environment such as the Web. On the other hand, validation checking approaches in RDF usually adopt a *closed world assumption* (CWA) with UNA, i.e., inferring a statement to be false on the basis of failure to prove it, and if two entities are named differently, they are assumed to be different entities. To deal with this, SHACL defines the so-called *UNA 2.0* which is a simple workaround where all entities are treated as different, unless explicitly stated otherwise by `owl:sameAs`. From a practical point of view, it is a desirable feature for mining algorithms to consider the semantics of RDF graphs avoiding misinterpretations of the data. Figure 1 (left) shows an example where the adoption of normal UNA will lead to a count of five different entities (i.e., `ex:A`, `ex:B`, `ex:C`, `ex:D`, `ex:E`), and for `ex:A` the property `ex:p1` would have cardinality 1. While adopting UNA 2.0 (Figure 1, right) changes the counts: now we have four different entities (i.e., `ex:A`, `ex:B`, `ex:C`, `ex:E`), and the cardinality of `ex:p1` in `ex:A` is 2. Here, we call *rewriting* the process of applying UNA 2.0 to an unnormalised KB.

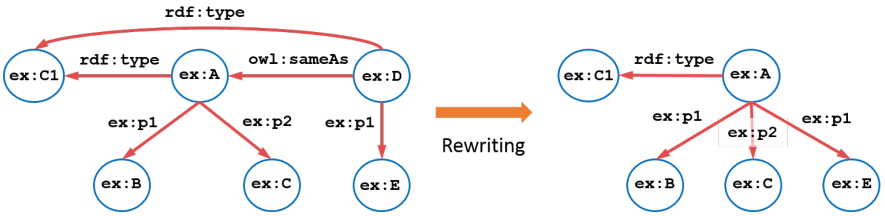

Fig. 1: Example of UNA 2.0 as defined in SHACL.

Since the cardinality of a property is severely affected when `owl:sameAs` axioms are not considered, hereafter, we adopt UNA 2.0 and satisfy this requirement, allowing us to correctly interpret the KB semantics.

4 Cardinality Bounds in RDF

In this section we introduce a definition of cardinality bounds in KBs that generalises the semantics of the definitions discussed in Section 3.

Cardinality (also known as multiplicity) covers a critical aspect of relational data modelling, referring to the number of times an instance of one entity can be related with instances of another entity. A *cardinality bound* is a restriction on the number of elements in the relation. In particular, in KBs we have relations between entities of a given type through properties, and we want to specify bounds for such relationships. For example, we would like to express that a drug has only one molecular formula, but can be associated to a finite set (of known or unknown size, the latter denoted as ∞) of adverse drug reactions.

Definition 1 *A* cardinality bound *in RDF knowledge bases restricts the number of property values related with an entity in a given context (i.e., a particular type, or the whole KB). Formally, a cardinality bound φ is an expression of the form $card(P, \tau) = (min, max)$ where $P \subseteq \mathcal{P}$, τ is an entity type, and where $min \in \mathbb{N}$ and $max \in \mathbb{N} \cup \{\infty\}$ with $min \leq max$. Here $|P|$ denotes the number of properties in φ, min is called the* lower bound*, and max the* upper bound *of φ. If τ is defined ($\tau \neq \varepsilon$), we say that φ is* qualified*; otherwise we say that φ is* unqualified*.*

The semantics of this definition of cardinality bounds limits the maximum and minimum count that a given set of properties can have in a given context as in SHACL, DSP, ICV and OWL. The lower bound of a cardinality may take on values in $\mathbb{N}$, whilst upper bounds can be ∞ to represent that there is an unknown upper limit. An unqualified bound is independent of a type (context), i.e., it holds for a set of properties independently of its context, whereas a qualified bound holds only for a set of properties in combination with subject entities of a given type. Herein, we focus on qualified constraints given their interestingness and relevance for structural analyses of KBs.

