

Ontology Integration for Linked Data

Lihua Zhao · Ryutaro Ichise

Received: 11 May 2013 / Revised: 14 April 2014 / Accepted: 20 April 2014 / Published online: 25 May 2014
© The Author(s) 2014. This article is published with open access at Springerlink.com

Abstract The Linked Open Data cloud contains tremendous amounts of interlinked instances with abundant knowledge for retrieval. However, because the ontologies are large and heterogeneous, it is time-consuming to learn all the ontologies manually and it is difficult to learn the properties important for describing instances of a specific class. To construct an ontology that helps users to easily access various data sets, we propose a semi-automatic system, called the Framework for InTegrating Ontologies, that can reduce the heterogeneity of the ontologies and retrieve frequently used core properties for each class. The framework consists of three main components: graph-based ontology integration, machine-learning-based approach for finding the core ontology classes and properties, and integrated ontology constructor. By analyzing the instances of linked data sets, this framework constructs a high-quality integrated ontology, which is easily understandable and effective in knowledge acquisition from various data sets using simple SPARQL queries.

Keywords Semantic Web · Graph pattern · Ontology integration · Ontology alignment · Knowledge acquisition · Machine learning

L. Zhao (✉)
Computational Intelligence Laboratory, Toyota Technological
Institute, Nagoya, Japan
e-mail: lihua@toyota-ti.ac.jp

R. Ichise
National Institute of Informatics, The Graduate University
for Advanced Studies, Tokyo, Japan
e-mail: ichise@nii.ac.jp

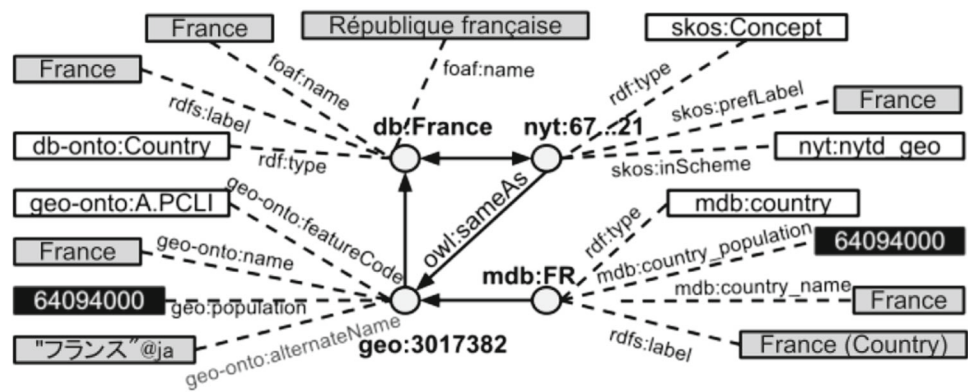
1 Introduction

The Linked Open Data (LOD) cloud, which has been growing rapidly over the past years, consists of 295 machine-readable data sets with over 31 billion Resource Description Framework (RDF) triples (as of Sept. 2011). The data sets in the LOD cloud are mainly categorized into seven domains: cross-domain, geographic, media, life sciences, government, user-generated content, and publications. Same instances in different data sets are interlinked with *owl:sameAs*, which is a built-in Web Ontology Language (OWL) property [4]. Currently, approximately 504 million *owl:sameAs* links are in the LOD cloud. Although some built-in properties such as *owl:equivalentClass* and *owl:equivalentProperty* are available for linking equivalent classes or properties, only a few of these kinds of links are in the LOD cloud [12]. Hence, it is difficult to understand the ontology alignments between different data sets.

OWL, which is a semantic markup language developed as a vocabulary extension of RDF, has more vocabularies for describing classes and properties [3]. RDF is a general-purpose language for representing information on the Web. RDF Schema, which is a semantic extension of RDF, provides mechanisms for describing groups of related resources and the relationships between these resources [5]. OWL 2 Web Ontology Language [34] provides the same classes and properties as in the older OWL 1 [3], but OWL 2 has richer data types, data ranges, disjoint properties, etc.

Our research mainly focuses on integrating ontologies from various data sets so that Semantic Web application developers can easily understand the ontologies and gain access to data sets. In this paper, we present our solution to resolve the following problems:

Fig. 1 Interlinked instances of “France”



1. Ontology Heterogeneity Problem

Data sets are published according to the Linked Data principles and also provide links to other data resources [4]. However, no standard ontology fits all data sets, and so the many kinds of ontologies can cause the ontology heterogeneity problem. The research in [10] categorized the ontology heterogeneity problem into four different types: syntactic heterogeneity, terminological heterogeneity, conceptual heterogeneity, and semiotic heterogeneity. We mainly focus on terminological and conceptual heterogeneity problems.

- The terminological heterogeneity problem occurs when the same entities in different ontologies are represented differently, e.g., `openingDate` vs. `establishedDate` and `shortDescription` vs. `abstract`.
- The conceptual heterogeneity, which is also called semantic heterogeneity in [9] and logical mismatch in [18], occurs due to the use of different axioms for defining concepts or due to the use of totally different concepts. For example, most airport instances are described with the type of `db-onto:Airport` that is a subclass of `db-onto:Infrastructure` (`db-onto:Infrastructure` is a subclass of `db-onto:ArchitecturalStructure`). However, some airports are described using `db-onto:-Building`, which is a subclass of `db-onto:ArchitecturalStructure`.

Figure 1 shows the interlinked instances of “France”. All the properties (labeled on the dotted line) that are connected to gray boxes (objects) represent the name of “France” and the properties that are connected to the black boxes represent the population. To access various data sets simultaneously, we have to understand their heterogeneous ontologies in advance to achieve semantic interoperability.

2. Difficulty in Identifying Core Ontology Entities

The instances of each class are described by part of the ontology properties. When the ontology is large, it is time-consuming to identify important properties used for describing instances of a specific class. Retrieving fre-

quently used core classes and properties from various data sets can help Semantic Web developers easily understand the ontology entities used for describing instances in each data. The core ontology entities help us to construct SPARQL queries and to discover missing information in the data sets. The core ontology entities consist of top-level classes and frequently used core properties.

- Top-level class: If the data set is ontology-based, top-level classes are all the direct subclasses of `owl:Thing`. Otherwise, we use the top categories as top-level classes. For example, `db-onto:Agent` and `db-onto:Place` are top-level classes in DBpedia, and `nyt:nytd_geo` and `nyt:nytd_org` are top-level classes in NYTimes.
- Frequent core property: The frequently used properties describing instances in the data sets are considered as frequent core properties. For example, the properties `db-onto:kingdom`, `db-onto:-class`, and `db-onto:family` are frequently used to describe instances defined with the class of `db-onto:Species`.

3. Missing Domain or Range Information

The relations between the ontology classes and properties are described with the property `rdfs:domain`, which indicates that the properties are designed to be used for the instances of a specific class. Furthermore, the range information of the values can help users better understand the data sets. However, in real data sets, many ontologies have missing domain or range information. In addition to the domain and range information, we should retrieve the description of each ontology class and property to construct an easily understandable integrated ontology.

To solve the above problems, we introduce the Framework for Integrating Ontologies (FITON), which decreases the ontology heterogeneity in the linked data sets, retrieves core ontology entities, and automatically enriches the integrated ontology by adding the domain, range, and annotations. FITON applies the different techniques listed below:

1. Ontology Similarity Matching on the SameAs Graph Patterns

Ontology integration is defined as the process that generates a single ontology from different existing ontologies [6]. However, having only few links at the class or property level makes it difficult to directly retrieve equivalent classes and properties for ontology integration. Ontology alignment, or ontology matching is commonly used to find correspondences between ontologies to solve the ontology heterogeneity problem [29]. We combined string-based and WordNet-based ontology matching methods on the predicates and objects to discover similar concepts.

Since the same instances are linked by *owl:sameAs*, we can create undirected graphs with the linked instances and analyze the graphs to retrieve related classes and properties. By analyzing the graph patterns, we can observe how the same concepts are represented differently in various data sets. Moreover, to reduce the ontology heterogeneity problem, we perform different similarity matching methods on the SameAs graph patterns to integrate the various ontologies.

2. Machine Learning for Core Ontology Entity Extraction
Machine learning methods such as association rule learning and rule-based classification can be applied to discover core properties for describing instances in a specific class. Apriori is a well-known algorithm for learning association rules in a big database [1], while the rule-based learning method, Decision Table, can retrieve a subset of properties that leads to high prediction accuracy with cross-validation [19]. By applying machine learning approaches to linked data sets, we can retrieve core ontology entities that are important for describing instances in the data sets.

3. Automatic Ontology Enrichment

The domain and the range information in the ontologies are critical for users to understand the relations between ontology entities. However, much missing domain and range information exists in the published LOD cloud. Hence, we propose the integrated ontology constructor that automatically enriches the integrated ontology by adding the missing domains and ranges, as well as annotations.

We randomly select some samples of the instances described by triples containing the retrieved classes and properties. From the contents of the sample instances, we automatically retrieve the domain information to link the properties and classes. We also retrieve the default range information of the properties and analyze the values of properties from the sample instances, which are mainly categorized into String and Resource. By analyzing these sample instances, we can reduce the analysis time and also retrieve the domain or range information.

The default annotations are added to make the integrated ontology easily understandable.

In this paper, we propose FITON, which adds the additional automatic ontology enrichment component as an extension of the research presented in [38]. The remainder of this paper is organized as follows. In Sect. 2, we discuss some related work and limitations of the methods. In Sect. 3, we introduce FITON, which contains three main components. Section 4 describes experiments with FITON, including assessing the performance with two machine learning methods, a comparison between FITON and other ontology matching tools, and an evaluation of the integrated ontology. In Sect. 5, we discuss possible applications using the graph patterns and the integrated ontology created with FITON. We conclude and propose our future work in Sect. 6.

