IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 14, 2022, accepted February 26, 2022, date of publication March 2, 2022, date of current version March 10, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3155816

Inductive Learning of OWL 2 Property Chains

JEDRZEJ POTONIEC
Institute of Computing, Poznań University of Technology, 60-965 Poznań, Poland
e-mail: jpotoniec@cs.put.poznan.pl

This work was supported in part by the Poznań University of Technology under Grant 09/91/DSMK/0659.

ABSTRACT We present an algorithm to inductively learn Web Ontology Language (OWL) 2 property chains to be used in object subproperty axioms. For efficiency, it uses specialized encodings and data structures based on hash-maps and sparse matrices. The algorithm is based on the frequent pattern search principles and uses a novel measure called *s-support*. We prove soundness and termination of the algorithm, and report on evaluation where we mine axioms from DBpedia 2016-10. We extensively discuss the 36 mined axioms and conclude that 30 (83%) of them are correct and could be added to the ontology.

INDEX TERMS Ontology learning, owl 2, property chains, semantic web.

I. INTRODUCTION

The Semantic Web, as envisioned in the paper by Berners-Lee *et al.* [1], is the Internet where the semantics of information is explicitly represented and thus comprehensible for both humans and machines in an unambiguous way. The Semantic Web enables services such as personal agents, that leverage semantic information to perform intelligent tasks on behalf of their owners. The representation of choice for the Semantic Web are node- and edge-labeled graphs expressed in Resource Description Framework (RDF) and accompanied by formal semantics expressed in Web Ontology Language 2 (OWL 2), both detailed in subsection II-A.

Unfortunately, such a state of the Web is still to be achieved. One of the main blockers, present not only in the Semantic Web, but in all other knowledge-oriented systems, is the so-called knowledge acquisition bottleneck, which represents the slowness and complexity of collecting and formalizing general knowledge, e.g., as a formal ontology.

To address this blocker in the context of the Semantic Web, a problem of ontology learning was posed in 2001 by Maedche and Staab [2], and extensively researched since then. Ontology learning is a well-developed area in the research on the Semantic Web, concerned with a (semi-)automatic creation of ontologies from various, preexisting resources, structured and unstructured alike. A reader interested in a general overview on ontology learning is referred to a book edited by Lehmann and Völker [3]. In this paper, we are interested in learning an ontology from structured data, namely from a preexisting RDF graph, and works in such a setup were already conduced.

Most of such works concentrated on mining axioms related to the hierarchy of classes. For example, Potoniec *et al.* proposed *Swift Linked Data Miner*, a method for mining class expression serving as super-classes in class inclusion axioms [4]. Völker *et al.* proposed algorithms for mining class hierarchy [5] and class disjointness [6] using association rule mining. Li and Sima developed a method for parallel learning of an OWL 2 EL ontology from a large Linked Data repository [7]. Hellmann *et al.* presented DL-Learner, a tool for learning class descriptions from very large knowledge bases [8]. Recently, a method for mining OWL cardinality restrictions from knowledge graphs has been proposed by Potoniec [9]. Finally, Melo *et al.* reported on a method on adding type assertion to entities in RDF knowledge bases [10].

The body of work on automatic acquisition of hierarchy of properties is far smaller. Ell *et al.* researched on a bridge between class hierarchy and property hierarchy by proposing a method for automatic, joint discovery of domain and range restrictions from an RDF dataset [11]. Irny and Kumar presented a work on inverse and symmetric property axioms [12], while Fleischhacker *et al.* proposed an application of association rule mining to discover property characteristic axioms, such as functionality [13]. An approach for discovering property characteristic axioms based on recurrent neural networks was recently proposed by Potoniec [14].

In this paper we consider a subproblem of the ontology learning, namely learning object subproperty axioms with property chains, which were introduced by the Web Ontology

The associate editor coordinating the review of this manuscript and approving it for publication was Zhe Xiao.

Language 2 (OWL 2) [15]. Intuitively, a property chain enables expressing reasoning rules on the chains of relations, e.g., *If X is a parent of Y and Y is a parent of Z, then X is a grandparent of Z.* The goal of the proposed algorithms is to discover such axioms from frequently repeating patterns in an RDF graph, and then measuring their confidence to distinguish between coincidences and true patterns suitable to be added to the ontology of the graph.

Ontology learning can use various resources, e.g., text, databases, RDF graphs. In this paper we opt for the RDF graphs, i.e., we learn axioms directly from the actual use of vocabulary in an RDF graph. This is rooted in an assumption that it is much easier to use a vocabulary to describe data in an RDF graph than to properly axiomatize it to form an ontology. This is also in line with the vast amounts of RDF data available in the Linked Open Data [16]. To the best of our knowledge, this is the first algorithm capable of learning OWL 2 property chains from the structured data of RDF graphs. The necessary data structures used by the algorithm are described in subsection II-B and its details are presented in subsection II-C.

To evaluate the algorithm, we use DBpedia, a result of a complex knowledge extraction process from Wikipedia [17]. We mine a set of axioms using DBpedia 2016-10 and discuss extensively which of them should be included in the DBpedia ontology and why. The details of the experiment and the discussion are reported in section III and section IV. We remark that due to the nature of encyclopedic data, some of the examples in the paper discuss political entities, their names and locations. Presented states and conclusions are drawn from the DBpedia and the Wikipedia and are, by no means, any reflection of the political views of the author or a try to solicit one state of affairs over another.

The contributions of the paper are as follows:

- We introduce *s-support*, a new measure for frequent pattern mining and prove its properties.
- We introduce a new, efficient algorithm suitable for construction OWL 2 object subproperty axioms, in particular axioms containing property chains.
- We evaluate the algorithm on a real-world dataset.

II. MATERIALS AND METHODS

A. PRELIMINARIES

1) RESOURCE DESCRIPTION FRAMEWORK

In the paper, we consider data represented using Resource Description Framework (RDF) [18]. A basic unit of information in RDF is an *RDF triple*, consisting of a *subject*, a *predicate* and an *object*. Meaning of the information is considered within a universe of discourse. Such a triple means that the entity represented by the subject is connected with the entity represented by the object with the relation denoted by the predicate. A set of RDF triples is called an *RDF graph*. A set of all subjects and objects form a set of *RDF nodes*. An RDF node can be one of the following: an *Internationalized Resource Identifier* (IRI), a *blank node* or a *literal*.

```
:Arietty      :director     :Yonebayashi   ;
              :participant  :Yonebayashi   ;
              :studio       :Ghibli        .
:SA           :director     :Miyazaki      ;
              :participant  :Miyazaki      ;
              :studio       :Ghibli        ;
              :next         :Arietty       .
:Laputa       :next         :Totoro, :SA   ,
                            :Arietty       .
:Totoro       :next         :SA, :Arietty  .
:Miyazaki     :member       :Ghibli        .
:Yonebayashi  :member       :Ghibli        .
```

Listing 1. An RDF graph used as a running example in the paper, expressed in Turtle.

An IRI is a global identifier for an entity from the universe of discourse, that is the same IRI denote the same entity in different graph, whereas a blank node is a local identifier for an entity, that is the same blank node may denote different entities in different graphs. A literal is a concrete value such as a string of characters or a number. The subject of a triple is necessarily an IRI or a blank node and the predicate is an IRI. An IRI that occurs in the predication position of a triple is called a *property*. Let $\mathbb{G}$ be an RDF graph, $\mathbb{I}$ denote the set of all valid IRIs, $\mathbb{B}$ the set of all valid blank nodes and $\mathbb{L}$ the set of valid literal, then:

$$\mathbb{G} \subseteq (\mathbb{I} \cup \mathbb{B}) \times \mathbb{I} \times (\mathbb{I} \cup \mathbb{B} \cup \mathbb{L})$$

In this paper, we typeset IRIs using a monospace font, e.g, `http://dbpedia.org/resource/Poland`. To make the text more readable, we use prefixes to represent common namespaces: we replace `http://dbpedia.org/resource/` with `dbr:` and `http://dbpedia.org/ontology/` with `dbo:`. We use `:` as the prefix in examples. To present RDF graphs, we use Turtle notation, in which a comma separates two triples sharing a subject and a predicate, a semicolon separates two triples sharing a subject, and a dot separates two triples with different subjects [19]

Throughout the paper we use an RDF graph presented in Listing 1 as the running example. The graph describes four movies by *Studio Ghibli* (`:Ghibli`): *Laputa: Castle in the Sky* (`:Laputa`), *My Neighbor Totoro* (`:Totoro`), *Spirited Away* (`:SA`) and *The Secret World of Arrietty* (`:Arietty`). Here the movies are ordered from the oldest to the newest and this is represented in the graph by the `:next` property, which links a movie with all the later movies by *Studio Ghibli*. The first three movies were directed by Hayao Miyazaki (`:Miyazaki`), while the last was directed by Hiromasa Yonebayashi (`:Yonebayashi`).

In principle, a single entity may be represented using multiple identifiers, e.g., `:SpiritedAway`, `:SA` and `:Spirited_Away` refer all to the same movie *Spirited Away*. It is a research problem on its own to identify such cases, we thus make the Unique Name Assumption (UNA), that is we assume that different identifiers necessarily denote different entities. For a given RDF graph, if the assumption

is not met, but it is known which identifiers correspond to the same entity, the graph can be easily normalized by selecting a single identifier for each entity and replacing the remaining identifiers with the selected one. Otherwise, a method for entity resolution must be employed prior to using methods proposed in this paper.