Definition 2 *Consider a KB $\mathcal{G}$. We say that φ is a cardinality bound in $\mathcal{G}$ for a set of properties $P_\varphi \subseteq \mathcal{P}$, a lower bound min_φ, and upper bound max_φ, if*

$$\forall s \in (\mathcal{R} \cup \mathcal{B})\ (min_\varphi \leq |\{p : p \in P_\varphi \wedge \exists o\ (s, p, o) \in \mathcal{G}\}| \leq max_\varphi).$$

```
SELECT $this
WHERE {
        $this $PROPERTY ?value .
    }
} GROUP BY $this
HAVING (COUNT(?value) < $minCount)
```

Fig. 2: SPARQL 1.1 definition of a minimum cardinality constraint.

If φ is qualified to τ then to satisfy φ, $\mathcal{G}$ also needs to satisfy the condition that $\forall s \in (\mathcal{R} \cup \mathcal{B})\ (s, \texttt{rdf:type}, \tau) \in \mathcal{G}$.

Although our approach is able to compute an upper bound cardinality, this limit is uncertain when considering RDF's open world assumption (OWA). For instance, even when the data show that an entity person has maximum two children, this might be wrong when considering other entities. More certain cardinality bounds can be mined from reliable or complete graphs usually existent within specific domains. Therefore, we refer to cardinality bounds as "patterns" when they are automatically extracted from raw KBs, and as "constraints" when normatively assessed by a user and applied in order to restrict a KB.

In practice, cardinality bounds can be used to validate KBs using SPARQL 1.1 queries. For instance, Fig. 2 shows the SPARQL query proposed to validate a lower bound min_φ (`$minCount`). The query represents restrictions on the number of values, `?value`, that the `$this` node may have for the given property. A validation result must be produced if the number of value nodes is less than `$minCount`. Similarly, to validate an upper bound (max_φ) restriction for a property, we can change the `HAVING` condition to '`>`'. Note that SHACL, ShEx, and other constraint languages only allow the definition of one condition at a time per property. Therefore, to validate our cardinalities with multiple properties, one must apply an SPARQL 1.1 query like the one in Fig. 2 independently for each property and bound. In Section 5 we will show how a single SPARQL 1.1 query can be used to extract both minimum and maximum bounds at once.

Example 1 *The following expressions define cardinality bounds for different entity types in different domains.*

1. $card(\{\texttt{mondial:name}, \texttt{mondial:elevation}\}, \texttt{mondial:Volcano}) = (1, 1)$,
2. $card(\{\texttt{mondial:hasCity}\}, \texttt{mondial:Country}) = (1, \infty)$,
3. $card(\{\texttt{dcterms:contributor}\}, \texttt{bibo:Book}) = (0, \infty)$,
4. $card(\{\texttt{dcterms:language}\}, \texttt{bibo:Book}) = (1, 2)$.

As suggested in the previous example, when the upper bound is unclear we use ∞ in the cardinality bound to express that uncertainty.

5 Mining Cardinality Patterns

In the following, we introduce our main algorithm for mining cardinality patterns from KBs. We also present two different implementations: one based on SPARQL 1.1 that uses a graph databases approach to normalise and extract cardinalities; and another based on Apache Spark that applies a MapReduce or divide-and-conquer strategy to divide the data and run the steps in parallel.

Algorithm 1 CardBounds: Extraction of cardinality bounds.

Input: a knowledge base $\mathcal{G}$; and a context τ
Output: a set Σ of cardinality bounds

1: $\mathcal{G}' \leftarrow \text{Normalise}(\mathcal{G}, \tau)$
2: $P \leftarrow \text{Pred}_{\mathcal{G}'}(\tau)$ ◁ *Retrieve all predicates for entities of type* τ
3: **for all** $p \in P$ **do**
4: $\quad \mathcal{D} \leftarrow \text{Triples}_{\mathcal{G}'}(p)$ ◁ *Retrieve all triples with property* p
5: $\quad \mathcal{M}\,\langle u, v\rangle \leftarrow \text{CardPatterns}(\mathcal{D})$ ◁ u *is an entity, and* v *a cardinality*
6: $\quad inliers \leftarrow \text{FilterOutliers}(\mathcal{M}.v)$
7: $\quad \Sigma.add(card(\{p\}, \tau) = (\texttt{MIN}(inliers), \texttt{MAX}(inliers)))$
8: **end for**

5.1 Algorithm

We present Algorithm 1 as an efficient solution to mine accurate and robust cardinality patterns from any KB. This algorithm is designed to mine qualified cardinalities, i.e., there is a context type; however, it can be easily extended to mine unqualified cardinalities. From a data quality perspective, it is desirable that the mined cardinalities bounds (see Section 4) are accurate and robust. Algorithm 1 outputs a set of cardinality patterns, which are called "accurate" because we consider the semantics of `owl:sameAs` axioms, and "robust" because we perform an outliers detection and filtering over noisy cardinality counts.