2 Related Work

The authors in [20] introduced a closed frequent graph mining algorithm to extract frequent graph patterns from the Linked Data Cloud. Then, they extracted features from the entities of the extracted graph patterns to detect hidden *owl:sameAs* links or relations in geographic data sets such as the U.S. Census, Geonames, DBpedia, and World Factbook. They applied a supervised learning method on the frequent graph patterns to discover useful attributes for linking the same instances in various data sets. However, their approach only focused on the geographic domain and did not discuss the kinds of features important for finding the hidden links.

A debugging method for mapping lightweight ontologies is introduced in [25]. These authors applied machine learning technology to determine the disjointness of any pair of classes using the features of taxonomic overlap, semantic distance, object properties, label similarity, and WordNet similarity. Although their method performs better than other state-of-the-art ontology matching systems, the method is limited to expressive lightweight ontologies.

In [27], the authors focused on finding concept coverings between two sources by exploring the disjunctions of restriction classes. Their approach produces coverings where concepts at different levels in the ontologies can be mapped even if there is no direct equivalence. However, the ontology alignments are limited to two sources only.

The analysis of the basic properties of the SameAs network, the Pay-Level-Domain network, and the Class-Level Similarity network are discussed in [8]. Those researchers compared the five most frequent types to examine how data publishers are connected. However, considering only the types is not sufficient to detect related instances, which normally contain many data type properties.

The authors in [26] proposed constructing an intermediate layer ontology using an automatic alignment method on the linked data. However, the authors focused only on analyzing at the class level. If they had also considered the alignments at the property level, they could better understand how the instances are interlinked.

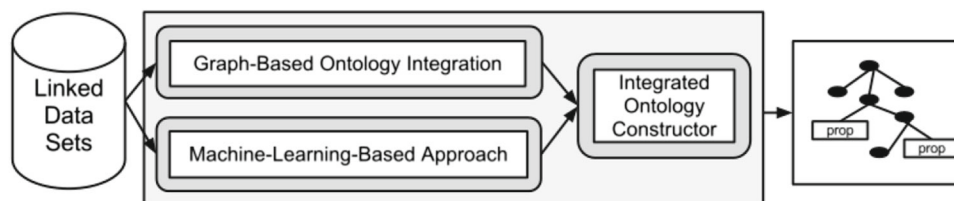
In contrast to the related research described above, FITON finds ontology alignments at both the class and the property levels. Furthermore, in each data set, we discover frequently used core properties and classes that can help data publishers detect misuses of the ontologies in the published data sets. FITON is domain-independent and successfully integrates heterogeneous ontologies by extracting related properties and classes that are critical for interlinking instances. In addition, for the instances of a specific class, we recommend core properties that are frequently used for instance description.

3 Ontology Integration Framework

Semantic Web developers often want to integrate data sets from various domains, but it is time-consuming to manually learn all the ontologies in different data sets. Moreover, large ontologies and heterogeneous ontologies make it difficult to manually map ontologies. Constructing a global ontology by integrating heterogeneous ontologies of linked data can help effectively integrate various data resources. We can decrease the ontology heterogeneity problem by retrieving related classes and properties from the interlinked instances. In addition, we also need the top-level classes and frequent core properties in each data set, which can be extracted using machine learning methods. For instance, the Decision Table algorithm can retrieve a subset of properties that leads to high prediction accuracy with cross-validation and the Apriori algorithm can discover properties that occur frequently in the instances of the top-level classes.

In this section, we introduce the semi-automatic ontology integration framework FITON that finds alignments between different ontologies and extracts core ontology entities used to describe instances. In addition to the approach introduced in [38], we improved the integrated ontology by enriching it with annotations, and domain and range information that can help users easily understand the ontology. As shown in Fig. 2, the framework of FITON consists of three main components:

Fig. 2 Framework for InTegrating Ontologies (FITON)



graph-based ontology integration [37], machine learning-based approach [38], and an integrated ontology constructor that can automatically enrich the integrated ontology with useful information. In the following, we describe each component in detail.

3.1 Graph-Based Ontology Integration

The instances that are interlinked by *owl:sameAs* are used to construct graphs and we can apply ontology matching methods on the graphs to find the alignments for the related classes and properties. Figure 3 shows the architecture of the graph-based ontology integration component, which contains five main steps. In the following, we describe each step of the graph-based ontology integration framework introduced in [37]:

3.1.1 Graph Pattern Extraction

We collect all the instances that have the *owl:sameAs* (SameAs) links to construct graph patterns that can be analyzed for mapping different ontology classes and properties. In the following, we list definitions of the terms SameAs Triple, SameAs Instance, SameAs Graph and graph pattern as follows:

Definition 1 *SameAsTriple*. A SameAs Triple is an RDF triple that contains the *owl:sameAs* predicate.

Definition 2 *SameAsInstance*. A SameAs Instance is a tuple $SI = (U, T, L)$, where U is the URI of the instance that appears in a SameAs Triple $\langle U, owl:sameAs, X \rangle$ or $\langle X, owl:sameAs, U \rangle$, T is the number of the distinct SameAs Triples that contain U , and L is the label of the data set that includes the instance.

Definition 3 *SameAsGraph*. This is an undirected SameAs Graph $SG = (V, E, I)$, where V is a set of vertices that are the labels of data sets having the linked SameAs Instances, $E \subseteq V \times V$ is a set of sameAs edges, and I is a set of URIs of the interlinked SameAs Instances.

Here, we give an example of the SameAs Graph constructed with the interlinked instances of “France” shown in Fig. 1. The SameAs Graph $SG_{France} = (V, E, I)$, where $V =$

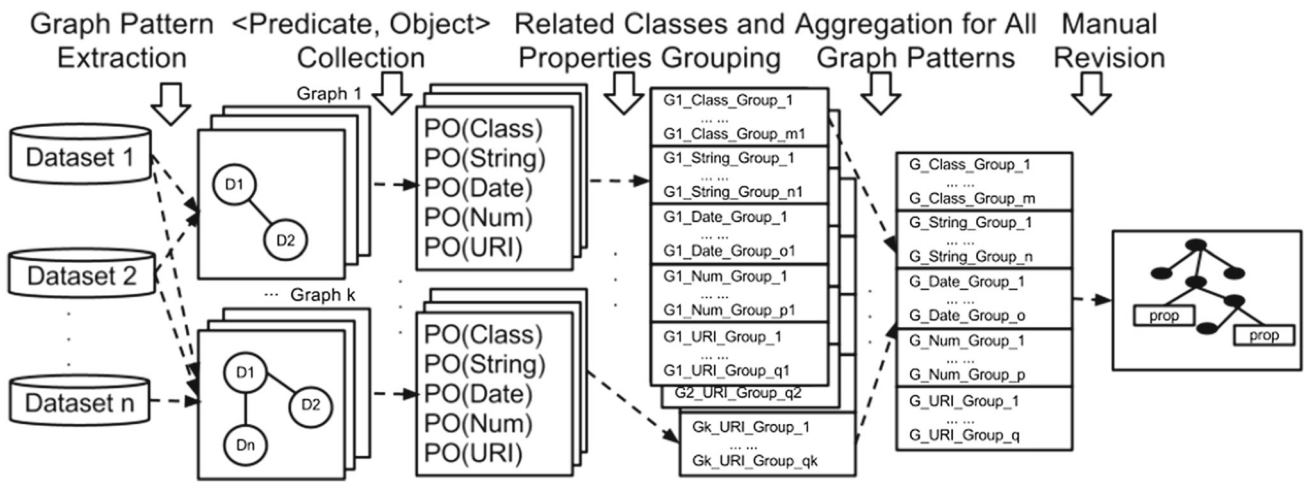


Fig. 3 Architecture of the graph-based ontology integration

Algorithm 1: SameAs Graphs extraction.

```

Input : IndexSI: A set of SameAs Instances
Output: SetSG: A set of SameAs Graphs
Variable: LinkedInst: Linked instances of SI
begin
    SetSG ← ∅
    for SI ∈ IndexSI do
        if SI.visited = false then
            SG ← ∅
            SI.visited ← true
            SG ← SearchGraph(SG, SI)
            SetSG.put(SG)
    return SetSG

```

```

SearchGraph(SG, SI)
begin
    SG.V.put(SI.L)
    SG.I.put(SI.U)
    LinkedInst ← ∅
    for X ∈ (< SI, owl:sameAs, X > ∪
    < X, owl:sameAs, SI >) do
        if X.visited = false then
            LinkedInst.put(X)
    for X ∈ LinkedInst do
        if X.visited = false then
            SG.E ← (SI, X)
            X.visited ← true
            SG ← SearchGraph(SG, X)
    return SG

```

{M, D, G, N}, $E = \{(D, G), (D, N), (G, M), (G, N)\}$, $I = \{mdb\text{-}country:FR,¹ db:France,² geo:3017382,³ nyt:67...21⁴\}$. M, D, G, and N represent the labels of data sets LinkedMDB, DBpedia, Geonames, and NYTimes, respectively.

To collect all the SameAs Graphs in the linked data sets using Algorithm 1, we extract all the SameAs Instances and rank them based on the value of T, which is the number of

distinct SameAs Triples. The ranked SameAs Instances are indexed in *IndexSI*, from which we extract a set of SameAs Graphs *SetSG* from the linked data sets.