To query an RDF graph, there are multiple query languages, but *SPARQL Query Language* (SPARQL) is the W3C recommendation and by far the most popular of these. In this paper we use SPARQL very sparingly and only in section III to formalize some points in the discussion. The language is in form similar to SQL, but it denotes variables by prefixing them with ? and expresses the conditions in a WHERE clause as graph patterns, i.e., RDF graphs with some of the entities replaced by variables. The details on syntax and semantics of SPARQL can be found in [20].

2) WEB ONTOLOGY LANGUAGE

An RDF graph represents assertional knowledge, i.e., knowledge about concrete entities. It can be accompanied by an ontology, i.e., a representation of taxonomic knowledge. Within the Semantic Web, ontologies are usually expressed using the OWL 2 Web Ontology Language [15]. It is a language underpinned by the Description Logics, enabling a very expressive deductive reasoning. There are three basic notions in OWL: an *entity* is a reference to an entity from a universe of discourse; an *expression* is a complex description formed as a combination of entities; a *axiom* is a single piece of knowledge expressed by the ontology. Entities can be separated into three types: *individuals*, representing concrete entities; *classes*, representing groups of entities having some common feature; *properties*, representing relations between entities. When OWL is used together with RDF, individuals correspond to IRIs and blank nodes and properties to (RDF) properties. Properties in OWL are separated further into *object properties*, i.e., properties that link two entities, and *data properties*, i.e., properties that link an entity with a literal. From the RDF point of view, a triple using an object property must belong to $(\mathbb{I} \cup \mathbb{B}) \times \mathbb{I} \times (\mathbb{I} \cup \mathbb{B})$, while a triple using a data property must belong to $(\mathbb{I} \cup \mathbb{B}) \times \mathbb{I} \times \mathbb{L}$.

We express OWL using the Manchester syntax [21]. We present only the relevant part of syntax and semantics of OWL, concerned with the hierarchy of properties. The paper concentrates on object subproperty axioms, which define a hierarchy of object properties. Such an axiom is of form p SubPropertyOf: r, denoting that whenever (s, p, o) is true, (s, r, o) is also true. The left-hand side of such an axiom may contain a *property chain* instead of a single property: $p_1 \circ p_2 \circ \ldots \circ p_n$ SubPropertyOf: r, meaning that whenever there exists $t_1, \ldots, t_{n-1}$ such that the triples $(s, p_1, t_1), (t_1, p_2, t_2), \ldots (t_{n-1}, p_{n-1}, o)$ are true, then (s, r, o) is also true. It may be the case that there are two axioms: p SubPropertyOf: r and r SubPropertyOf: p. Such a situation means that p and r are equivalent, i.e., (s, p, o) is true if, and only if, (s, r, o) is true. For short we denote it by p EquivalentTo: r.

3) OPEN-WORLD ASSUMPTION

Both RDF and OWL make the Open-world assumption (OWA), stating that if a piece of information is not known to be true nor known to be false, its truth value is unknown. This is very different from the Close-world assumption made in databases or in Prolog programming language: if a piece of information is not known to be true, it is assumed to be false.

This poses a serious challenge for learning from RDF and OWL data, as for a piece of information not present in the data (e.g., an RDF triple not present in the considered graph) one cannot easily distinguish between (a) the piece is untrue, but the negative knowledge was not asserted; (b) the piece is true and not asserted.

This inseparable overlap of two very different situations requires making some assumptions to enable mining. First of all, all the mined axioms may be only considered as suggestions for an ontology engineer, who must then make a separate decision for each axiom. The purpose of the method may be seen as to detect and model knowledge that is present in the data, but is not reflected in the ontology.

Secondly, we assume that one can very roughly separate the overlap in the following way: for a given RDF triple, if there are no similar triples this probably means that the triple is untrue (e.g., an ontology does not forbid a river to be a mountain at the same time, but if in the corresponding RDF graph there are no objects that are rivers and mountains that probably means that it is either impossible or at least unlikely). On the other hand, if there are multiple similar triples, this hints that the considered triple may be true and was omitted from the graph. This assumption is formulated in very broad and coarse terms and the rest of the paper is dedicated to formalizing it and verifying the efficiency and efficacy of the method based on it.

The presented approach is based on machine learning techniques what makes it capable of dealing with noisy and incomplete data through employing jointly s-support and confidence measures. Unfortunately, due to the OWA there is usually very little, if any, negative information to use as negative examples in machine learning. This may easily lead to overly general axioms generated by the algorithm mistaking lack of information with negative information. Unfortunately, this cannot be easily addressed in an automated way without making additional assumptions about the considered data, particularly because RDF and RDFS do not allow to express any negative knowledge. In this work we follow the similarity assumption described above and infer from it that the entities in the considered knowledge graph rather suffer from omissions than purposeful lack of information. In some cases, like in a highly curated, manually-created graph, this assumption will likely be untrue and many of the generated axioms may be needlessly and overly general. This again underlines the necessity of treating the generated axioms only as suggestions which must be further validated by the ontology engineer in order to ensure the quality of the ontology.

4) REMARKS ON NOTATION

The paper uses multiple complex data structures and operations on them. To avoid confusion, we consistently use the following notation: square brackets [] denote a vector of variable length, while round braces () denote a tuple of fixed length and curly braces {} denote an unordered set. An element with an index (resp. a key) y of a vector (resp. a map) X is denoted by $X[y]$. To concatenate two vectors we use operator $\circ$, remarking that if an argument of the operator is a singleton, we omit the braces. Element-wise multiplication is denoted by $\odot$, whereas $\cdot$ and $\prod$ denote matrix multiplication.

We denote the presence of a triple (s, p, o) in an RDF graph $\mathbb{G}$ by $(s, p, o) \in \mathbb{G}$. To shorten the notation we abbreviate

$$\exists t_1, \ldots, t_{n-1} \colon (s, p_1, t_1), \ldots, (t_{n-1}, p_n, o) \in \mathbb{G}$$

to $(s, p_1 \circ p_2 \circ \ldots \circ p_n, o) \in \mathbb{G}$. When $\mathfrak{p} = p_1 \circ p_2 \circ \ldots \circ p_n$, we write $(s, \mathfrak{p}, o) \in \mathbb{G}$ with the same meaning.

B. DATA STRUCTURES

1) ENCODING IRIs AS NUMBERS

To enable efficient representations, described in the next subsections, we begin with encoding all the IRIs in the considered RDF graph $\mathbb{G}$ as numbers. Let n_{so} be the number of distinct IRIs occurring in the subject or object positions in the triples of the graph $\mathbb{G}$, i.e.,

$$n_{so} = |\{s \colon \exists p, o \colon (s, p, o) \in \mathbb{G}\} \cup \{o \colon \exists s, p \colon (s, p, o) \in \mathbb{G}\}|$$

Let n_p be the number of distinct IRIs occurring in the predicate position of the triples of the graph $\mathbb{G}$, i.e.,

$$n_p = |\{p \colon \exists s, o \colon (s, p, o) \in \mathbb{G}\}|$$

In the paper we assume that the IRIs in the subject and object positions are represented by integer numbers from 0 to $n_{so} - 1$ and IRIs in the predicate position are represented by integer numbers from 0 to $n_p - 1$. In the case when an IRI is used in both contexts, it is assigned two, possibly distinct, numbers. The encoding from IRIs to numbers is done while loading the data, when it can be efficiently executed using two hash-maps from IRIs to numbers. The decoding from numbers to IRIs is needed only when outputting the final patterns and can be efficiently realized using two arrays mapping numbers to IRIs. In the remainder of the paper we will assume that IRIs are represented as numbers and that the graph $\mathbb{G}$ is composed of triples of numbers. However, for clarity, we will still be referring to them as to IRIs, to underline the semantic of the numbers.

A sample encoding of IRIs from Listing 1 is presented in Listing 2.

2) MAP-BASED INDICES FOR EFFICIENT SEARCHING

The algorithm proposed in subsection II-C requires an efficient way to query for all predicate and object pairs present in the graph for a given subject s. To address this, we propose to use a double index based on hash maps, which we denote henceforth by *SPO*. An example of such an index, representing the graph from Listing 1 using the encoding from Listing 2, is presented in Figure 1. The first level of the index consists of a map from subjects present in the graph to the pointers to separate maps in the second level. Each map in the second level maps properties present for the given subject in the graph to the vectors in the third level. Each vector in the third level contains all the objects present in the graph for the subject and property determined by the higher levels.

Later in the paper we use another index, called *PSO*. Its structure is the same, but the order differs: the first level contains properties, the second subjects and the third objects.

3) MATRIX-BASED INDICES FOR EFFICIENT QUERYING AND COUNTING

The algorithm requires an efficient way to jointly consider pairs of subjects and objects for a given property p. For this, we use the following matrix PSO^p:

$$PSO^p_{s,o} = \begin{cases} 1 & (s, p, o) \in \mathbb{G} \\ 0 & \text{otherwise} \end{cases}$$

A single matrix is created for each property in the graph. These matrices are sparse, so a suitable representation, supporting efficient matrix multiplication and element-wise multiplication, is needed. Moreover, we are interested in efficient iterating over of all non-zero elements and efficient counting them. These properties can be achieved by using compressed sparse row representation (CSR) [22].

A CSR matrix of type $n \times n$, containing k non-zero elements is represented as three vectors of numbers: *AA* a vector of length k of real values representing the non-zero elements of the matrix, ordered row-by-row; *JA* a vector of length k of integer values representing the column numbers of the corresponding elements of the previous vector; *IA* a vector of length $n + 1$ of integer values representing the beginning of each row in the previous vector, with the last element being $k + 1$.