Our mining algorithm has three main parts: (1) KB normalisation: represented by the Normalise(·) function, receives an (unnormalised or with multiple equal entities) KB as input and applies an on-the-fly rewriting process to consider the semantics of `owl:sameAs` (or other alike relation), where one can build cliques from grouping equal entities. This function could be considered optional though in cases where users want information about unnormalised bounds—at the cost of accuracy. (2) Cardinalities extraction: performed by the function CardPatterns(·), it is called to retrieve cardinality pairs (entity, cardinality) from the data for a given property. The cardinalities for a fixed property are stored in a map which is after used to identify noisy values. (3) Outliers filtering: represented by the FilterOutliers(·) function, receives a map of (entity, cardinality) pairs and applies unsupervised univariate statistical methods to identify and remove noisy or outside of a range values to ensure robustness.

Next, we present an example for the application of Algorithm 1, and describe each of its part in more details in Sections 5.2 to 5.4.

Example 2 *Let us consider a KB with entities* `ex:s1` *and* `ex:s2`*, and properties* `ex:p1` *and* `ex:p2`*. The* Normalise *function takes all triples in the KB and replaces duplicates by one representative element equivalence type induced by* `owl:sameAs`*-cliques. Then, for each property it extracts the cardinality values using the function* CardPatterns *to get values:* $[1, 1, 1]$ *for* `ex:p1`*, and* $[3, 3, 25]$ *for* `ex:p2`*. Next, the function* FilterOutliers *determines that there are no outliers for property* `ex:p1`*, but that a cardinality of 25 is an outlier for* `ex:p2`*. Thus, 25 is removed from the patterns leaving* $[3, 3]$ *as robust cardinalities for* `ex:p2`*. Finally, the cardinality bounds (min, max) are extracted from the remaining inlier cardinalities by using simple* `MIN` *and* `MAX` *functions.*

Table 1: An axiomatisation for reduction on equality

$\frac{(s,p,o) \wedge (s',p',o') \wedge (s', \texttt{owl:sameAs}, s)}{(s,p,o),(s,p',o')}$ (subject-equality)	$\frac{(s,p,o) \wedge (s',p',o') \wedge (o', \texttt{owl:sameAs}, o)}{(s,p,o),(s',p',o)}$ (object-equality)

5.2 Knowledge Bases Equality Normalisation

Knowledge bases contain different types of axioms, being `owl:sameAs` and alike the most important when computing cardinalities. Regardless of the approach, by not considering these axioms a method loses its accuracy and cannot ensure that the cardinality bounds are consistent with the data. Unlike [21], we perform an on-the-fly normalisation of the graph in order to capture the semantics of `owl:sameAs` axioms without having to modify the underlying data. A simple SPARQL query using the `COUNT` operator will return two instances of `ex:C1` instead of the expected count 1 for the example in Figure 1 (left). To overcome this issue, we propose an axiomatisation with two rules, namely, *subject-equality* and *object-equality* (Table 1), where duplicated elements are replaced by one representative element equivalence type induced by `owl:sameAs`. This normalisation can be done replacing the underlying data [12] or on-the-fly (without modification) when needed. However, if the underlying data is modified, the links to other KBs stated by the `owl:sameAs` axioms are overwritten and lost. Instead, here we follow an on-the-fly overwrite (line 1 in Algorithm 1) which performs the modifications in memory. This was also used by Schenner et al. in [25]. The representatives are selected from the so-called `owl:sameAs`-*cliques*, which are sets of entities all of which are equal to each other [12]. In practice, the axiomatisation of Table 1 can be implemented on-the-fly either by using SPARQL 1.1 or programmatically. We briefly introduce these two options as follows:

SPARQL rewrite. We make use of the nested SPARQL 1.1 query in Figure 4, which contains three sub-queries and aims to obtain (entity, cardinality) pairs for a given property and entity type (line 3). Embedded in `SQ-1` are `SQ-2` and `SQ-3` performing the clique generations for subject and object, respectively. Sub-query SQ-2 applies the subject-equality rule, and sub-query SQ-3 applies the object-equality rule. In a clique generation, a graph search is done in all directions of the graph to find equal entities, which incurs in a high complexity. For each clique found, a representative is selected, thus omitting all "clone" entities. Intuitively, the query used here can be seen as complex and resource demanding. Hence, we also propose a faster solution that works outside of a SPARQL endpoint.

Programmatic rewrite. We can also frame the extraction of cardinality patterns as the well-known words count problem. So, we can easily parallelise the algorithm using frameworks such as Apache Spark[7]. By using Spark and the `filer` and `map` operations, we implemented a parallel rewrite, where the `owl:sameAs`-cliques are generated and used to normalise the KB triple by triple (see Figure 3, left). We generate the `owl:sameAs`-cliques as follows: for each (s, `owl:sameAs`, o)

[7] `http://spark.apache.org/` (version 2.1.0)

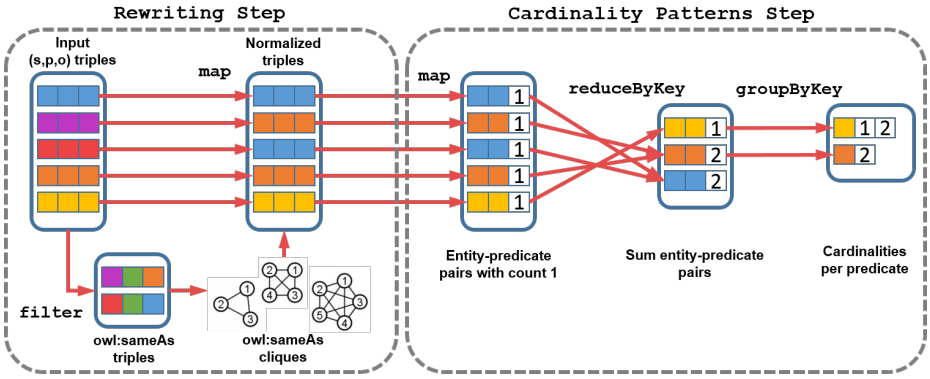

Fig. 3: Cardinality patterns extraction using Apache Spark.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT ?first_subj (COUNT(DISTINCT ?first_obj) AS ?nbValues) WHERE {
    { SELECT DISTINCT ?first_subj ?first_obj WHERE {                    % (SQ-1)
        ?subj $property ?obj .
        { SELECT ?subj ?first_subj WHERE {                              % (SQ-2)
            ?subj a $type .
            ?subj ((owl:sameAs|^owl:sameAs)*) ?first_subj .
                ?notfirst ((owl:sameAs|^owl:sameAs)*) ?first_subj .
                FILTER (STR(?notfirst) < STR(?first_subj))
            } FILTER(!BOUND(?notfirst))
        }}
        { SELECT ?obj ?first_obj WHERE {                                % (SQ-3)
            ?obj ((owl:sameAs|^owl:sameAs)*) ?first_obj .
                ?notfirst ((owl:sameAs|^owl:sameAs)*) ?first_obj .
                FILTER (STR(?notfirst) < STR(?first_obj))
            } FILTER(!BOUND(?notfirst))
        }}
    }}
} GROUP BY ?first_subj
```

Fig. 4: Query the cardinality of a property for every entity of a given type.

triple we lexically compare s and o and select the minimum (e.g., s), which becomes the *representative*; add a mapping from the minimum to the other (e.g., from s to o); if the no-minimum (e.g., o) was the representative of other entities, then we update their mappings in cascade with the new representative (e.g., s). We then apply a `map` operation over each initial triple and overwrite it according to the `owl:sameAs`-cliques to obtain $\mathcal{G}'$ in $O(1)$.

5.3 Detection of Cardinality Patterns

After the normalisation step, cardinalities can be collected for each property (line 5 in Algorithm 1) ensuring their accuracy, which is a major difference w.r.t. previous approaches such as Rivero et al. [21]. In the SPARQL-based approach, Figure 4 shows a query which performs both the normalisation of $\mathcal{G}$ and the detection of cardinality patterns in one place. Complex SPARQL queries are hard to evaluate and optimise [27], making this approach very inefficient and

poorly scalable. On the other hand, the Spark-based approach can make use of multiple machines to scale and process the KB in splits. We show a comparison of both approaches latter in Section 6. Regardless of the approach, the output of the cardinality extraction is a map of (entity, cardinality) pairs for a given property and type. These cardinalities already could be taken as cardinality patterns by users. However, several works have shown that KBs frequently contain noise and outliers (e.g., [7,17,16]). In order to address this, we carry out a filtering of outliers from the cardinalities, which is described in the next section.

5.4 Outlier Detection and Filtering