In Algorithm 1, for each unvisited *SI* in *IndexSI*, we create an empty SameAs Graph *SG* and construct a SameAs Graph by using the function SearchGraph(*SG*, *SI*). We put *L* and *U* of *SI* into *SG*, and then search for the instances linked with *SI* and put them in *LinkedInst*. For each unvisited instance *X* in *LinkedInst*, we put the edge (*SI*, *X*) into the *SG*, and mark *X* as visited. Then we iteratively search with *SG* and *X* and assign the returned value to *SG* until all the instances in the *LinkedInst* are visited. The function SearchGraph(*SG*, *SI*) returns a SameAs Graph *SG* and all the SameAs Graphs constructed with the instances in the *IndexSI* are stored in *SetSG*.

Definition 4 *graphpattern*. Two SameAs Graphs SG_i and SG_j have the graph pattern (GP), if $SG_i.V = SG_j.V$ and $SG_i.E = SG_j.E$.

All the same SameAs Graphs form a graph pattern, from which we can detect related classes and properties.

3.1.2 <Predicate, Object> Collection

An instance can be represented as a collection of RDF triples in the form of <subject, predicate, object>, where the subject is the URI of an instance. Since a SameAs Graph contains linked instances, we collect all the Predicate, Object> pairs of the interlinked instances as the content of the SameAs Graph. Hereafter, *PO* represents <Predicate, Object>.

To avoid comparison between different types of objects, we classify the *PO* pairs into five different types: Class, String, Date, Number, and URI. The type of Class can be

¹ mdb-country: <http://data.linkedmdb.org/resource/country/>

² db: <http://dbpedia.org/resource/>

³ geo: <http://sws.geonames.org/>

⁴ nyt: <http://data.nytimes.com/>

Table 1 Type classification

Type	Built-in data types
String	http://www.w3.org/2001/XMLSchema#string http://www.w3.org/2001/XMLSchema#date
Date	http://www.w3.org/2001/XMLSchema#gYear http://www.w3.org/2001/XMLSchema#gMonthDay http://www.w3.org/2001/XMLSchema#integer
Number	http://www.w3.org/2001/XMLSchema#float http://www.w3.org/2001/XMLSchema\protect\LY1\textdollardouble http://www.w3.org/2001/XMLSchema#int
URI	http://www.w3.org/2001/XMLSchema#anyURI

identified from the predicates `rdf:type`⁵ and `skos:inScheme`⁶. The other four types of *PO* pairs can be identified from the object values with the built-in data types listed in Table 1. Usually the data types of objects are followed by the symbol “^^”. If the data types are not given expressively in the RDF triples, we analyze the object values in the following way:

- Number: The value consists of all numbers.
- URI: Starts with “http://”.
- String: All the other values that can not be classified.

Table 2 shows an example of the collected *PO* pairs of the interlinked instances shown in Fig. 1 and the types of *PO* pairs in SG_{France} . The first two columns list the *PO* pairs and the last column lists the types of the *PO* pairs.

3.1.3 Related Class and Property Grouping

To find related classes and properties for each graph pattern, we analyze the collected *PO* pairs of the SameAs Graphs. In the following, we describe how to discover related classes by checking the subsumption relations and how to find related properties by using ontology alignment methods.

1. Related Class Grouping

The ontology classes have subsumption relations such as `owl:subClassOf` and `skos:inScheme`. The two triples $\langle C_1, owl:subClassOf, C_2 \rangle$ and $\langle C_1, skos:inScheme, C_2 \rangle$ mean that the concept of C_1 is more specific than the concept of C_2 . To identify the types of linked instances, we focus on the most specific classes from the linked instances by tracking the subsumption relations. The classes and subsumption relations form a tree, and the most specific classes are called leaf nodes in the tree.

⁵ `rdf:` <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

⁶ `skos:` <http://www.w3.org/2004/02/skos/core#>

Table 2 *PO* pairs and types for SG_{France}

Predicate	Object	Type
<code>rdf:type</code>	<code>db-onto:Country</code>	Class
<code>rdfs:label</code>	“France”@en	String
<code>foaf:name</code>	“France”@en	String
<code>foaf:name</code>	“République française”@en	String
<code>db-onto:wikiPageExternalLink</code>	http://us.franceguide.com/	URI
<code>db-prop:populationEstimate</code>	65447374	Number
.....
<code>geo-onto:name</code>	France	String
<code>geo-onto:alternateName</code>	“France”@en	String
<code>geo-onto:featureCode</code>	<code>geo-onto:A.PCLI</code>	Class
<code>geo-onto:population</code>	64768389	Number
.....
<code>rdf:type</code>	<code>mdb:country</code>	Class
<code>mdb:country_name</code>	France	String
<code>mdb:country_population</code>	64094000	Number
<code>rdfs:label</code>	France (Country)	String
.....
<code>rdf:type</code>	<code>skos:Concept</code>	Class
<code>skos:inScheme</code>	<code>nyt:nytd_geo</code>	Class
<code>skos:prefLabel</code>	“France”@en	String
<code>nyt-prop:first_use</code>	2004-09-01	Date

A class which has no subsumption relation is considered as a leaf node.

From each SameAs Graph, we construct trees with the classes extracted from the *PO* pairs classified in the type Class. Then we group the leaf nodes, which represent the most specific class information of an instance. For each data set, we pre-define the properties of the class type and the subsumption relations. For example, we use `geo:featureCode` as the class type instead of `rdf:type`, and use `skos:inScheme` as the subsumption relation in Geonames. In NYTimes, we use `skos:inScheme` for the class type instead of `rdf:type` because it can categorize NYTimes data into four different types. Hence, `nyt:nytd_geo` is used as the class node instead of `skos:Concept`.

Figure 4 shows a collection of classes extracted from SG_{France} . The classes are connected with the subsumption relations `owl:subClassOf` and `skos:inScheme`. The gray nodes are `mdb:country`⁷, `db-onto:Country`⁸, `geo-onto:A.PCLI`⁹ and `nyt:nytd_geo`, which are the leaf nodes in Fig. 4. Therefore, we can group these four classes, that are used for describing countries in different data sets.

2. Related Property Grouping

⁷ `mdb:` <http://data.linkedmdb.org/resource/movie/>

⁸ `db-onto:` <http://dbpedia.org/ontology/>

⁹ `geo-onto:` <http://www.geonames.org/ontology/> #

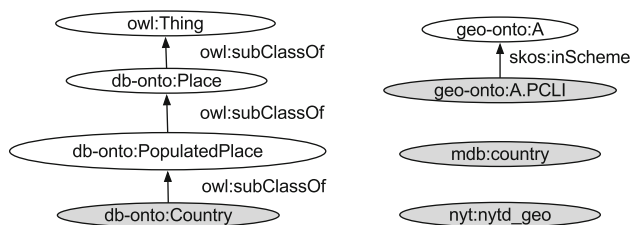


Fig. 4 Collected classes from SG_{France}

We perform exact and similarity matching methods on the collected PO pairs to find related properties, which are also used as predicates in the PO pairs. This is an extension of the similarity matching method introduced in [39].

(a) Exact Matching for Creating the Initial Sets of PO Pairs

The first step in the predicate grouping is to create the initial sets of PO pairs by the exact string matching method. For each classified type of PO pairs, we perform a pairwise comparison of PO_i and PO_j , and create the initial sets S_1, S_2, \dots, S_k by checking whether they have identical predicates or objects. Here, S is a set of PO pairs. For example, in Table 2, the predicates `rdfs:label`,¹⁰ `mdb:country_name`, `foaf:name`, `skos:prefLabel`, `geo-onto:name`, and `geo-onto:alternateName` have the same value “France”, and the predicate `foaf:name` has another object, “République française”@en. Hence, these six PO pairs are grouped together to create an initial set. After creating initial sets by exact matching, we create an initial set for each PO pair that has not yet been grouped.

(b) Similarity Matching on the Initial Sets of PO Pairs

The identical predicates of PO pairs that are classified into Date and URI can be discovered by exact matching. However, for the types of Number and String, the objects may be slightly different.

To find related initial sets, we apply similarity matching methods on the PO pairs of two initial sets and merge them if the similarity of any two PO pairs is higher than the predefined similarity threshold.

String-based and WordNet-based similarity matching methods are commonly used for matching ontologies at the concept level [10]. In our approach, we adopt three string-based similarity measures, namely, Jaro-Winkler distance [35], Levenshtein distance, and n-gram, as introduced in [14]. String-based similarity measures are applied to compare the objects of PO pairs that are classified in String. $ObjSim(PO_i, PO_j)$, which is the similarity of objects between two PO pairs, is calculated as

follows:

$$ObjSim(PO_i, PO_j) = \begin{cases} 1 - \frac{|O_{PO_i} - O_{PO_j}|}{O_{PO_i} + O_{PO_j}} & \text{if } O_{PO} \text{ is Number} \\ StrSim(O_{PO_i}, O_{PO_j}) & \text{if } O_{PO} \text{ is String} \end{cases}$$

where $StrSim(O_{PO_i}, O_{PO_j})$ is the average of the three string-based similarity values and the term O_{PO} indicates the object of PO .