For example, consider the following matrix M:

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 0 & 5 \end{bmatrix}$$

Its CSR representation is as follows:

$$AA = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$
$$JA = \begin{bmatrix} 1 & 1 & 2 & 1 & 3 \end{bmatrix}$$
$$IA = \begin{bmatrix} 1 & 2 & 4 & 6 \end{bmatrix}$$

Observe that with such a definition, matrix multiplication is equivalent to computing all the resources connected by the chain corresponding to the used matrices. Similarly, element-wise multiplication corresponds to computing all the pairs that share the corresponding properties.

Theorem 1: For a given property chain $p_1 \circ p_2 \circ \ldots \circ p_n$ $(n \geq 2)$ and for any IRIs s, o:

$$(s, p_1 \circ p_2 \circ \ldots \circ p_n, o) \in \mathbb{G} \iff Y_{s,o} \neq 0$$

```
:Arietty :SA :Laputa :Totoro :Miyazaki :Yonebayashi :Ghibli
    0     1     2       3         4            5          6
:director :participant :studio :next :member
    0           1         2      3      4
```

Listing 2. An example of encoding of IRIs as numbers using the IRIs from the graph in Listing 1.

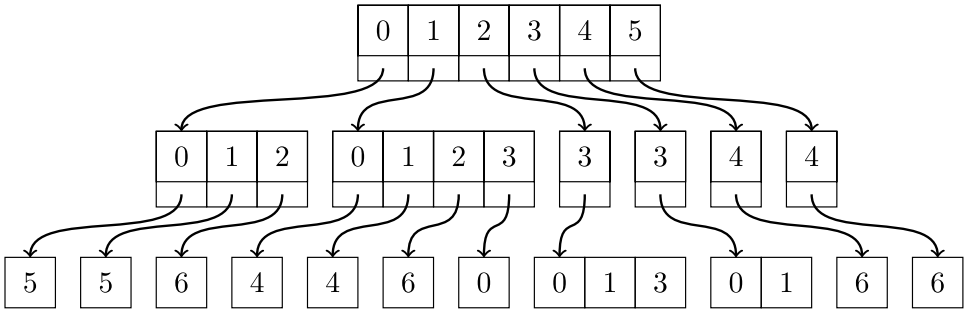

FIGURE 1. A map-based index encoding the sample RDF graph presented in Listing 1, using the IRIs as numbers encoding from Listing 2. There are three levels, the first representing subjects, the second predicates and the third objects. Each cell consists of a number and a pointer to the corresponding part of the next level. Cells with shared borders are represented together in a single structure that can be easily iterated over. For example, the path from 2 to 3 to 3 represents the triple `:Laputa :next :Totoro`.

where

$$Y = \prod_{i=1}^{n} PSO^{p_i}$$

Proof: Assume $n = 2$, that is

$$(s, p_1 \circ p_2, o) \in \mathbb{G} \iff (PSO^{p_1} \cdot PSO^{p_2})_{s,o} \neq 0 \qquad (1)$$

From the definition of matrix multiplication:

$$(PSO^{p_1} \cdot PSO^{p_2})_{s,o} = \sum_t PSO^{p_1}_{s,t} PSO^{p_2}_{t,o}$$

This sum is nonzero if for at least one t

$$PSO^{p_1}_{s,t} = 1 \wedge PSO^{p_2}_{t,o} = 1$$

From the definition of the PSO matrices, this means that

$$(s, p_1, t) \in \mathbb{G} \wedge (t, p_2, o) \in \mathbb{G}$$

This is the left-hand side of Equation 1, which concludes the proof for $n = 2$. For $n > 2$, the proof is by induction. □

Theorem 2: For any given set of properties $\{p_1, p_2, \ldots, p_n\}$ $(n \geq 2)$ and for any IRIs s, o:

$$(s, p_1, o), (s, p_2, o), \ldots, (s, p_n, o) \in \mathbb{G} \iff Y_{s,o} \neq 0$$

where

$$Y = PSO^{p_1} \odot PSO^{p_2} \odot \ldots \odot PSO^{p_n}$$

Proof: Assume $n = 2$, that is

$$(s, p_1, o), (s, p_2, o) \in \mathbb{G} \iff (PSO^{p_1} \cdot PSO^{p_2})_{s,o} \neq 0$$

From the definition of the *PSO* matrices:

$$PSO^{p_1}_{s,o} = 1 \iff (s, p_1, o) \in \mathbb{G}$$
$$PSO^{p_2}_{s,o} = 1 \iff (s, p_2, o) \in \mathbb{G}$$

It follows that $PSO^{p_1}_{s,o} \cdot PSO^{p_2}_{s,o} = 1$ if, and only if, $(s, p_1, o) \in \mathbb{G}$ and $(s, p_2, o) \in \mathbb{G}$.

For $n > 2$, the proof is by induction. □

Recall the graph from Listing 1 and encoding from Listing 2. For the property `:next`, we obtain the following matrix:

$$PSO^{:\texttt{next}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Multiplying $PSO^{:\texttt{next}}$ by itself we obtain:

$$PSO^{:\texttt{next}} \cdot PSO^{:\texttt{next}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The coordinates of non-zero values are $(2, 0)$, $(2, 1)$, $(3, 0)$ and decoding with Listing 2 we obtain the pairs connected through the property chain `:next ∘ :next`: (`:Laputa`, `:Arietty`), (`:Laputa`, `:SA`), (`:Totoro`, `:Arietty`).

The functions using the proposed theorems are presented as function `Query` in Algorithm 1 and as function `Count` in Algorithm 2. The arguments of the functions are n property chains $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$. The result of `Query` is the set of pairs (s, o) such that for all $i = 1, \ldots, n$ $(s, \mathfrak{p}_i, o) \in \mathbb{G}$, i.e., o is reachable from s by each of the property chains $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$.

The second function only returns the number of such pairs, as it is more efficient than running `Query` and counting the results.

Algorithm 1 An Algorithm for Querying for Pairs (s, o) Such That o Can Be Reached From s on All the Property Chains $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$

```
function Query(p_1, ..., p_n)
    for i = 1, ..., n do
        X_i ← ∏_{p∈p_i} PSO^p
    end
    Y ← X_1 ⊙ ... ⊙ X_n
    return {(s, o): Y_{s,o} ≠ 0}
end
```

Algorithm 2 An Algorithm for Counting the Number of Pairs (s, o) Such That o Can Be Reached From s on All the Property Chains $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$

```
function Count(p_1, ..., p_n)
    for i = 1, ..., n do
        X_i ← ∏_{p∈p_i} PSO^p
    end
    Y ← X_1 ⊙ ... ⊙ X_n
    return number of non-zero elements in Y
end
```

C. MINING PROPERTY INCLUSION AXIOMS

1) USED MEASURES

The proposed algorithm is based on frequent pattern mining. For this, we must first define what constitutes a pattern and when a pattern becomes frequent. In this paper, any property chain is a pattern and to avoid confusion we avoid the word *pattern* and use *property chain* instead.

Definition 3: The support set *for a property chain* $\mathfrak{p}$ *in an RDF graph* $\mathbb{G}$*, denoted* $\mathbb{S}_{\mathbb{G}}(\mathfrak{p})$*, is the following set of pairs*:

$$\mathbb{S}_{\mathbb{G}}(\mathfrak{p}) = \{(s, o) \colon (s, \mathfrak{p}, o) \in \mathbb{G}\}$$

Definition 4: The subject support set *for a property chain* $\mathfrak{p}$ *in an RDF graph* $\mathbb{G}$*, denoted* $\mathbb{S}^s_{\mathbb{G}}(\mathfrak{p})$*, is the following set*:

$$\mathbb{S}^s_{\mathbb{G}}(\mathfrak{p}) = \{s \colon \exists o \colon (s, o) \in \mathbb{S}_{\mathbb{G}}(\mathfrak{p})\}$$

Theorem 5: For any property chain $\mathfrak{p}$ *and any property* p*, the subject support set of a property chain* $\mathfrak{p} \circ p$ *is a subset of the subject support set of the property chain* $\mathfrak{p}$*:*

$$\forall \mathfrak{p} \forall p \colon \mathbb{S}^s_{\mathbb{G}}(\mathfrak{p}) \supseteq \mathbb{S}^s_{\mathbb{G}}(\mathfrak{p} \circ p)$$

Proof: Starting from the right-hand side:

$$\begin{aligned}
\mathbb{S}^s_{\mathbb{G}}(\mathfrak{p} \circ p) &= \{s \colon \exists o \colon (s, o) \in \mathbb{S}_{\mathbb{G}}(\mathfrak{p} \circ p)\} \\
&= \{s \colon \exists o, t \colon [(s, t) \in \mathbb{S}_{\mathbb{G}}(\mathfrak{p}) \wedge (t, p, o) \in \mathbb{G}]\} \\
&= \{s \colon \exists t \colon [(s, t) \in \mathbb{S}_{\mathbb{G}}(\mathfrak{p}) \wedge \exists o \colon (t, p, o) \in \mathbb{G}]\} \\
&\subseteq \{s \colon \exists t \colon (s, t) \in \mathbb{S}_{\mathbb{G}}(\mathfrak{p})\} = \mathbb{S}^s_{\mathbb{G}}(\mathfrak{p})
\end{aligned}$$