Considering the adverse effects that outliers could have in the method described so far, we now present techniques that can be used to detect and remove outliers (line 6 in Algorithm 1). Several supervised and unsupervised approaches can be used for the detection of outliers in numerical data (see [18] for details); however, we did not find any labelled dataset for valid cardinality values. Therefore, we only consider unsupervised approaches for univariate data. We address the detection of outliers in a sequence of numbers as a statistical problem. Interestingly, outlier detection approaches determine a lower and upper bound on the range of data, similarly to the semantics of a cardinality bound. The *extreme studentized deviation* (ESD) identifier [22] is one of the most popular approaches and computes the mean μ and standard deviation σ values and considers as outlier any value outside of the interval $[\mu - t \cdot \sigma, \mu + t \cdot \sigma]$, where $t = 3$ is usually used. The problem with ESD is that both the mean and the standard deviation are themselves sensitive to the presence of outliers in the data. *Hampel* identifier [18] appears as an option, where the mean is replaced by the median *med*, and the standard deviation by the median absolute deviation (MAD). The range for outliers is now: $[med - t \cdot MAD, med + t \cdot MAD]$. Since the median and MAD are more resistant to the influence of outliers than the mean and standard deviation, Hampel identifier is generally more effective than ESD. Although, Hampel sometimes could be considered too aggressive, declaring too many outliers [18]. Boxplot appears as a third option, and defines the range: $[Q1 - c \cdot IQD, Q3 + c \cdot IQD]$, where $Q1$ and $Q3$ are the lower and upper quartiles, respectively, and $IQD = Q3 - Q1$ is the interquartile distance—a measure similar to the standard deviation. The parameter c is similar to t in Hampel and ESD, and is commonly set to $c = 1.5$. Boxplot is better suited for distributions that are moderately asymmetric, because it does not depends on an estimated "centre" of the data. Thus, in our evaluation we use boxplot rule to determine cardinality outliers.

6 Evaluation

Next, we evaluate the application of our mining algorithm in its two variants against real-world and synthetic KBs. After, we use the mined cardinality bounds to analyse the notions of completeness and consistency of KBs.

Table 2: Datasets characteristics

Dataset	№ triples	№ types	№ prop.	№ sameAs
OpenCyc	2,413,894	7,613	165	360,014
UOBM	2,217,286	40	29	0
British National Library	210,820	24	45	14,761
Mondial	186,534	27	60	0
New York Times People	103,496	1	20	14,884

6.1 Settings

Datasets. We used five datasets with different number of triples and `owl:sameAs` axioms. The chosen datasets are diverse in domain, features, and represent real-world and synthetic data. We present their characteristics in Table 2, and describe them as follows:

- OpenCyc[8] is a large general KB released in 2012 that contains hundreds of thousands of terms in the domain of human knowledge covering places, organisations, business related things, people among others.
- UOBM[9] is a synthetic dataset that extends the Lehigh University Benchmark (LUMB), a university domain ontology, that contains information about faculties and students.
- British National Library[10] is a dataset published by the National Library of the UK (second largest library in the world) about books and serials.
- Mondial[11] is a database compiled from geographical Web data sources such as CIA World Factbook, and Wikipedia among others.
- New York Times People[12] is a compilation of the most authoritative people mentioned in news of the New York Times newspaper since 2009.

Test settings. We implemented our cardinality mining Algorithm 1 using Python 3.4 and Apache Spark 2.1.0. We use a Intel Core i7 4.0 GHz machine with 32 GB of RAM running Linux kernel 3.2 to run experiments on different KBs. Although Spark can run on multiple machines, we only tested it on a single machine using multiple parallel processes—one per core using 8 cores in total.

6.2 Results

First, we quantitatively compare the runtime of Algorithm 1 with both implementations, SPARQL and Spark, to normalise (rewrite) KBs, retrieve, and filter cardinality bounds. We then analyse qualitatively the use of the identified cardinality bounds to assess two crucial notions of KBs and databases: completeness and consistency.