The WordNet-based similarity matching method is required to group semantically similar predicates as discussed in [39]. WordNet::Similarity¹¹ provides nine similarity measures based on the lexical database WordNet [30]. Resnik [32], Lin [23], and Jiang and Conrath (JCN) [17] are based on the information content of the least common subsumer (LCS) of concepts, and Leacock and Chodorow (LCH) [21], Wu and Palmer (WUP) [36], and PATH are based on path lengths between a pair of concepts. The other three methods measure relatedness between concepts, which are Hirst and StOnge (HSO) [13], LESK [2], and VECTOR [28]. We adopt the same approach to calculate the similarity of predicates $PreSim(PO_i, PO_j)$ using the following formula:

$$PreSim(PO_i, PO_j) = WNSim(T_{PO_i}, T_{PO_j})$$

where T_{PO} indicates the pre-processed terms of the predicates in PO and $WNSim(T_{PO_i}, T_{PO_j})$ is the average of the nine applied WordNet-based similarity values. $Sim(PO_i, PO_j)$, which is the similarity between PO_i and PO_j , is calculated as follows:

$$Sim(PO_i, PO_j) = \frac{ObjSim(PO_i, PO_j) + PreSim(PO_i, PO_j)}{2}$$

If $Sim(PO_i, PO_j)$ is higher than the predefined similarity threshold, we consider that these two PO pairs are similar and merge the two sets S_m and S_n that contain PO_i and PO_j , respectively. In this work, we set the default similarity threshold to 0.5. After comparing all the pairwise initial sets, we remove the initial set S_i if it has not been merged during this process and has only one PO pair.

(c) Refine Sets of PO Pairs

The final step of the related property grouping is to split the predicates of each S_i according to the relation `rdfs:domain` [5]. Even though the objects or terms of the predicates are similar, the predicates may belong to different domains. For further refinement, we keep only

¹⁰ `rdfs:http://www.w3.org/2000/01/rdf-schema#`

¹¹ <http://wn-similarity.sourceforge.net/>

frequent pruned S_i that appears more than the predefined frequency threshold.

From the sets of PO pairs retrieved from each graph pattern, we collect the classes and properties. Then we construct integrated groups of classes and properties for the types Date, String, Number, and URI.

3.1.4 Aggregation of All Integrated Classes and Properties

In this step, we aggregate the integrated classes and properties from all the graph patterns to construct a preliminary integrated ontology according to the following rules:

1. Select a Term for Each Set

To perform automatic term selection, we pre-process all the terms of the classes and properties in each set by tokenization, stop word removal, and stemming. We keep the original terms because sometimes a single word is ambiguous for representing a set of terms. For example, “area” and “areaCode” have different meanings, but they may have the same frequency because the former is extracted from the latter. Hence, when two terms have the same frequency, we choose the longer one. The predicate `ex-onto:ClassTerm` is designed to represent a class, where “ClassTerm” is automatically selected and starts with a capitalized character. The predicate `ex-prop:propTerm` is designed to represent a property, where “propTerm” is automatically selected and starts with a lowercase character.

2. Construct Relations

We use the predicate `ex-prop:hasMemberClasses` to link the integrated classes with `ex-onto:ClassTerm`, and the predicate `ex-prop:hasMemberProperties` to link the integrated properties with `ex-prop:propTerm`. Here, we use the relation `ex-prop:hasMemberClasses` and `ex-prop:hasMemberProperties` instead of the existing `owl:equivalentClass` or `owl:equivalentProperty` to easily observe how each class and property are connected and how they are used for describing instances. Each class or property can belong to different groups and might be used in a different way. Furthermore, even if classes or properties are in the same group, it does not mean that they are equivalent. However, if we can guarantee they are equivalent, we can easily connect them with `owl:equivalentClass` or `owl:equivalentProperty`.

3. Construct Preliminary Integrated Ontology

A preliminary integrated ontology is automatically constructed with the integrated sets of related classes and properties, the selected terms `ClassTerm` and `propTerm`, and the `ex-prop:hasMemberClasses` and `ex-prop:hasMemberProperties`.

3.1.5 Manual Revision

The automatically constructed preliminary integrated ontology includes related classes and properties from different data sets. However, not all the terms of the classes and properties are properly selected, and some statements of `rdfs:domain` are missing. Hence, we need experts to revise the integrated ontology by choosing a proper term for each group of properties and by amending wrong groups of classes and properties. Since the integrated ontology is much smaller than the original ontologies, it is lightweight work.

3.2 Machine-Learning-Based Approach

Although, the graph-based ontology integration method can retrieve related classes and properties from different ontologies, this method may miss some core classes and frequently used properties that might be important for describing instances. Since the method relies too much on `SameAs` links, it cannot retrieve related classes or properties if no links or no other similar classes or properties exist. Therefore, we need another method to extract top-level classes and frequent core properties, which are essential for describing instances. Even though each instance is described by using one top-level class and subclasses of the top-level class that have more specific class information, not all the instances are described by specific class information and the number of instances per specific class is not balanced. Hence, we consider the top-level classes as part of the core ontology entities and perform machine learning methods.

By applying machine learning methods, we can find core properties that are frequently used to describe instances of a specific class. The Decision Table is a rule-based algorithm that can retrieve a subset of core properties, and the Apriori algorithm can find a set of associated properties that are frequently used for describing instances. Therefore, we apply the Decision Table and the Apriori algorithm to retrieve top-level classes and frequent core properties from the linked data sets.

To perform the machine learning methods, we randomly select a fixed number of instances for each top-level class from the data sets. For the data sets built based on an ontology, we track the subsumption relations to retrieve the top-level classes. For example, we track the `owl:subClassOf` subsumption relation to retrieve the top-level classes in DBpedia and track `skos:in-Scheme` in Geonames. However, some data sets use categories without any structured ontology. For this kind of data set, we use the categories as the top-level classes. As an example, NYTimes instances are only categorized into people, locations, organizations, and descriptors. We use this strategy to collect the top-level classes in each data set, and then extract properties that appear more than the frequency threshold θ . The selected instances, properties, and

top-level classes are used for performing machine learning methods.

3.2.1 Decision Table

The Decision Table is a simple rule-based supervised learning algorithm that leads to high performance with a simple hypothesis [19]. The Decision Table algorithm can retrieve a subset of core properties that can predict unlabeled instances with high accuracy. Therefore, the properties retrieved by the Decision Table play an important role in the data description.

We convert the instances of the linked data sets into data that is adaptable to the Decision Table algorithm. The data consists of a list of weights of the properties and the class labels. The weight represents the importance of a property in an instance and the labels are top-level classes. The weight of a property in an instance is calculated in a similar way as the Term Frequency-Inverse Document Frequency (TF-IDF), which is often used as a weighting factor in information retrieval and text mining [24]. The TF-IDF value reflects how important a word is to a document of a collection or a corpus. Similarly, in the Decision Table algorithm, the weight of each property in an instance is defined as the product of the property frequency (PF) and the inverse instance frequency (IIF). The property frequency $pf(prop, inst)$ is the frequency of the property $prop$ in the instance $inst$.

The inverse instance frequency of the property $prop$ in the data set D is $iif(prop, D)$, calculated as follows:

$$iif(prop, D) = \log \frac{|D|}{|inst_{prop}|}$$

where $inst_{prop}$ indicates an instance that contains the property $prop$. The value of $iif(prop, D)$ is the logarithm of the ratio between the number of instances in D and the number of instances that contain $prop$. If $prop$ appears in $inst$, the weight of $prop$ is calculated according to the following equation:

$$weight(prop, inst) = pf(prop, inst) \times iif(prop, D)$$

The properties retrieved in each data set by the Decision Table are critical for describing instances in that data set. Thus, we use these retrieved properties and top-level classes as parts of the final integrated ontology.

3.2.2 Apriori

An association rule learning method can extract a set of properties occurring frequently in the instances of a specific class. Apriori is a classic association rule mining algorithm, designed to operate on the databases of transactions. A frequent itemset is an itemset whose support is greater than the user-specified minimum support. Each instance in a specific class represents a transaction, and the properties that describe

the instance are treated as items. Hence, the frequent itemsets represent the frequently used properties for describing the instances of a specific class. The frequent core properties can be recommended to data publishers or help them finding missing important descriptions of the instances.

For each instance, a top-level class and all the properties that appear in the instance are collected as a transaction. The Apriori algorithm can extract associated sets of properties that occur frequently in the instances of a top-level class. Hence, the retrieved sets of properties are essential for describing the instances of a specific class. Furthermore, we can either identify commonly used properties in each data set or unique properties used in the instances of each class. Therefore, the properties extracted with the Apriori algorithm are necessary for the integrated ontology.

3.3 Integrated Ontology Constructor

The third component is an integrated ontology constructor, which merges the ontology classes and properties extracted from the previous two components. The graph-based ontology integration component outputs groups of related classes and properties, whereas the machine learning-based approach outputs a set of core properties retrieved by the Decision Table and a set of properties along with a top-level class retrieved by the Apriori algorithm. The global integrated ontology can help us to easily access various data sets and discover missing links. Furthermore, the domain information of the properties is automatically added using the results of the Apriori algorithm.

To construct an easily understandable ontology, we enrich the definition of the retrieved ontology classes and properties by adding annotations, and domain and range information. This integrated ontology constructor mainly consists of ontology enrichment, ontology merger, and naming validator. In the following, we describe each part in detail.

3.3.1 Ontology Enrichment

Most of the retrieved ontology classes and properties lack clear definitions. Therefore, an ontology enrichment method is necessary for a better understanding of the ontology definitions and the relations between the classes and properties. We enrich the retrieved classes and properties in the following way:

- *Annotation*: We collect all the default annotation definitions of the classes and properties from the data sets. In this process, for each group of classes and properties, we simply remove the duplicated annotations and the simple annotations that are included in the more comprehensive ones.