□

Definition 6: The support *of a property chain* $\mathfrak{p}$ *in an RDF graph* $\mathbb{G}$*, denoted* $\sigma_{\mathbb{G}}(\mathfrak{p})$*, is the cardinality of the support set*:

$$\sigma_{\mathbb{G}}(\mathfrak{p}) = |\mathbb{S}_{\mathbb{G}}(\mathfrak{p})|$$

Definition 7: The s-support *(an abbreviation of* subject support*) of a property chain* $\mathfrak{p}$ *in an RDF graph* $\mathbb{G}$*, denoted* $\sigma^s_{\mathbb{G}}(\mathfrak{p})$*, is the cardinality of the subject support set*:

$$\sigma^s_{\mathbb{G}}(\mathfrak{p}) = \left|\mathbb{S}^s_{\mathbb{G}}(\mathfrak{p})\right|$$

Theorem 8: For any property chain $\mathfrak{p}$ *and any property* p *the following holds*:

$$\sigma_{\mathbb{G}}(\mathfrak{p}) \geq \sigma^s_{\mathbb{G}}(\mathfrak{p}) \geq \sigma^s_{\mathbb{G}}(\mathfrak{p} \circ p)$$

Proof: Follows directly from 4 and 5. □

Definition 9: The limited subject support set *for a property chain* $\mathfrak{p}$ *limited to the set* O *in an RDF graph* $\mathbb{G}$*, denoted* $\mathbb{S}^s_{O,\mathbb{G}}(\mathfrak{p})$*, is the following set*:

$$\mathbb{S}^s_{O,\mathbb{G}}(\mathfrak{p}) = \{s \colon \exists o \in O \colon (s, o) \in \mathbb{S}_{\mathbb{G}}(\mathfrak{p})\}$$

Definition 10: The limited s-support *for a property chain* $\mathfrak{p}$ *limited to the set* O *in an RDF graph* $\mathbb{G}$*, denoted* $\sigma^s_{O,\mathbb{G}}(\mathfrak{p})$*, is the cardinality of the corresponding limited subject support set*:

$$\sigma^s_{O,\mathbb{G}}(\mathfrak{p}) = \left|\mathbb{S}^s_{O,\mathbb{G}}(\mathfrak{p})\right|$$

Theorem 11: For any property chain $\mathfrak{p}$ *and any set* O *the following holds*:

$$\sigma^s_{\mathbb{G}}(\mathfrak{p}) \geq \sigma^s_{O,\mathbb{G}}(\mathfrak{p})$$

Proof: Follows directly from 4 and 9 □

Definition 12: A frequent property chain *is a property chain* $\mathfrak{p}$ *such that* $\sigma_{\mathbb{G}}(\mathfrak{p}) \geq$ *minsup, where minsup is a user-defined support threshold.*

A conclusion from the presented definitions and theorems is that the s-support and the limited s-support can be used as a monotonic measure for pruning during frequent pattern mining, meaning that if the (limited) s-support of $\mathfrak{p}$ is below a predefined threshold, any property chain extending $\mathfrak{p}$ also has the (limited) s-support below this threshold. Moreover, if $\sigma^s_{O,\mathbb{G}}(\mathfrak{p}) \geq$ *minsup*, then $\sigma_{\mathbb{G}}(\mathfrak{p}) \geq$ *minsup*, meaning that $\mathfrak{p}$ necessarily is a frequent property chain. In practice we use both: the s-support to decide whether a property chain is worth considering further and the limited s-support to decide whether a property chain is frequent within the set of IRIs that are of interest from the point of view of the task at hand.

The final measure of importance for this paper is *confidence*, measuring the percent of occurrences of the property chain supporting the axiom as a whole. On the first glance confidence may seem like a superfluous measure in the context of ontological knowledge, which is not statistical by nature. However, it may be the case that the RDF graph is noisy, e.g., due to humans errors arising from misunderstanding of the purpose of a property, or from incompleteness of the graph. Thus, it may be worth considering axioms with confidence less than 1, as they may still be correct axioms.

Definition 13: The confidence *of an axiom* $\mathfrak{p}$ SubPropertyOf: r *in an RDF graph* $\mathbb{G}$ *is given by the*

following formula:

$$\frac{|\mathbb{S}_{\mathbb{G}}(\mathfrak{p}) \cap \mathbb{S}_{\mathbb{G}}(r)|}{\sigma_{\mathbb{G}}(\mathfrak{p})}$$

2) PROBLEM STATEMENT

As stated in the introduction, we are interested in discovering OWL 2 object subproperty axioms with a special emphasis on axioms with property chains. We formalize the considered problem as follows: given an RDF graph $\mathbb{G}$, an object property r in this graph and three thresholds: the minimal confidence threshold *minconf*, the minimal s-support threshold *minsup* and the maximal length of the considered property chains *maxlen*, generate all axioms of form $\mathfrak{p}$ SubPropertyOf: r such that $\sigma^s_{\mathbb{G}}(\mathfrak{p}) \geq$ *minsup*, $\frac{|\mathbb{S}_{\mathbb{G}}(\mathfrak{p}) \cap \mathbb{S}_{\mathbb{G}}(r)|}{\sigma_{\mathbb{G}}(\mathfrak{p})} \geq$ *minconf* and $|\mathfrak{p}| \leq$ *maxlen*.

3) CONSTRUCTING OBJECT SUBPROPERTY AXIOMS OUT OF FREQUENT PROPERTY CHAINS

We begin the description by presenting an algorithm that, given a generator of frequent property chains, is able to construct object subproperty axioms. Its pseudocode is presented in Algorithm 3. The most important input to the algorithm is the property r that is to be placed in the right-hand side of the resulting axioms. There are also three numeric parameters that govern the behavior of the algorithm: the minimal s-support threshold *minsup*, the minimal confidence threshold *minconf* and the maximal length threshold *maxlen*. Finally, the graph for mining must be provided in the form of a hash-map index *SPO* and a family of matrix-based indices *PSO*.

The algorithm starts by querying for pairs of IRIs connected through the property r and constructing: a set of all objects O and a map from each subject to a set containing only this subject. Then, the `MineChains` function is called, which is detailed in the next section. For each of the returned frequent property chain $\mathfrak{p}$ confidence is computed using the function `Count`, presented in Algorithm 2. If the computed confidence reaches the minimal confidence threshold *minconf*, an axiom $\mathfrak{p}$ SubPropertyOf: r is added to the set of results. Finally, the set of results is returned to the user.

4) MINING FREQUENT PROPERTY CHAINS

The recursive function `MineChains` presented in Algorithm 4 mines frequent property chains using the limited s-support, as established in subsubsection II-C1. The function requires four arguments: a mapping S from the current nodes reached by the property chain to the starting points, a set of IRIs O representing end points for the property chain, *current*, a property chain constructed so far. In addition to that, the function uses the hash-map index *SPO* and two thresholds: *minsup* denoting a minimal value of s-support for a property chain to be kept as a candidate for a frequent property chain and *maxlen* denoting a maximal length of a property chain.

Algorithm 3 An Algorithm to Construct Object Subproperty Axioms With the Right-Hand Side Being a Given Property r

```
input : r, minconf, minsup, maxlen, SPO, PSO
S ← empty map
O ← ∅
foreach s, o ∈ Query([r]) do
    S[s] ← {s}
    O ← O ∪ {o}
end
result ← ∅
foreach 𝔭 ∈ MineChains(S, O, ∅) do
    conf ← Count(𝔭,[r]) / Count(𝔭)
    if conf ≥ minconf then
        result ← result ∪ {𝔭 SubPropertyOf: r}
    end
end
return result
```

First, the algorithm checks whether the property chain *current* successfully connects starting points and O by computing the limited s-support and comparing it with the threshold, and checks whether the property chain *current* is different from the property r, to avoid a trivial solution r SubPropertyOf: r. If it is so, the property chain *current* is returned and the execution of the function terminates. Then the length of the property chain *current* is checked and if it reached the threshold *maxlen*, the execution of the function is terminated. If both checks fail, the actual computation begins. A new index *POS* is constructed and filled by iterating over the index *SPO* using IRIs present in S. *POS* uses the same data structures as *SPO*, described in subsubsection II-B2, but it does not necessarily contain a subgraph of $\mathbb{G}$ and uses different ordering: properties in the first level, objects in the second and subjects in the third. After the execution of the triple-nested foreach, *POS* maps pairs (p, o) to sets of IRIs from which these pairs are reachable. Now *POS* is iterated over: for each property p in *POS*, the s-support is computed using the content of *POS*. If it reaches the threshold *minsup*, the function is executed recursively by using *POS*$[p]$, i.e., a map from IRIs to sets of IRIs from which they are reachable on the property chain *current* $\circ\, p$. Any results obtained this way are then returned from the function.