[8] `http://www.cyc.com/platform/opencyc`
[9] `https://www.cs.ox.ac.uk/isg/tools/UOBMGenerator/`
[10] `http://www.bl.uk/bibliographic/download.html`
[11] `http://www.dbis.informatik.uni-goettingen.de/Mondial/`
[12] `https://datahub.io/dataset/nytimes-linked-open-data`

Quantitative Evaluation. Intuitively, based on the scalability of Spark, one can foresee that the parallelised variant of our algorithm (Figure 3) will outperform the other that uses SPARQL. To test this we ran both implementations on the British National Library (BNL) and Mondial datasets, where only the first contains `owl:sameAs` axioms. For BNL, we considered the type $\tau = \mathsf{Book}$ with 7 predicates and obtained average runtime of 253.908 sec for the SPARQL implementation, and 15.634 sec for the Spark one. This shows that the Spark implementation is 16x faster than using SPARQL, while performing the same task on BNL dataset. For Mondial, we considered the type $\tau = \mathsf{River}$ with 8 predicates and obtained average runtime of 117.739 sec for the SPARQL implementation, and 2.948 sec for the Spark one. This shows that the Spark implementation is 40x faster than using SPARQL, while performing the same task on Mondial dataset. It is worth pointing out that the times on Mondial dataset are lower than for BNL because of the lower number of instances and the absence of `owl:sameAs` axioms. Finally, our experiments show that the outlier detection method (i.e., boxplot) does not add a significant overhead to the whole process and scales well for different data sizes.

Qualitative Evaluation. The characteristics of the datasets range between 1 up to 7,613 types and 20 up to 165 properties. To keep our study manageable, we selected randomly one entity type per dataset (5 in total) and five properties per type (25 in total). For each type, we show (see Table 3) the number of `owl:sameAs`-cliques generated, and the number of triples before and after the rewriting process. To show the benefits of studying cardinality constraints derived from automatically discovered bounds in KBs, we bring to the fore their use on the realm of completeness and consistency. We evaluate each entity type in these two dimensions from a common sense point of view. Because the consideration of cardinality bounds is application dependent, here we try to abstract (without loss of generality) from individual use cases. The cardinalities presented herein are considered robust bound assessed to be a constraint by a knowledge engineer. We consider that a property p in the context of a type τ is *complete* given a cardinality constraint if every entity s of type τ has the 'right number' of triples (s, p, o), and *incomplete* otherwise. For example, a constraint might be that all books must have at least one property `title`, but the same it is not true for property `comment`. Also, we consider that a property p in the context of a type τ is *consistent* if the triples with predicate p and subject s (of type τ) comply with the cardinality bounds, and *inconsistent* otherwise. For example, a constraint might be that all books must have always 1 `title`; however, we found five books which violate this constraint having 2 titles. Based on a set of verified discovered robust bounds, in Table 3 we show the ratios of completeness and consistency found in the 5 properties per type. For example, 2/5 completeness ratio in the entity type Book indicates that 2 out of 5 properties presented complete data, and the rest was incomplete. We did the same to measure consistency. In particular, we noticed a strong consistency on synthetic datasets, where it is normal to define an ontology that all instances generated will satisfy.

Table 3: Evaluation of completeness and consistency per dataset: one type and five random properties per type.

Class	№ sA-Cliques	№ Triples Befo./After	Complet. Ratio	Consist. Ratio
Fashion Model	118	1060/928	2/5	5/5
Research Assistant	0	135197/135197	4/5	5/5
Book	4515	97101/83556	2/5	3/5
Country	0	21766/21766	1/5	4/5
Concept	4979	58685/48780	2/5	5/5