- *Domain*: The domain information of a property should be included in the integrated ontology because it indicates the relation between a property and a class. This information can help users to easily understand the kinds of properties that can be used for a specific class. To retrieve the domain information of a property, we randomly select m number of samples of instances having the property. Then we collect all the class information of the sample instances and iteratively do the sampling process for n times. The class information can be collected by tracking with properties `rdf:type`, `skos:inScheme`, etc. Then we analyze the collected class information to retrieve the proper domain information for a property. We choose the most frequently appearing classes as the domains of a property, which are the classes that appear in almost every sample instance. However, we observed that some classes are also frequently used, but are missing in a few instances. Hence, we set a frequency threshold for the domain retrieval as $0.95 * Freq_{top}$, where $Freq_{top}$ is the highest frequency of a class. If we could not retrieve the frequent class information or default definition of domain information, we set `owl:Thing` as the domain information.
- *Range*: The range information of a property is also important for users when they create SPARQL queries or publish data sets. However, most of the ranges are missing and sometimes the values are published in various ranges. To retrieve the range information, we also use the same sample instances described above. Then we analyze the values of the properties in the sample instances. We can retrieve the built-in data types by tracking the symbol “`^^`”. For other values for which we do not expressly show the data types, we classify them into two types: Resource and String. If the value contains resource information, we classify it as a Resource, otherwise we consider it as a String.

3.3.2 Ontology Merger

We adopt OWL 2 for constructing an integrated ontology. During the merging process, we also add relations between classes and properties so that we can easily identify the kinds of properties used to describe the instances of a specific class. We obey the following rules to construct the integrated ontology, where “`ex-onto`” and “`ex-prop`” are the prefixes of the integrated ontology.

- *Class* Related classes are collected from the graph-based ontology integration component and the top-level classes in each data set are collected from the machine learning-based ontology entity extraction component.
 1. *Groups of classes from the graph-based ontology integration*

Related classes from different data sets are extracted by analyzing *SameAs* graph patterns and then grouped into $cgroup_1, cgroup_2, \dots, cgroup_z$. We define $ex-onto : ClassTerm$ for each group, where *ClassTerm* is the most frequent term in the group. For all $c_i \in cgroup_k$, $\langle ex-onto : ClassTerm_k, ex-prop : hasMemberClasses, c_i \rangle$ is added automatically.

2. *Classes from the machine-learning-based approach* Top-level classes in each data set are added to the integrated ontology. If a top-level class $c_i \notin cgroup_k (1 \leq k \leq z)$, we create a new group $cgroup_{z+1}$ for each class c_i and create a new term $ex-onto : ClassTerm_{z+1}$ for the new group. Then we add a triple $\langle ex-onto : ClassTerm_{z+1}, ex-prop : hasMemberClasses, c_i \rangle$.

- *Property* The extracted properties from two components are merged according to the following rules. First, we extract the existing property type and the domain information of each property from the data sets. The property type is mainly defined by `rdf:Property`, `owl:DataTypeProperty`, and the object property `owl:ObjectProperty`. If the type is not clearly defined, we set the type as `rdf:Property`.

1. *Groups of properties from graph-based ontology integration* Related properties from various data sets are extracted by analyzing the *SameAs* graph patterns and then grouped into $pgroup_1, pgroup_2, \dots, pgroup_p$. For each group, we choose the most frequent term $ex-onto:propTerm$. Next, for each property $prop_i \in pgroup_t (1 \leq t \leq p)$, we add a triple $\langle ex-onto : propTerm_t, ex-prop : hasMemberProperties, prop_i \rangle$ and the triple $\langle ex-onto : propTerm_t, rdfs:domain, dInfo \rangle$, where *dInfo* is retrieved domain information of $prop_i$ in the ontology enrichment process.
2. *Properties from machine learning-based approach* We automatically add domain information for the properties retrieved by the Apriori method. For each property $prop$ extracted from the instances of class c , $\langle prop, rdfs : domain, c \rangle$ is automatically added, if it is not defined in the data set.

3.3.3 Naming Validator

The naming validator corrects the terms that have a naming format different from others. In the pitfall catalog introduced in the OOPS! (Ontology Pitfall Scanner!) system [31], consistent naming criteria is suggested for validating the ontology quality. In our regulation, we do not allow any characters,

such as “-”, “_”, and “/”, in the terms. We convert names containing words separated by these special characters into camelCase.

4 Experiments

In this section, we introduce the experimental data sets and then discuss experimental results with the Decision Table and the Apriori algorithm, which retrieve the top-level classes and the frequent core properties. The comparison results with other ontology matching tools are also discussed using ontology reference alignments. Last, we evaluate the quality of the integrated ontology with an ontology validator and ontology reference alignments.

4.1 Data Sets

We selected DBpedia (v3.6), Geonames (v2.2.1), NYTimes, and LinkedMDB from the LOD cloud to evaluate FITON. DBpedia is a cross-domain data set with approximately 8.9 million URIs and more than 232 million RDF triples. Geonames is a geographic domain data set with more than 7 million distinct URIs. NYTimes and LinkedMDB are both from the media domain with 10,467 and 0.5 million URIs, respectively.

Figure 5 shows the SameAs links connecting the above four data sets, as plotted by Cytoscape [33]. In this figure, the size of a node is determined by the total number of distinct instances in a data set on a logarithmic scale. The thickness of an arc is determined by the number of SameAs links as labeled on each arc on a logarithmic scale.

The number of instances in our database is listed in the second column of Table 3. The graph-based ontology integration component uses all the instances in the data sets. However, for the machine learning methods, we randomly choose samples of the data sets to speed up the modeling process, as well as to use an unbiased data size for each top-level class. We randomly select 5,000 instances per top-level class in Geon-

ames and LinkedMDB, 3,000 instances per top-level class in DBpedia, and use all the instances in NYTimes. The number of selected instances of DBpedia is less than 84,000, because some classes include less than 3,000 instances.

The original number of classes and properties, the number of top-level classes and the selected properties for machine learning methods are listed in Table 3. We track the subsumption relations such as owl:subClassOf and skos:inScheme to collect the top-level classes. Since the number of properties in the data sets is large, we filter out infrequent properties that appear less than the frequency threshold θ . For each data set, we manually set a different frequency threshold θ as \sqrt{n} , where n is the total number of instances in the data sets.

4.2 Decision Table

The Decision Table algorithm is used to discover a subset of features that can achieve high prediction accuracy with cross-validation. Hence, we apply the Decision Table to retrieve the core properties essential in describing instances of the data sets. For each data set, we execute the Decision Table algorithm to retrieve core properties by analyzing randomly selected instances of the top-level classes. In this experiment, we evaluate whether the retrieved sets of properties are important for describing instances by testing their performance on instance classification.

In Table 4, we list the percentage of the weighted averages of precision, recall, and F -measure. The precision is the ratio of correct results to all the results retrieved, and the recall is the percentage of retrieved relevant results to all relevant results. The F -measure is a measure of a test’s accuracy, and it considers both the precision and the recall. The F -measure is the weighted harmonic mean of the precision and recall, calculated as follows:

$$F\text{-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F -measure reaches its best value at 1 and its worst value at 0. A higher F -measure value means the retrieved subset of properties can well classify the instances; this implies that these properties are important for describing the instances of a specific class. A lower F -measure fails to classify some instances, because the retrieved properties are commonly used in every instance. In the following, we discuss the experimental results using the Decision Table algorithm in each data set.

- *DBpedia*. The Decision Table algorithm retrieved 53 DBpedia properties from 840 selected properties. For example, the properties db-onto:formationYear, db-prop:city, db-prop:debut, and db-prop:stateName are extracted from DBpedia instances. The precision, recall, and F -

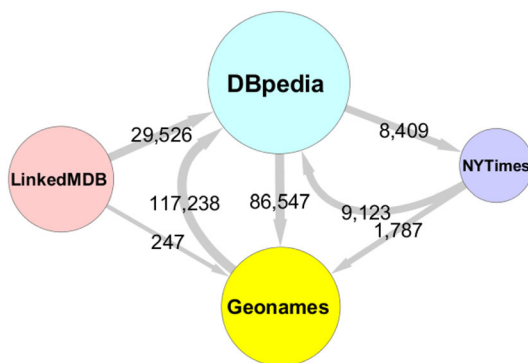


Fig. 5 SameAs links between data sets

Table 3 Data sets for experiments

Data set	Instances	Selected instances	Classes	Top-level classes	Properties	Selected properties
DBpedia	3,708,696	64,460	241	28	1,385	840
Geonames	7,480,462	45,000	428	9	31	21
NYTimes	10,441	10,441	5	4	8	7
LinkedMDB	694,400	50,000	53	10	107	60

Table 4 Results for the Decision Table algorithm

Data set	Average precision	Average recall	Average <i>F</i> -measure	Retrieved properties
DBpedia	0.892	0.821	0.837	53
Geonames	0.472	0.4	0.324	10
NYTimes	0.795	0.792	0.785	5
LinkedMDB	1	1	1	11

measure on DBpedia are 0.892, 0.821, and 0.837, respectively.

- *Geonames*. We retrieved 10 properties from 21 selected properties, such as `geo-onto:alternateName`, `geo-onto:countryCode`, and `wgs84_post:alt`, etc. Since all the instances of Geonames are from the geographic domain, the Decision Table algorithm cannot well distinguish different classes with these commonly used properties. Hence, the evaluation results on Geonames are very low with 0.472 precision, 0.4 recall, and 0.324 *F*-measure.
- *NYTimes*. Among the 7 properties used in the data set, 5 of them are retrieved using the Decision Table algorithm. We retrieved `skos:scopeNote`, `nyt:latest_use`, `nyt:topicPage`, `skos:definition`, and `wg-s84_pos:long`. In NYTimes, only a few properties describe news articles and most of them are commonly used in every instance. The cross-validation test results with NYTimes are 0.795 precision, 0.792 recall, and 0.785 *F*-measure.
- *LinkedMDB*. The algorithm can correctly classify all the instances in LinkedMDB with 11 properties retrieved from 60 properties. In addition to commonly used properties such as `foaf:page`, and `rfs:label`, we also extracted some unique properties such as `director_directorid`, `mdb:writer_writerid`, `md-b:performance_performanceid`, etc.