5) EXAMPLE

To better explain the proposed algorithm and highlight its capabilities, we use the sample RDF graph from Listing 1. For clarity of the example, we use IRIs instead of their encodings presented in Listing 2. We assume *minsup* $= 2$, *maxlen* $= 2$ and *minconf* $= 95\%$.

a: SINGLE SUBPROPERTY

In the first example, we mine for object subproperty axioms with `:participant` in the right-hand side, i.e.,

Algorithm 4 An Algorithm for Mining Frequent Property Chains

input : *minsup*, *maxlen*, *SPO*, *r*

function `MineChains` (*S*, *O*, *current*)
- **if** $|current| > 0 \wedge current \neq r \wedge \left|\bigcup_{s \in S \cap O} S[s]\right| \geq minsup$ **then**
  - **return** {*current*}
- **end**
- **if** $|current| \geq maxlen$ **then**
  - **return** $\emptyset$
- **end**
- $POS \leftarrow$ an empty index
- **foreach** $s \in S$ **do**
  - **foreach** $p \in SPO[s]$ **do**
    - **foreach** $o \in SPO[s, p]$ **do**
      - $POS[p, o] \leftarrow POS[p, o] \cup S[s]$
    - **end**
  - **end**
- **end**
- $result \leftarrow \emptyset$
- **foreach** $p \in POS$ **do**
  - $sup \leftarrow \left|\bigcup_{o \in POS[p]} POS[p, o]\right|$
  - **if** $sup \geq minsup$ **then**
    - $result \leftarrow result \cup$ `MineChains`$(POS[p], O, current \circ p)$
  - **end**
- **end**
- **return** *result*

end

$r =$ `:participant`. We begin by computing S and O:

$$S = \{\texttt{:Arietty} \mapsto \{\texttt{:Arietty}\}, \texttt{:SA} \mapsto \{\texttt{:SA}\}\}$$
$$O = \{\texttt{:Miyazaki}, \texttt{:Yonebayashi}\}$$

Then, `MineChains` is called. Neither condition in the beginning is met and the algorithms starts by constructing the index *POS*, iterating over all triples having `:Arietty` or `:SA` as the subject. The final index is presented in Listing 3. Now, the algorithm iterates over the properties in *POS*. Lets first consider $p =$ `:participant`: $sup = |\{\texttt{:Arietty}\} \cup \{\texttt{:SA}\}| \geq minsup = 2$, so a recursive call is made with $S = POS[\texttt{:participant}]$ and $current =$ `:participant`.

In the beginning of the recursive call the limited s-support condition is fulfilled, but the second part of the condition is not, so the computation continues. The length of *current* does not reach *maxlen*, so *POS* in the recursive call is computed:

$$POS = \{\texttt{:member} \mapsto \{\texttt{:Ghibli} \mapsto \{\texttt{:Arietty}, \texttt{:SA}\}\}\}$$

Again, the s-support is high enough and yet another recursive call is made with $current =$ `:participant` $\circ$ `:member`, but it returns $\emptyset$ due to reaching the maximal length threshold *maxlen*. We are now back at $current =$ `:participant`, but $result = \emptyset$ and this call also terminates.

We are back to the initial call to `MineChains` and consider $p =$ `:director`. Again, the minimal support requirement is fulfilled and we make a recursive call with $S = POS[\texttt{:director}]$ and $current =$ `:director`. In the recursive call $current \neq r$ and we compute the limited s-support: $|S[\texttt{:Miyazaki}] \cup S[\texttt{:Yonebayashi}]| = 2 \geq minsup$. Both conditions are satisfied and the recursive call returns a set containing a single frequent property chain: {`:director`}.

We are again back to the initial call to `MineChains` and consider $p =$ `:studio`. A recursive call is made with $current =$ `:studio` and the condition $current \neq r$ is satisfied, but the second part is not: $S \cap O = \emptyset$ and thus further computation is executed. As there are no triples with the subject `:Ghibli` in the sample graph, *POS* is empty and the call returns with $\emptyset$.

This concludes the loop in the initial call to `MineChains` and it returns $result =$ {`:director`}. The confidence for the axiom `:director` SubPropertyOf: `:participant` is now computed as $\frac{2}{2} \geq minconf$ and thus the axiom is returned to the user.

b: TRANSITIVITY

We now consider $r =$ `:next` and we will show that the algorithm is capable of finding the transitivity axioms of form $p \circ p$ SubPropertyOf: p. We begin again be preparing

$$S = \{\texttt{:SA} \mapsto \{\texttt{:SA}\}, \texttt{:Laputa} \mapsto \{\texttt{:Laputa}\}, \texttt{:Totoro} \mapsto \{\texttt{:Totoro}\}\}$$

and $O =$ {`:Totoro`, `:SA`, `:Arietty`}. We call `MineChains` and construct *POS* (for brevity, we omit remaining properties that will be dropped further on in the process):

$$POS[\texttt{:next}] = \{\texttt{:Arietty} \mapsto \{\texttt{:SA}, \texttt{:Laputa}, \texttt{:Totoro}\}, \texttt{:SA} \mapsto \{\texttt{:Laputa}, \texttt{:Totoro}\}, \texttt{:Totoro} \mapsto \{\texttt{:Laputa}\}\}$$

Now, a recursive call is made with $S = POS[\texttt{:next}]$ and $current =$ `:next`. The conditions in the beginning are not fulfilled and *POS* is computed again: $POS[\texttt{:next}] = \{\texttt{:Arietty} \mapsto \{\texttt{:Laputa}, \texttt{:Totoro}\}, \texttt{:SA} \mapsto \{\texttt{:Laputa}\}\}$. The s-support is computed $sup = |\{\texttt{:Laputa}, \texttt{:Totoro}\}| \geq minsup$ and another recursive call is made with $S = POS[\texttt{:next}]$ and $current =$ `:next` $\circ$ `:next`, but now the first condition is fulfilled: $|S[\texttt{:SA}] \cup S[\texttt{:Arietty}]| = |\{\texttt{:Laputa}, \texttt{:Totoro}\}| = 2 \geq minsup$. After returning from all the calls of `MineChains` $\mathfrak{p} =$ `:next` $\circ$ `:next`. The confidence of the axiom `:next` $\circ$ `:next` SubPropertyOf: `:next` is 1 and the axiom is returned to the user.

c: ARBITRARY PROPERTY CHAIN

Finally, we mine $r =$ `:studio`. We begin by constructing $S = \{\texttt{:Arietty} \mapsto \{\texttt{:Arietty}\}, \texttt{:SA} \mapsto \{\texttt{:SA}\}\}$

$$POS = \{\texttt{:director} \mapsto \{\texttt{:Yonebayashi} \mapsto \{\texttt{:Arietty}\}, \texttt{:Miyazaki} \mapsto \{\texttt{:SA}\}\}$$
$$\texttt{:participant} \mapsto \{\texttt{:Yonebayashi} \mapsto \{\texttt{:Arietty}\}, \texttt{:Miyazaki} \mapsto \{\texttt{:SA}\}\}$$
$$\texttt{:studio} \mapsto \{\texttt{:Ghibli} \mapsto \{\texttt{:Arietty}, \texttt{:SA}\}\}\}$$

Listing 3. A *POS* index constructed in the example described in paragraph II-C5.a.

$$POS = \{\texttt{:director} \mapsto \{\texttt{:Yonebayashi} \mapsto \{\texttt{:Arietty}\}, \texttt{:Miyazaki} \mapsto \{\texttt{:SA}\}\},$$
$$\texttt{:participant} \mapsto \{\texttt{:Yonebayashi} \mapsto \{\texttt{:Arietty}\}, \texttt{:Miyazaki} \mapsto \{\texttt{:SA}\}\},$$
$$\texttt{:studio} \mapsto \{\texttt{:Ghibli} \mapsto \{\texttt{:Arietty}, \texttt{:SA}\}\}\}$$

Listing 4. A *POS* index constructed in the example described in paragraph II-C5.c.

and $O = \{\texttt{:Ghibli}\}$. We then make an initial call to `MineChains` and compute *POS*, presented in Listing 4. We iterate over the properties in *POS* and make recursive calls to `MineChains`. Here, we consider only the one for `:director`, remarking that the call for `:participant` is identical, while the call for `:studio` leads nowhere and returns $\emptyset$. In the recursive call the limited s-support computed in the beginning is 0 and the maximal length is not reached, so *POS* is computed again:

$$POS = \{\texttt{:member} \mapsto \{\texttt{:Ghibli} \mapsto \{\texttt{:Arietty}, \texttt{:SA}\}\}\}$$

A recursive call is made with *current* = `:director` $\circ$ `:member`, but now the limited s-support is 2, which is enough to accept the the property chain as frequent. `MineChains` terminates returning two frequent property chains: `:director` $\circ$ `:member` and `:participant` $\circ$ `:member`. Each of them is transformed to an object subproperty axiom and the confidence is 1 in both cases.

6) SOUNDNESS, COMPLETENESS AND TERMINATION OF MINING FREQUENT PROPERTY CHAINS

From subsubsection II-C1 it directly follows that `MineChains` is sound, i.e., any property chain returned by the function is a frequent property chain. Unfortunately, the following does not necessarily hold for any property chain $\mathfrak{p}$ and any property p:

$$\sigma_{\mathbb{G}}(\mathfrak{p}) \geq \sigma_{\mathbb{G}}(\mathfrak{p} \circ p)$$

For a counterexample, consider the following RDF graph $\mathbb{G} = \{(a, p, b), (b, r, c_1), (b, r, c_2)\}$. Now $\mathbb{S}_{\mathbb{G}}(p) = \{(a, b)\}$ and thus $\sigma_{\mathbb{G}}(p) = 1$, while $\mathbb{S}_{\mathbb{G}}(p \circ r) = \{(a, c_1), (a, c_2)\}$ and thus $\sigma_{\mathbb{G}}(p \circ r) = 2$. However, in both cases, $\sigma^s_{\mathbb{G}}(p) = \sigma^s_{\mathbb{G}}(p \circ r) = |\{a\}| = 1$.

Due to this `MineChains` is heuristic: it is sound, but not complete, i.e., there may be frequent property chains of arbitrary length, that will not be returned by the algorithm. We also observe that a complete algorithm would be, necessarily, non-terminating. Consider the following graph: $\mathbb{G} = \{(a, p, a)\}$. Assuming *minsup* = 1, we observe the following frequent property chains: p, $p \circ p$, $p \circ p \circ p$ etc. To avoid this problem, we require that all the mined frequent property chains must be no longer than *maxlen* properties. This ensures termination, at a cost of omitting all the property chains longer than *maxlen*.