```
card({mondial:name}, mondial:Volcano)=(1,1)
card({mondial:elevation}, mondial:Volcano)=(1,1)
card({mondial:longitude}, mondial:Volcano)=(1,1)
card({mondial:latitude}, mondial:Volcano)=(1,1)
card({mondial:locatedIn}, mondial:Volcano)=(1,3)
card({mondial:lastEruption}, mondial:Volcano)=(0,1)
```

Fig. 5: Cardinality bounds subset for entity type `mondial:Volcano`.

Figure 5 shows a subset of the cardinality bounds extracted for the entity type Volcano in Mondial KB. At a first glance, the extracted patterns correspond to what a knowledge engineer could expect: all volcanoes have a name and latitude/longitude coordinates, not all have information about their last eruption, and they have 1 to 3 locations for those that are in the intersection of different countries. Notice that even when we obtain a cardinality for `mondial:locatedIn` with upper bound 3, this cannot be considered as a constraint and used for validation until it is assessed by a user. We found a similar situation with the property `mondial:hasCity` in the type `mondial:Country`, where the robust cardinality identified was $card(\{\texttt{mondial:hasCity}\}, \texttt{mondial:Country}) = (1, 31)$. However, based on that upper limit, there are 25 identified outliers, which are not real outliers. For instance, we have China with 306 cities, USA with 250 cities, Brazil with 210 cities, Russia with 171 cities, and India with 99 cities, which are outside of the range considered as robust. Based on this information, data users and knowledge engineers could be able to determine whether a given cardinality pattern should be promoted to become a constraint (i.e., verified discovered robust bounds) or not. We argue that by detecting cardinality inconsistencies and incompleteness we can determine structural problems at the instance level. This can be used to guide repair methods and move towards better quality of KBs.

7 Conclusions and Outlook

KBs contain implicit patterns and constraints not always stated in ontologies or schemata neither clear for data creators and consumers. In this paper, we have introduced an extensional approach to the discovery of so-called cardinality bounds in KBs. Cardinality bounds are proposed as a solution for the need to (partially) cope with the lack of explicit structure in KBs. We presented two implementations of our approach, namely SPARQL- and Spark-based, and

evaluated them against five different datasets including real-world and synthetic data. Our analysis shows that cardinality bounds can be mined efficiently, and are useful to understand the structure of data at a glance, and also to analyse the completeness and consistency of data.

Future work in this area could go into various directions. First, we notice that cardinality bounds give an indication about the completeness of data, and thus can be used to guide methods for knowledge completion or link prediction [5]. Second, further research is required on the assessment of the discovered cardinality bounds by users, and whether they can be promoted as normative constraints that could be used for validation [19]. Finally, the structure provided by cardinality patterns could serve to generate schema graphs that represent the characteristics that KBs naturally exhibit, unlocking management problems such as query optimisation [27].

Acknowledgements. This work has been supported by TOMOE project funded by Fujitsu Laboratories Ltd., Japan and Insight Centre for Data Analytics at National University of Ireland Galway, Ireland.

References

1. Bosch, T., Eckert, K.: Guidance, please! towards a framework for RDF-based constraint languages. In: Proc. of the International Conference on Dublin Core and Metadata Applications. (2015)
2. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. **41**(3) (2009)
3. Ferrarotti, F., Hartmann, S., Link, S.: Efficiency frontiers of XML cardinality constraints. Data and Knowledge Engineering **87** (2013) 297–319
4. Fleischhacker, D., Paulheim, H., Bryl, V., Völker, J., Bizer, C.: Detecting errors in numerical Linked Data using cross-checked outlier detection. In: Semantic Web Conference (1). Volume 8796 of Lecture Notes in Computer Science., Springer (2014) 357–372