The experimental results show that the properties extracted from LinkedMDB can well distinguish the types of instances, because the properties are unique IDs of different types of instances. The performance of the classifications in DBpedia and NYTimes is lower because the retrieved properties contain commonly used properties. Since most of the instances in Geonames are described by common properties, the performance in predicting the types of instances is also low. We observed that the Decision Table either retrieves unique properties or commonly used properties, both of which are important for describing instances. Furthermore, instances of

the same class type are consistently described by the retrieved properties. The Decision Table retrieves core properties in the data sets; therefore, it is necessary for FITON to create an integrated ontology.

4.3 Apriori

The Apriori algorithm is a classic algorithm for retrieving frequent itemsets based on the transaction data. We list representative examples of the top-level class and the corresponding set of properties that are retrieved by the Apriori algorithm. Furthermore, we analyze the performance of the algorithm in each data set with examples. For the experiment, we use the parameters for the upper and lower bound of minimum support as 1 and 0.2, respectively. We set the minimum confidence as 0.9. With a lower minimum support, we can retrieve more properties that frequently appear in the data.

We retrieve frequently appearing core properties using the Apriori algorithm. Some examples are listed in Table 5. The first column lists the experimental data sets, and the second column lists samples of the top-level classes in each data set. The third column lists some of the representative properties retrieved for each top-level class.

In DBpedia and LinkedMDB, we retrieved some unique properties in each class such as `db-onto:kingdom`, `db-onto:family`, `mdb:actor_name`, and `mdb:actor_netflix-_id`. We can easily identify the class type with the unique properties of IDs in LinkedMDB, but we need combinations of unique properties to identify the class type in some types of DBpedia. From Geonames and NYTimes, we only retrieved commonly used properties in the data sets. It was difficult to predict the class type with high accuracy. From the instances of Geonames, we found the commonly used properties `geo-onto:alternateName`, `wgs84_pos:alt`, and `geo-onto:countryCode`. NYTimes has only a few properties, such as property `wgs84_pos:long` in the `nyt:nytd_geo` class and

Table 5 Examples of the retrieved properties with the Apriori algorithm

Data set	Class	Properties
DBpedia	db:Event	db-onto:place, db-prop:date, db-onto:related/geo.
	db:Species	db-onto:kingdom, db-onto:class, db-onto:family.
	db:Person	foaf:givenName, foaf:surname, db-onto:birthDate.
Geonames	geo-onto:P	geo-onto:alternateName, geo-onto:countryCode
	geo-onto:R	wgs84_pos:alt, geo-onto:name, geo-onto:countryCode.
NYTimes	ny.t:nytd_geo	wgs84_pos:long
	nyt:nytd_des	skos:scopeNote
LinkedMDB	mdb:actor	mdb:performance, mdb:actor_name, mdb:actor_netflix_id.
	mdb:film	mdb:director, mdb:performane, mdb:actor, dc:date.

property skos:scopeNote in the nyt:nytd_des class, that are commonly used in every instance.

We retrieved frequent sets of properties in most of the cases except in the db:Planet class, because db:Planet contains 201 different properties for describing instances that are sparsely used. In addition, we retrieved only db-onto:title and rdfs:type from db:PersonFunction and only rdfs:type property from db:Sales. This is caused by the lack of descriptions in the instances: most of the instances in db:PersonFunction and db:Sales defined only the class information without other detailed descriptions.

The set of properties retrieved from each class implies that the properties are frequently used for the instance descriptions of the class. Hence, for each property *prop* retrieved from the instances of class *c*, we automatically added $\langle prop, rdfs:domain, c \rangle$ to assert that *prop* can be used for describing instances in class *c*. Therefore, we can automatically recommend the missing core properties for an instance based on its top-level class.

4.4 Comparison with Other Ontology Matching Tools

Many state-of-the-art ontology matching tools have been developed, but most of them accept only two ontologies as inputs. Furthermore, since these tools need ontologies for analysis, they cannot find alignments for data sets, such as NYTimes and LinkedMDB, that do not contain structured ontologies. Hence, we compare the alignments between DBpedia and Geonames with other ontology matching tools. The following two comparison experiments are conducted using DB-Geo alignments¹² and BLOOMS alignments¹³.

4.4.1 Comparison with DB-Geo Alignments

Since LOD schema alignments have no standard benchmark, an expert manually created some DB-Geo alignments between DBpedia and Geonames. The DB-Geo

Table 6 Comparison with DB-Geo alignment references

DBpedia-Geonames	AROMA	FITON
Precision	0.18	0.64
Recall	0.04	0.37
<i>F</i> -measure	0.07	0.47

alignments contain 49 reference alignments, such as (db-onto:totalPopulation and geo-onto:population), (db-onto:location and geo-onto:location), (db-onto:Airport and geo-onto:S.AIRP), and (db-onto:Mountain and geo-onto:T.MT). We compare FITON with the AROMA [7] ontology matching system, which can find alignments between DBpedia and Geonames.

The AROMA system uses association rule mining on the data for matching ontologies [7]. AROMA found 11 alignments between DBpedia and Geonames, where only 2 were correct alignments. As shown in Table 6, the precision, recall, and *F*-measure of the AROMA system are 0.18, 0.04, and 0.07, respectively. FITON found alignments with 0.64 precision, 0.37 recall, and 0.47 *F*-measure. We found 28 alignments in total, where 18 of them were correctly matched. The experimental results show that FITON performs much better than the AROMA system.

In some cases, we found correct matches, but some of them were categorized as incorrect mappings according to the manual alignments because of the misuses of the schemas in the real data. For example, in the reference alignments, db-onto:totalPopulation is matched to geo-onto:population, but db-onto:totalPopulation never appears in the DBpedia data set. In fact, the population is described with other properties, such as db-onto:populationTotal, db-prop:populationEstimate, and db-prop:population. FITON successfully grouped these properties with geo-onto:population.

4.4.2 Comparison with BLOOMS Alignments

The BLOOMS ontology matching approach utilizes the Wikipedia category hierarchy and constructs BLOOMS

¹² <http://ri-www.nii.ac.jp/~lihua/Alignments.xlsx>

¹³ <http://wiki.knoesis.org/index.php/BLOOMS>

forests to find alignments between two ontologies [15]. For evaluation, BLOOMS provides 48 reference alignments between DBpedia and Geonames. BLOOMS matched the geo-onto:SpatialThing to most of the DBpedia ontology classes. This result means BLOOMS matched all the geographic information in the DBpedia ontology classes to geo-onto:SpatialThing. However, in DB-Geo alignments, we matched the ontologies with more specific geographic feature codes. Hence, to compare the alignment results with BLOOMS and other systems with BLOOMS alignments, we assume all the feature codes of geonames are subclasses of geo-onto:SpatialThing.

According to [15], only S-Match [11] can find ontology alignments between DBpedia and Geonames, while RiMOM [22] causes errors and all other systems including BLOOMS failed to find ontology alignments. The failure in ontology alignment is caused by ontologies that have ambiguous meaning of the concepts or the absence of corresponding concepts in the target data set. However, FITON can find alignments for poorly structured data sets by analyzing the contents of the interlinked instances. In Table 7, we list the results of BLOOMS, RiMOM, and S-Match from the evaluation results in [15]. The precision of FITON is 0.65, which is approximately 3 times the precision of S-Match: this means 65% of the found alignments are correct with FITON. However, the recall is lower than that of S-Match, where S-Match can find all the alignments used in the evaluation. The *F*-measure of FITON and S-Match is the same, 0.37.

4.5 Evaluation of the Integrated Ontology

The final integrated ontology contains 135 classes and 453 properties that are grouped into 87 and 97 groups, respectively. In this section, we evaluate the integrated ontology using the OOPS! ontology validator and evaluate the quality using the ontology reference alignments created by an ontology expert.

4.5.1 Evaluation with OOPS! validator

The OOPS! (Ontology Pitfall Scanner!) analyzes whether an ontology contains anomalies or pitfalls [31]. Currently, they use 29 pitfalls from four dimensions, such as human

understanding, logical consistency, modeling issues, and real-world representation. We validated the integrated ontology constructed from previous work [38] and this improved FITON using the OOPS! validator. We did not perform any optimization on FITON for the evaluation with OOPS!.

The OOPS! detected 55 missing ranges, 9 missing domains, and different naming criteria pitfalls from the integrated ontology in our previous work. However, we did not detect any of these pitfalls in our current work, because we automatically added ranges, domains, and naming mistakes in the integrated ontology constructor component. In fact, we found 26 missing domains and automatically added the domain info by analyzing the object values. This finding shows that OOPS! did not detect all the missing domains of the properties.

With the integrated ontology constructor component, we successfully removed the pitfalls caused by missing domains, missing ranges, and inconsistent naming criteria. The domain information can help users to understand the relations between properties and classes, and the range information can help us to easily validate or add values for properties.

4.5.2 Evaluation with Ontology Reference Alignments

The quality of the integrated ontology is evaluated with the ontology reference alignments created by an expert familiar with the LOD data sets. The expert created alignments among DBpedia, Geonames, LinkedMDB, and NYTimes.

As shown in Table 8, the precision reaches 1 for the alignments of DBpedia-LinkedMDB, LinkedMDB-NYTimes, and Geonames-NYTimes. For DBpedia-Geonames and DBpedia-NYTimes we found some incorrect alignments, but we could not get any alignment for LinkedMDB-Geonames. The system performs best while finding the alignments between DBpedia and Geonames. One of the reasons is that most of the links are between DBpedia and Geonames, as shown in Fig. 5, while only 247 links are between LinkedMDB and Geonames, which is not sufficient number of links to find correct alignments.