III. RESULTS

To validate whether the proposed method is useful, we posed the following research question: *Are the axioms proposed by the method true, or are they rather statistical coincidences?* To answer the question, we conducted a computational experiment described below.

A. DATASET

In the evaluation, we used DBpedia, an effect of knowledge extraction from Wikipedia [17]. DBpedia consists of two main parts: the DBpedia ontology, containing the terminological knowledge, and the DBpedia datasets, containing the actual results of extraction. We used *Mappingbased Objects*, a subset consisting of RDF triples using only object properties from the DBpedia ontology, containing 18, 111, 905 triples.

The process of construction of this subset requires detailed description, as we use artifacts used in the process to discuss the results of the experiment. To describe it, we must first introduce Wikipedia infoboxes, that is tables of attribute-value pairs present on most of the Wikipedia pages, typically displayed in the top right corner of the page. Each infobox follows a template, that specifies a list of allowed attributes in the infobox. There is a plethora of templates (e.g., a separate template for a Swiss town `Infobox_swiss_town` and for a German town `Infobox_town_de`), each defining its own set of attributes, not necessarily consistently with other templates (e.g., one uses `birthplace`, while another `placeofbirth`). On top of that, editors of Wikipedia not necessarily follow the guidelines for using the template, making the infoboxes as a whole quite a noisy environment for knowledge extraction. The effect of a simple transformation: an attribute of an infobox to the property in a triple and its value to the object in the triple is in *Infobox Properties* subset and uses properties from the `http://dbpedia.org/property/` namespace.

In order to obtain more consistent and cleaner results, an additional step is executed. A set of mappings from infoboxes and their attributes to DBpedia ontology classes and properties is used. The mappings are edited

collaboratively on DBpedia Mappings Wiki.[1] These mappings form a documentation about the intended usage of properties, which is not necessarily reflected in the ontology. For example, in the ontology there is the property `dbo:intercommunality`, with unspecified domain and the range of `dbo:PopulatedPlace`. It is only through analysis of the mappings, that one can discover that this property is used exclusively for `Infobox_French_commune`, and that its intended domain is, in fact, `dbo:Settlement` and `dbo:country` hasValue `dbr:France`. The result of application of the mappings to *Infobox Properties* is the *Mappingbased Objects* subset. DBpedia Mappings Wiki is currently relatively stable with apparently no edits in the mappings for the English Wikipedia infoboxes since at least 28 September 2018 [23]. Due to these properties, we use Mappings Wiki as a documentation for the ontology, to verify whether the mined axioms are meaningful and justified.

B. SETUP

We ran the proposed method on DBpedia 2016-10 with the following parameters: $minsup = 100$, $minconf = 0.95$, $maxlen = 2$. We removed the properties with the following words in their labels (`rdfs:label`): *spouse, successor, predecessor, previous, following, subsequent*, as we observed that such properties, while yielding a substantial amount of patterns, do not lead to axioms worth considering. The full implementation used during the experiment is available at https://github.com/jpotoniec/PropertyChains. The results of the experiment are presented in Table 1.

IV. DISCUSSION

First, we checked whether any of the presented axioms logically follow from the DBpedia ontology. This is the case only for axiom 7, `dbo:owningOrganisation` SubPropertyOf: `dbo:owner`.

A. AXIOMS WITH A SINGLE PROPERTY

We observe that axioms that have a single property on the left-hand side can be discussed and verified using the DBpedia Mappings Wiki. For each such a property we searched the mappings for it and analyzed from what infobox attributes and templates it could be derived. We used an RML representation of the mappings, available at https://github.com/dbpedia/mappings-tracker. The mappings validated an axiom if all the mappings yielding a triple with the subproperty of the axiom also yielded a triple with the superproperty of the axiom. In 18 out of 19 cases we were able to validate the axiom and we report the details in Table 2, with the sole exception of axiom 28.

The mappings for the infobox template `Infobox_football_club_season` confirm axiom 28, whereas the mappings for the infobox template `Infobox_CFL_team` do not. This hints on a usage inconsistency, possibly due to the differences between American and British English, as the later infobox template is intended for American football teams. Nevertheless, we conclude that the axiom is incorrect.

There are also 9 counterexamples to axiom 24. Searching for these 9 pairs in DBpedia Live [24] shows that this is no longer the case: except for `dbr:TolumiDE`, which got removed, in the remaining 8 cases both properties occur. This hints on extraction problems in DBpedia 2016-10, which were later solved.

We remark here that, while the results are consistent with DBpedia Mappings Wiki, they are not necessarily correct from the ontological point of view. For example, source of a river (`dbo:sourcePlace`) is not necessarily a mountain (`dbo:sourceMountain`). Nevertheless, the proposed axioms are consistent with intended usage, as defined by DBpedia Mappings Wiki, and with the actual dataset.

B. AXIOMS WITH PROPERTY CHAINS

There is not much to say about axioms 14, 18 and 22: they have 100% confidence and follow common sense. Axioms 19 and 20, on the other hand, look very suspicious and they both transpire from a confusion of `dbr:Azores` (an archipelago in Portugal) and `dbr:Atlantic/Azores`, a time zone of the Azores. Even though their confidence is 100%, they must be rejected as caused by a mistake in the data. Axiom 21 is correct, but very narrow: `dbo:intercommunality` can be extracted only from the infobox template `Infobox_French_commune` (infobox property: `intercommunality`), so one can immediately suspect that the country in question is France, and the data fully confirms.

Starting from axiom 25, we must also discuss the counterexamples that were observed during mining, as these axioms have confidence below 100%. For axiom 25, `dbo:metropolitanBorough` can be extracted only from the infobox template `Infobox_UK_place`, hinting that the country in question must be the Unitied Kingdom. However, there are six boroughs that do not fit the pattern: `dbr:Aston,_South_Yorkshire` and `dbr:Swallownest`, both located in the metropolitan borough of Rotherham (denoted in both cases by `dbr:Rotherham`, not by `dbr:Metropolitan_Borough_of_Rotherham`), which has two objects for the property `dbo:country`: `dbr:United_Kingdom`, which is expected, and the unexpected `dbr:England_football_team`. The remaining four counterexamples are `dbr:Hare_Hatch`, `dbr:Holme_Green`, `dbr:Gardeners_Green`, `dbr:Upper_Culham`, all four located in `dbr:Wokingham`. The single object for the `dbo:country` property for them is `dbo:United_Kingdom`, whereas there are two such objects for `dbr:Wokingham`: `dbo:United_Kingdom` and `dbo:England`. This does not pose any contradiction to the proposed axiom, but merely shows the incompleteness of the data.

[1] http://mappings.dbpedia.org/

TABLE 1. The axioms mined using the proposed algorithm and the setup described in subsection III-B. The column *conf.* contains the confidence of the axioms, while *CE* reports the number of pairs in the considered graph that do not follow the axiom, but follow its left-hand side. The summary of the discussion in the main body of text is in the column *correct*.

#	axiom	s-support	conf.	CE	correct
1	dbo:associatedMusicalArtist SubPropertyOf: dbo:associatedBand	131338	100%	0	✓
2	dbo:musicalBand SubPropertyOf: dbo:musicalArtist	56270	100%	0	✓
3	dbo:musicalArtist SubPropertyOf: dbo:musicalBand	56270	100%	0	✓
4	dbo:stateOfOrigin SubPropertyOf: dbo:nationality	38461	100%	0	✓
5	dbo:mouthPlace SubPropertyOf: dbo:mouthMountain	15489	100%	0	✓
6	dbo:mouthMountain SubPropertyOf: dbo:mouthPlace	15489	100%	0	✓
7	dbo:owningOrganisation SubPropertyOf: dbo:owner	13186	100%	0	✓
8	dbo:sourcePlace SubPropertyOf: dbo:sourceMountain	6147	100%	0	✓
9	dbo:sourceMountain SubPropertyOf: dbo:sourcePlace	6147	100%	0	✓
10	dbo:distributingCompany SubPropertyOf: dbo:distributingLabel	1796	100%	0	✓
11	dbo:distributingLabel SubPropertyOf: dbo:distributingCompany	1796	100%	0	✓
12	dbo:designCompany SubPropertyOf: dbo:designer	1528	100%	0	✓
13	dbo:officialLanguage SubPropertyOf: dbo:language	561	100%	0	✓
14	dbo:associationOfLocalGovernment ∘ dbo:country SubPropertyOf: dbo:country	412	100%	0	✓
15	dbo:legalForm SubPropertyOf: dbo:type	358	100%	0	✓
16	dbo:sourceConfluencePlace SubPropertyOf: dbo:sourceConfluenceMountain	205	100%	0	✓
17	dbo:sourceConfluenceMountain SubPropertyOf: dbo:sourceConfluencePlace	205	100%	0	✓
18	dbo:administrativeCollectivity ∘ dbo:country SubPropertyOf: dbo:country	160	100%	0	✓
19	dbo:timeZone ∘ dbo:state SubPropertyOf: dbo:timeZone	159	100%	0	✗
20	dbo:daylightSavingTimeZone ∘ dbo:state SubPropertyOf: dbo:daylightSavingTimeZone	158	100%	0	✗
21	dbo:intercommunality ∘ dbo:country SubPropertyOf: dbo:country	154	100%	0	✓
22	dbo:administrativeDistrict ∘ dbo:country SubPropertyOf: dbo:country	142	100%	0	✓
23	dbo:lowestMountain SubPropertyOf: dbo:lowestPlace	133	100%	0	✓
24	dbo:associatedBand SubPropertyOf: dbo:associatedMusicalArtist	131338	100%	9	✓
25	dbo:metropolitanBorough ∘ dbo:country SubPropertyOf: dbo:country	784	99%	6	✓
26	dbo:unitaryAuthority ∘ dbo:country SubPropertyOf: dbo:country	111	99%	1	✓
27	dbo:isPartOf ∘ dbo:state SubPropertyOf: dbo:isPartOf	56987	99%	540	✗
28	dbo:homeStadium SubPropertyOf: dbo:ground	7318	99%	85	✗
29	dbo:countySeat ∘ dbo:state SubPropertyOf: dbo:state	149	99%	2	✓
30	dbo:countySeat ∘ dbo:country SubPropertyOf: dbo:country	2584	98%	40	✓
31	dbo:locatedInArea ∘ dbo:locatedInArea SubPropertyOf: dbo:locatedInArea	1139	98%	29	✓
32	dbo:largestCity ∘ dbo:state SubPropertyOf: dbo:state	150	97%	4	✓
33	dbo:frazioni ∘ dbo:country SubPropertyOf: dbo:country	116	97%	4	✓
34	dbo:starring ∘ dbo:network SubPropertyOf: dbo:network	325	96%	15	✓
35	dbo:leaderParty ∘ dbo:country SubPropertyOf: dbo:country	2180	95%	106	✗
36	dbo:mayor ∘ dbo:nationality SubPropertyOf: dbo:country	100	95%	5	✗