5. Galárraga, L., Razniewski, S., Amarilli, A., Suchanek, F.M.: Predicting completeness in knowledge bases. In: WSDM, ACM (2017) 375–383
6. Glimm, B., Hogan, A., Krötzsch, M., Polleres, A.: OWL: Yet to arrive on the Web of Data? In: LDOW. Volume 937 of CEUR Workshop Proceedings., CEUR-WS.org (2012)
7. Hogan, A., Harth, A., Passant, A., Decker, S., Polleres, A.: Weaving the Pedantic Web. In: LDOW. Volume 628 of CEUR Workshop Proceedings., CEUR-WS.org (2010)
8. Kellou-Menouer, K., Kedad, Z.: Evaluating the gap between an RDF dataset and its schema. In: Proc. of the Advances in Conceptual Modeling - ER 2015 Workshops. (2015) 283–292
9. Lausen, G., Meier, M., Schmidt, M.: SPARQLing constraints for RDF. In: EDBT. (2008) 499–509
10. Liddle, S.W., Embley, D.W., Woodfield, S.N.: Cardinality constraints in semantic data models. Data & Knowledge Engineering **11**(3) (dec 1993) 235–270
11. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. Web Semantics: Science, Services and Agents on the World Wide Web **7**(2) (2009) 74–89

12. Motik, B., Nenov, Y., Piro, R.E.F., Horrocks, I.: Handling Owl:sameAs via rewriting. In: AAAI, AAAI Press (2015) 231–237
13. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Web Ontology Language structural specification and functional-style syntax (second edition). http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/ (2012)
14. Muñoz, E.: On learnability of constraints from RDF data. In: ESWC. Volume 9678 of Lecture Notes in Computer Science., Springer (2016) 834–844
15. Neumann, T., Moerkotte, G.: Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In: ICDE, IEEE Computer Society (2011) 984–994
16. Paulheim, H.: Knowledge graph refinement: A survey of approaches and evaluation methods. Semantic Web – Interoperability, Usability, Applicability an IOS Press Journal (2016)
17. Paulheim, H., Bizer, C.: Improving the quality of Linked Data using statistical distributions. Int. J. Semantic Web Inf. Syst. **10**(2) (2014) 63–86
18. Pearson, R.K.: Mining Imperfect Data: Dealing with Contamination and Incomplete Records. Society for Industrial and Applied Mathematics (2005)
19. Prud'hommeaux, E., Gayo, J.E.L., Solbrig, H.R.: Shape expressions: an RDF validation and transformation language. In: SEMANTICS, ACM (2014) 32–40
20. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The VLDB Journal **10**(4) (dec 2001) 334–350
21. Rivero, C.R., Hernández, I., Ruiz, D., Corchuelo, R.: Towards discovering ontological models from big RDF data. In: ER Workshops. Volume 7518 of Lecture Notes in Computer Science., Springer (2012) 131–140
22. Rosner, B.: Percentage points for a generalized ESD many-outlier procedure. Technometrics **25**(2) (1983) 165–172
23. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edition. 2 edn. Pearson Education (2009)
24. Ryman, A.G., Hors, A.L., Speicher, S.: OSLC resource shape: A language for defining constraints on Linked Data. In: Proceedings of the WWW2013 Workshop on Linked Data on the Web. (2013)
25. Schenner, G., Bischof, S., Polleres, A., Steyskal, S.: Integrating distributed configurations with RDFS and SPARQL. In: Configuration Workshop. Volume 1220 of CEUR Workshop Proceedings., CEUR-WS.org (2014) 9–15
26. Schmidt, M., Lausen, G.: Pleasantly consuming Linked Data with RDF data descriptions. In: COLD, CEUR-WS.org (2013)
27. Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL query optimization. In: ICDT, ACM (2010) 4–33
28. Thalheim, B.: Fundamentals of cardinality constraints. In: Proc. of the 11th International Conference on the Entity-Relationship Approach: Entity-Relationship Approach. ER '92 (1992) 7–23
29. Töpper, G., Knuth, M., Sack, H.: DBpedia ontology enrichment for inconsistency detection. In: I-SEMANTICS, ACM (2012) 33–40
30. Völker, J., Niepert, M.: Statistical schema induction. In: ESWC (1), Springer (2011) 124–138
31. Wienand, D., Paulheim, H.: Detecting incorrect numerical data in DBpedia. In: ESWC. Volume 8465 of Lecture Notes in Computer Science., Springer (2014) 504–518