Table 7 Comparison with BLOOMS alignments

Geonames-DBpedia	Alignments from BLOOMS			
	FITON	BLOOMS	RiMOM	S-Match
Precision	0.65	0	err	0.23
Recall	0.26	0	err	1
<i>F</i> -measure	0.37	N/A	N/A	0.37

Table 8 Evaluation of the integrated ontology

Data pair	Precision	Recall	<i>F</i> -measure
DBpedia-Geonames	0.64	0.37	0.47
DBpedia-LinkedMDB	1	0.1	0.2
DBpedia-NYTimes	0.93	0.02	0.04
LinkedMDB-NYTimes	1	0.07	0.13
LinkedMDB-Geonames	0	0	n/a
Geonames-NYTimes	1	0.04	0.08

Since no statistics are available on the number of missing SameAs links among these data sets, it is difficult to judge how many links among data sets are necessary for FITON to perform effectively. However, according to the existing SameAs links in the data sets, FITON can perform well when at least 4% of the instances in the smaller data set are linked to the other data set. In addition, even with no direct links between LinkedMDB and NYTimes, FITON can perform well with the indirect links through DBpedia.

Since we only analyzed the interlinked instances for finding alignments, we cannot find alignments if no links or too few links exist between instances. Therefore, the recall is low between most of the data set pairs. Although it cannot find some alignments that the expert created, FITON could find some alignments that the expert did not discover. For example, nyt:nytd_geo, mdb-movie:country, geo-onto:A.PCLI, geo-onto:A.PCLD, and db-onto:Country are integrated as the members of ex-onto:Country with FITON. In the ontology enrichment process, FITON added the label “country” as the annotation of ex-onto:Country, which is the default annotation definition of db-onto:Country. Therefore, the users can easily understand that all the integrated classes represent countries. However, the expert could not discover that geo-onto:A.PCLI and geo-onto:A.PCLD are used to represent countries in Geonames, because these two feature codes have no annotation.

The expert could only match the given ontologies, but some of the ontology entities might be mistakenly used in the real data. For example, as listed in Table 9, among the 7 different properties that indicate the birthday of a person, only the property “db-onto:birthDate” has the domain definition with the class db-onto:Person and has the highest frequency of usage that appeared in 287,327 DBpedia instances. The second column in Table 9 represents the number of distinct instances using the property listed in the first column. From the definitions of the properties and the number of instances that contain the corresponding properties, we can assume that properties except “db-onto:birthDate” are mistakenly used when the data providers publish the DBpedia data. The property “db-onto:birthDate” is well defined

Table 9 Predicates grouped in ex-prop:birthDate

Property	Number of instances	rdfs:domain
db-onto:birthDate	287,327	db-onto:Person
db-prop:datebirth	1,675	N/A
db-prop:dateofbirth	87,364	N/A
db-prop:dateOfBirth	163,876	N/A
db-prop:born	34,832	N/A
db-prop:birthdate	70,630	N/A
db-prop:birthDate	101,121	N/A

with rdfs:domain and has the highest usage in the DBpedia instances. Therefore, we suggest “db-onto:birthDate” as the standard property to represent the birthday of a person, and correct the other properties with this standard property. Using FITON, we can integrate heterogeneous ontologies and also recommend standard ontology entities so that we can validate whether the ontologies are correctly used in the data.

It is time-consuming to find alignments for the data sets, especially for DBpedia, which has hundreds of classes and thousands of properties. Furthermore, it was impossible for the expert to find alignments of the heterogeneous ontologies. Therefore, FITON can dramatically decrease the time cost for finding the ontology alignments.

5 Discussion

In this section, we discuss possible applications with the graph patterns extracted by FITON and with the integrated ontology.

5.1 Discovering Missing Links with Graph Patterns

The graph-based ontology integration component retrieved 13 different graph patterns from the SameAs Graphs listed in Fig. 6 [37]. The labels of nodes M, D, N, and G represent the data sets LinkedMDB, DBpedia, NYTimes, and Geonames, respectively. The number on the right side of each graph pattern is the number of SameAs Graphs and is used to decide the frequency threshold for refining the sets of PO pairs.

The type of instances shared by the four data sets is Country. The integrated class that represents Country is ex-onto:Country, which consists of geo-onto:A.PCLI, geo-onto:A.PCLD, mdb:country, nyt:nytd_geo, and db-onto:Country. As we can see in Fig. 6, the graph patterns GP₉, GP₁₀, GP₁₁, and GP₁₂ are sub-graphs of GP₁₃. However, GP₁₃ is not a complete graph and has missing links between (M, N) and (M, D). Hence, with ex-onto:Country, we can link the missing links of countries among these four data sets. We can also add missing links for all the other incomplete graph patterns, such as GP₅, GP₆, GP₇, and GP₈.

5.2 Discovering Missing Links with Integrated Ontology

Here, we show two examples, shown in Table 10, that can find missing SameAs links with the integrated ontology. The first example finds the missing links for island instances between DBpedia and Geonames. Here, db-onto:Island and geo-onto:T.ISL are used for island instances in DBpedia and Geonames, respectively. Example 1 in Table 10 shows a SPARQL query to find the same islands that have the same name in DBpedia and Geonames. In total, we retrieved 509 links, among which 97 existing links are from DBpedia to

Fig. 6 SameAs graph patterns

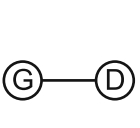
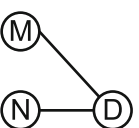
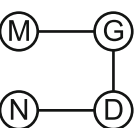
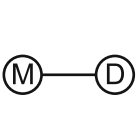
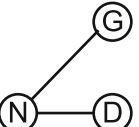
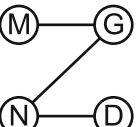
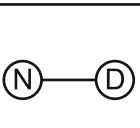
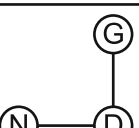
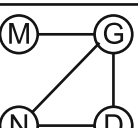
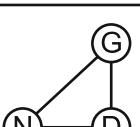
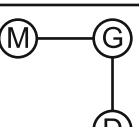
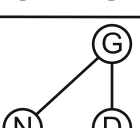
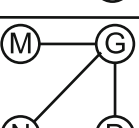
 GP ₁ 124,243	 GP ₆ 569	 GP ₁₁ 92
 GP ₂ 28,141	 GP ₇ 791	 GP ₁₂ 68
 GP ₃ 6,702	 GP ₈ 263	 GP ₁₃ 25
 GP ₄ 232	 GP ₉ 13	M LinkedMDB D DBpedia
 GP ₅ 403	 GP ₁₀ 45	N NYTimes G Geonames

Table 10 Finding missing links

Example 1: Link Islands

```
SELECT DISTINCT ?geo ?db ?string
where {
?geo geo-onto:featureCode geo-onto:T.ISL.
?geo ?gname ?string.
ex-onto:name ex-prop:hasMemberProperties ?gname.
?db rdf:type db-onto:Island.
ex-onto:name ex-prop:hasMemberProperties ?dname.
?db ?dname ?string. }
```

Example 2: Link Countries

```
SELECT DISTINCT ?geo ?db
where {
ex-onto:name ex-prop:hasMemberProperties ?gname.
{ ?geo geo-onto:featureCode go-onto:A.PCLI. }
UNION
{ ?geo geo-onto:featureCode geo-onto:A.PCLD. }
?geo ?gname ?string.
?db rdf:type db-onto:Country.
ex-onto:name ex-prop:hasMemberProperties ?dname.
?db ?dname ?string. }
```

Geonames, 211 links are from Geonames to DBpedia, and 90 bidirectional links are between DBpedia and Geonames. Hence, we discovered 291 missing links that have the same island name.

The second example finds missing links for country instances. The class db-onto:Country is integrated with geo-onto:A.PCLI and geo-onto:A.PCLD. With the SPARQL query in Example 2 in Table 10, we retrieved 663 links, including 221 existing SameAs links. Among the existing links 30 are from DBpedia to Geonames, 220 are from Geonames to DBpedia, and 29 links are bidirectional links. As a result, we discovered 442 new links for countries with exact matching on the names.

The above SPARQL examples show that we can find missing links with the integrated ontology using exact matching on the labels of instances. If we can analyze the labels with similarity string matching, we could discover more links with the integrated ontology. However, the SPARQL endpoint currently does not support a similarity matching query.

5.3 More Answers with the Integrated Ontology

The heterogeneous ontologies make it difficult to find proper properties to construct a SPARQL query. However, the integrated ontology grouped all kinds of properties used in real data sets, including properties that are not defined in the ontology. Hence, we can discover more query results simply using the classes and properties in the integrated ontology. Here, we discuss the two SPARQL examples shown in Table 11. We selected two questions from the QALD-1 Open Challenge.¹⁴

¹⁴ <http://www.sc.cit-ec.uni-bielefeld.de/qald-1>

Table 11 SPARQL examples from QALD-1 Open Challenge

Standard Query	Query with the Integrated Ontology
Give me all the cities with more than 10,000,000 inhabitants	
<pre>SELECT DISTINCT ?uri ?string WHERE { ?uri rdf:type db-onto:City. ?uri db-prop:populationTotal ?inhabitants. FILTER (?inhabitants > 10000000). OPTIONAL { ?uri rdfs:label ?string. FILTER (lang(?string) = 'en') }}</pre>	<pre>SELECT DISTINCT ?uri ?string WHERE { ?uri rdf:type db-onto:City. ex-onto:population ex-prop:hasMemberProperties ?prop. ?uri ?prop ?inhabitants. FILTER (?inhabitants > 10000000). OPTIONAL { ?uri rdfs:label ?string. FILTER (lang(?string) = 'en') }}</pre>
How tall is Claudia Schiffer?	
<pre>SELECT DISTINCT ?height WHERE { res:Claudia_Schiffer db-onto:height ?height. }</pre>	<pre>SELECT DISTINCT ?height WHERE { ex-onto:height ex-prop:hasMemberProperties ?hprop res:Claudia_Schiffer ?hprop ?height. }</pre>

The first query is to find the cities that have a population of more than 10 million. The SPARQL query on the left side is the standard one given by the QALD-1 Open Challenge, which utilizes the db-onto:City class and the standard property db-prop:populationTotal. The query on the right side uses all the population properties used in the real DBpedia. As a result, we retrieved 20 distinct cities with our SPARQL query, in comparison with only 9 cities retrieved with the standard query.