There is a single counterexample for axiom 26: dbr:Inchgarvie, with a very similar problem as previously, reflected in the following three triples:

```
dbr:Inchgarvie dbo:country dbr:Scotland;
     dbo:unitaryAuthority dbr:Edinburgh.
dbr:Edinburgh dbo:country
     dbr:United_Kingdom.
```

Again, no contradiction here, only data incompleteness.

For axiom 27 the number of counterexamples is quite large. However, if we assume transitivity of dbo:isPartOf, e.g., using the star operator in a SPARQL property path, as in the following query, the number of counterexamples decreases to only 13, hinting that the problem is mostly due to the data incompleteness.

```
SELECT DISTINCT ?s ?o
WHERE {
    ?s dbo:isPartOf ?x.
    ?x dbo:state ?o.
    FILTER NOT EXISTS
    {?s dbo:isPartOf* ?o.}
}
```

However, in these 13, one can find real counterexamples. For example, the dbr:Herring_Bay is a bay in the state of Maryland in the United States and a part of dbr:Chesapeake_Bay, which is located in the state of Maryland and in the state of Virginia. However, the dbr:Herring_Bay is not in the state of Maryland. This hints that the axiom is, in fact, incorrect, as there may be entities that span across multiple states, yet some of their parts lie strictly within a single state.

Axiom 29 has two counterexamples in DBpedia: dbr:Albany_County,_New_York and dbr:Rensselaer_County,_New_York, both located in dbr:New_York_(state) instead of dbr:New_York. These are due to the UNA rather than to any real problem with the axiom.

There are 40 counterexamples for axiom 30. Of these, 33 are due to the UNA: dbr:America, dbr:United_States_of_America and dbr:United_States denote the same entity, but they are not recognized as such. Of the remaining 7 counterexamples, 4 are due to non-unique names, e.g., dbr:Athens_County,_Ohio with county seat in dbr:Athens instead of dbr:Athens,_Ohio; similarly for dbr:Isle_of_Wight_County,

TABLE 2. The axioms that could be justified using DBpedia Mappings Wiki, along with the respective infobox properties and templates.

#	Axiom	Infobox properties	Infobox templates
1, 24	dbo:associatedMusicalArtist EQUIVALENTTO: dbo:associatedBand	Associated_acts, associated_acts	Infobox_musical_artist
2, 3	dbo:musicalBand EQUIVALENTTO: dbo:musicalArtist	Artist	Infobox_single
4	dbo:stateOfOrigin SUBPROPERTYOF: dbo:nationality	nationality	Infobox_person, Infobox_sportsperson
5, 6	dbo:mouthPlace EQUIVALENTTO: dbo:mouthMountain	mouth_location	Geobox, Infobox_river
8, 9	dbo:sourcePlace EQUIVALENTTO: dbo:sourceMountain	source_location	Geobox, Infobox_river
		source1_location	Geobox
10, 11	dbo:distributingCompany EQUIVALENTTO: dbo:distributingLabel	distributor	Infobox_record_label
12	dbo:designCompany SUBPROPERTYOF: dbo:designer	designer	Infobox_Automobile_generation, Infobox_automobile, Infobox_North_American_Automobile
13	dbo:officialLanguage SUBPROPERTYOF: dbo:language	official_languages	Infobox_country, Infobox_Geopolitical_organization, Infobox_former_country, Infobox_former_subdivision
15	dbo:legalForm SUBPROPERTYOF: dbo:type	company_type	Infobox_law_firm
16, 17	dbo:sourceConfluencePlace EQUIVALENTTO: dbo:sourceConfluenceMountain	source_confluence_location	Geobox
23	dbo:lowestMountain SUBPROPERTYOF: dbo:lowestPlace	lowest_location	Geobox

_Virginia, dbr:Medina_County,_Ohio and dbr:Somerset, _County_Pennsylvania. The remaining 3 are: dbr:Marathon_County,_Wisconsin, dbr:Wilkes_County,_North_Carolina and Wasco_County,_Oregon. For the county seats of these states, the dbo:country property links to the county itself instead to the country where the state lies. This is an irregular usage of the property and does not contradict the axiom.

Axiom 31 is a transitivity axiom for the property dbo:locatedInArea and there are 29 counterexamples for it. However, these are only due to the incompleteness in the data. If we instead consider the transitive closure of the property, e.g., by using the star operator of SPARQL, we obtain no counterexamples whatsoever, as in the following SPARQL query:

```
SELECT DISTINCT ?s ?o
WHERE {
    ?s dbo:locatedInArea ?x.
    ?x dbo:locatedInArea ?o.
    FILTER NOT EXISTS
    {?s dbo:locatedInArea* ?o}
}
```

As for 4 counterexamples for axiom 32, three of them are due to the confusion between dbr:New_York and dbr:New_York_(state). The fourth is a non-typical usage:

```
dbr:Delaware dbo:largestCity
     dbr:Wilmington,_Delaware.
dbr:Wilmington,_Delaware dbo:state
     dbr:Delaware.
```

However, the triple dbr:Delaware dbo:state dbr:Delaware is not present (nor is any other triple with dbo:state for dbr:Delaware).

The counterexamples for axiom 33 are apparently due to the incorrect values for dbo:frazioni, due to the lack of appropriate page in Wikipedia and corresponding lack of resource in DBpedia (e.g., dbr:Castiglione_di_Garfagnana) or due to the name confusion (e.g. dbr:Triei). We also remark that *frazioni* is an Italian administrative structure, and thus, by definition this axiom is correct, but somewhat limited, similarly to axiom 21.

Consider axiom 34. According to the DBpedia ontology, the property dbo:starring is equivalent to http://schema.org/actors, which expects a movie, radio series etc. on the left-hand side and a person on the right-hand side, and the DBpedia generally follows suit. However, in all 15 counterexamples for this axiom, this is not the case and the right-hand side of dbo:starring is not a person, but a media content of sorts, e.g. dbr:Poor_Paul. From this, we conclude that the axiom is correct.

Axiom 35 is a political rather than ontological topic and we abstain from discussing the counterexamples here, counting the axiom as an incorrect one.

Finally, axiom 36 assumes that country and nationality are denoted by the same resource. This is generally, but not necessarily, true in DBpedia, e.g., there exists dbr:Belgian_people. We thus conclude that the axiom, however well supported by the statistics, is incorrect.

C. FURTHER EVALUATION

The considered problem may be viewed in terms of information retrieval measures, based on a confusion matrix consisting of four parts: true positives *TP*, false positives *FP*, true negatives *TN*, false negatives *FN*. Each mined axiom is either correct or incorrect, which immediately translates to, respectively, a *true positive* or a *false positive*. From Table 1 we can compute the number of true positives $TP = 30$ and the

TABLE 3. The axioms mined using the proposed algorithm on the Lehigh University Benchmark. The column *conf.* contains the confidence of the axioms, while *CE* reports the number of pairs in the considered graph that do not follow the axiom, but follow its left-hand side. The prefix `univ:` corresponds to the namespace `http://swat.cse.lehigh.edu/onto/univ-bench.owl#`.

#	axiom	s-support	conf.	CE
1	`univ:advisor` ∘ `univ:worksFor` SUBPROPERTYOF: `univ:memberOf`	3101	100%	0
2	`univ:advisor` ∘ `univ:headOf` SUBPROPERTYOF: `univ:memberOf`	127	100%	0
3	`univ:headOf` SUBPROPERTYOF: `univ:worksFor`	15	100%	0

number of false positives $FP = 6$ and from this the *precision*:

$$\text{precision} = \frac{TP}{TP + FP} = \frac{30}{36} = 83\%$$

On the other hand, computing *TN* and *FN* is impossible, as it requires having an ontology complete w.r.t. all subproperty axioms which could be relevant. In particular, the most interesting piece of information, namely the set of false negatives, i.e., axioms that should have been generated, but were not, is not available and would require a collective effort of a group of experts to complete the ontology first in that regard.