The second query is to answer the height of Claudia Schiffer. With the standard property db-onto:height, we can retrieve the answer of 1.8(m). However, with our integrated ontology, which consists of db-onto:height, db-prop:height, db-prop:heightIn, db-prop:heightFt, and db-onto:Person/height, we found one more answer, 180.0 (centimeter), which uses a different measurement unit.

Since we integrated properties that are not defined in the ontologies, although commonly used in real data sets, we can retrieve more answers with the integrated ontology. Hence, it is helpful for discovering more related answers with simple queries.

6 Conclusion and Future Work

In this paper, we introduced FITON, which constructs an integrated ontology for Semantic Web developers. The framework consists of three main components: graph-based ontology integration, machine-learning-based approach, and integrated ontology constructor. The graph-based ontology integration component retrieves related ontology classes and properties by analyzing the graph patterns of the interlinked instances. This component reduces the heterogeneity of ontologies in the LOD cloud. The integrated ontology contains top-level classes and frequent core properties retrieved from the machine-learning-based approach. This component

helps Semantic Web application developers to more easily find core properties used for a specific instance. We also enriched the integrated ontology with annotations, domain, and range information, and validated it during the ontology construction process. With the integrated ontology, we can also detect misuses of the ontologies in data sets and recommend core properties for describing instances. Furthermore, we can detect missing links using the integrated ontology and use it for QA systems to retrieve more related results.

In future work, we would like to compare FITON with other latest state-of-the-art ontology alignment tools such as BLOOMS+ [16], which is an improved version of BLOOMS. However, BLOOMS+ is not publicly available and the developers have only provided comparison results of linked data sets with PROTON ontology. Hence, it is difficult to compare the newer version with FITON. We will perform a comparison between FITON and BLOOMS+ when that tool is available. We will also work on a link discovery system that can utilize the integrated ontology to find missing links in linked data sets. By iteratively feeding discovered missing links to the data sets, we can update the integrated ontology with newly linked instances. Moreover, we can retrieve more missing links by applying similarity matching methods on the object values in the link discovery process.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

References

1. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th international conference on very large data, bases, pp 487–499

2. Banerjee S, Pedersen T (2003) Extended gloss overlaps as a measure of semantic relatedness. Proceedings of the 18th international joint conference on Artificial intelligence. Morgan Kaufmann Publishers Inc., Los Altos, pp 805–810
3. Bechhofer S, van Harmelen F, Hendler J, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2004) OWL Web Ontology Language Reference. W3C Recommendation. <http://www.w3.org/TR/owl-ref/>
4. Bizer C, Heath T, Berners-Lee T (2009) Linked data - the story so far. *Int J Semant Web Infor Syst* 5(3):1–22
5. Brickley D, Guha R (2004) RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation. <http://www.w3.org/TR/rdf-schema/>
6. Choi N, Song IY, Han H (2006) A survey on ontology mapping. *ACM SIGMOD Rec* 35:34–41
7. David J, Guillet F, Briand H (2006) Matching directories and owl ontologies with AROMA. In: Proceedings of the 15th ACM international conference on information and knowledge management, pp 830–831. ACM
8. Ding L, Shinavier J, Shanguan Z, McGuinness DL (2010) SameAs networks and beyond: analyzing deployment status and implications of owl:sameAs in linked data. In: Proceedings of the 9th international semantic web conference, pp 145–160
9. Euzenat J (2001) Towards a principled approach to semantic interoperability. In: Proceedings of IJCAI workshop on ontologies and information sharing, pp 19–25
10. Euzenat J, Shvaiko P (2007) *Ontology matching*. Springer, Berlin
11. Giunchiglia F, Shvaiko P, Yatskevich M (2004) S-match: an algorithm and an implementation of semantic matching. In: Proceedings of the 1st European semantic web symposium. LNCS. Springer, Berlin, pp 61–75
12. Heath T, Bizer C (2011) *Linked data: evolving the web into a global data space*. Morgan & Claypool, San Rafael
13. Hirst G, St Onge D (1998) Lexical chains as representation of context for the detection and correction malapropisms. The MIT Press, Cambridge, pp 305–332
14. Ichise R (2010) An analysis of multiple similarity measures for ontology mapping problem. *Int J Semant Comput* 4(1):103–122
15. Jain P, Hitzler P, Sheth AP, Verma K, Yeh PZ (2010) Ontology alignment for linked open data. Proceedings of the 9th international semantic web conference, vol 6496, LNCS. Springer, Berlin, pp 402–417
16. Jain P, Yeh PZ, Verma K, Vasquez RG, Damova M, Hitzler P, Sheth AP (2011) Contextual ontology alignment of lod with an upper ontology: a case study with proton. In: Proceedings of the 8th extended semantic web conference on the semantic web. Springer, Berlin, pp 80–92
17. Jiang J, Conrath D (1997) Semantic similarity based on corpus statistics and lexical taxonomy. In: Proceedings of the international conference on research in, computational linguistics, pp 19–33
18. Klein M (2001) Combining and relating ontologies: an analysis of problems and solutions. In: Proceedings of IJCAI workshop on ontologies and information sharing, pp 53–62
19. Kohavi R (1995) The power of decision tables. In: Proceedings of the 8th European conference on machine learning. Springer, Berlin, pp 174–189
20. Le NT, Ichise R, Le HB (2010) Detecting hidden relations in geographic data. In: Proceedings of the 4th international conference on advances in semantic processing, pp 61–68
21. Leacock C, Miller GA, Chodorow M (1998) Using corpus statistics and wordnet relations for sense identification. *Comput Linguist* 24(1):147–165
22. Li J, Tang J, Li Y, Luo Q (2009) Rimom: a dynamic multistrategy ontology alignment framework. *IEEE Trans Knowl Data Eng* 21(8):1218–1232
23. Lin D (1998) An information-theoretic definition of similarity. In: Proceedings of the 15th international conference on machine learning. Morgan Kaufmann Publishers Inc., Los Altos, pp 296–304
24. Manning CD, Raghavan P, Schütze H (2008) *Introduction to information retrieval*. Cambridge University Press, Cambridge
25. Meilicke C, Völker J, Stuckenschmidt H (2008) Learning disjointness for debugging mappings between lightweight ontologies. Proceedings of the 16th international conference on knowledge engineering and knowledge management. LNCS, vol 5268. Springer, Berlin Heidelberg, pp 93–108
26. Parundekar R, Knoblock CA, Ambite JL (2010) Linking and building ontologies of linked data. In: Proceedings of the 9th international semantic web conference, pp 598–614
27. Parundekar R, Knoblock CA, Ambite JL (2012) Discovering concept coverings in ontologies of linked data sources. In: Proceedings of the 11th international semantic web conference, LNCS, vol 7649. Springer, Berlin, pp 427–443
28. Patwardhan S (2003) Incorporating dictionary and corpus information into a context vector measure of semantic relatedness. Master's thesis, University of Minnesota
29. Pavel S, Euzenat J (2013) Ontology matching: state of the art and future challenges. *IEEE Trans Knowl Data Eng* 25(1):158–176
30. Pedersen T, Patwardhan S, Michelizzi J (2004) Wordnet:similarity: measuring the relatedness of concepts. In: Proceedings of the 19th national conference on artificial intelligence, association for computational linguistics, pp 1024–1025
31. Poveda-Villalón M, del Carmen Suárez-Figueroa M, Gómez-Pérez A (2012) Validating ontologies with OOPS! In: Knowledge engineering and knowledge management. LNCS, vol 7603. Springer, Berlin, pp 267–281
32. Resnik P (1995) Using information content to evaluate semantic similarity in a taxonomy. In: Proceedings of the 14th international joint conference on artificial intelligence, pp 448–453
33. Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, Ideker T (2003) Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res* 13(11):2498–2504
34. W3C OWL Working Group: OWL 2 Web Ontology Language Document Overview. W3C Recommendation (2012). <http://www.w3.org/TR/owl2-overview/>
35. Winkler WE (2006) Overview of record linkage and current research directions. Tech. rep., Statistical Research Division U.S. Bureau of the Census
36. Wu Z, Palmer M (1994) Verbs semantics and lexical selection. In: Proceedings of the 32nd annual meeting on association for computational linguistics. Association for, computational linguistics, pp 133–138
37. Zhao L, Ichise R (2012) Graph-based ontology analysis in the linked open data. In: Proceedings of the 8th international conference on semantic systems, pp 56–63. ACM
38. Zhao L, Ichise R (2013) Instance-based ontological knowledge acquisition. In: Proceedings of the 10th extended semantic web conference. Springer, Berlin, pp 155–169
39. Zhao L, Ichise R (2013) Integrating ontologies using ontology learning approach. *IEICE Trans Inform Syst* E96-D(1):40–50