D. COMPUTATIONAL COMPLEXITY

Consider a graph using m different predicates, each used in n triples. In `MineChains`, when constructing the index *POS* each triple is visited at most once, yielding the worst-case complexity of $n \cdot m$. Then, for each predicate it is possible to make a recursive call to `MineChains`, yielding m recursive calls in the worst-case. This cost can be formalized as the following function $c(i)$, where l is the maximal recursion depth:

$$c(l) = \begin{cases} 1 & l = 0 \\ m(n + c(l-1)) & l > 0 \end{cases}$$

This definition can be simplified to $c(l) = n \sum_{i=1}^{l} m^i + m^{l+1}$, which yields the worst-case complexity of $O(c(maxlen)) = n \cdot m^{maxlen+1}$.

E. SUMMARY

We presented an approach to mine OWL object subproperty axioms from usage of properties in an RDF graph. The mined axioms can be used by an ontology engineer to extend an ontology. To the best of our knowledge, this is the first algorithm capable of mining object subproperty axioms with property chains in their left-hands sides. We discussed the theoretical properties of the algorithm: its soundness and termination, and we explained why such an algorithm cannot be complete. We evaluated the algorithm using the newest DBpedia available, i.e., DBpedia 2016-10 and extensively discussed the obtained results. We argued that from the 36 mined candidate axioms, 30 (83%) is correct and could be added to the DBpedia ontology.

This paper considered only a static ontology and a static RDF graph. Further work could include the dynamic aspect where the graph and the ontology are modified in time and one should detect axioms that become plausible to suggest their addition to the ontology engineer, and conversely, detect axioms that become implausible to suggest their removal.

APPENDIX
EVALUATION USING THE LEHIGH UNIVERSITY BENCHMARK

To check whether the proposed approach is extendable to other datasets, we performed the following experiment. We used the data generator *UBA 1.7* from the Lehigh University Benchmark [25] and executed it with the parameters `-index 0 -seed 0`, as suggested on its website http://swat.cse.lehigh.edu/projects/lubm/. This yielded an RDF graph distributed over 15 files, from which we extracted all triples such that their object is an individual or a class, obtaining a subgraph of 67,494 triples. On the subgraph, we executed the proposed algorithm, setting $minsup = 10$, $minconf = 0.8$, $maxlen = 10$. The axioms obtained are reported in Table 3.

We verified the first axiom with the source code of the data generator and it indeed follows the behaviour of the generator, which always selects an advisor for a student from within the department being generated, as per methods `_getId` and `_selectAdvisor` of the class `Generator`. The second axiom is also correct, as according to the ontology accompanying the generator, `univ:headOf` is a subproperty of `univ:worksFor`. Finally, the third axiom is asserted in the ontology, and thus correct as well.

We observe that our approach helped to uncover a behaviour that is implemented in the generator, but it is not documented in the ontology.

REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Sci. Amer.*, vol. 284, no. 5, pp. 28–37, 2001.

[2] A. Maedche and S. Staab, "Ontology learning for the semantic web," *IEEE Intell. Syst.*, vol. 16, no. 2, pp. 72–79, Mar. 2001, doi: 10.1109/5254.920602.

[3] J. Lehmann and J. Völker, *Perspectives on Ontology Learning* (Studies on the Semantic Web), vol. 18. Amsterdam, The Netherlands: IOS Press, 2014, doi: 10.3233/978-1-61499-379-7-i.

[4] J. Potoniec, P. Jakubowski, and A. Ławrynowicz, "Swift linked data miner: Mining OWL 2 EL class expressions directly from online RDF datasets," *J. Web Semantics*, vols. 46–47, pp. 31–50, Oct. 2017, doi: 10.1016/j.websem.2017.08.001.

[5] J. Völker and M. Niepert, "Statistical schema induction," in *Proc. 8th Extended Semantic Web Conf.* in Lecture Notes in Computer Science, vol. 6643, G. Antoniou, M. Grobelnik, E. P. B. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Z. Pan, Eds. Heraklion, Crete, Greece: Springer, Jun. 2011, pp. 124–138, doi: 10.1007/978-3-642-21034-1_9.

[6] J. Völker, D. Fleischhacker, and H. Stuckenschmidt, "Automatic acquisition of class disjointness," *J. Web Semantics*, vol. 35, pp. 124–139, Dec. 2015, doi: 10.1016/j.websem.2015.07.001.

[7] H. Li and Q. Sima, "Parallel mining of OWL 2 EL ontology from large linked datasets," *Knowl.-Based Syst.*, vol. 84, pp. 10–17, Aug. 2015, doi: 10.1016/j.knosys.2015.03.023.

[8] S. Hellmann, J. Lehmann, and S. Auer, "Learning of OWL class descriptions on very large knowledge bases," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 2, pp. 25–48, Apr. 2009, doi: 10.4018/jswis.2009040102.

[9] J. Potoniec, "Mining cardinality restrictions in OWL," *Found. Comput. Decis. Sci.*, vol. 45, no. 3, pp. 195–216, Sep. 2020. [Online]. Available: https://content.sciendo.com/view/journals/fcds/45/3/article-p195.xml
[10] A. Melo, J. Völker, and H. Paulheim, "Type prediction in noisy RDF knowledge bases using hierarchical multilabel classification with graph and latent features," *Int. J. Artif. Intell. Tools*, vol. 26, no. 2, pp. 1–32, 2017, doi: 10.1142/S0218213017600119.
[11] B. Ell, S. Hakimov, and P. Cimiano, "Statistical induction of coupled domain/range restrictions from RDF knowledge bases," in *Knowledge Graphs and Language Technology* (Lecture Notes in Computer Science), vol. 10579, M. van Erp, S. Hellmann, J. P. McCrae, C. Chiarcos, K. Choi, J. Gracia, Y. Hayashi, S. Koide, P. N. Mendes, H. Paulheim, and H. Takeda, Eds. Kobe, Japan: Springer, 2016, pp. 27–40, doi: 10.1007/978-3-319-68723-0_3.
[12] R. Irny and P. S. Kumar, "Mining inverse and symmetric axioms in linked data," in *Semantic Technology* (Lecture Notes in Computer Science), vol. 10675, Z. Wang, A. Turhan, K. Wang, and X. Zhang, Eds. Cham, Switzerland: Springer, 2017, pp. 215–231, doi: 10.1007/978-3-319-70682-5_14.
[13] D. Fleischhacker, J. Völker, and H. Stuckenschmidt, "Mining RDF data for property axioms," in *On the Move to Meaningful Internet Systems: OTM* (Lecture Notes in Computer Science), vol. 7566, R. Meersman, H. Panetto, T. S. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, and I. F. Cruz, Eds. Rome, Italy: Springer, 2012, pp. 718–735, doi: 10.1007/978-3-642-33615-7_18.
[14] J. Potoniec, "Learning owl 2 property characteristics as an explanation for an RNN," *Bull. Polish Acad. Sci., Tech. Sci.*, vol. 68, no. 6, pp. 1481–1490, 2020.
[15] B. Parsia, S. Rudolph, P. Patel-Schneider, P. Hitzler, and M. Krötzsch, "OWL 2 web ontology language primer (second edition)," W3C, Cambridge, MA, USA, Tech. Rep., Dec. 2012. [Online]. Available: http://www.w3.org/TR/2012/REC-owl2-primer-20121211/
[16] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data—The story so far," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, Jul. 2009, doi: 10.4018/jswis.2009081901.
[17] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "DBpedia—A large-scale, multilingual knowledge base extracted from Wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015, doi: 10.3233/SW-140134.
[18] Y. Raimond and G. Schreiber, "RDF 1.1 primer," W3C, Cambridge, MA, USA, Tech. Rep., Jun. 2014. [Online]. Available: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/
[19] G. Carothers and E. Prud'hommeaux, "RDF 1.1 turtle," W3C, Cambridge, MA, USA, Tech. Rep., Feb. 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-turtle-20140225/
[20] S. Harris and A. Seaborne, *SPARQL 1.1 Query Language*. Cambridge, MA, USA: W3C Recommendation, Mar. 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/
[21] M. Horridge and P. Patel-Schneider, "OWL 2 web ontology language Manchester syntax (second edition)," W3C, Cambridge, MA, USA, Tech. Rep., Dec. 2012. [Online]. Available: http://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/
[22] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: SIAM, 2003, doi: 10.1137/1.9780898718003.
[23] *Dbpedia Mappings Wiki: Recent Changes*. Accessed: Nov. 15, 2021. [Online]. Available: http://mappings.dbpedia.org/index.php?namespace=2&invert=1&days=365&lim%it=5000&title=Special%3ARecentChanges
[24] M. Morsey, J. Lehmann, S. Auer, C. Stadler, and S. Hellmann, "DBpedia and the live extraction of structured data from Wikipedia," *Program*, vol. 46, no. 2, pp. 157–181, Apr. 2012, doi: 10.1108/00330331211221828.
[25] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *J. Web Semantics*, vol. 3, nos. 2–3, pp. 158–182, 2005, doi: 10.1016/j.websem.2005.06.005.

JEDRZEJ POTONIEC received the B.S., M.S., and Ph.D. degrees in computing from the Poznań University of Technology, Poznań, Poland, in 2011, 2012, and 2018, respectively.

From 2012 to 2018, he was a Teaching and Research Assistant with the Poznań University of Technology, where he has been an Adjunct, since 2018. He is the author of 22 peer-reviewed articles. His research interests include artificial intelligence with special focus on knowledge representation, machine learning, and their integration.

Dr. Potoniec is a member of the Polish Association of Artificial Intelligence (PSSI).